МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

МІКРОПРОЦЕСОРНА ТЕХНІКА

Підручник

Друге видання, доповнене
і перероблене

Дніпро
НТУ «ДП»
2022

УДК 004.31 (075.8)
М 59

Рекомендовано вченою радою як підручник для здобувачів-бакалаврів вищих навчальних закладів спеціальностей 151 Автоматизація та комп’ютерно-інтегровані технології і 152 Метрологія та інформаційно-вимірювальна техніка (протокол № 12 від 22.07.2021).

Рецензенти:

С.А. Положаєнко, д-р техн. наук, проф., завідувач кафедри комп’ютеризовані системи керування Одеського національного політехнічного університету;

А.І. Купін, д-р техн. наук, проф., завідувач кафедри комп'ютерних систем і мереж ДВНЗ «Криворізький національний університет»;

А.Й. Наконечний, д-р техн. наук, проф., завідувач кафедри комп’ютеризовані системи управління Національного університету «Львівська політехніка».

Мікропроцесорна техніка : підручник / В.В. Ткачов, С.М. Проценко, М.В. Козар, В.І Шевченко; М-во освіти і науки України, НТУ «Дніпровська політехніка». – 2-ге вид., допов. і переробл. – Дніпро : НТУ «ДП». – 2022. – 230 с.

ISBN 978–966–350–773–6

Розглянуто питання побудови архітектури мікроконтролерів, сигналів керування й системи команд однокристальних мікроконтролерів MCS-8051 та MCS 1T8051, організації режимів переривань, тимчасових затримок, передачі інформації в послідовному форматі. Наведено приклади створення систем керування об’єктами з неперервними і дискретними характеристиками.

Зміст видання відповідає програмі дисципліни «Мікропроцесорна техніка», яку вивчають студенти спеціальностей 151 Автоматизація та комп’ютерно-інтегровані технології, 152 Метрологія та інформаційно-вимірювальна техніка, а також для студентів інших спеціальностей.

УДК 004.31 (075.8)

ISBN 978–966–350–773–6

ЗМІСТ

ПЕРЕДМОВА

Дисципліна «Мікропроцесорна техніка» є нормативною, оскільки компетенції, які формуються дисципліною визначені стандартом бакалавра спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології.

Завдання предмета – формування здатності використовувати знання фізики, електротехніки, електроніки і мікропроцесорної техніки в обсязі, необхідному для розуміння процесів, що відбуваються у системах автоматизації та комп'ютерно-інтегрованих технологіях, вміння обґрунтовувати вибір структури та розробляти прикладне програмне забезпечення для мікропроцесорних систем керування на базі локальних засобів автоматизації, промислових логічних контролерів та програмованих логічних матриць і сигнальних процесорів.

Знань з організації архітектури сучасних мікропроцесорів, мікроконтролерів, пам'яті та системи команд, паралельного й послідовного інтерфейсів, режимів тимчасових затримок і переривань, а також набуття навичок розробки алгоритмів і програм керування неперервними і дискретними об'єктами.

Підручник «Мікропроцесорна техніка» враховує досвід багатьох вчених зі створення і впровадження сучасних мікропроцесорних технологій керування, з удосконалення лабораторної бази дисципліни, методичного та мультимедійного забезпечення, а також значного оновлення елементної бази мікропроцесорної техніки.

Матеріал підручника відповідає лекційному курсу дисципліни, яка впродовж кількох років викладалася у Національному технічному університеті «Дніпровська політехніка». Підручник може бути використаний студентами заочної і дуальної форм навчання. З цією метою під час викладення теоретичного матеріалу наводяться приклади розв'язання задач, наприкінці кожного розділу містяться контрольні питання, надається програма та методичні вказівки для лабораторного практикуму.

З урахуванням навчального призначення у підручнику не робиться бібліографічних посилань на запозичення з відомих джерел. Список літератури містить тільки вказівки на матеріали, що рекомендуються як додаткові для поглиблення знань та розширення навичок у галузі мікропроцесорної техніки.

Автори із вдячністю отримають зауваження та побажання, що сприятимуть подальшому покращенню змісту підручника.

1. ЗАГАЛЬНІ ПОНЯТТЯ І СТРУКТУРА СИСТЕМ КЕРУВАННЯ

Курс лекцій «Мікропроцесорна техніка» базується на дисциплінах «Теоретичні основи електротехніки», «Вимірювання електричних і неелектричних величин», «Електроніка й мікросхемотехніка», «Програмування», «Елементи і пристрої систем керування».

З грецької мови слово «система» перекладається як «складний», «багатокомпонентний», «взаємозалежний».

Під системою керування (СК) розуміється комплекс логічно-взаємодіючих механічних, апаратних і програмних засобів, а також рішень, прийнятих людиною, яка виконує завдання контролю й керування технологічним процесом або об'єктом.

Технологічний процес – це послідовність дій і операцій перетворення вихідної сировини, матеріалу або енергії з метою одержання товарної продукції із заданими параметрами.

Керування – це сукупність впливів на керований об'єкт, спрямованих на підтримку його функціонування відповідно до вибраних правил на базі інформації, що надходить від об'єкта керування й зовнішнього середовища, у якому він функціонує. У випадку, коли керуючі впливи виконуються без участі людини, керування називається автоматичним. Якщо ж деякі функції під час вибору керуючих дій виконуються людиною, то керування називається автоматизованим. За аналогією системи також поділяються на автоматичні й автоматизовані. Узагальнена структурна схема такої СК зображена на рис. 1.1.

Наведена схема не є абсолютною з погляду «тільки так і ніяк інакше», але вона дає можливість найбільш повно уявити сукупність компонентів СК й розібратися з їхніми призначенням і взаємозв'язками.

Для СК джерелами інформації, як правило, є датчики, рішення, прийняті людиною, а також інші системи або підсистеми, з якими обумовлено спільне функціонування.

За допомогою датчиків виконується контроль параметрів технологічного процесу, зокрема, переміщення або положення машин і механізмів, зусилля, температура, тиск, що спостерігаються, і т.ін. Тому датчики – це обладнання, де відбувається перетворення контрольованої фізичної величини в електричний сигнал. Вони поділяються на два основні типи – дискретні й аналогові, і узагальнено їх можна назвати технологічними.

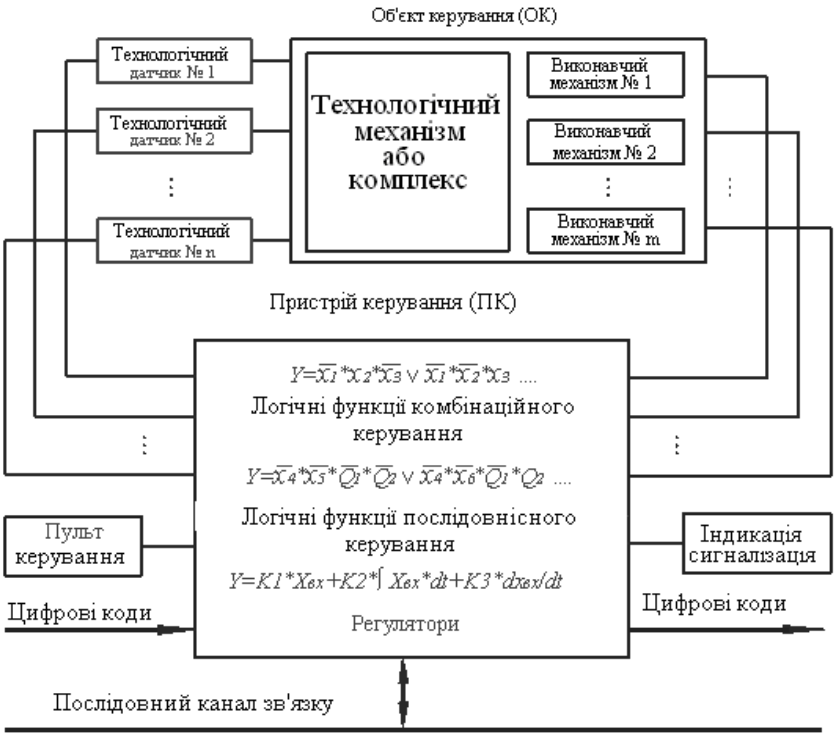


Рисунок 1.1 – Структурна схема системи керування

У датчиках дискретного типу зміна вихідного сигналу відбувається східчасто – від мінімального значення до максимального, або навпаки. В аналогових датчиках вихідний сигнал змінюється безупинно, залежно від зміни контрольованого технологічного параметра, і в кожний момент часу може приймати величину, що перебуває між мінімальним і максимальним значеннями. До групи аналогових слід віднести також датчики, вихідними параметрами яких є зміна тривалості сигналу або частоти вихідних імпульсів. У кожному разі ці параметри пропорційні зміні контрольованої фізичної величини.

Інформація від людини передається в систему через систему пультів керування (кнопки, що задають елементи, маніпулятори), а також через комп'ютерні засоби шляхом вибору й впровадження у дію відповідних завдань, програм або режимів.

Інформація від підсистем, з якими обумовлено спільне функціонування, може надходити як у вигляді окремих сигналів (дискретних або аналогових), так і цифрових кодів, переданих по паралельних або послідовних каналах зв'язку.

Процес керування передбачає зняття інформації з об'єкта (вимірювання параметрів об'єкта або процесу), аналіз цієї інформації й створення керуючих впливів згідно із заданим законом керування.

Керований об'єкт, яким може бути технологічний механізм, лінія або комплекс, виділено в окремий прямокутник ОК. Під ним мається на увазі деяка

сукупність виконавчих обладнань і механізмів, що приводять у дію технологічне устаткування. Ними можуть бути електричні, пневматичні або гідравлічні приводи, нагрівальні елементи, маслостанції, компресори тощо. У цю частину входять також магнітні пускачі, контактори, силові перетворювачі, гідравлічні й пневматичні розподільники, через які енергія надходить до відповідних приводів і на які безпосередньо спрямовані керуючі впливи від пристроїв керування (ПК).

На входи ПК можуть надходити аналогові й дискретні сигнали від технологічних датчиків і пультів керування, а також інформація у вигляді цифрових кодів у тих випадках, коли за умов перешкодозахищеності кодування знижує ймовірність її спотворення. Джерелом вхідної інформації є також і послідовний канал зв'язку, за допомогою якого може відбуватися інформаційний обмін і взаємодія між суміжними підсистемами й верхніми рівнями ієрархії всієї структури підприємства.

По суті, пристрій керування є основним елементом СК, що сприймає вхідну інформацію, перетворює її за заданими законами й алгоритмами і створює керуючі впливи на ОК. Крім цього, ПК виконує функції надання інформації обслуговуючому персоналу у вигляді світлової й звукової сигналізації або графічної візуалізації, а також спільно взаємодіючим підсистемам і вищим за ієрархією рівням керування.

1.1. Дискретні або цифрові системи керування

Якщо припустити, що об'єкт керування відомий, тобто відомі виконавчі механізми та засоби комутації енергії до них, вибрані відповідні датчики, визначені зовнішні зв'язки, сформульовані необхідні вимоги до алгоритмів керування й параметрів регулювання, то залишається створити або розробити ПК. За своєю суттю ПК є пристроєм перетворення вхідної інформації у вихідну. Але оскільки вхідні і вихідні сигнали неоднакові, то при розробці необхідно застосовувати різні елементи і пристрої як аналогові, так і дискретні.

Дискретні способи перетворення сигналів мають декілька безперечних переваг. Насамперед, це велика точність перетворення, висока перешкодозахищеність і загальна надійність як способів перетворення, так і пристроїв, що їх реалізують. Дискретне перетворення дає реальну можливість використовувати складні алгоритми обробки сигналів з точним цифровим відображенням вхідної й вихідної інформації. Самі пристрої мають відносно низьку вартість. Зазначені переваги такі, що у деяких випадках вигідніше змінити характер вхідних або вихідних сигналів для збереження однорідності функцій перетворення.

У системах керування, де вхідними й вихідними є дискретні сигнали, побудова самих пристроїв керування найчастіше зводиться до розробки рішень, основаних на законах логічного керування, описуваних відповідними логічними функціями. До них відносяться пристрої, що реалізують комбінаційне або послідовнісне керування. У першому випадку значення функцій виходів залежать тільки від комбінацій вхідних дискретних сигналів, що розглядаються як аргументи або логічні змінні. У другому випадку вихідні функції формуються так само, але з урахуванням стану елементів пам'яті. У класиці дискретної логіки це розглядається як комбінаційні автомати без пам'яті й послідовнісні автомати з пам'яттю.

Вирішуючи питання дискретизації при реалізації функцій неперервного регулювання, у технічних системах найчастіше використовують аналого-цифрові (АЦП) і цифро-аналогові (ЦАП) перетворювачі. Таким чином, вхідні аналогові сигнали перетворюються в послідовності кодів (чисел), які піддаються математичній обробці, а вихідні аналогові сигнали формуються з послідовності кодів (чисел), одержуваних у результаті цієї обробки. За зазначеним принципом будуються контури неперервного регулювання, де потрібно або підтримувати, або змінювати регульовану величину вихідного сигналу відповідно до вибраного закону керування. Такі регулятори одержали назву цифрових, а системи – цифрового керування.

1.2. Основи обчислювальної техніки

1.2.1. Основи булевої алгебри

Для опису роботи дискретних і цифрових пристроїв використовується математичний апарат алгебри логіки або булевої алгебри. Основними поняттями булевої алгебри є логічна змінна й логічна функція.

Логічна змінна – це величина, яка може приймати тільки два можливі значення, одне з яких за твердженням вважається «справжнім» а друге – «помилковим». У булевій алгебрі справжнє значення змінної позначається символом «1» (логічна одиниця), неправильне – символом «0» (логічний нуль). Самі змінні частіше позначають символами x_1, x_2, ..., x_n. У силу визначення логічні змінні можна називати також двійковими.

Логічна функція – це двійкових змінних (аргументів), яка також може приймати тільки одне з двох значень, яке так само, за твердженням, може бути справжнім (що дорівнює «1») або помилковим (що дорівнює «0»). Значення деякої логічної функції n змінних задається для кожної комбінації двійкових

змінних, кількість можливих різних наборів яких дорівнює 2^n. При цьому, оскільки сама функція на кожному наборі може приймати значення «0» або «1», то загальне число можливих функцій від n змінних дорівнює $2 \cdot 2^n$. Позначається логічна функція звичайно символом y.

Для безлічі значень, які можуть приймати як аргументи, так і функції, в булевій алгебрі визначається відношення еквівалентності, позначуване символом рівності «=», і три операції: логічного додавання (диз'юнкції), логічного множення (кон'юнкції), логічного заперечення (інверсії), позначувані відповідно символами:

$\vee$ (+) – операція диз'юнкції;

$\wedge$ (&) – операція кон'юнкції;

$\bar{*}$ – операція інверсії.

З огляду на постулати при виконанні перелічених операцій відношення еквівалентності мають такий вигляд:

а)	б)	в)
$0 + 0 = 0$	$0 \cdot 0 = 0$	$\bar{0} = 1$
$0 + 1 = 1$	$0 \cdot 1 = 0$	$\bar{1} = 0$
$1 + 0 = 1$	$0 \cdot 1 = 0$	
$1 + 1 = 1$	$1 \cdot 1 = 1$	

(1.1)

На підставі постулатів (1.1) можна вивести такі співвідношення (закони) алгебри логіки:

1. Закони одинарних елементів: універсальної множини – а); нульової множини – б); тавтології – в).

а)	б)	в)
$x + 1 = 1$	$x + 0 = x$	$x+x= x$
$x \cdot 1 = x$	$x \cdot 0 = 0$	$x \cdot x = x$

(1.2)

2. Закони заперечення: подвійного заперечення – а); додатковості – б), подвійності – в).

а)	б)	в)
$\bar{\bar{x}} = x$	$x + \bar{x} = 1$	$\overline{x_1 + x_2} = \bar{x}_1 \cdot \bar{x}_2$
	$x \cdot \bar{x} = 0$	$\overline{x_1 \cdot x_2} = \bar{x}_1 + \bar{x}_2$

(1.3)

3. Закони абсорбції або поглинання – а), склеювання – б).

$$\begin{array}{cc} \text{а)} & \text{б)} \\ x_1 + x_1 \cdot x_2 = x_1 & x_1 \cdot x_2 + x_1 \cdot \bar{x}_2 = x_1 \\ x_1 \cdot (x_1 + x_2) = x_1 & (x_1 + x_2) \cdot (x_1 + \bar{x}_2) = x_1 \end{array} \qquad (1.4)$$

Крім законів, зазначених вище, які не мають аналогів у звичайній алгебрі (алгебрі чисел), для алгебри логіки справедливі закони звичайної алгебри: комутативні або переставні, дистрибутивні або розподільні, асоціативні або сполучні.

Будь-яка логічна функція y для n двійкових змінних $x_1, x_2, ..., x_n$ може бути задана за допомогою таблиць. Вони одержали назву таблиць істинності, що містять 2^n рядків, у які записуються всі можливі двійкові набори значень аргументів, а також відповідне кожному із цих наборів значення функції.

У дискретній і цифровій схемотехніці широко використовуються функції заперечення, кон’юнкції, заперечення кон’юнкції, диз'юнкції, заперечення диз'юнкції й додавання за модулем два. Для реалізації цих функцій розроблені й виготовляються інтегральні мікросхеми. Їх часто називають базовими логічними елементами «НІ», «І», «І – НІ», «АБО», «АБО – НІ», а пристрої, синтезовані на їхній основі, називають комбінаційними автоматами або пристроями комбінаційного типу.

1.2.2. Базові логічні елементи

Найпростішою є функція заперечення, значення якої завжди протилежно значенню аргументу. Аналітична форма запису цієї функції має вигляд

$$y = \bar{x}. \qquad (1.5)$$

Функція заперечення, а відповідно й логічна операція «НІ» є унарною, тобто має всього один аргумент. Таблиця істинності цієї функції, умовна позначка елемента «НІ» і його тимчасові діаграми наведені на рис. 1.2.

Інверсія мовою дискретної або цифрової техніки означає, що значення вихідного сигналу (y) завжди протилежно значенню вхідного (x).

Логічні функції кон'юнкції і диз'юнкції є бінарними, тому що являють собою результати дій більш ніж над однією змінною.

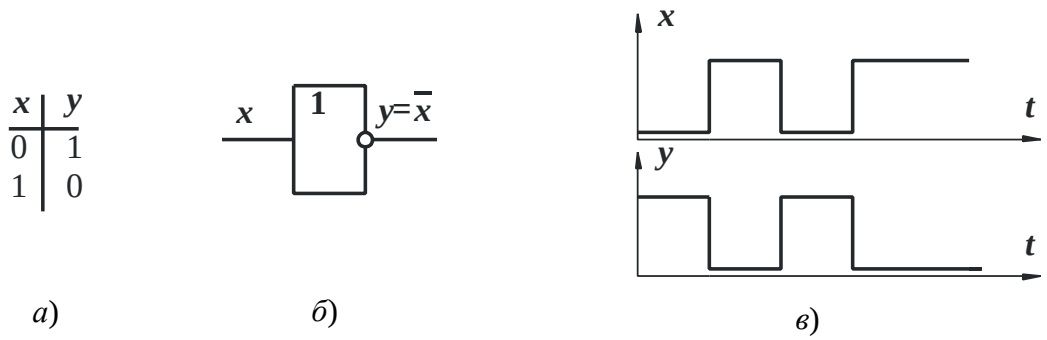


Рисунок 1.2 – Елемент «НІ»: *а*) – таблиця істинності; *б*) – умовне зображення; *в*) – тимчасові діаграми

Аналітична форма запису функції кон'юнкції двох аргументів x_1 і x_2 має вигляд

$$y = x_1 \cdot x_2 \text{, або } y = x_1 \wedge x_2 \text{, або } y = x_1 \,\&\, x_2 . \qquad (1.6)$$

Функція кон'юнкції дійсна або дорівнює 1 тоді й тільки тоді, коли всі її аргументи набувають істинного значення або дорівнюють 1.

Логічний елемент, що реалізує функцію кон'юнкції, називають кон'юнктором або елементом «І». На рис. 1.3 наведені таблиця істинності, умовне зображення, тимчасові діаграми двовходового кон'юнктора.

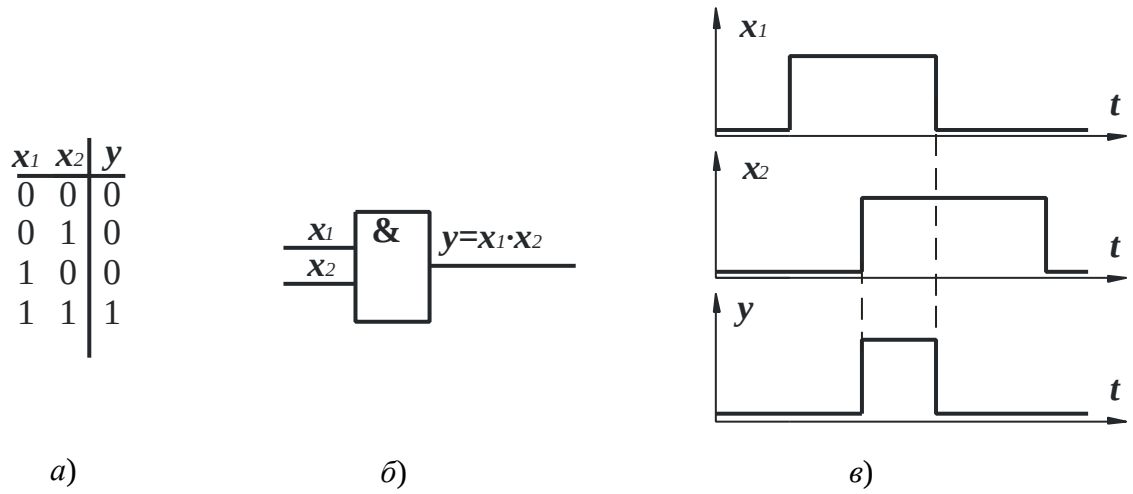


Рисунок 1.3 – Елемента «І»: *а*) – таблиця істинності, *б*) – умовне позначення, *в*) – тимчасові діаграми

Елементи «І» часто використовують для керування потоком інформації. При цьому на один з його входів надходять сигнали, що несуть деяку інформацію, а на інший – керуючий сигнал: пропустити інформацію – 1, не пропустити – 0.

Функція заперечення кон'юнкції («І – НІ») одержується шляхом інвертування самої функції кон'юнкції. Запис такої функції має такий вигляд:

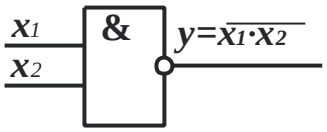
$$y = \overline{x_1 \cdot x_2}\,. \qquad (1.7)$$

Форма запису функції диз'юнкції двох аргументів x_1 і x_2 аналогічна:

$$y = x_1 + x_2 \quad \text{або} \quad y = x_1 \vee x_2\,. \qquad (1.8)$$

Умовне зображення елемента «І – НІ» і таблиця істинності функції (1.7) наведені на рис. 1.4.

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

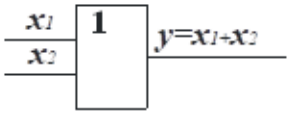


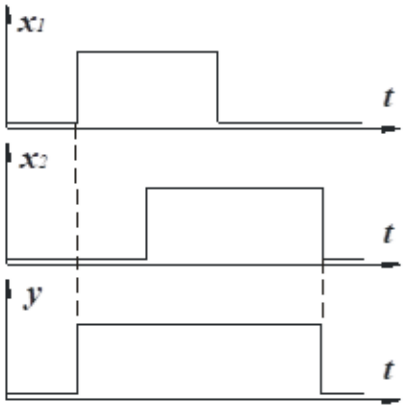
а) *б*)

Рисунок 1.4 – Елемент «І – НІ»: *а*) – таблиця істинності, *б*) – умовне зображення

Функція диз'юнкції істинна або дорівнює 1, якщо хоча б один з її аргументів набуває істинного значення або дорівнює 1. Така функція часто називається функцією логічного додавання. Логічний елемент, що реалізує функцію диз'юнкції, називають диз’юнктором або елементом «АБО». На рис. 1.5 для нього наведені таблиця істинності, умовне зображення, тимчасові діаграми.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1





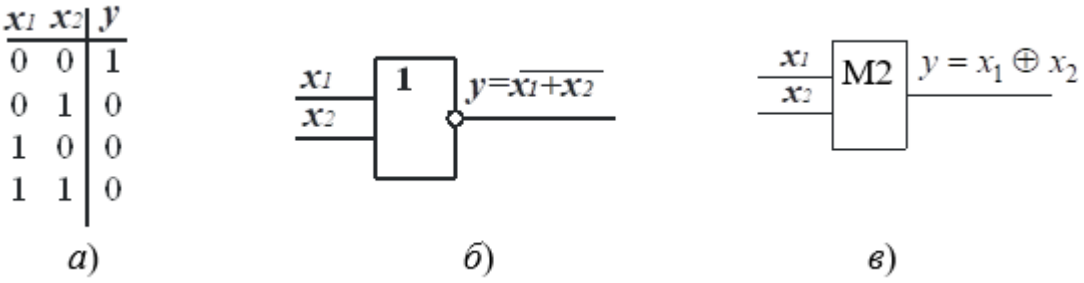
а) *б*) *в*)

Рисунок 1.5 – Елемент «АБО»: *а*) – таблиця істинності, *б*) – умовне позначення, *в*) – тимчасові діаграми

Функція заперечення диз'юнкції («АБО – НІ») одержується шляхом інвертування самої функції диз'юнкції та записується так:

$$y = \overline{x_1 + x_2}\,. \qquad (1.9)$$

Умовне зображення елемента «АБО – НІ» й таблиця істинності функції (1.9) наведені на рис. 1.6.



а) *б*) *в*)

Рисунок 1.6 – Елемент «АБО – НІ»: *а*) – таблиця істинності; *б*) – умовне зображення

Функція «додавання за модулем два» (М2) найчастіше використовується як функція двох аргументів (рис. 1.6, *в*). У цьому випадку її називають «виключне АБО», також її називають функцією «нерівнозначності». Аналітична форма запису такої функції має вигляд

$$y = x_1 \oplus x_2 . \tag{1.10}$$

Назва функції пов'язана з тим, що $x_1 \oplus x_2$ є арифметичною сумою двійкових чисел x_1 і x_2 у межах одного розряду: 0+0=1; 0+1=1; 1+0=1; 1+1=0. В останньому випадку виникаючий перенос у старший розряд ігнорується, а в розряді самих доданків одержуємо нуль. Логічні елементи такого типу широко застосовуються при синтезі підсумовуючих пристроїв.

Функція М2 має деяку властивість, яку корисно знати.

$$y = x_1 \oplus x_2 = \overline{x_1} \cdot x_2 \wedge x_1 \cdot \overline{x_2} . \tag{1.11}$$

Таким чином, логічна функція y дорівнює 1, якщо змінні x_1 і x_2 мають протилежні значення (0, 1), а при рівних значеннях (00, 11) функція дорівнює 0. У цьому випадку реалізується логічна функція «рівнозначності» $x_1 \equiv x_2$. Вона дорівнює 1, якщо $x_1 = x_2$. Очевидні також і співвідношення (1.12).

$$x \oplus 0 = x; \qquad x \oplus 1 = \overline{x}; \quad x \oplus x = 0; \quad x \oplus \overline{x} = 1. \tag{1.12}$$

1.2.3. Реалізація логічних елементів (ЛЕ)

Релейно-контактні логічні елементи

Логічну функцію, яка визначає залежність стану вихідних сигналів обладнання керування від вхідних, можна реалізувати на релейно-контактних логічних (РКЛ) елементах, що являють собою електромагнітні реле. Вхід такого елемента – обмотка,

а вихід – стан контактів. Звичайно на схемах зображуються не обмотки, а тільки контакти реле. Логічна змінна x може бути зумовлена станом контакту, а також ухвалювати одне із двох значень. Значенню «істина» відповідає замкнений стан контакту, значенню «хибний» – розімкнений (рис. 1.7). Часто «істинний» і «хибний» стани відповідають термінам «високий» і «низький» рівень, «так» – «ні», «увімкнено» – «вимкнено», «одиниця (1)» – «нуль (0)».

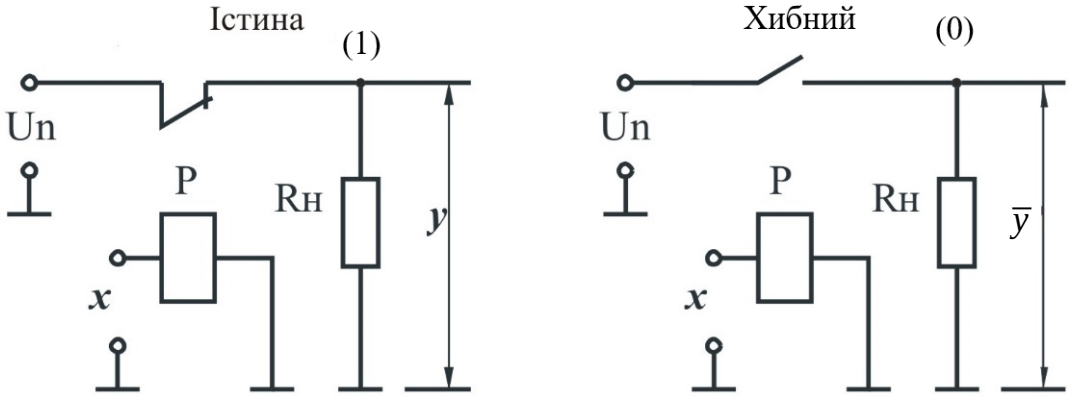


Рисунок 1.7 – Релейно-контактні логічні елементи

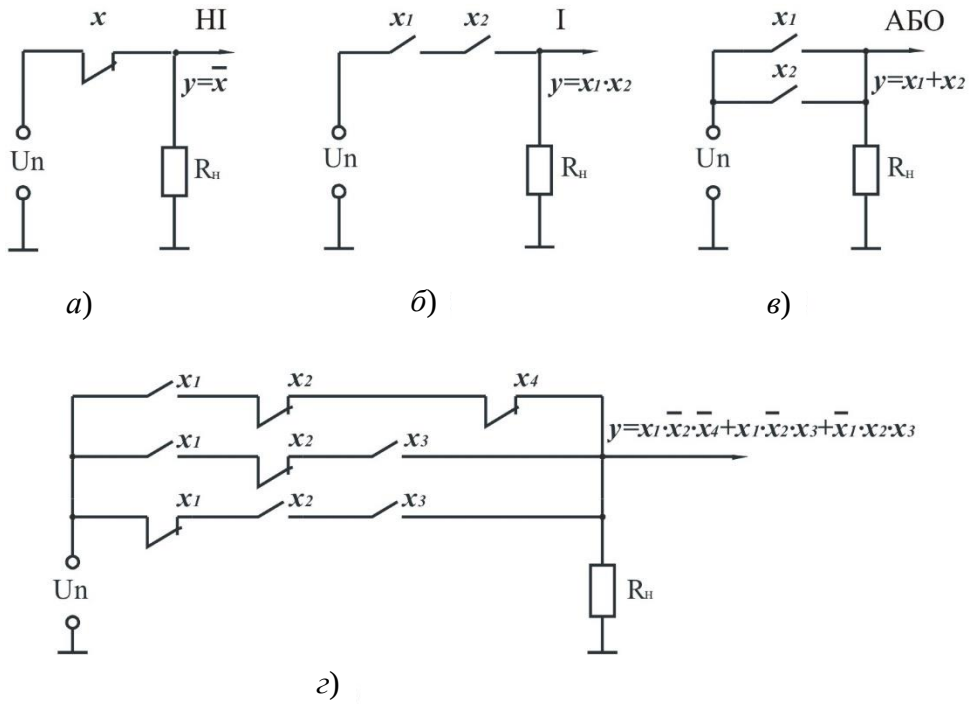
Реалізація функцій диз'юнкції, кон’юнкції, інверсії на релейно-контактних елементах показана на рис. 1.8. Функцію інверсії «НІ» можна реалізувати, використовуючи розмикальний контакт реле x (рис. 1.8, *а*). Функція кон’юнкції припускає послідовне увімкнення контактів, і вона буде дорівнювати 1, якщо всі змінні одночасно на вході дорівнюють 1 (рис. 1.8, *б*). Оскільки функція диз'юнкції дорівнює 1, якщо хоча б одна змінна на вході дорівнює 1, то це рівносильно паралельному увімкненню контактів реле (рис. 1.8, *в*).

Приклад реалізації складної логічної функції на релейно-контактних елементах наведено на рис. 1.8, *г*.

Діодні логічні елементи

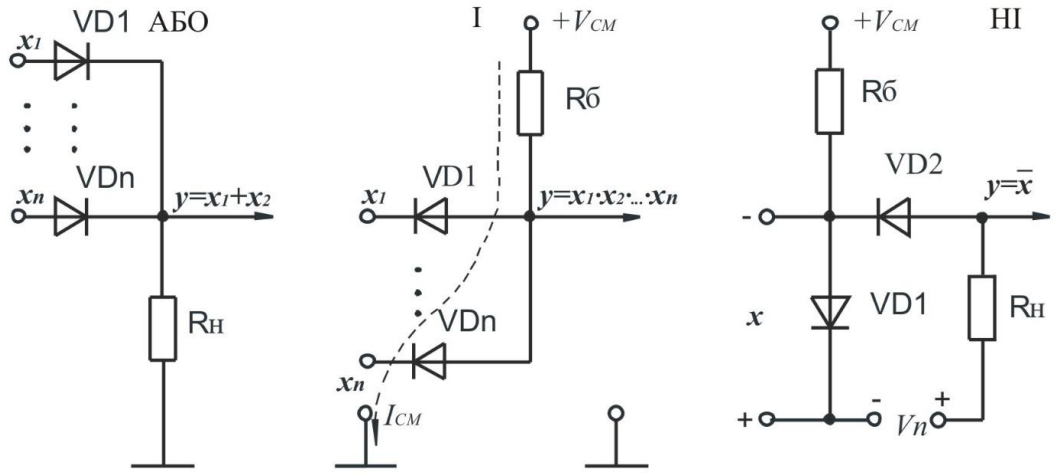
Реалізувати основні логічні операції «І», «АБО», «НІ» можна на напівпровідникових діодах. У схемі «АБО» (рис. 1.9, *а*) не потрібно додаткового джерела енергії, тому що під час подачі позитивного потенціалу на один з виходів x через відповідний діод і резистор $R_н$ проходить струм, під дією якого на $R_н$ спадає напруга, що відповідає на виході логічній одиниці. Функція «І» (рис. 1.9, *б*) реалізується з використанням додаткового джерела зсуву $U_{зс}$. За відсутності вхідних сигналів x_1, x_2, ..., x_n через резистор $R_б$, діоди $VD1$, ..., VDn і вхідні опори джерел сигналу x проходить струм зсуву, і вся напруга

виділяється на резисторі $R_б$, а на виході – низький потенціал (0). Під час подачі на вхід сигналів x діоди $VD1, ..., VDn$ замикаються, тому на виході буде високий потенціал (1). Якщо навіть на одному із входів відсутній вхідний сигнал, то на виході буде 0, що й відповідає функції «І».



а) *б*) *в*) *г*)

Рисунок 1.8 – Реалізація логічних функцій на РКЛ-елементах



а) *б*) *в*)

Рисунок 1.9 – Діодні логічні елементи

Функція «НІ» реалізується за допомогою двох джерел живлення (рис. 1.9, *в*). Якщо відсутній вхідний сигнал x, то діод $VD1$ відкритий і на виході буде високий потенціал (1) від Vn. Якщо на вхід поданий негативний потенціал, то діод $VD1$

закривається, струм через $R_н$ не проходить, і на виході потенціал буде дорівнювати 0. Мала навантажувальна здатність і велике згасання сигналів – основні недоліки діодної ЛЕ.

Транзисторні логічні елементи

Транзисторні ЛЕ набули найбільшого поширення, оскільки мають декілька схемних рішень реалізації логічних функцій.

Реалізація основних логічних операцій з використанням транзисторів наведена на рис. 1.10. Операція «АБО» (рис. 1.10, *а*) реалізується, якщо відкритий один із транзисторів $VT1$ або $VT2$, тобто $y = x_1 \vee x_2$; операція «І» (рис. 1.10, *б*) реалізується, якщо відкриті транзистори $VT1$ й $VT2$, тобто $y = x_1 \cdot x_2$. Операція інверсії реалізується за допомогою звичайного ключа з навантаженням у колі колекторам (рис. 1.10, *в*). За відсутності сигналу на вході $(x = 0)$ транзистор VT закритий, і на виході буде високий потенціал (1). За наявності сигналу на вході $(x = 1)$ транзистор відкривається, потенціал на виході дорівнює потенціалу «загальної точки» (тобто «0»).

Використовуючи діод-транзисторні ЛЕ, можна реалізувати логічну функцію у функціонально повному базисі «І – НІ» (рис. 1.10, *г*), а застосовуючи резистивно-транзисторні ЛЕ – у функціонально повному базисі «АБО – НІ» (рис. 1.10, *д*).

Транзисторно-транзисторні логічні елементи (ТТЛ) з'явилися внаслідок їх подальшого розвитку завдяки заміні діодів багатоемітерним транзистором. Основна відмінність багатоемітерного транзистора від звичайного полягає в тому, що він має декілька емітерів, розташованих таким чином, що пряма взаємодія між ними виключена.

ТТЛ дозволяють реалізувати логічні функції у функціонально повному базисі «І – НІ» (рис. 1.10, *е*). Якщо на входи x_1 й x_2 подано високий потенціал, то проходить струм I_1, який відкриває транзистори $VT2$ й$VT4$, і на виході буде низький потенціал $(y = 0)$. Якщо навіть на один із входів x_1 або x_2 подано низький потенціал (тобто заземлено один із входів), то в цьому випадку стане проходити струм I_2, транзистори $VT2$ й $VT4$ будуть закриті, і на виході утворюється високий потенціал $(y = 1)$.

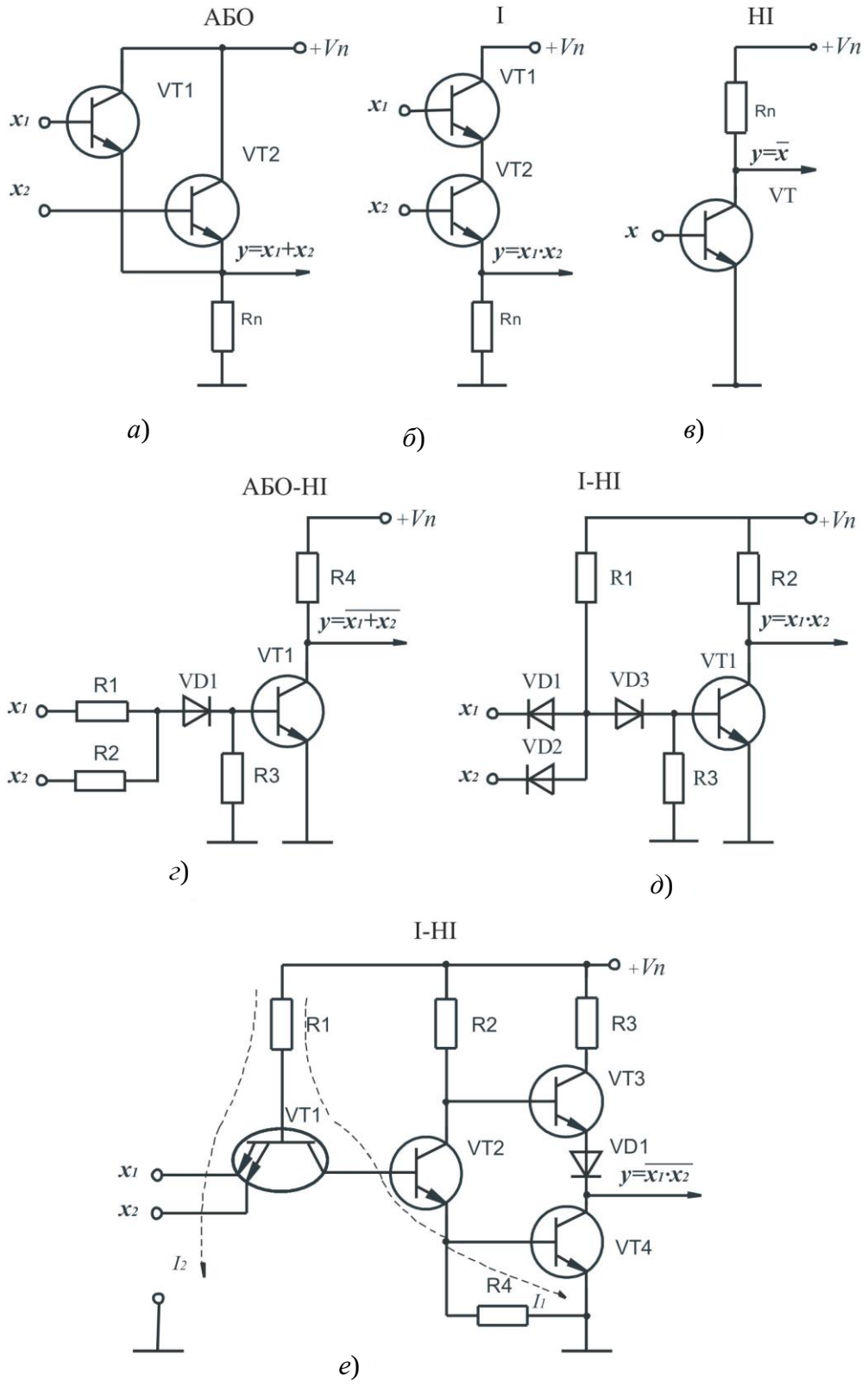


Рисунок 1.10 – Транзисторні логічні елементи

ТТЛ-елементи застосовуються винятково в інтегральних схемах, що мають високу щільність упаковки електрично з'єднаних елементів і кристалів, коли вартість транзисторів не є великою, а простота проектованого пристрою – не головна вимога. Елемент, наведений на рис. 1.10, *е*, є базовим або типовим для ТТЛ і виготовляється серійно з різною кількістю входів – від двох до восьми.

1.2.4. Комбінаційні пристрої

Комбінаційні пристрої повністю описуються логічними функціями. У деяких випадках при реалізації пристроїв керування об'єктами їх виявляється цілком достатньо. Але це тільки ті випадки, коли функції керування залежать тільки від станів або комбінацій вхідних сигналів.

В обчислювальній і мікропроцесорній техніці до пристроїв комбінаційного типу належать дешифратори, мультиплексори, суматори, тригери й деякі інші. Побудову таких пристроїв буде розглянуто нижче.

Дешифратори, мультиплексори, демультиплексори

Дешифраторами називають цифрові пристрої комбінаційного типу, що мають декілька входів і виходів, у яких певним комбінаціям вхідних сигналів відповідають цілком певні комбінації вихідних сигналів. Іноді такі пристрої називають декодерами. Різноманітність дешифраторів дуже велика, але в мікропроцесорній техніці звичайно використовуються так звані лінійні дешифратори.

Лінійним дешифратором називається пристрій, що здійснює перетворення n-розрядного двійкового коду в m-розрядний унітарний код. Унітарний код може бути прямим, якщо одна «1» є у деякому розряді m-розрядного двійкового коду і $m-1$ нулів, або зворотним, якщо один «0» і $m-1$ одиниць. Наприклад, запис унітарного коду для $m=4$ буде мати такий вигляд:

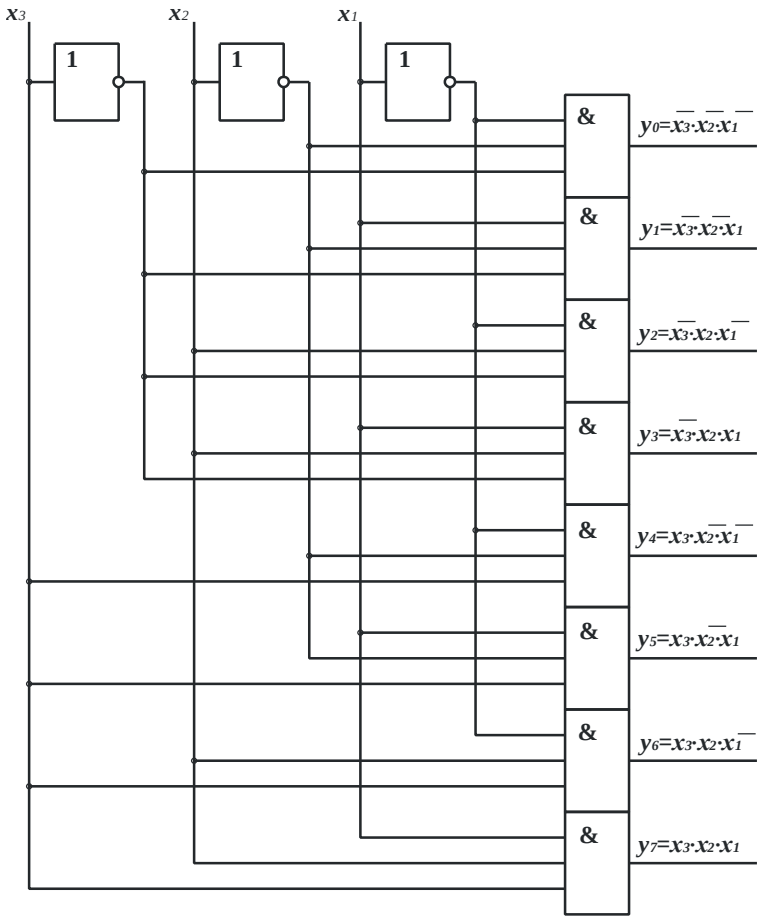
прямого – 1000, ..., 0010, 0001
зворотного – 0111, ..., 1101, 1110

Схема дешифратора має n входів, на які надходять відповідні розряди двійкового коду x_n, x_{n-1}, ..., x_2, x_1, й m виходів, на яких формуються розряди унітарного коду y_{m-1}, ..., y_1, y_0. В табл. 1.1 наведено приклад для дешифратора, що має $n=3$ входів і $m=2^n=8$ виходів. Функціональна схема й умовне графічне позначення показані на рис. 1.11.

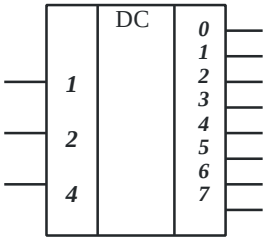
Функціональна схема дешифратора побудована відповідно до рівнянь $y_7 ... y_0$ і являє собою вісім кон'юнкторів з трьома входами, кожний з яких реалізує одну з вихідних функцій. Якщо виходи дешифратора проінвертувати, то вийде дешифратор зі зворотним унітарним кодом, або, як його ще називають, дешифратор з інверсними виходами.

Таблиця 1.1 – Приклад дешифратора, в якого n дорівнює трьом входам

x_3 x_2 x_1	y_0 y_1 y_2 ... y_6 y_7
0 0 0	1 0 0 ... 0 0
0 0 1	0 1 0 ... 0 0
0 1 0	0 0 1 ... 0 0
.....	.....
1 1 0	0 0 0 ... 1 0
1 1 1	0 0 0 ... 0 1



а)



б)

Рисунок 1.11 – Функціональна схема дешифратора – *а*) та її умовне позначення – *б*)

Варіант схемної реалізації мультиплексора «4–1» і його умовне графічне зображення наведені на рис. 1.12.

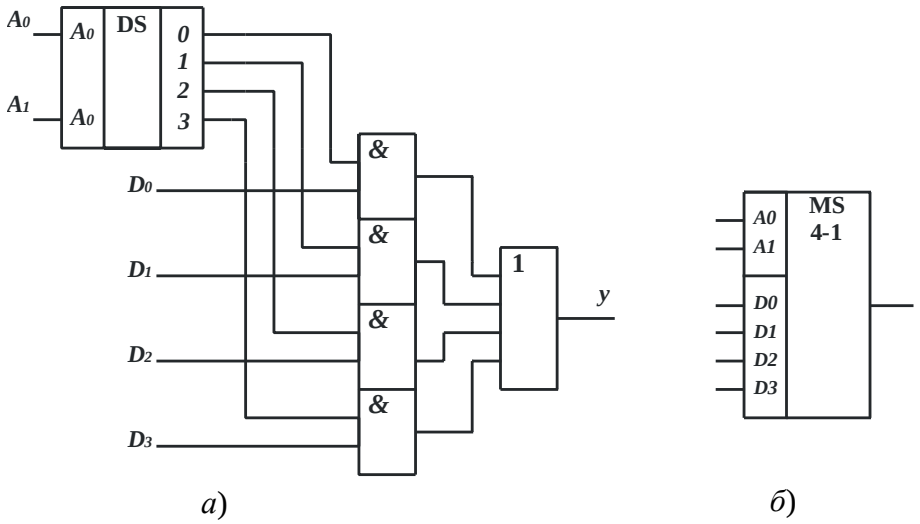


Рисунок 1.12 – Функціональна схема мультиплексора – *а*) та її умовне зображення – *б*)

Мультиплексор являє собою сукупність дешифраторів двійкового адресного коду й чотирьох двовходових елементів «І», виходи яких об'єднані елементом «АБО». Таким чином залежності від адресної комбінації A_0, A_1 на вихід мультиплексора (y) передається стан відповідного входу $Д_i$.

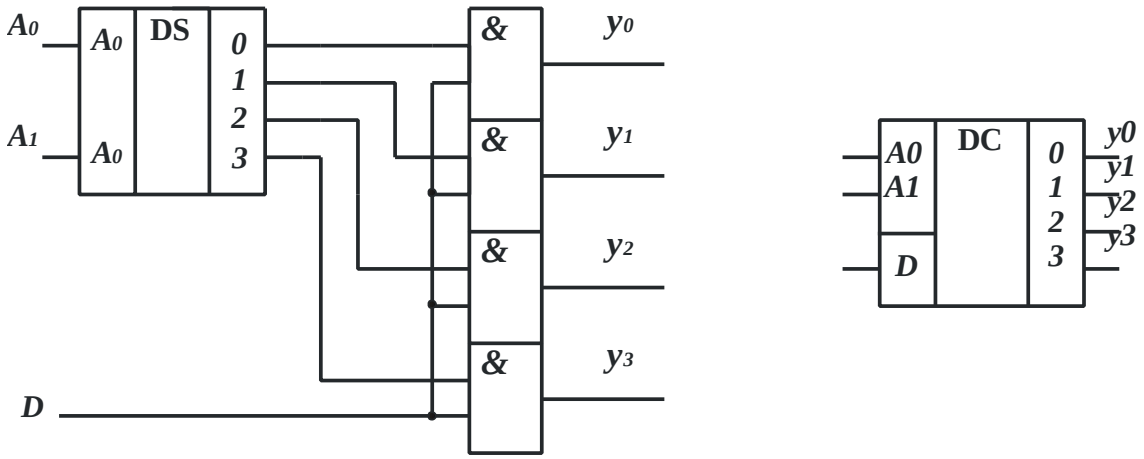
Демультиплексор – це схема, що виконує функцію, яка є зворотною для функції мультиплексора, тобто це комбінаційна схема, що має один інформаційний вхід Д, n інформаційних виходів $y_0, y_1, \ldots, y_{n-1}$ і k керуючих (адресних) входів $A_0, A_1, \ldots, A_{k-1}$. Двійковий код, що надходить на адресні входи, визначає один з n виходів, на який передається значення змінної з інформаційного входу Д.

Приклад функціонування демультиплексора, що має $n = 4$ інформаційних виходів (y_0, y_1, y_2, y_3) і $k = 2$ адресних входів(A_0, A_1), розглянутий в табл. 1.2.

Таблиця 1.2 – Приклад функціонування демультиплексора, який має 4 виходи

$Д$	A_0, A_1	$в_0\ в_1\ в_2\ в_3$
0	0 0	0 0 0 0
1	0 0	1 0 0 0
0	0 1	0 0 0 0
1	0 1	0 1 0 0
0	1 0	0 0 0 0
1	1 0	0 0 1 0
0	1 1	0 0 0 0
1	1 1	0 0 0 1

Схема демультиплексора і його умовне зображення наведена на рис. 1.13.



а) *б*)

Рисунок 1.13 – Функціональна схема демультиплексорів – *а*) та її умовне зображення – *б*)

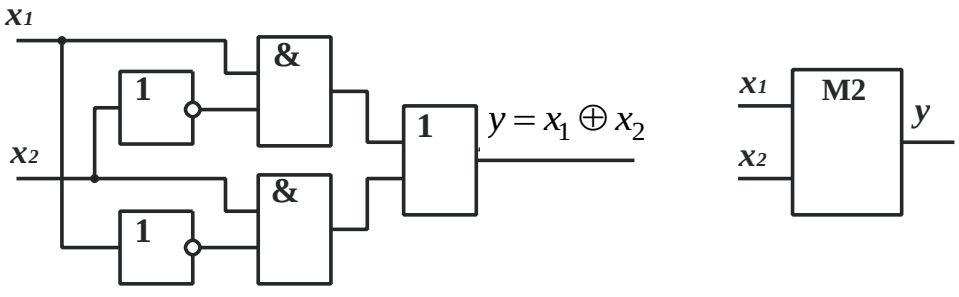
Схему реалізовано за допомогою тривходових елементів «І» і двох інверторів. Передача сигналу з входу Д на один з виходів буде відповідати адресній комбінації на входах A_0, A_1.

1.2.5. Арифметичні пристрої

Напівсуматор

Напівсуматор – це пристрій, що реалізує функцію «сума за модулем 2» (М2). Напівсуматори застосовуються при побудові арифметико-логічних пристроїв (АЛП) як елементи, що реалізують функцію нерівнозначності, і як складові частини повних арифметичних суматорів. Таблиця істинності функції, її функціональна схема й умовне зображення наведено на рис. 1.14. Очевидно, що на виході напівсуматора буде одиниця тільки при комбінаціях вхідних сигналів 01 і 10.

x_2	x_1	y
0	0	0
0	1	1
1	0	1
1	1	0



а) *б*) *в*)

Рисунок 1.14 – Таблиця істинності – *а*), функціональна схема – *б*)
і умовне зображення – *в*) функції М2

Повні двійкові суматори

Правила виконання арифметичних операцій

Арифметичні належать до поширених операцій, що виконуються цифровими пристроями.

Правила виконання арифметичних операцій над двійковими числами є аналогічними до відповідних правил десяткової арифметики і зведені в табл. 1.3 і 1.4.

Таблиця 1.3 – Правила і приклад виконання арифметичного додавання

Двійкове додавання			
Доданки k-го розряду	Сума k-го розряду	Перенос у $k+1$-й розряд	Приклад
0 + 0 =	0	0	1100 – перенос
0 + 1 =	1	0	1101 – 1-й доданок
1 + 0 =	1	0	+ 1100 – 2-й доданок
1 + 1 =	0	1	11001 – сума

Таблиця 1.4 – Правила і приклад виконання арифметичного віднімання

Двійкове віднімання						
Зменшуване k-го розряду		Від'ємне k-го розряду		Різниця k-го розряду	Додатне з $k+1$-го розряду	Приклад
0	–	0	=	0	0	010 – додатне
0	–	1	=	1	1	1101 – зменшуване
1	–	0	=	1	0	– 1010 – від'ємне
1	–	1	=	0	0	0011 – різниця

Для виконання арифметичних операцій над двійковими числами зі знаком уводять додатковий (знаковий) розряд, який указує, чи є число додатним або від'ємним. Якщо число додатне, у знаковий розряд проставляється 0, якщо від'ємне, то 1. Наприклад, число (+5) з урахуванням знакового розряду (відділяється крапкою) запишеться як 0.101, а число (–3) як 1.011.

При додаванні чисел з однаковими знаками вони складаються, і сумі присвоюється код знака доданків, наприклад:

$$\begin{array}{r} +3 \\ +2 \\ \hline 5_{10} \end{array} \Rightarrow \begin{array}{r} 0.011 \\ +0.010 \\ \hline 0.101_2 \end{array} \qquad \begin{array}{r} -3 \\ -2 \\ \hline -5_{10} \end{array} \Rightarrow \begin{array}{r} 0.011 \\ +0.010 \\ \hline 0.101_2 \end{array}$$

Ускладняється операція додавання чисел з різними знаками (алгебраїчне додавання), що рівносильно відніманню чисел. У цьому випадку необхідно визначити більше за модулем число, зробити віднімання й присвоїти різниці знак більшого (за модулем) числа.

Для спрощення виконання цієї операції доданки зображуються у зворотному або додатковому кодах, оскільки відомо, що операція віднімання (алгебраїчного додавання) зводиться до операції простого арифметичного додавання двійкових чисел, зображених у зворотному або додатковому кодах. Додатні числа в прямому, зворотному й додатковому кодах мають той самий вид, а від'ємні– різний.

Щоб зобразити від'ємне двійкове число у зворотному коді треба поставити в знаковий розряд 1, а у всіх інших розрядах прямого коду замінити одиниці нулями, а нулі – одиницями, тобто проінвертувати число.

При записі від'ємного двійкового числа в додатковому коді треба поставити 1 у знаковий розряд, а інші розряди одержати зі зворотного коду числа за допомогою додавання 1 до молодшого розряду.

Наведемо приклади запису двійкових чисел зі знаками в прямому, зворотному й додатковому кодах.

Число	Прямий код	Зворотний код	Додатковий код
+6	0.110	0.110	0.110
-5	1.101	1.010	1.011
-11	1.1011	1.0100	1.0101

Пояснимо процедуру віднімання чисел 5 і 3, 3 і 5. Послідовність і взаємозв'язок операцій наведено у табл. 1.5.

З прикладів випливає, що при використанні зворотного коду в пристрої, який забезпечує підрахування багаторозрядних двійкових чисел, – двійковому суматорі, необхідно передбачити коло циклічного переносу. У випадку використання додаткового коду це коло відсутнє.

З наведеного вище можна зробити такий висновок. Застосування простого математичного «трюку» (подання двійкових чисел у зворотному або додатковому коді) дозволяє пристосувати двійковий суматор для виконання як операцій додавання двійкових чисел, так і операцій їх віднімання.

Більше того, за допомогою двійкового суматора можна забезпечити також виконання й операцій множення й ділення двійкових чисел (тобто всіх чотирьох

арифметичних дій), оскільки множення являє собою послідовне додавання, а ділення – послідовне віднімання.

Таблиця 1.5 – Правила віднімання чисел

Дія	Зворотний код	Додатковий код
$5-3=5+(-3)=2$	$x_1=0,101$ $x_2=1,100$ 0,101 + 1,100 10,001 0,010 Перенос у молодший розряд. Сума додатна	$x_1=0,101$ $x_2=1,101$ 0,101 + 1,101 10,010 x 0,010 Одиниця переносу в молодший розряд ігнорується. Сума додатна
$3-5=3+(-5)=-2$	$x_1=0,011$ $x_2=1,010$ 0,011 + 1,010 1,101 Перенос у молодший розряд відсутній. Сума від'ємна й зображена у зворотному коді	$x_1=0,011$ $x_2=1,011$ 0,011 + 1,011 1,110 Сума від'ємна й зображена в додатковому коді

Однорозрядний двійковий суматор

Підрахування багаторозрядних двійкових чисел $A=a_n a_{n-1}...a_0$ і $B=b_n b_{n-1}...b_0$ проводиться шляхом їх порозрядного додавання з переносом між розрядами. Тому основним вузлом багаторозрядних суматорів є комбінаційний однорозрядний суматор, який виконує арифметичне додавання трьох однорозрядних чисел (цифр): цифри даного розряду першого доданка (a_i), цифри даного розряду другого доданка (b_i) й цифри (1 або 0) переносу із сусіднього молодшого розряду (p_i). У результаті додавання для кожного розряду виходять дві цифри – сума для цього розряду (S_i) і перенос у наступний старший розряд (p_{i+1}).

Принцип дії однорозрядного суматора наведений в табл. 1.6.

Для синтезу схеми однорозрядного суматора необхідно записати логічні рівняння для функцій виходів S_i і P_{i+1} і виконати перетворення відповідно до правил алгебри логіки. Для цього необхідно перегрупувати вихідне рівняння за прямим і інверсним значеннями аргумента p_i, винести їх за дужки й перетворити у функції нерівнозначності (М2). Це перетворення розглянуто за допомогою таких формул:

$$S_i = a_i \bar{b}_i \bar{p}_i + \bar{a}_i b_i \bar{p}_i + \bar{a}_i \bar{b}_i p_i + a_i b_i p_i = (a_i \bar{b}_i + \bar{a}_i b_i)\bar{p}_i + (\overline{a_i \oplus b_i}) p_i = (a_i \oplus b_i)\bar{p}_i + (\overline{a_i \oplus b_i}) p_i = (a_i \oplus b_i) \oplus p_i. \quad (1.13)$$

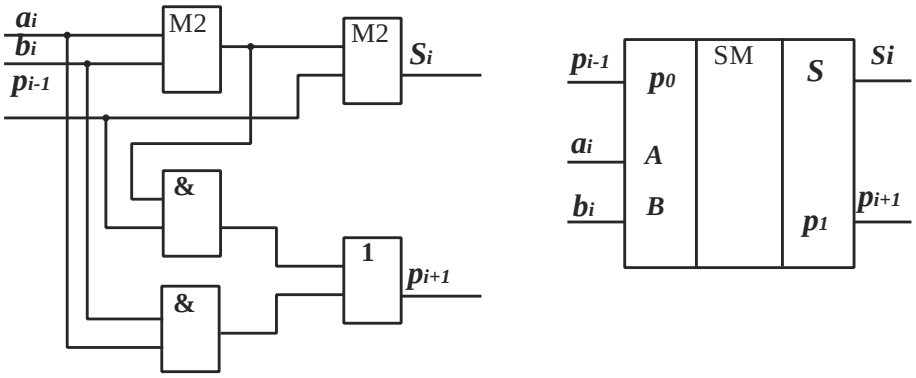
Таблиця 1.6 – Принцип дії однорозрядного суматора

a_i	b_i	p_i	S_i	p_{i+1}
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Аналогічно записується і рівняння для функції P_{i+1}:

$$P_{i+1} = (a_i \oplus b_i) p_i + a_i b_i. \quad (1.14)$$

Умовне позначення і функціональна схема однорозрядного суматора, яка побудована відповідно за отриманими виразами, наведено на рис. 1.15.



а) *б*)

Рисунок 1.15 – Функціональна схема однорозрядного суматора – *а*) і його умовне позначення – *б*)

Багаторозрядний паралельний суматор може бути побудований з однорозрядних суматорів, число яких дорівнює числу розрядів, що складаються шляхом з'єднання виходів, на яких формується сигнал переносу даного розряду із входом для сигналу переносу сусіднього старшого розряду. Такий спосіб організації переносу називається послідовним.

Багаторозрядний суматор

Приклад побудови чотирирозрядного паралельного суматора демонструє рис. 1.16. У суматорах цього типу перенос поширюється послідовно від розряду до розряду з формуванням суми в кожному розряді. Так, наприклад, при додаванні чисел 11…11 і 00…01 буде мати місце «пробіг» одиниці переносу через увесь суматор від самого молодшого до самого старшого розряду. Даний тип суматора найбільш простий з погляду схеми кіл поширення переносу.

Нескладно побачити, що на базі суматора легко реалізувати і функцію віднімання чисел. Для цього, наприклад, віднімач повинен бути в додатковому коді. Останнє досягається інвертуванням розрядів, що віднімаються, й подачею (додаванням) «1» до молодшого розряду. Це показано в лівій частині рис. 1.16.

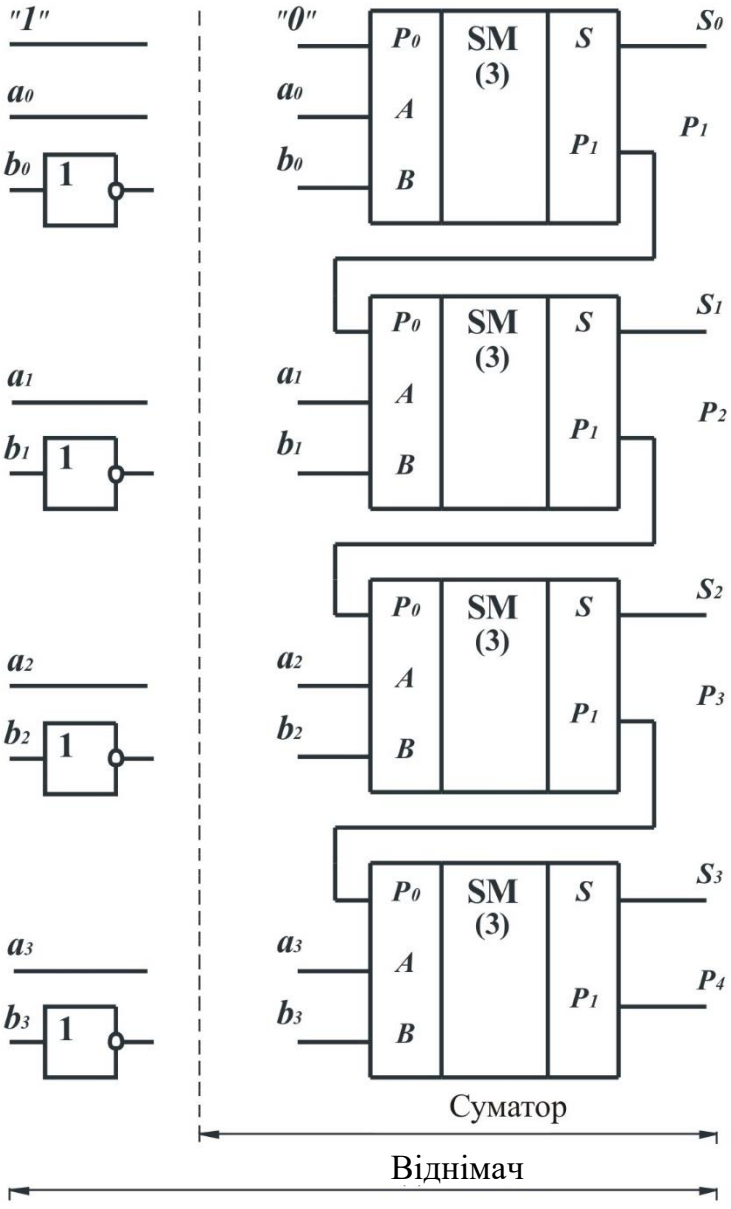


Рисунок 1.16 – Багаторозрядний суматор-віднімач

Арифметико-логічні пристрої

Вершиною розвитку мікросхемотехніки середнього ступеня інтеграції стали арифметико-логічні пристрої (АЛП), що об'єднали в собі можливість виконання даних логічних і арифметичних операцій. У порівнянні із приладами, що працюють на твердій логіці, вони являють собою пристрої більш високого класу. В АЛП перетворення виконуються над багаторозрядними двійковими операндами.

До основних виконуваних операцій належать:

- логічні – порозрядні диз'юнкція або кон’юнкція;
- арифметичні – додавання, віднімання, додавання за модулем 2;
- спеціальні – інверсія, інкремент або декремент, зсув вліво або вправо одного з операндів.

Звичайно в АЛП є дві групи входів для операндів і група входів для вибору виду виконуваної операції. Крім цього, є вихід, що означає виникнення переносу в старшому розряді, й вхід, на який може надходити перенос із молодшого розряду. Це дає можливість збільшувати розрядність оброблюваних операндів шляхом каскадного вмикання декількох АЛП разом.

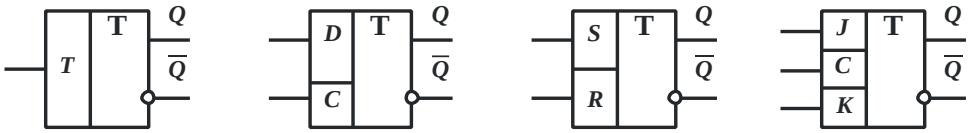
В архітектурі мікроконтролерів АЛП є базовим елементом, за допомогою якого виконується арифметична й логічна обробка даних.

1.2.6. Цифрові пристрої послідовнісного типу

Тригери

Тригери – це елементи цифрової, дискретної техніки, що мають здатність перебувати в одному із двох стійких станів протягом необмеженого часу. Перший стан називається «одиничним», інший «нульовим». Тригери мають два виходи, один з яких є прямим Q, а другий – інверсним $\bar{Q}$. Прийнято вважати, що стан тригера відповідає сигналу на його прямому виході.

За логікою функціонування розрізняють тригери типів $R-S$, D, T і $J-K$, що відповідає назві їх керуючих входів. Умовне зображення цих тригерів показано на рис. 1.17. За допомогою вхідних сигналів здійснюється керування перемиканням або установкою тригера в необхідний стан. До моменту впливу на керуючий вхід тригер зберігає той стан, у котрий його було встановлено. Тимчасові інтервали, протягом яких ці стани зберігаються незмінними, називають тактами або кроками роботи.



а) *б*) *в*) *г*)

Рисунок 1.17 – Умовне зображення: T – *а*), D – б), $R-S$ – *в*), $J-K$ – *г*) – тригерів

Логіка функціонування тригерів залежить не тільки від значення вхідних змінних (керуючих сигналів), що надходять при поточному кроці, але й від стану самого тригера, в якому він перебував на попередньому кроці. Для опису їх роботи застосовуються логічні таблиці виходів і переходів, які часто об'єднують. У таких таблицях стану тригери на попередньому й наступних кроках записуються як Q^k і Q^{k+1}. Функціонування переходів і виходів для T- і D-тригера наведені в табл. 1.7. У цих тригерах є по одному керуючому входів, які позначаються відповідно літерами T і D.

Таблиця 1.7 – Функціонування переходів і виходів: T – *а*) і D – *б*) – тригерів

T^k	Q^k	Q^{k+1}
0	0	0
1	0	1
0	1	1
1	1	0

а)

C	D^k	Q^k	Q^{k+1}
1	0	0	0
1	1	0	1
1	0	1	0
1	1	1	1

б)

З табл. 1.7, *а* можна побачити, що T-тригер перемикається в протилежний стан щоразу при подачі на його вхід сигналу рівня «1» незалежно від того, в якому стані він перебував. Такий тригер називається рахунковим або «тригер з рахунковим входом». Він використовується як дільник на 2.

Для D-тригера властиво те, що він є повторювачем вхідного сигналу. Щоб забезпечити функцію зберігання інформації, потрібен додатковий керуючий сигнал, за яким дозволяється запис у тригері. Такий сигнал називається синхронізуючим і позначається літерою C. У проміжку між синхросигналами стан тригера не змінюється. Функціонування D-тригера будуть розглянуто нижче.

Стан переходів і виходів для тригерів $R-S$ і $J-K$ наведено в табл. 1.8. У цих тригерах є окремі входи для встановлення виходів у стан «1» відповідно S і J, і для встановлення в «0» – входи R і K. Входи R і K називають також входами скидання тригера. За відсутності керуючих сигналів тригери зберігають свій стан.

Таблиця 1.8 – Стан переходів і виходів для тригерів $R-S$ і $J-K$

$R^k\left(K^k\right)$	$S^k\left(J^k\right)$	Q^k	Q^{k+1}	
			$R-S$	$J-K$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	*	1
1	1	1	*	0

Відмінності між $R-S$ і $J-K$ -тригерами проявляються в їхніх реакціях на одночасну подачу сигналів рівня «1» на обидва керуючих входи. Для $R-S$ -тригера така комбінація сигналів є забороненою, а $J-K$ -тригер змінює свій стан на протилежний, перемикаючись як T -тригер.

Окремо слід сказати про деякі інші особливості $J-K$ -тригерів. Вони були розроблені, щоб виключити ефект, що одержав назву «перегони тригерів», який проявлявся при паралельній, груповій роботі декількох тригерів. Індивідуальною особливістю мікросхем тригерів, навіть, якщо вони того самого типу, є тривалість часу перемикання. Двох однакових немає! А це призводить до того, що при одночасному перемиканні декількох тригерів у якомусь проміжку з'являються неправильні комбінації вихідних сигналів, відбувається збій у роботі всього обладнання.

Функціонально $J-K$ -тригери складаються із двох ступенів, тригера, що ведучий, і ведений. Ці тригери завжди такі, що синхронізуються. Перший ступінь перемикається за фронтом синхроімпульсу, а другий – за зрізом. Час перемикання веденого ступеня значно менше, ніж у ведучого. І при груповій роботі декількох тригерів зміна вихідних сигналів відбувається дуже швидко, начебто одномоментно. Це дало можливість виключити появу непередбачених станів тригерів. У цифровій техніці $J-K$ -тригери широко застосовуються при побудові лічильників.

За способом керування тригери поділяються на асинхронні та синхронні, або такі, що синхронізуються. В асинхронних тригерах перехід до нового стану залежить тільки від вхідних керуючих сигналів. У синхронних тригерах додатково потрібно формувати сигнал синхронізації. Їх перемикання відбувається тільки за умови побудови такого сигналу.

За способом сприйняття синхросигналів тригери поділяються на статичні, керовані за рівнем і динамічні керовані за фронтом або зрізом синхроімпульсу. Умовні позначення входів синхронізації наведені на рис. 1.18.

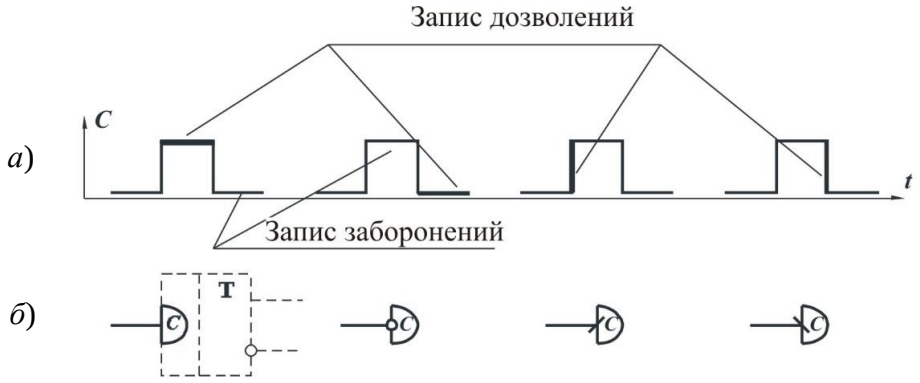


Рисунок 1.18 – Входи синхронізації: *а*) – тимчасові діаграми; *б*) – умовне позначення

Найпростішим є асинхронний $R-S$-тригер. Його схеми, що виконані на елементах «АБО – НІ» і «І – НІ» наведені на рис. 1.19.

Зіставлення схем *а* і *б* свідчить, що функціонування тригера не залежить від типу елементів, на яких вони реалізовані. Відмінністю є те, що тригер, виконаний на елементах «АБО – НІ», керується прямими вхідними сигналами R і S (рівень логічної 1), а на елементах «І – НІ» – інверсними сигналами $\overline{R}$ і $\overline{S}$ (рівень логічного 0). Для варіанта *а* на рис. 1.19 наведено тимчасову діаграму. Тригер встановлюється при $S=1$ і скидається при $R=1$.

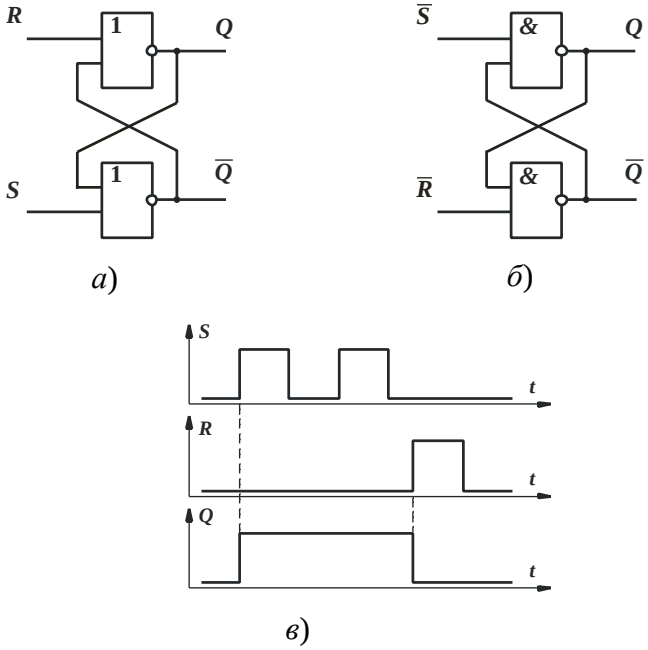
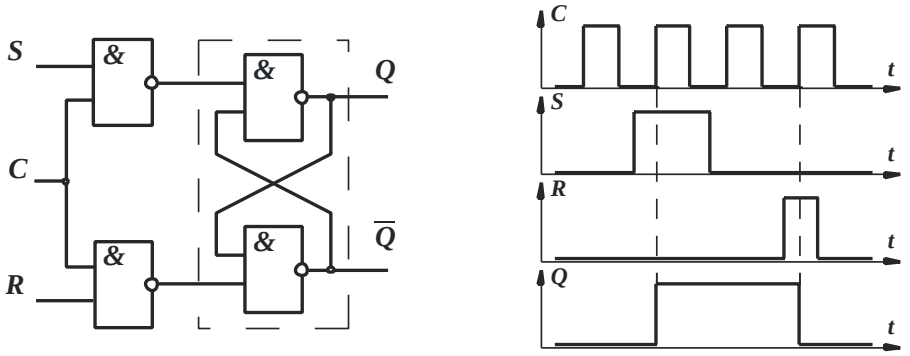


Рисунок 1.19 – Асинхронний $R-S$-тригер: *а*) – у базисі «АБО – НІ»; *б*) – у базисі «І – НІ»; *в*) – тимчасові діаграми

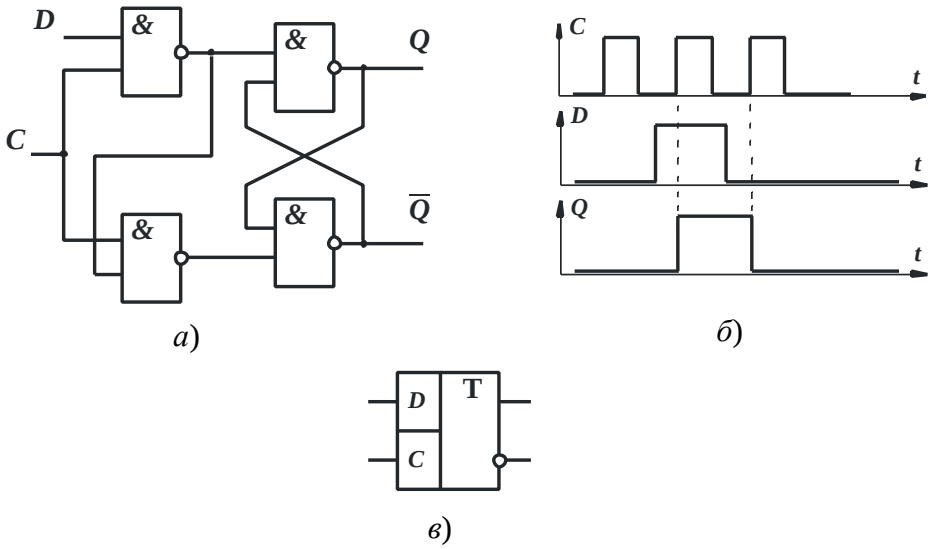
У синхронному $R-S$-тригері необхідно створити умови, щоб керуючі сигнали R або S надходили на вхід тригера тільки в момент присутності синхроімпульсу. Це легко вирішується за допомогою елементів «2І – НІ». Приклад такої схеми і тимчасові діаграми її роботи наведено на рис. 1.20.



а) *б*)

Рисунок 1.20 – Функціональна схема синхронного $R-S$ -тригера – *а*) та тимчасові діаграми – *б*)

На базі синхронного $R-S$ -тригера легко одержати D -тригер. Для цього вводиться зворотний зв'язок (рис. 1.21). Проходження сигналу із входу D можливо тільки за наявності «1» на вході C. Умовне графічне зображення і тимчасові діаграми роботи D -тригера так само можна побачити на рис. 1.21.

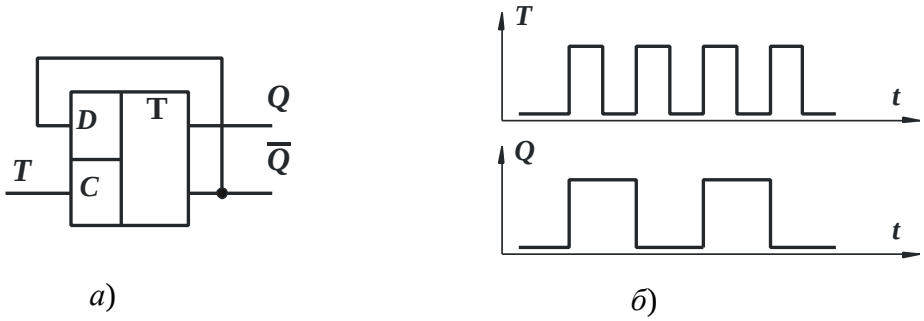


а) *б*)

в)

Рисунок 1.21 – Функціональна схема D -тригера – *а*), тимчасові діаграми – *б*) та умовне позначення – *в*)

У мікропроцесорних структурах D -тригер є базовим елементом при побудові регістрів, лічильників, а також оперативних запам'ятовувальних пристроїв. На рис. 1.22 зображений T -тригер, побудований на базі D -тригера та його тимчасові діаграми роботи.



а) *б*)

Рисунок 1.22 – Функціональна схема T - тригера – *а*) та його тимчасові діаграми – *б*)

Регістри

Регістрами називаються цифрові пристрої, використовувані для запису і зберігання n-розрядного двійкового слова. Вони будуються на базі тригерів, число яких у схемі регістра звичайно відповідає числу розрядів двійкового слова. Особливістю роботи схеми регістра є те, що запис, зчитування або перетворення інформації відбувається одночасно у всіх його тригерах. У багаторозрядних оперативних запам'ятовувальних пристроях (ОЗП) регістри є базовими елементами. Їх називають осередками пам'яті. У мікроконтролерах розрядність таких гнізд звичайно становить 8 або 16 біт.

Крім зберігання інформації, в деяких типах регістрів забезпечується можливість виконання додаткових функцій для її перетворення. До них відносяться: перетворення даних з паралельної форми подання в послідовну, і навпаки; перетворення прямого коду у зворотний; виконання зсуву на один або декілька розрядів убік молодшого або старшого розрядів; інкрементування або декрементування вмісту регістра. У структурі мікропроцесора це основний елемент.

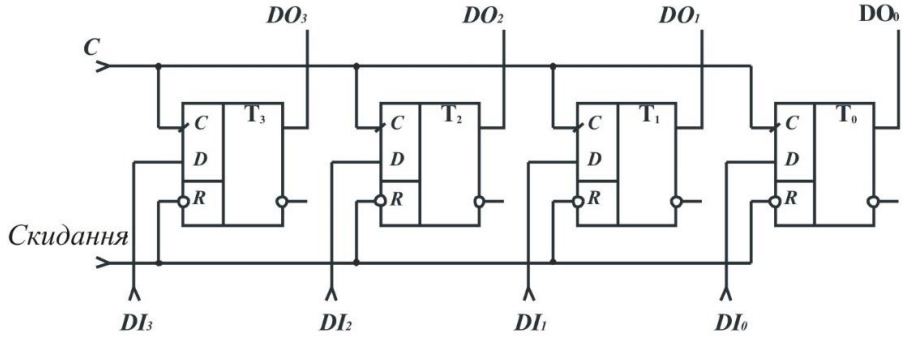
Паралельний регістр

Паралельним називають регістр, у якому n-розрядне двійкове слово записується і зчитується одночасно у всіх n розрядах. Причому, після зчитування інформація, що зберігається в ньому, він не змінюється.

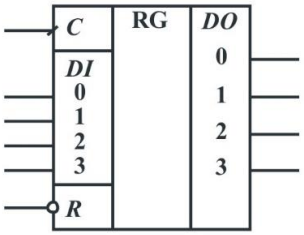
Паралельні регістри найчастіше будуються на синхронних D або $R-S$-тригерах. Приклад схеми чотирирозрядного паралельного регістра і його умовне позначення наведено на рис. 1.23.

Загальними для розрядів регістра є кола синхронізації (C) і скидання (R). Значення розрядів двійкового слова 0, 1, 2, 3 записуються у відповідних тригерах тільки після подачі імпульсу синхронізації $C=1$. Після цього тригери переходять у

режим зберігання. Зчитування слова з регістра здійснюється з прямих Q-виходів тригерів. Інверсні виходи $\left(\overline{Q}\right)$ використовуються мало.



а)



б)

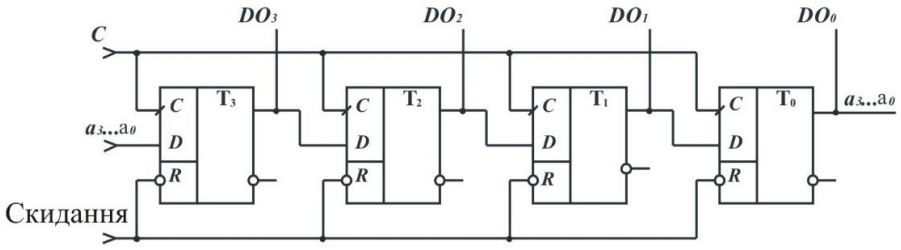
Рисунок 1.23 – Функціональна схема паралельного регістра – *а*) та його умовне позначення – *б*)

Слід нагадати, що після подачі живлення на схему регістра тригери встановлюються в довільні, заздалегідь невизначені стани. Тому в наведеній схемі є окрема лінія установки регістра в «нульовий» або вихідний стан для скидання тригерів. На R-вхід кожного з них подається імпульс скидання. У схемі такий імпульс формується «0», тому що R-входи інверсні.

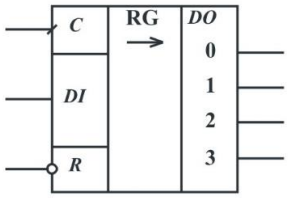
Послідовний регістр

Послідовним називають регістр, у якому здійснюється послідовне (розряд за розрядом) прийняття і видача інформації зі зсувом. Такий регістр називається регістром зрушення. Він являє собою ряд послідовно з'єднаних тригерів, число яких визначається розрядністю двійкового слова. За напрямком зсуву регістри бувають прямого зсуву убік молодшого розряду і зворотного – убік старшого розряду. Іноді вживають терміни «зсув вправо» або «зсув вліво». Існують також і реверсивні регістри, що допускають зрушення в обох напрямках.

На рис. 1.24 наведено приклад функціональної схеми й умовне позначення чотирирозрядного регістра, що виконує зсув вправо.



а)



б)

Рисунок 1.24 – Функціональна схема регістра зсуву вправо – *а*)
та його умовне позначення – *б*)

Регістр побудовано на D-тригерах, які синхронізуються фронтом імпульсу. При записі інформації молодший розряд слова, що вводиться, подається на вхід крайнього лівого тригера T_3. Але запис розряду відбувається тільки при надходженні імпульсу синхронізації на вхід C. За наступним синхроімпульсом значення з виходу Q_3 передається в тригер T_2, а на вхід T_3 повинен надходити наступний розряд слова, що вводиться. Таким чином, із надходженням кожного наступного синхроімпульсу відбувається зсув інформації на один розряд вправо, і після четвертого синхроімпульсу регістр буде заповнений усіма розрядами слова, що вводиться.

Для реалізації процедури зсуву вліво використовуються двоступеневі тригери, у яких за фронтом синхроімпульсу встановлюється перший ступень, а за зрізом – другий. На рис. 1.25 наведено схему такого регістра. Для передачі інформації з молодшого до старшого розряду вихід кожного тригера молодшого розряду з'єднаний із входом тригера старшого розряду. До закінчення синхроімпульсу кожний тригер утримує інформацію, записану на попередньому кроці. Захоплювання ж інформації відбувається за фронтом синхроімпульсу, що й забезпечує її зсув вліво.

Комбінуючи схеми зсуву вправо й уліво та використовуючи керуючі сигнали, можна побудувати реверсивний регістр. А, якщо в схемі забезпечити з'єднання виходу останнього розряду із входом першого, то регістр зсуву легко перетворюється в кільцевий регістр.

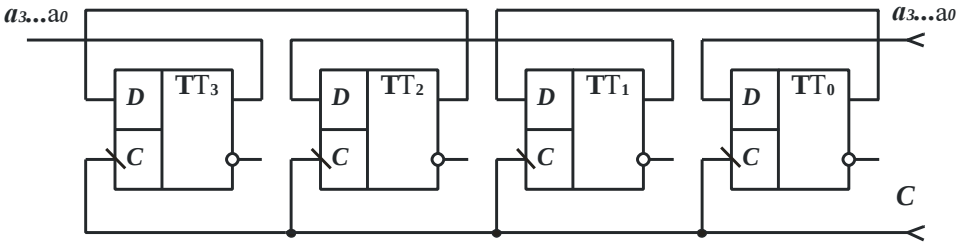


Рисунок 1.25 – Схема регістра зсуву вліво

Лічильники

Цифровим лічильником називають функціональний вузол, який здійснює підрахунок числа імпульсів, що надходять на його вхід, формує результат обчислення в заданому коді. Такий код звичайно є двійковим й тому ці лічильники називаються двійковими.

Залежно від напрямку підрахунку розрізняють підсумовуючі (із прямим підрахунком), віднімальні (зі зворотним підрахунком) і реверсивні (як із прямим, так і зі зворотним підрахунком) лічильники. Під ємністю лічильника мається на увазі максимальне число імпульсів, які можуть бути підраховані лічильником, звичайно 2^k. Де k дорівнює числу розрядів лічильника.

Двійкові лічильники можуть бути побудовані і на синхронних D-тригерах, і $J-K$-тригерах, перетворених у T-тригери.

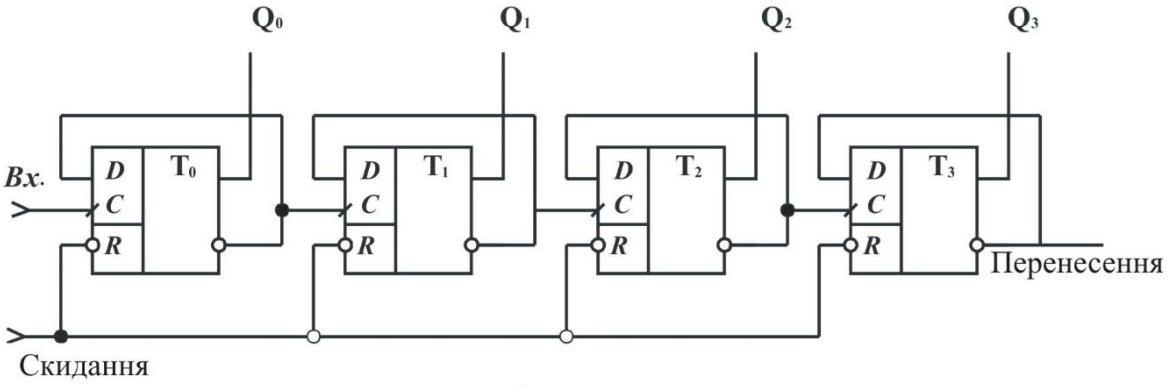
Підсумовуючі двійкові лічильники

На рис. 1.26 наведено схему і тимчасові діаграми чотирирозрядного підсумовуючого двійкового лічильника з колами послідовного переносу. Інверсний вихід i-го розряду (тригера) з'єднано із входом $(i+1)$-го розряду. Лічильник побудовано на D-тригерах, які перетворені в асинхронні T-тригери, що тактуються фронтом синхроімпульсу.

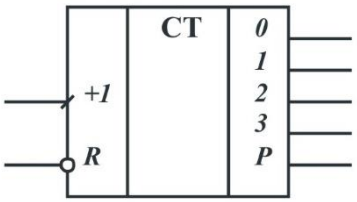
Входом лічильника служить вхід крайнього лівого тригера (T_0). Двійковий код результату підрахунку формується на виходах тригерів Q_0, Q_1, Q_2, Q_3 (Q_0 – молодший, а Q – старший). Ємність розглянутого лічильника $2^4 = 16$, тому максимальне його показання відповідно до подачі на вхід становить 15 рахункових імпульсів. Останній 16-й рахунковий імпульс установлює всі тригери у вихідний (нульовий) стан, отже шина «скидання» (установлення в 0) необхідна лише на початку роботи лічильника.

Після подачі кожного наступного вхідного імпульсу T-тригер переходить у протилежний стан. З тимчасової діаграми можна побачити, що період проходження імпульсів на виходах кожного розряду у два рази більший, ніж на

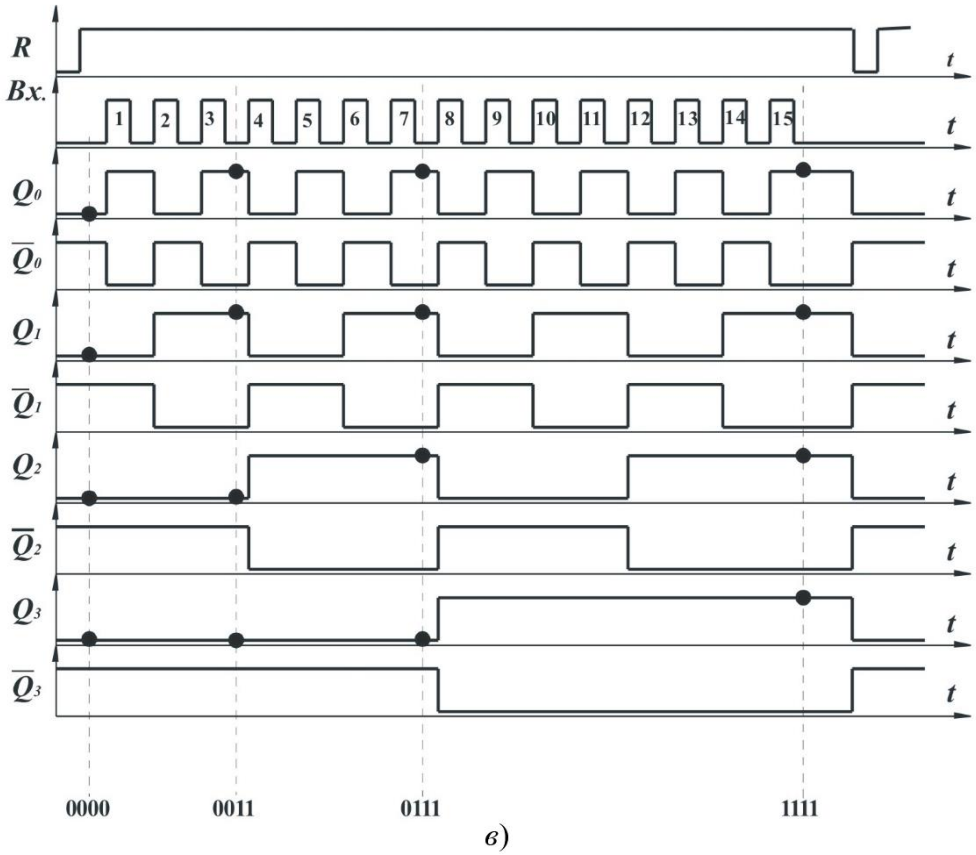
його вході. У будь-який момент часу стан тригерів лічильника однозначно визначає число імпульсів, що надійшли на його вхід.



а)



б)



в)

Рисунок 1.26 – Функціональна схема підсумовуючого двійкового лічильника – *а*) умовне позначення – *б*) та його тимчасові діаграми – *в*)

1.2.7. Організація пам'яті

Запам'ятовувальні пристрої (ЗП) призначені для зберігання інформації (даних). Вони поділяються на постійні запам'ятовувальні пристрої (ПЗП) і оперативні запам'ятовувальні пристрої (ОЗП).

ПЗП (ROM – Read Only Memory) – пристрій для зберігання й зчитування незмінних даних. Цей вид пам'яті призначено для зберігання програми, а також незмінних констант. Така пам'ять є енергонезалежною – при відключенні живлення записана інформація зберігається. Після поновлення живлення вона може бути зчитана.

ОЗП (RAM – Random-Access Memory) – пристрій для запису, зберігання і зчитування змінюваних даних. У даному виді пам'яті зберігається інформація, що модифікується і використовується в процесі роботи. Це можуть бути різні змінні або результати проміжних і остаточних обчислень. Інформація при такому виді пам'яті зникає після вимикання живлення.

Основним параметром пам'яті є її обсяг, тобто кількість бітів інформації, які можуть у ній зберігатися. Для позначення обсягу використовуються такі спеціальні одиниці:

- 1 К – це 1024 (210) і читається як «кіло-» або «ка-». Обсяг пам'яті приблизно дорівнює однієї тисячі;
- 1 М – це 1048576 (220) і читається як «мега-». Обсяг пам'яті приблизно дорівнює одному мільйону;
- 1 Г – це 1073741824 (230) і читається як «гіга-». Обсяг пам'яті приблизно дорівнює одному мільярду.

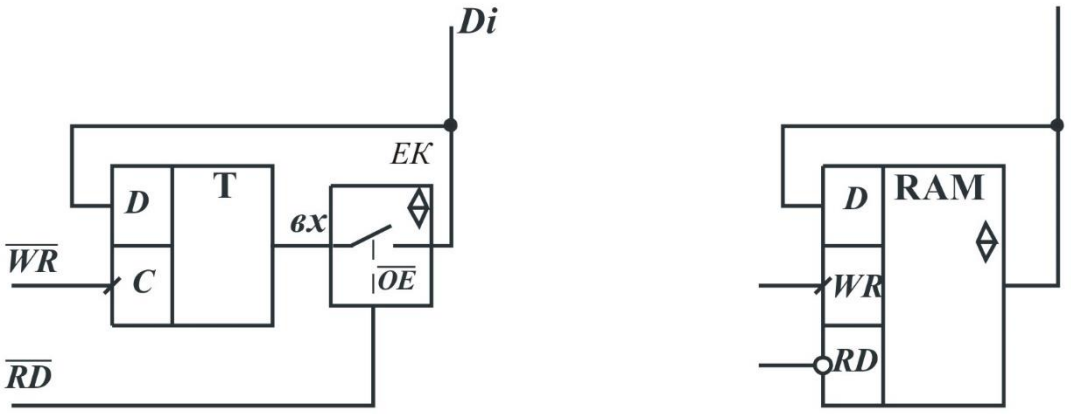
За видом організації пам'ять може бути однорозрядною з індивідуальним доступом до кожного біта та багаторозрядною з паралельним доступом до групи бітів. Групова організація має байтову структуру, від одного до 2^n байтів. У свою чергу кожний байт складається з 8 бітів.

Під осередком пам'яті може матися на увазі як однорозрядна, так і багаторозрядна структура. У другому випадку в позначенні додається цифра, що вказує на кількість одночасно доступних розрядів. Наприклад, RAM 2Kx8 має 2048 восьмирозрядних осередків пам'яті типу ОЗП.

Оперативні запам'ятовувальні пристрої

Основою елементарного осередка ОЗП є тригер. На рис. 1.27 зображена її функціональна схема. Запам'ятовування і зберігання інформації відбувається в D-тригері, який має динамічний вхід синхронізації із записом інформації за

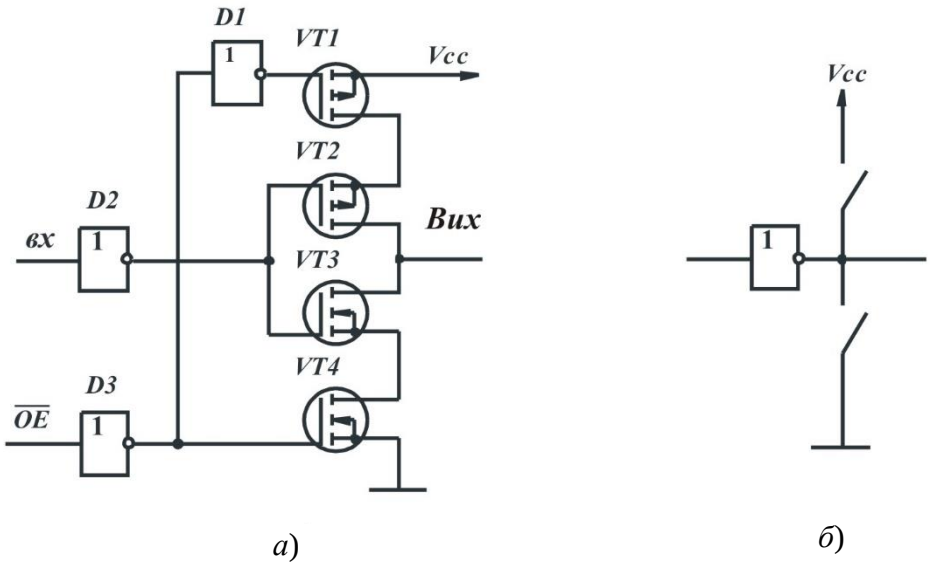
фронтом синхросигналу. Вихід тригера увімкнено через електронний ключ (ЕК), що реалізує лінію із трьома станами.



а) *б*)

Рисунок 1.27 – Вигляд елементарного осередка оперативних запам'ятовувальних пристроїв

Функціональна схема ЕК зображена на рис. 1.28. Якщо на керуючому вході OE присутній сигнал високого рівня, то транзистори $VT1$ й $VT4$ замкнені і вихідна лінія перебуває у високоімпедансному стані. Еквівалентно це показано у вигляді розімкнених ключів. Низький рівень сигналу призводить до відключення транзисторів $VT1$, $VT4$. На вихідній лінії буде сигнал такого самого рівня, що й на вході. Оскільки ключ, виконаний на транзисторах $VT2$, $VT3$, інвертує сигнал, то для забезпечення функції повторення на елементі $D2$ є інвертор.



а) *б*)

Рисунок 1.28 – Схема електронних ключів

Організація осередка пам'яті у такий спосіб дає можливість об'єднати її вхід і вихід на одній лінії даних Di. Для керування записом і читанням інформації існують керуючі сигнали WR і RD. Сигнал WR подається на синхровхід

тригера, а сигнал RD – на вхід керування електронним ключем OE. Тимчасові діаграми запису і читання наведені на рис. 1.29.

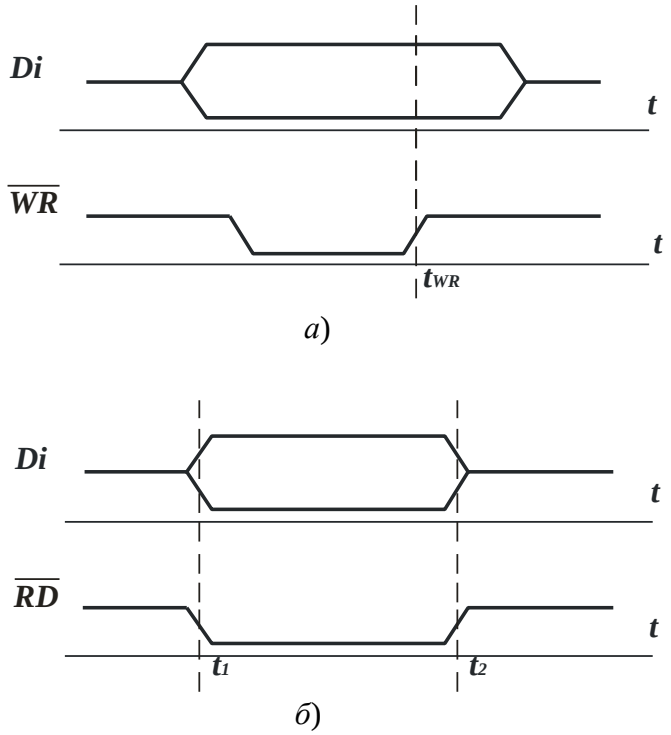


Рисунок 1.29 – Тимчасові діаграми запису оперативних запам'ятовувальних пристроїв – *а*) та читання – *б*) відповідно

При записі інформації дані виставляються на лінію Di, після чого формується строб-імпульс на лінії WR. Установлення або скидання тригера відбувається за фронтом строб-імпульсу (t_{WR}). Під час читання інформації дані із тригера виставляються на лінію Di в проміжку часу між $t_1 - t_2$ та у момент дії сигналу RD.

Для організації багаторозрядного ОЗП елементарні, однорозрядні осередки пам'яті необхідно об'єднати в реєстрові структури. На рис. 1.30 зображена функціональна схема 8-розрядного ОЗП ємністю 2^n. Вибір конкретної лінійки пам'яті проводиться за допомогою дешифратора адреси, на виходи якого надходить відповідна адресна комбінація $A0...An$. Таким чином, кожен осередок пам'яті має адресу, за якою зберігається інформація.

В ОЗП є три керуючих входи. Для запису або читання – входи WR, RD, для дозволу роботи дешифратора – CS. Декодування адреси й установлення 1 на одному з виходів дешифратора відбувається тільки під час подачі сигналу нульового рівня на вхід CS. Високий рівень на одному з виходів дешифратора дозволяє проходження сигналу WR або RD до відповідної лінійки осередків в ОЗП. За сигналом WR виконується паралельний запис у пам’ять інформації із

шини даних $D0...D7$, а за сигналом RD – читання або видача на шину даних інформації з пам'яті.

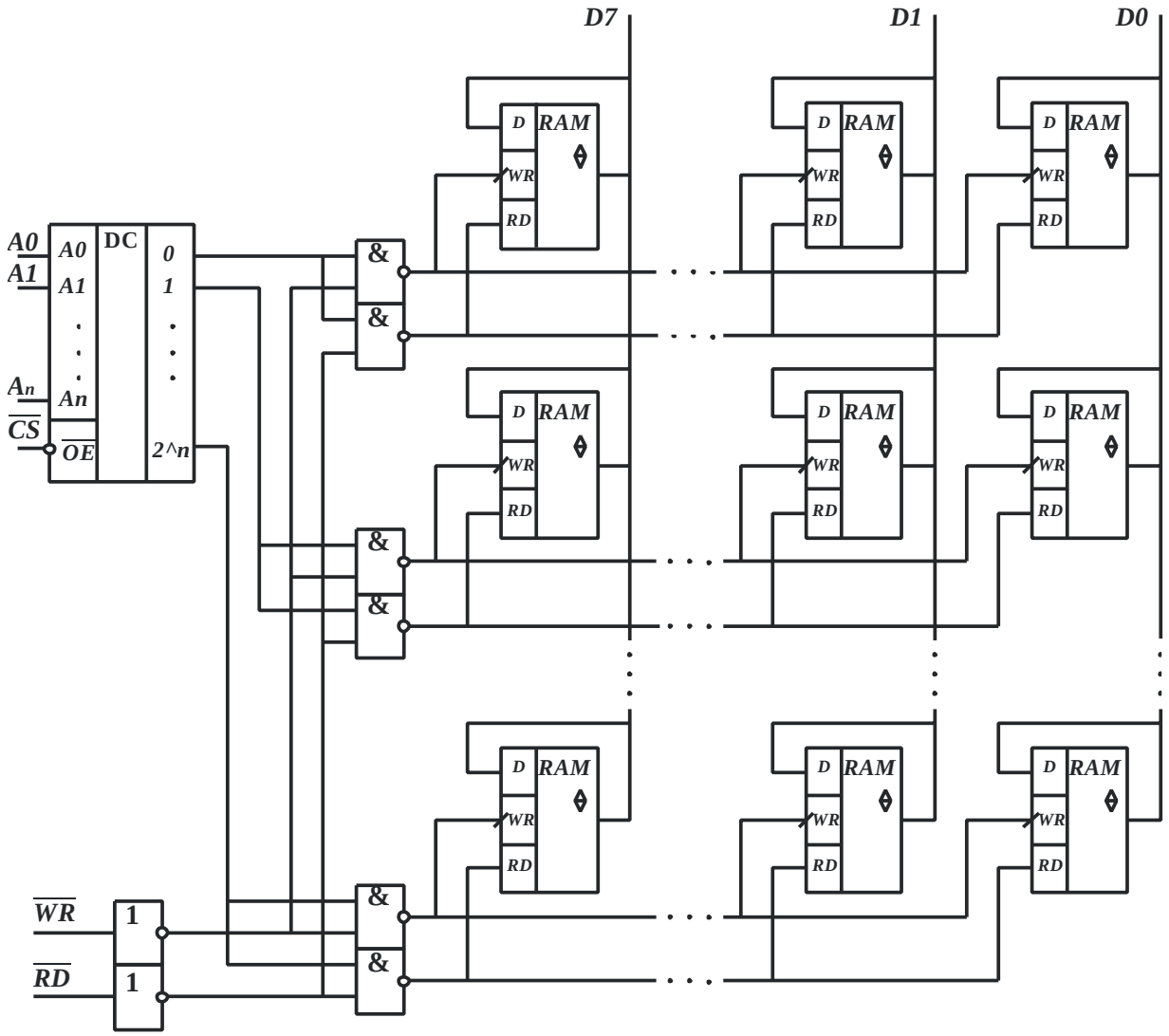


Рисунок 1.30 – Схема організації багаторозрядного оперативного запам'ятовувального пристрою

Умовне зображення ОЗУ і тимчасові діаграми його роботи зображені на рис. 1.31. З діаграм можна побачити, що в першу чергу повинна з’являтися адреса осередка пам'яті, потім сигнал CS і далі під час читання сигнал RD, а при записі – спочатку дані і тільки після цього сигнал WR.

Постійні запам'ятовувальні пристрої

Постійні запам'ятовувальні пристрої (ПЗП) різняться за принципом запису і стирання інформації. Сам процес занесення або запису інформації в ПЗП називається «програмуванням». На сьогоднішній момент існує п'ять видів ПЗП.

ROM (Read Only Memory) – ПЗП програмуються одного разу на стадії виготовлення. Цей вид застосовують при великому тиражі виробів, коли є завдання зниження їх вартості.

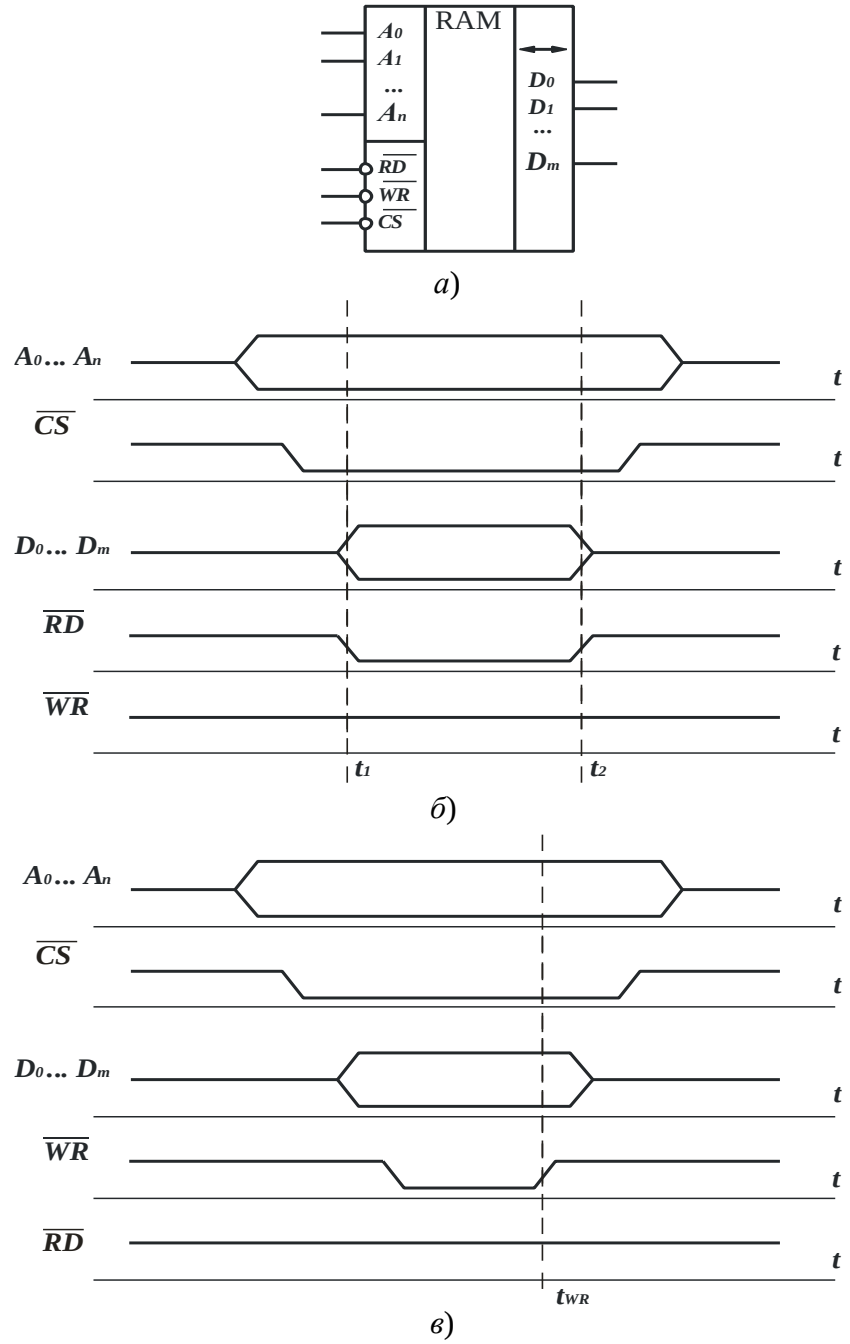


Рисунок 1.31 – Умовне позначення багаторозрядного оперативного запам'ятовувального пристрою – *а*) та тимчасові діаграми читання – *б*) й запису – *в*)

PROM (Programmable Read Only Memory) – однократно програмувальні ПЗП. Запис інформації в них здійснюється за допомогою спеціального пристрою, який називається програматором.

EPROM (Erasable Programmable Read Only Memory) – багаторазово перепрограмувальні ПЗП. Запис інформації здійснюється за допомогою програматора, стирання – шляхом ультрафіолетового опромінення мікросхеми через спеціальне вікно в корпусі.

EEPROM (Electrically Erasable Programmable Read Only Memory) – електрично програмувальні та такі ПЗП, що стираються. Запис і стирання інформації здійснюється за допомогою програматора.

Flash Memory – електрично програмувальні та такі ПЗП, що стираються. Інформація може записуватися й стиратися за допомогою електричних сигналів без застосування спеціальних програматорів. Використовується найчастіше для енергонезалежного зберігання даних.

У сучасних мікроконтролерах для зберігання програм широко використовуються ПЗП типу EEPROM із багаторозрядною організацією доступу до даних. Як елементарний осередок пам'яті використовується тригер, який виконано на базі польового транзистора з утримуючою ємністю. Функціональна схема такого осередка зображена на рис. 1.32. При зарядженій ємності $C1$ транзистор $VT1$ закритий і на його виході утримується високий рівень сигналу (логічна 1). Ємність заряджається при програмуванні ПЗП та у тих бітах, які повинні зберігати нуль. Заряджається і розряджається ємність $C1$ за допомогою ключів ЕК1 і ЕК3.

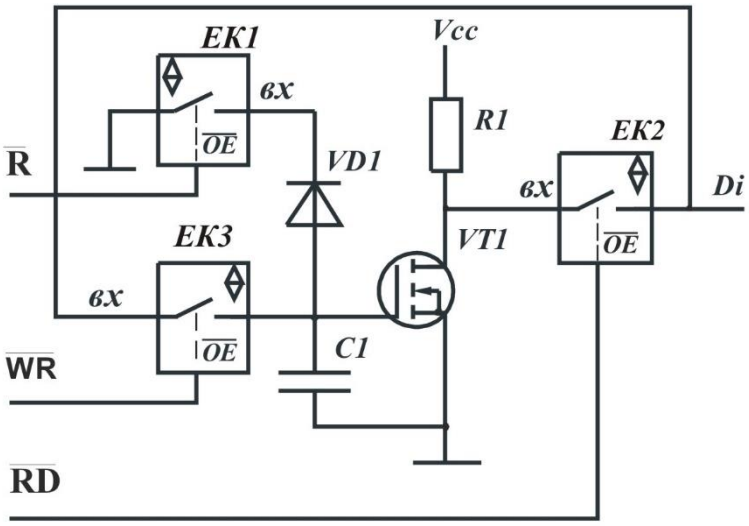


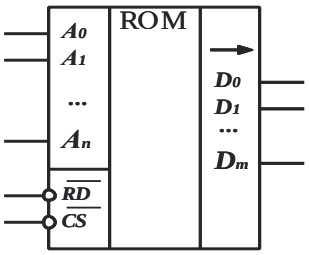
Рисунок 1.32 – Вигляд елементарного осередка оперативного запам'ятовувального пристрою

При стиранні ПЗП керуючий сигнал подається на ключ ЕК1, і ємність розряджається через діод $VD1$. Після стирання всі осередки пам'яті встановлюються в одиничний стан. Керування записом інформації виконується за допомогою ключа ЕК3. Під час його вмикання на ємність $C1$ надходить сигнал з лінії даних (Di). Його високий рівень створює заряд ємності. Зчитування інформації із $VT1$ здійснюється через ключ ЕК2, для чого повинен бути сформований керуючий сигнал на лінії RD. Оскільки електронні ключі

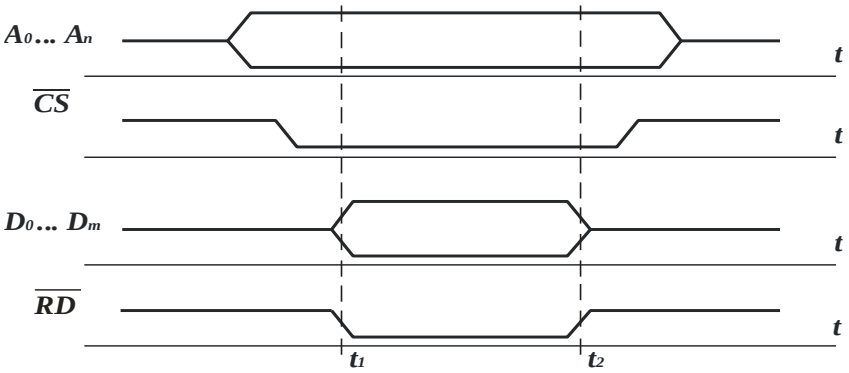
забезпечують лінії з трьома станами, то вхід і вихід осередка пам'яті об'єднуються.

Спосіб організації ПЗП є аналогічним організації багаторозрядного ОЗП. При звичайній роботі із ПЗП проводиться тільки зчитування інформації, тому на схемах умовного позначення лінії запису й стирання часто не зображуються. На рис. 1.33 наведено умовне позначення й тимчасові діаграми роботи ПЗП.

У ПЗП є два керуючих входи RD і CS, а також адресні входи $A0...An$. Кількість адресних входів визначає ємність пам'яті. Так, при n, що дорівнює 14, ємність ПЗП становить 32К. З тимчасових діаграм можна побачити послідовність формування вхідних сигналів. Спочатку встановлюється адреса, потім формується сигнал CS і далі сигнал RD. Читання інформації відбувається в проміжку часу між $t_1...t_2$.



а)



б)

Рисунок 1.33 – Умовне позначення постійних запам'ятовувальних пристроїв – *а*) та його тимчасові діаграми – *б*)

1.2.8. Методи і способи реалізації цифрових систем керування

На сучасному етапі реалізація дискретних і цифрових пристроїв може бути виконана різними методами, а також на базі різних технічних засобів. Це може бути метод традиційного логічного проектування з використанням дискретних інтегральних схем типу «І», «АБО», «НІ», або схем середнього ступеня інтеграції, типу дешифраторів або мультиплексорів, їх реалізація з

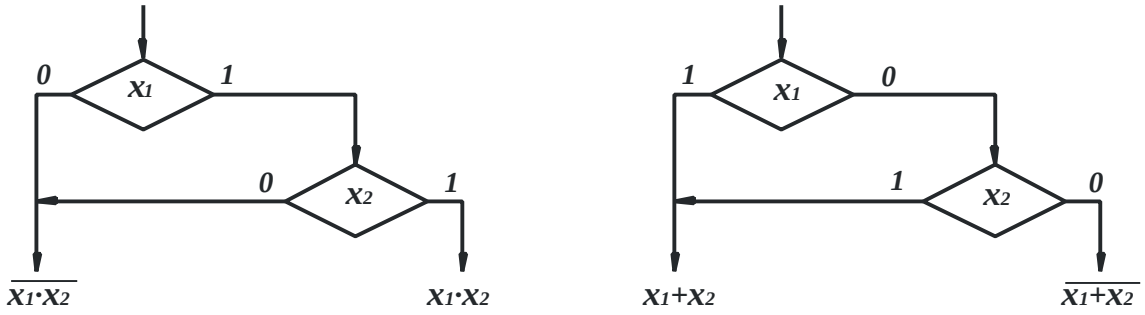
використанням великих інтегральних схем (ВІС), ПЗП або програмних логічних матриць (ПЛМ), а також засобів мікропроцесорної техніки. Усе це є інструментом для вирішення поставленого питання. Кожний зі способів має свої переваги і недоліки. Вирішальними факторами під час вибору способу реалізації можуть виступати ступінь складності завдання, вимоги швидкодії або економічна доцільність.

Невеликі й досить прості логічні функції, а відповідно і пристрої можуть бути реалізовані за допомогою дискретних вентильних схем або на базі ВІС ПЗП, ПЛМ. Крім того, існує чимало стандартних елементів, які виготовляються промисловістю. Наприклад, перетворювачі двійково-десяткового коду в двійковий, двійкового коду в додатковий і код знаків, багаторозрядні суматори, арифметико-логічні пристрої та багато інших. У сукупності це цілком може задовольнити потреби розробника. При такому підході слід ураховувати, що, якщо надалі виникне необхідність зміни алгоритму функціонування пристрою, то це призведе до його повної переробки, фактично до створення нового пристрою.

Застосовуючи програмні способи реалізації, часто достатньо лише зробити перепрограмування наявного пристрою без будь-якої зміни апаратних засобів. Інакше кажучи, програмний підхід дозволяє застосовувати ті самі елементи пристрою для реалізації складних логічних функцій будь-якого виду. Для цієї мети переважно використовуються однокристальні програмувальні процесори (мікропроцесори) або більш функціонально повні мікроконтролери. Програмно можна реалізувати як логічні, так і математичні функції.

При програмній реалізації логічні функції зображують у вигляді структурних схем, за якими надалі створюється відповідна програма. Між логічними булевими функціями і структурними схемами програм існує взаємно однозначна відповідність. Вони еквівалентні. У цьому можна переконатися на наступному прикладі. Припустимо, необхідно програмно реалізувати логічні функції «І» і «АБО». Граф-схема алгоритму реалізації зазначених функцій зображена на рис. 1.34.

Як при використанні дискретної схемотехніки, так і їхніх програмних еквівалентів можна отримати логічну функцію будь-якої складності, причому з меншими апаратними витратами, оскільки вони формуються тим самим пристроєм. Однак функції в цьому випадку формуються послідовно в часі, тобто знижується швидкодія їх виконання. При використанні інтегральної схемотехніки обидві функції можна було б реалізувати одночасно. Ця обставина іноді буває вирішальною на користь застосування вентильних схем.



а) *б*)

Рисунок 1.34 – Граф-схема алгоритму реалізації логічних функцій «І» і «АБО»

Таким чином, будь-яка логічна функція, реалізована програмним шляхом, може бути перетворена в еквівалентно реалізовану апаратними засобами, і навпаки. Якщо функція (закон) керування являє собою логічну функцію комбінаційного типу або послідовнісного, то реалізація на базі ПЗП або на ПЛМ являє собою програму, яку потрібно записати у ВІС. При зміні функції керування переписується тільки програма.

Однак у багатьох випадках у процесі реалізації закону керування необхідно здійснювати обчислення керуючих впливів. Вони виконуються у свою чергу за іншими законами, тобто необхідно мати пристрої керування (ПК), арифметико-логічний пристрій (АЛП) і пам'ять для зберігання програми обчислень, а це є загальновідомою структурою електронних обчислювальних машин (ЕОМ). У цій структурі ПК і АЛП являють собою процесор ЕОМ (рис. 1.35). Коли досягнення технології дозволили ПК і АЛП розмістити на одному кристалі, тоді й з'явився термін мікропроцесор (префікс мікро- – більше стосується розмірів і вартості мікропроцесора).

Рисунок 1.35 – Узагальнена структурна схема ЕОМ або комп’ютера

З часом розвитку нові технології дозволили виконати на одному кристалі мікропроцесор, пам'ять даних і програм, систему переривань, пристрої введення і виведення інформації, пристрої формування системного часу (таймери), а також був уведений термін мікроконтролер, структурна схема якого зображена на рис. 1.36.

Рисунок 1.36 – Структурна схема мікроконтролера

Систему переривань, пристрої введення і виведення інформації, пристрої формування системного часу називають периферійними пристроями.

Принцип роботи мікропроцесора розглянемо на прикладі його взаємодії з пам’яттю (рис. 1.37). Мікропроцесор містить шину адреси (16 двійкових розрядів), шину даних (як правило 8 двійкових розрядів), систему сигналів читання (RD) і запису інформації (WR), які визначають напрям пересилання інформації до мікропроцесора (RD) або від мікропроцесора (WR). Мікропроцесор на шину адреси завжди пересилає дані лічильника команд (PC). При подачі напруги живлення на мікропроцесор формується сигнал скидання (RESET), при цьому обнуляються всі вузли мікропроцесора, і дані лічильника команд складають 0000h. Це й є адресою нульового осередка. При цьому формується сигнал читання RD і дані нульового осередка пам’яті по шині даних пересилається до МП. Таким чином, мікропроцесор завжди починає свою роботу з читання нульового осередка пам’яті, вміст якої декодується і визначається, яку операцію надалі буде виконувати мікропроцесор. Вміст лічильника команд PC

збільшується на одиницю. При цьому формується адреса наступного осередка пам’яті.

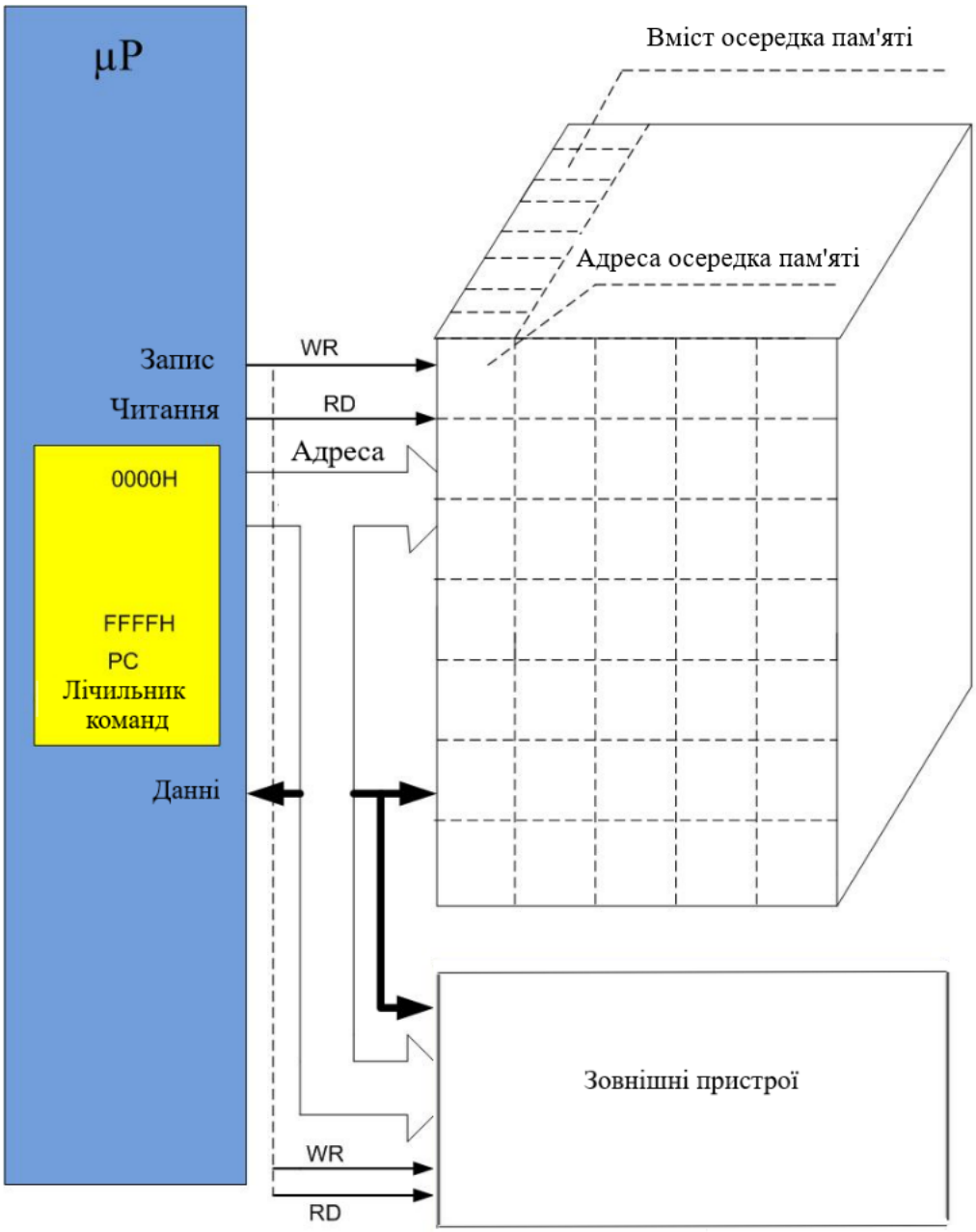


Рисунок 1.37 – Функціональна схема взаємодії мікропроцесора з вузлами комп’ютера

Контрольні питання

1. Які види інформації можуть надходити на вхід системи керування?
2. Що таке об'єкт керування? Які бувають види об'єктів керування?
3. Що таке логічна функція керування?
4. Що є аргументами функції керування?
5. Назвіть основні закони булевої алгебри.
6. Сформулюйте правила складання таблиць істинності.
7. Наведіть приклади базових логічних елементів.

8. Складіть таблицю істинності для повного лінійного дешифратора на 4 входи. Наведіть приклад функціональної схеми.

9. Опишіть функціональні схеми мультиплексора і демультиплексора, які побудовані на базі дешифратора на 3 входи.

10. Складіть таблицю істинності для суматора за модулем 2. Наведіть приклад функціональної схеми.

11. Опишіть процедуру віднімання двійкових чисел для випадків, коли зменшуване більше від'ємного й коли зменшуване менше від'ємного.

12. Складіть таблицю істинності для однорозрядного двійкового суматора та наведіть приклад функціональної схеми.

13. Що таке тригер? Які бувають види тригерів?

14. Складіть таблицю переходів і виходів D-тригера. Наведіть його функціональну схему і тимчасову діаграму.

15. Складіть таблицю переходів і виходів Т-тригера. Наведіть його функціональну схему, виконану на базі D-тригера і тимчасову діаграму.

16. Що таке паралельний регістр? Опишіть його функціональну схему.

17. Що таке послідовний регістр і які бувають його види? Опишіть їх функціональні схеми.

18. Опишіть призначення й принцип роботи двійкового суматора.

19. Опишіть призначення і види запам'ятовувальних пристроїв.

20. Які одиниці використовуються для виміру ємності пам'яті?

21. Розкрийте принцип побудови елементарного осередка ОЗУ. Наведіть приклад функціональної схеми і тимчасової діаграми роботи.

22. Розкрийте принцип побудови багаторозрядного ОЗУ. Наведіть приклад функціональної схеми.

23. Розкрийте принцип побудови елементарного осередка ПЗУ. Наведіть приклад функціональної схеми і тимчасову діаграму роботи.

24. Які методи і способи застосовуються для реалізації дискретних і цифрових систем керування?

2. ЗАГАЛЬНІ ВІДОМОСТІ ПРО МІКРОКОНТРОЛЕРИ

Перші мікропроцесорні пристрої з'явилися на ринку в 1973 р. Фірма «Intel» запропонувала власне мікропроцесорний комплект – сам мікропроцесор Intel 8080 у вигляді одного кристала і комплекта периферійних пристроїв у вигляді окремих ВІС, що містять паралельний інтерфейс, контролер переривань, таймер триканальний, контролер прямого доступу до пам'яті й модуль послідовного інтерфейсу. Побудова систем керування, що працюють у реальному часі, коли реакція системи на мінливі параметри об'єкта повинна бути миттєвою (швидкою), виходила громіздкою і недешевою. Для удосконалення мікропроцесорів (збільшення розрядності й частоти тактування відбувається відповідно до прогресу технологій, а для систем керування – працюючих у реальному часі, фірмою «Intel» була запропонована структура ВІС, у якій на один кристал, крім мікропроцесора, інтегровано всі периферійні пристрої: паралельний інтерфейс, таймери, контролер переривань, послідовний інтерфейс, пам'ять програм і пам'ять даних. Це дозволило скоротити апаратні витрати й здешевити системи керування. Такі ВІС одержали назву мікроконтролери (МК). Перший МК фірмою «Intel» мав марку 8051 і відповідно позначення MCS-51.

Архітектура сімейства MCS-51 виявилася настільки вдалою, що серед 8-розрядних мікроконтролерів на світовому ринку вже багато років вона займає лідируючі позиції. Крім фірми «Intel», мікроконтролери з архітектурою MCS-51 виготовляють фірми «Siemens», «Philips», «Atmel» та ін.

Удосконалення технології і підвищення ступеня інтеграції розширюють функції периферійних пристроїв, збільшують продуктивність, обсяги пам'яті програм і даних, але саме ядро команд MCS-51 залишається повністю сумісним з молодшими моделями, що дозволяє використовувати величезний накопичений досвід розробки і налагодження програмного забезпечення.

У 2020 році компанія «Novaton» виставила на ринок удосконалений мікроконтролер 1T8051, в якому повністю збережена система команд MCS-51 і 8-бітний високопродуктивний мікроконтролер на базі 1T8051 з вбудованою флеш-пам'яттю. Збільшена продуктивність процесора і значно розширені функції периферійних пристроїв. Тому в процесі вивчення мікроконтролерів будуть розглядатися базові питання функціонування процесів обробки інформації та вдосконалені функції периферійних пристроїв.

2.1. Структура і функціональні можливості базової моделі MCS-51 (МК51)

Незважаючи на велику різноманітність мікроконтролерів різних виробників, обчислювальне ядро МК51 у всіх контролерів однакове. МК51 містить 8-розрядний мікропроцесор, генератор тактових імпульсів, схеми керування й синхронізації, внутрішню пам'ять програм і внутрішню пам'ять даних, два таймери, паралельні порти введення/виведення, контролер переривання. Мікросхема живиться від одного джерела енергії напругою +5В, потужністю до 1,5 Вт, допускає експлуатацію в діапазоні температур від –40 до +100°С. Загальними характеристиками для всієї сім'ї групи МК51 є:

- 32 двонаправлені лінії введення/виведення, зображені у вигляді 4-, 8-розрядних портів;
- два 16-розрядних таймери-лічильники;
- контролер переривання;
- 16-розрядний лічильник команд (PC);
- 16-розрядний покажчик даних ДРТR;
- синхронно-асинхронний приймач-передавач послідовного порту зі змінюваною швидкістю передачі;
- генератор тактових імпульсів (ГТІ);
- усі вхідні і вихідні сигнали узгоджуються ТТЛ рівнем.

МК51 виготовляється в корпусі DIP-40. Принципову схему МК51 наведено на рис. 2.1, *а* призначення виводів – у табл. 2.1.

Внутрішню структуру МК51 наведено на рис. 2.2. Основні функціональні вузли МК51 об'єднані двонаправленою 8-розрядною магістраллю. Усі операції виконуються в АЛП, який являє собою 8-розрядний пристрій паралельного типу, що виконує всі арифметичні операції (+, –, ·, :), логічні (AND, OR, NOT, XOR), зсуви, скидання, установку бітів. Найважливішою особливістю АЛП є можливість роботи із бітами, що адресуються прямо в ОЗП і РСФ. АЛП оперує з чотирма видами інформації: 1 біт, 4 біта (тетрада), 8 бітів (тайм), 16 бітів (адреса).

Більшість команд МК51 виконуються за допомогою акумулятора – регістра з бітовою адресацією, що значно спрощує аналіз даних з бітовою структурою інформації. При використанні акумулятора передбачено два види адресації:

- регістрову, код адреси розміщений в коді операції команди (як <A>);

– пряму, при цьому вказується пряма адреса акумулятора (ЕОН) або символьне чи «АСС» в адресній частині команди.

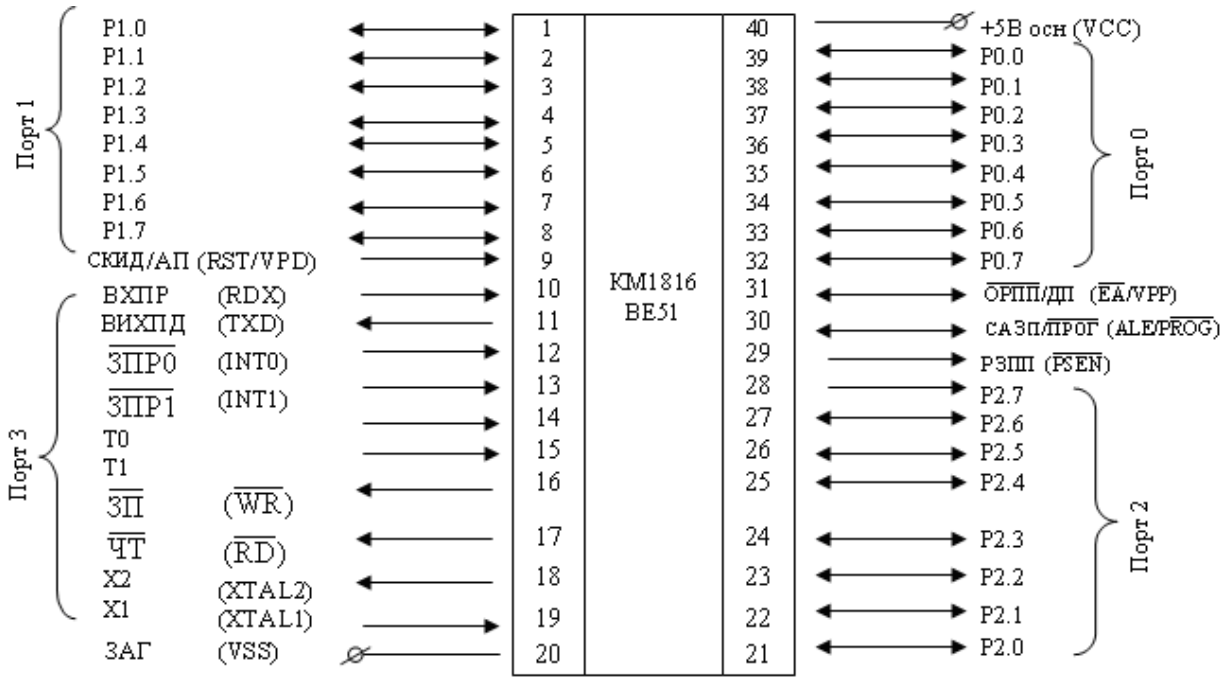


Рисунок 2.1 – Принципова схема мікроконтролера МК51

Таблиця 2.1 – Призначення виводів мікроконтролера МК51

Позначення	Тип	Функція виводу або групи виводів
Порти введення/виведення		
P0.0…P0.7	Вхід-Вихід	Порт 0 (P0) – 8-розрядний двонаправлений порт введення/виведення, передачі коду адреси (молодший байт) або коду даних у мультиплексному режимі під час звертання до зовнішньої пам'яті, введення/виведення при програмуванні й під час перевірки РПП
P1.0…P1.7	Вхід-Вихід	Порт 1 (P1) – 8-розрядний квазідвонаправлений порт введення/виведення для обміну інформацією із зовнішніми пристроями. Використовується також для введення молодших розрядів коду адреси під час програмування й перевірки РПП МК
P2.0…P2.7	Вхід-Вихід	Порт 2 (P2) – 8-розрядний квазідвонаправлений порт введення/виведення, передачі старших розрядів коду адреси під час звертання до зовнішньої пам'яті, а також для введення старших розрядів коду адреси і сигналів керування під час програмування і перевірки РПП
P3.0…P3.7	Вхід-Вихід	Порт 3 (P3) – 8-розрядний квазідвонаправлений порт введення/виведення, звичайно використовується для реалізації периферійних функцій
P3.0	Вхід	Rxd – вхід приймача послідовного порту в асинхронному режимі або вхід-вихід даних у синхронному режимі
P3.1	Вихід	TxD – вихід передавача послідовного порту в асинхронному режимі або видача синхроімпульсів у синхронному режимі
P3.2	Вхід	$\overline{\text{INT0}}$ – вхід запиту від зовнішнього джерела переривання з умовним номером 0

P3.3	Вхід	$\overline{\text{INT1}}$– вхід запиту від зовнішнього джерела переривання з умовним номером 1
P3.4	Вхід	T0 – вхід таймера-лічильника з номером 0
P3.5	Вхід	T1 – вхід таймера-лічильника з номером 1
P3.6	Вихід	$\overline{WR}$ – «запис» – вихідний сигнал запису байта в зовнішню пам'ять даних (ЗПД). Активний рівень сигналу – логічний «0»
P3.7	Вихід	$\overline{RD}$ – «читання» – вихідний сигнал читання байта із зовнішньої пам'яті даних (ЗПД). Активний рівень сигналу – логічний «0»
Сигнали керування МК		
ALE/#PROG	Вихід (вхід)	Стробування адреси зовнішньої пам'яті. Використовується для керування режимом мультиплексування адреси і даних, які передаються через порт P0 при звертанні до ЗП. Якщо ALE=1, то на контактах виведення порту P0 перебуває адреса. При програмуванні МК на вивід подається логічний «0»
#PSEN	Вихід	Дозвіл зовнішньої пам'яті програм. Виконує роль стробування прийняття байта команди в МК під час вибірки команд із ЗПП. Активний рівень сигналу – логічний «0»
#EA/VPP	Вхід	Сигнал вимкнення резидентної пам'яті програм (РПП). Якщо подано #EA=1, то будуть виконуватися команди, розміщені в РПП, при цьому (PC)=0000….0FFFh. Якщо подано #EA=0, то будуть виконуватися команди, розміщені тільки у ЗПП (РПП повністю недоступна). Під час програмування МК на цей вивід подається імпульс напругою +21 В
#EA/VPP	Вхід	Сигнал вимкнення резидентної пам'яті програм (РПП). Якщо подано #EA=1, то будуть виконуватися команди, розміщені в РПП, при цьому (PC)=0000….0FFFH. Якщо подано #EA=0, то будуть виконуватися команди, розміщені тільки у ЗПП (РПП повністю недоступна). Під час програмування МК на цей вивід подається імпульс напругою +21 В
RST/VRD	Вхід	Сигнал скидання МК (тобто переведення в початковий стан). Рівень сигналу 3,5 В повинен утримуватися не менш ніж 2 мкс. Використовується також для увімкнення аварійного джерела живлення
Сигнали синхронізації МК		
XTAL1	Вхід	Вхід генератора-підсилювача синхросигналів. Підключається до зовнішнього джерела синхронізації (кварцового резонатора, увімкненого за схемою із «середньою точкою»), рис. 2.5
XTAL2	Вихід	Вихід генератора-підсилювача синхросигналів. Підключення є аналогічним підключенню XTAL1
Ucc	-	Підключення до джерела живлення напругою *U*cc=+5 В±10%
Vss	-	«Загальний» контакт

Регістр PSW відображає ознаку або стан програми. У ньому формуються ознаки результатів, що отримуються в АЛП. Регістр PSW забезпечує керування вибором робочого банку регістрів, допускає побітову адресацію з використанням символічних імен PSW.3, PSW.4.

Зазначений регістр DPTR має двобайтову структуру та є базовим при звертанні до зовнішньої пам'яті. Якщо зовнішня пам'ять відсутня, то DPTR можна використовувати як регістр загального призначення.

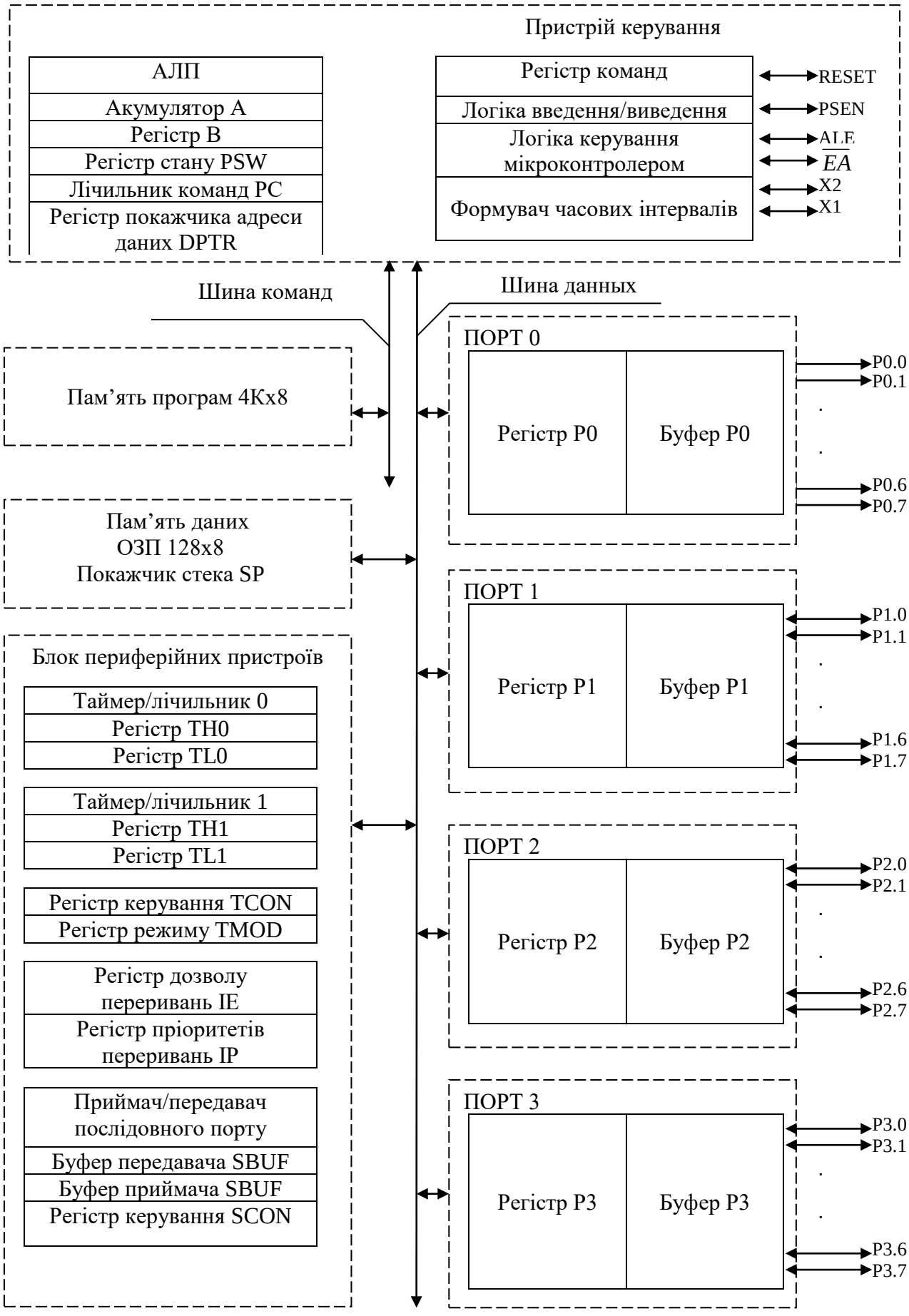


Рисунок 2.2 – Внутрішня структура МК51

Функціональну структуру зв'язків наведено на рис. 2.3, з якої видно, що всі периферійні пристрої взаємодіють із мікропроцесором через контролер переривань шляхом формування сигналу – запит на переривання (два сигнали від таймерів T0, T1, два сигнали від зовнішнього середовища контролера $\overline{\text{INT0}}$, $\overline{\text{INT1}}$) і один сигнал від буфера послідовного обміну ($\text{RI} \vee \text{TI}$).

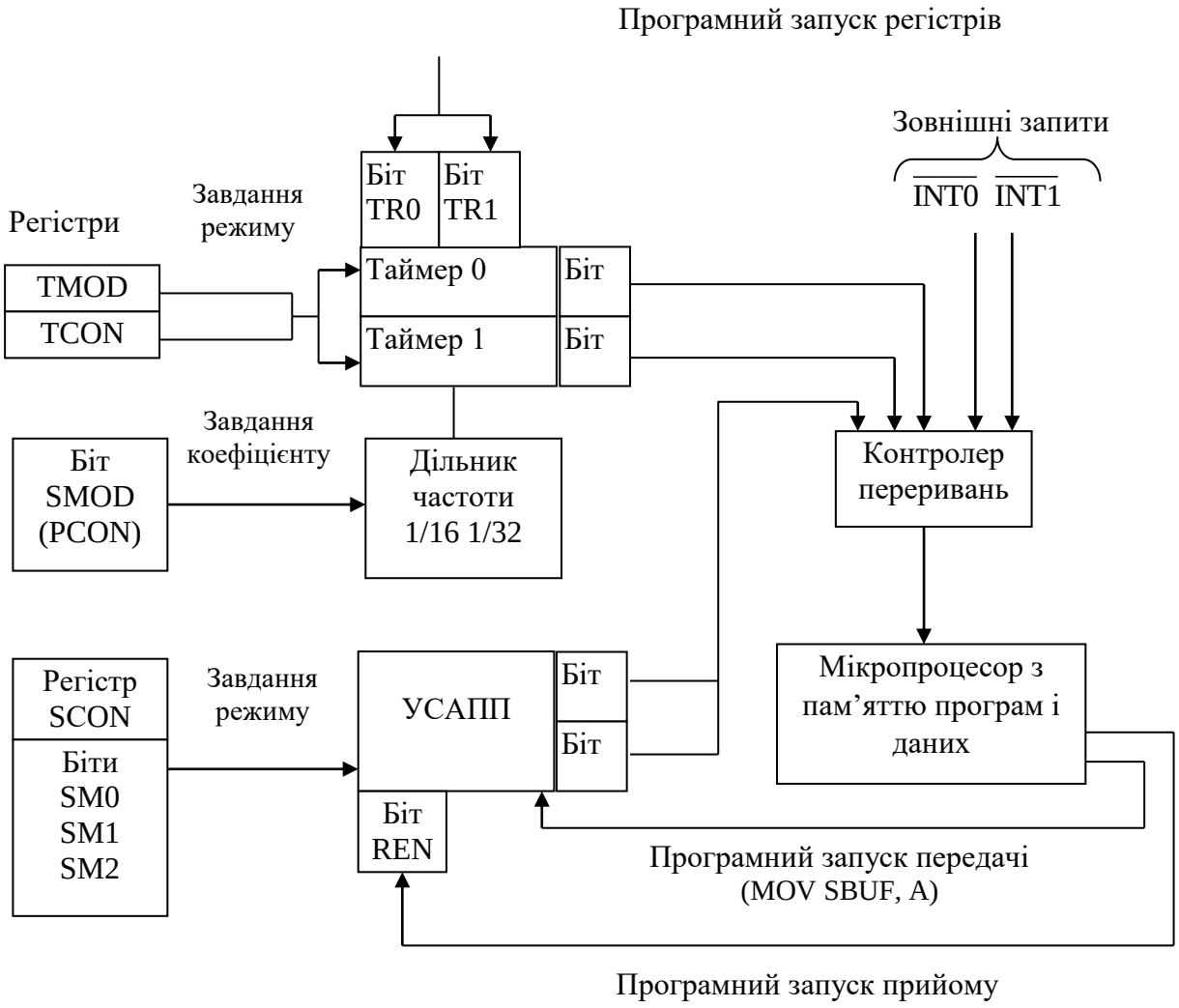


Рисунок 2.3 – Функціональна структура зв'язків внутрішніх вузлів МК51

Роботу мікроконтролера поділено на машинні цикли, а машинні цикли – на машинні такти (рис. 2.4).

Кварцовий резонатор підключається до зовнішніх виводів X1 і X2 МК51 (рис. 2.5, *а*) і визначає частоту коливань внутрішнього генератора, який формує всі сигнали керування і синхронізації. При подачі напруги живлення в системі завжди формується сигнал скидання (RST) (рис. 2.5, *б*), і після цього робота мікроконтролера поділяється на машинні цикли, а машинні цикли – на такти (рис. 2.4).

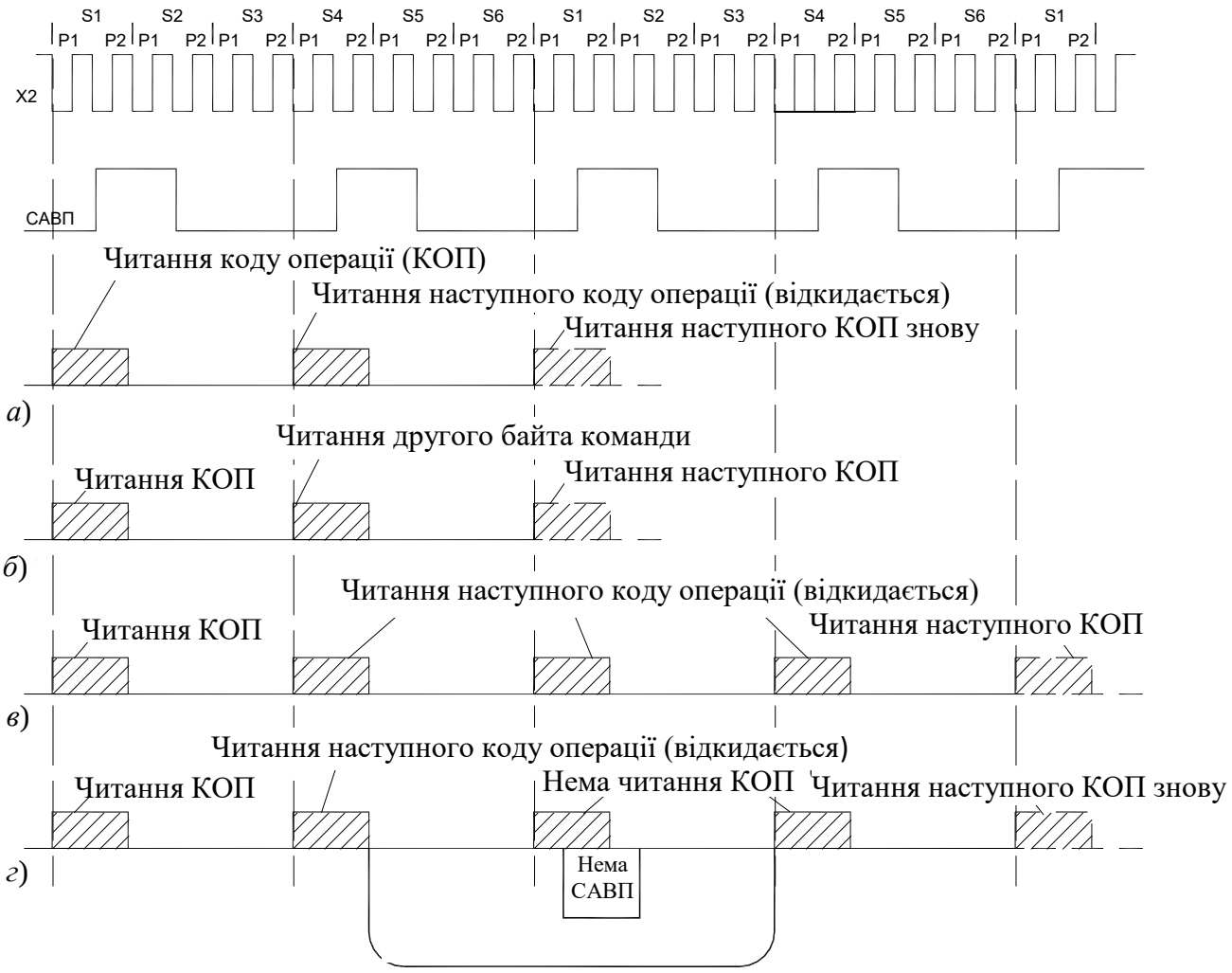


Рисунок 2.4 – Основні машинні цикли роботи МК51

Машинні цикли мають фіксовану тривалість, яка дорівнює 12-м періодам коливань резонатора або шести станам первинного керуючого автомата (S1-S6). Кожний стан керуючого автомата складається з двох фаз (P1, P2) сигналів резонатора. У фазі P1, як правило, виконуються операції в АЛП, а у фазі P2 – межрегістрові пересилання інформації. Увесь машинний цикл складається з 12 фаз, починаючись з фази S1P1 і закінчуючись фазою S6P2 (рис. 2.4). Усі заштриховані сигнали, наведені на рис. 2.4, є внутрішніми і недоступні користувачеві для спостереження.

Спостережуваними є сигнали резонатора і сигнал стробування адреси зовнішньої пам'яті (ШАЗП), який формується двічі за один машинний цикл (S1P2-S2P1 і S4P2-S5P1). Цей сигнал використовується для керування процесом пересилання інформації від МК51 і до зовнішньої пам'яті.

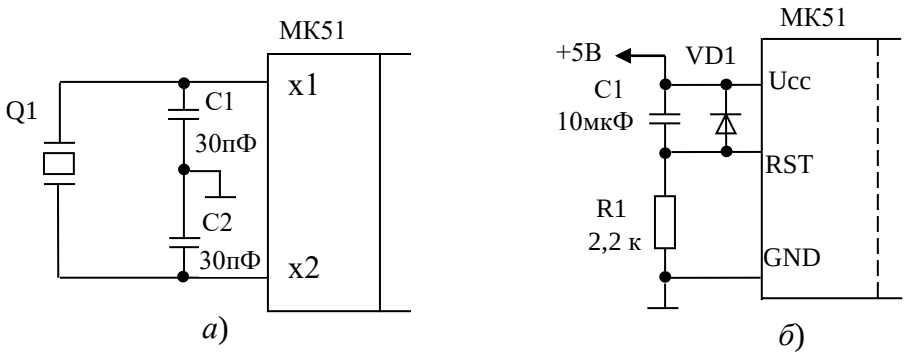


Рисунок 2.5 – Схеми підключення кварцових резонаторів до МК51

2.2. Програмно доступні ресурси МК51

У МК51 використовується гарвардська архітектура, у якій пам'ять програм і даних розділено, тобто мають свої шини адреси й даних.

Структура пам'яті може мати вигляд як на рис. 2.6. Пам'ять програм розділено на внутрішню (4 кбайт) і зовнішню (60 кбайт). При звертанні до молодших 4 кбайт адрес-сигнал $\overline{EA} = 0$. При звертанні до старших 60 кбайт адрес-сигнал $\overline{EA} = 1$.

Пам'ять даних також розділено на зовнішню й внутрішню. Внутрішня пам'ять даних у базовій моделі МК51 має розмір 256 – 128 байт внутрішнього ОЗП і 128 адрес регістрів спеціальних функцій.

З них молодші 32 адреси визначено за банками регістрів від 00F до 1Fh. Осередки за адресами 20h до 2Fh визначені за бітами, що адресуються прямо.

Область внутрішнього ОЗП визначена за адресою OOh до 7Fh (рис. 2.6). Чотири банки регістрів загального призначення мають по вісім байтів у кожному. При роботі мікропроцесора один з банків вважається робочим (активним). Вибір робочого банку здійснюється програмно, установкою/скиданням двох бітів у регістрі стану програми (PSW). Адресація до робочих регістрів здійснюється символічними іменами R0…R7. Регістри R0 і R1 кожного банку можуть використовуватися для непряморегістрової адресації до 128 осередків внутрішньої пам'яті даних.

Стекова пам'ять (пам'ять магазинного типу – першим увійшов, останнім вийшов) служить для запам'ятовування проміжних даних при виконанні підпрограм. Адресація до осередків стекової пам'яті здійснюється за допомогою 8-розрядного покажчика стека (SP). Звичайно стекова пам'ять розміщується у внутрішній пам'яті даних (ОЗП), починаючи за адресою 40h и вище. При увімкненні живлення або за сигналом скидання (RESET) в SP автоматично

завантажується код 07h. При завантаженні стека командами PUSH вміст SP збільшується, а при вивантаженні командами POP і RET, RETI вміст SP зменшується. Шляхом запису в SP значень від 0 до 127 адресна область стека може бути розміщена в будь-якій області адресного простору РПД.

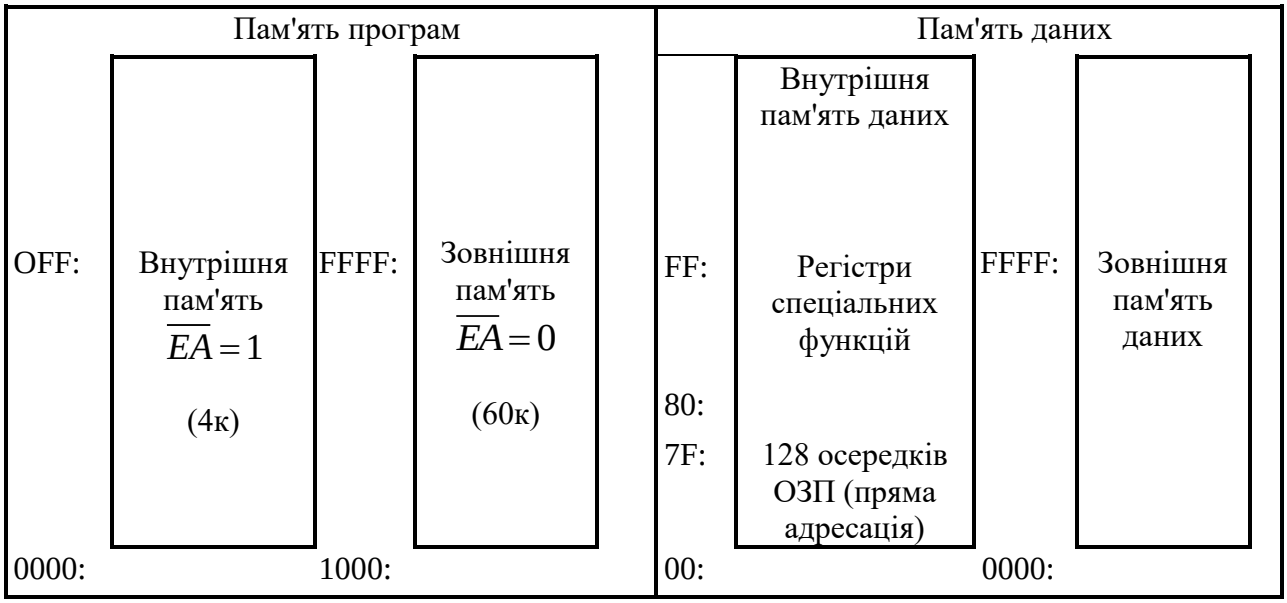


Рисунок 2.6 – Адресний простір мікроконтролера

Таблиця 2.2 – Структура внутрішньої пам'яті даних

Шістнадцяткова адреса							Десяткова адреса		
7F:									127
2F:	7F	7E	7D	7C	7B	7A	79	78	47
2E:	77	76	75	74	73	72	71	70	46
2D:	6F	6E	6D	6C	6B	6A	69	68	45
2C:	67	66	65	64	63	62	61	60	44
2B:	5F	5E	5D	5C	5B	5A	59	58	43
2A:	57	56	55	54	53	52	51	50	42
29:	4F	4E	4D	4C	4B	4A	49	48	41
28:	47	46	45	44	43	42	41	40	40
27:	3F	3E	3D	3C	3B	3A	39	38	39
26:	37	36	35	34	33	32	31	30	38
25:	2F	2E	2D	2C	2B	2A	29	28	37
24:	27	26	25	24	23	22	21	20	36
23:	1F	1E	1D	1C	1B	1A	19	18	35
22:	17	16	15	14	13	12	11	10	34
21:	0F	0E	0D	0C	0B	0A	09	08	33
20:	07	06	05	04	03	02	01	00	32
1F: 18:	R7 . . R0				БАНК3				31 24
17:	R7 .				БАНК2				23

10:	. R0								16
0F: 08:	R7 . . R0				БАНК1				15 8
07: 00:	R7 . . R0				БАНК0				7 0

Таблиця 2.3 – Адресний простір регістрів спеціальних функцій

Шістнадцяткова адреса							Десяткова адреса		Ім'я регістра
Регістр загального призначення									
0FF:									
0F0:	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0:	E7	E6	E5	E4	E3	E2	E1	E0	ACC
0D0:	CY D7	AC D6	F0 D5	RS1 D4	RS0 D3	OV D2		P D0	PSW
Регістр керування портами									
0B0:	B7	B6	B5	B4	B3	B2	B1	B0	P3
0A0:	A7	A6	A5	A4	A3	A2	A1	A0	P2
90:	97	96	95	94	93	92	91	90	P1
80:	87	86	85	84	83	82	81	80	PO
Регістр керування таймерами									
8C:									TH0
8A:									TL0
8D:									TH1
8B:									TL1
89:									TMOD
88:	TF1 8F	TR1 8E	TF0 8D	TR0 8C	IE1 8B	IT1 8A	IE0 89	IT0 88	TCON
Регістр керування системою переривань									
0B8:			PT2 BD	PS BC	PT1 BB	PX1 BA	PT0 B9	PX0 B8	IP
0A8:	EA AF		ET2 AD	ES AC	ET1 AB	EX1 AA	ET0 A9	EX0 A8	IE
Регістр керування послідовним портом									
99:									SBUF
98:	SM0 9F	SM1 9E	SM2 9D	REN 9C	TB8 9B	RB8 9A	TI 99	RI 98	SCON
87:									PCON
Регістр керування стековою пам’яттю									
81:									SP
Регістр-покажчик зовнішньої пам’яті									
83:									DPH
82:									DPL

Адреси внутрішньої пам'яті даних відведені для регістрів спеціальних функцій (РСФ) наведені в табл. 2.3. Усі регістри мають, крім адреси, як осередки внутрішньої пам'яті даних, так і символічні імена. Частина регістрів допускає побітову адресацію.

Прикладом є регістр стану програми PSW.

Призначення окремих розрядів регістра PSW наведено в табл. 2.4.

Таблиця 2.4 – Призначення окремих розрядів регістра PSW

<table>
<tr><th>Назва біта</th><th>Позиція</th><th>Призначення</th></tr>
<tr><td>C</td><td>PSW.7</td><td>Прапорець переносу. Встановлюється і скидається апаратно при виконанні арифметичних, логічних і бітових операцій, а також програмно</td></tr>
<tr><td>AC</td><td>PSW.6</td><td>Прапорець допоміжного переносу. Встановлюється і скидається тільки апаратно при виконанні команд підсумовування й віднімання у випадку виникнення переносу або позики в біті 3 акумулятора</td></tr>
<tr><td>F0</td><td>PSW.5</td><td>Вільний прапорець. Може бути змінений програмно і використовується за призначенням, встановленим програмістом</td></tr>
<tr><td>RS1
RS0</td><td>PSW.4
PSW.3</td><td>Вибір банку регістрів. Біти встановлюються і скидаються програмно для вибору активного (робочого) банку регістрів:
<table>
<tr><th>RS1</th><th>RS0</th><th>Активний банк</th><th>Адреси РПД</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>00h-07h</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>08P-0Fh</td></tr>
<tr><td>1</td><td>0</td><td>2</td><td>10h-17h</td></tr>
<tr><td>1</td><td>1</td><td>3</td><td>18h-1 Fh</td></tr>
</table></td></tr>
<tr><td>OV</td><td>PSW.2</td><td>Прапорець переповнення. Встановлюється і скидається апаратно при виконанні арифметичних операцій у випадку переповнення акумулятора. Дає можливість коректно виконувати дії над числами, зображеними в додатковому коді</td></tr>
<tr><td>-</td><td>PSW.1</td><td>Не використовується</td></tr>
<tr><td>P</td><td>PSW.0</td><td>Прапорець паритету. Встановлюється і скидається апаратно в кожному циклі команди, фіксує факт непарної кількості «1» в акумуляторі</td></tr>
</table>

До РСФ можна звертатися як до звичайних осередків РПД. Більш детальне призначення і функціональні характеристики регістрів спеціальних функцій (РСФ) буде розглянуто при вивченні периферійних пристроїв.

Внутрішня структура удосконаленого мікроконтролера (1T8051) наведена на рис. 2.7.

Удосконалений 1T8051 містить до 18 кбайт основної флеш-пам'яті, названої APROM, в якій розміщується програмне забезпечення користувача, рис. 2.8.

Flash підтримує функцію внутрішнього програмування (IAP), яка включає оновлення вбудованого ПЗ. IAP також дозволяє налаштувати будь-який блок коду користувача за допомогою використання команд MCS8051.

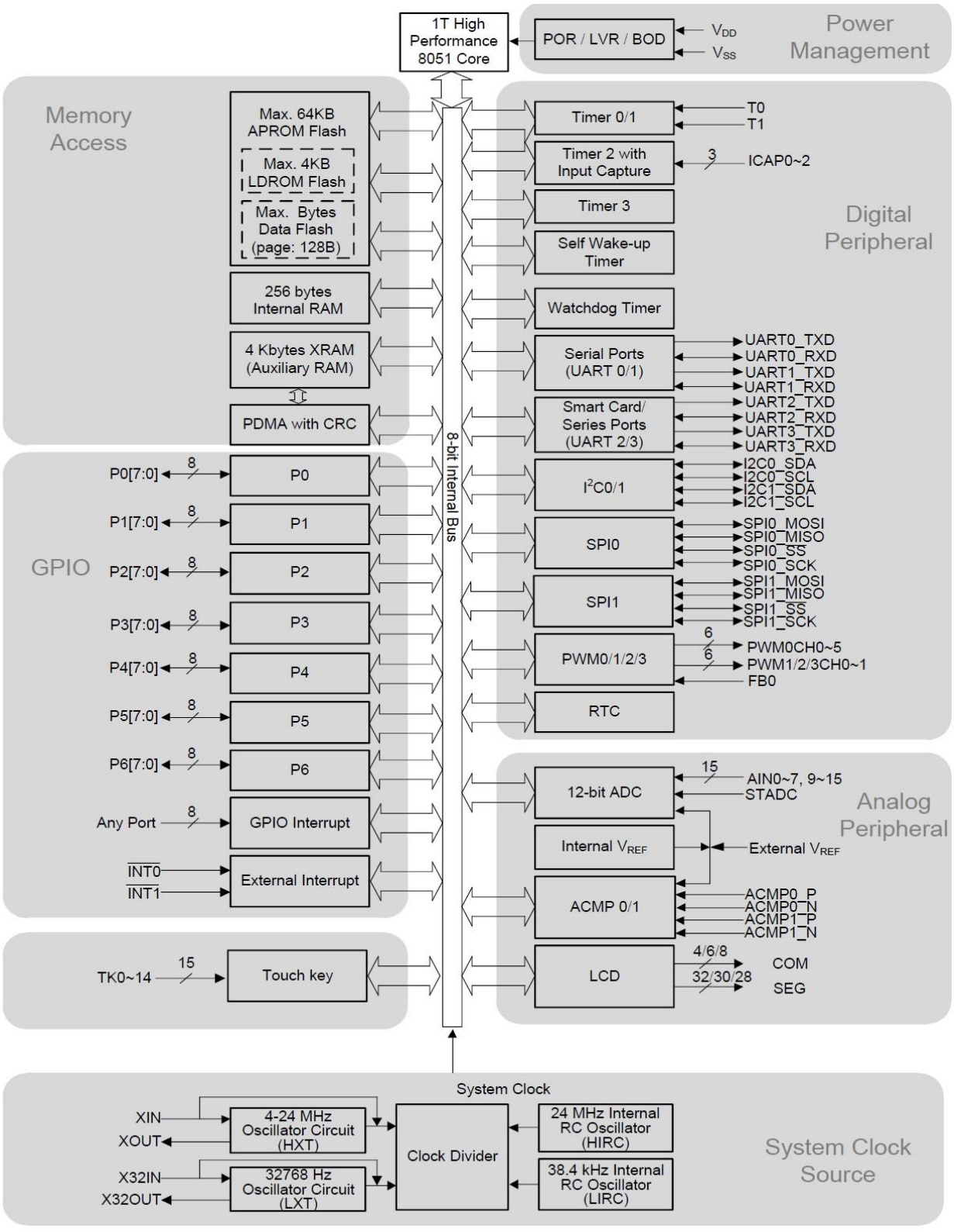


Рисунок 2.7 – Внутрішня структура мікроконтролера 1T8051

Існує додаткова флеш-пам'ять під назвою LDROM (табл. 2.5) для проведення внутрісистемного програмування (ISP). Розмір LDROM налаштовується до 4 квytes за допомогою конфігурування бітів регістра спеціальних функцій (рис. 2.8).

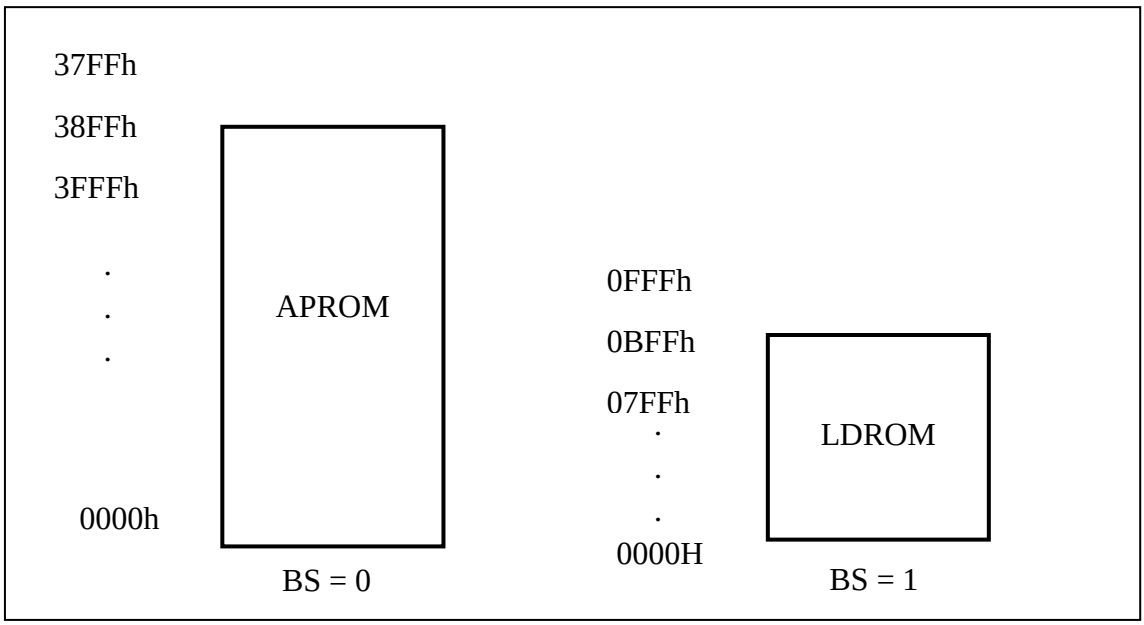


Рисунок 2.8 – Структура пам’яті програм мікроконтролера 1T8051

Для полегшення програмування і перевірки Flash дозволяє програмувати і читати вміст осередків пам’яті. Удосконалений мікроконтролер забезпечує багату периферію, включаючи 256 байтів SRAM, 768 байтів допоміжної оперативної пам'яті XRAM (рис. 2.9), в якій використовуються команди адресації до зовнішньої пам’яті, до 18-ти входів/виходів загального призначення, два 16-розрядний таймера-лічильника 0/1, один 16-розрядний таймер 2 з трьома модулями захоплення вхідного каналу, один сторожовий таймер (WDT), один таймер самовідтворення (WKT), чотири послідовних інтерфейси, два UART, один SPI-інтерфейс, один I2C, п'ять поліпшених вихідних каналів ШІМ, 12-розрядний АЦП, периферійні пристрої 18-ти джерел з можливістю переривання з пріоритетом 4-го рівня. 1T8051 використовує три джерела імпульсів і підтримує перемикання на льоту за допомогою програмного забезпечення. Три джерела синхронізації включають у себе зовнішній вхідний сигнал, внутрішній генератор 10 кГц і один внутрішній точний генератор 16 МГц, який налаштовується на заводі до ±1% при кімнатній температурі забезпечує 4-рівнева виявлення відключення, яке можна скинути при вмиканні напруги джерела.

Таблиця 2.5 – Налаштування розміру внутрісистемної пам’яті LDROM

Bit	Name	Description
2:0	CONFIG1[2:0]	**LDROM size select** This field selects the size of LDROM 111 = No LDROM. APROM is 18 kbytes 110 = LDROM is 1 kbytes. APROM is 17 kbytes 101 = LDROM is 2 kbytes. APROM is 16 kbytes 100 = LDROM is 3 kbytes. APROM is 15 kbytes 0xx = LDROM is 4 kbytes. APROM is 14 kbytes

Рисунок 2.9 – Структура пам’яті даних мікроконтролера 1T8051

Незаймані адреси в просторі SFR (табл. 2.6) позначені «–» і зарезервовані для подальшого використання. Доступ до цих областей матиме невизначений ефект і його слід уникати.

Таблиця 2.6 – Програмування режимів роботи периферійних пристроїв за допомогою регістрів спеціальних функцій

SFR Page	Addr	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
0 1	F8	SCON_1	PDTEN	PDNCNN	PMEN	PMD	PORDIS	EIP1	EIPH1
0 1	F0	B	CAPCON3	CAPCON4	SPCR SPCR2	SPSR	SPDR	AINDIDS	EIPH
0 1	E8	ADCCON0	PICON	PINEN	PIPEN	PIF	C2L	C2H	EIP
0 1	E0	ACC	ADCCON1	ADCCON2	ADCDLY	C0L	C0H	C1L	C1H
0 1	D8	PWMCON0	PWMPL	PWM0L	PWM1L	PWM2L	PWM3L	PIOCON0	PWMCON1
0 1	D0	PSW	PWMPH	PWM0H	PWM1H	PWM2H	PWM3H	PNP	FBD
0 1	C8	T2C0N	T2MOD	RCMP2L	RCMP2H	TL2 PWM4L	TH2 PWM5L	ADCMPL	ADCMPH
0 1	C0	I2CON	I2ADDR	ADCRL	ADCRH	T3CON PWM4H	RL3 PWM5H	RH3 PIOCON1	TA

0 1	B8	IP	SADEN	SADEN_1	SADDR_1	I2DAT	I2TAT	I2CLK	I2TOC
0 1	B0	P3	P0M1 P0S	P0M2 P0SR	P1M1 P1S	P1M2 P1SR	P2S	-	IPH PWMINTC
0 1	A8	IE	SADDR	WDCON	BODCON1	P3M1 P3S	P3M2 P3SR	IAPFD	IAPCN
0 1	A0	P2	-	AUXR1	BODCON0	IAPTRG	IAPUEN	IAPAL	IAPAH
0 1	98	SCON	SBUF	SBUF_1	EIE	EIE1	-	-	CHPCON
0 1	90	P1	SFRS	CAPCON0	CAPCON1	CAPCON2	CKDIV	CKSWT	CKEN
0 1	88	TCON	TMOD	TL0	TL1	TH0	TH1	CKCON	WKCON
0 1	80	P0	SP	DPL	DPH	RCTRIM0	RCTRIM1	RWK	PCON

Контрольні питання

1. Який максимальний обсяг зовнішньої пам’яті даних у MCS-51?
2. Скільки програмованих портів містить MCS-51?
3. Який мінімальний обсяг пам’яті програм у MCS-51?
4. Скільки розрядів містить лічильник команд MCS-51 ?
5. Скільки запитів на переривання опрацьовується в MCS-51 ?
6. Скільки двійкових розрядів містить універсальний програмований порт у MCS-51?
7. Яку кількість розрядів містить покажчик стекової пам’яті в MCS-51?
8. Скільки таймерів-лічильників містить MCS-51?
9. Скільки двійкових розрядів містить шина-даних MCS-51?
10. Скільки двійкових розрядів містить шина-адреса MCS-51?
11. Які сигнали мікропроцесора визначають напрямок передачі інформації?
12. При виконанні мікропроцесором програми вміст лічильника-команд зменшується чи збільшується?
13. Скільки двійкових розрядів містить регістр банку даних MCS-51?
14. Скільки регістрів містить банк даних MCS-51 ?
15. Скільки банків даних містить MCS-51?
16. Який максимальний обсяг зовнішньої пам’яті програм у MCS-51?

3. СИСТЕМА КОМАНД МІКРОКОНТРОЛЕРА СЕРІЇ MCS-51

3.1. Загальні положення

Команда мікропроцесора являє собою машинний код. Для виконання команди мікропроцесор завжди здійснює певну послідовність дій:

- витяг команди – читання з осередка пам'яті, адреса якої міститься в лічильнику команд і завантаження її в регістр команд;
- декодування команди;
- додаткове читання інформації з пам’яті у випадку, якщо команда має розмірність більше ніж один байт (розмірність команди визначається на етапі декодування команди);
- виконання команди – формування керуючих сигналів (залежно від результату декодування) для різних складових частин мікропроцесора, під впливом яких відбувається виконання операцій, визначених даною командою.

Написання програмного забезпечення в машинних кодах досить трудомісткий процес, тому для спрощення цього процесу створена мова програмування – «Асемблер». Це машинно-орієнтована мова програмування низького рівня. Вона являє собою систему позначень, яка використовується для подання в зручній формі програм, що написані в машинному коді.

Переклад програми з мови «Асемблер» в здійсненний машинний код (обчислення виразів, розкриття макрокоманд, заміна мнемонік власне машинними кодами і символьних адрес на абсолютні або відносні адреси) проводиться асемблером – програмою-транслятором, яка і дала мові «Асемблер» його назву.

У мові «Асемблер» застосовується мнемонічне позначення команд (машинних кодів) мікропроцесора. Мнемонічні позначення – це декілька символів (як правило, три – чотири літери) від скороченої назви операції (функції), що реалізується мікропроцесором при виконанні даної команди. Команда (у вигляді асемблера) залежно від функції, що здійснюється командою, може в своєму складі мати операнди. Таким чином, команда може бути без операндів або мати один або декілька операндів (зазвичай кількість операндів від 1 до 3).

Щодо мікроконтролерів серії MCS-51 система команд має 42 мнемонічних позначення команд для реалізації 33 функцій. У машинному коді команди займають один, два або три байта і виконуються за один або два машинних цикли (команди множення і ділення – за чотири).

Синтаксис більшості команд на мові «Асемблер» складається з мнемонічного позначення функції, за яким ідуть операнди, що вказують методи

адресації і типи даних. Різні типи даних або режими адресації визначаються зазначеними операндами, а не зміною мнемонічних позначень функції.

В загальному вигляді команда мікроконтролера серії MCS-51 в асемблерній формі має такий вигляд:

Команда Операнд 1, Операнд 2, Операнд 3

Як вже відзначалося, команда може бути без операндів чи мати один або декілька операндів. Операнди можуть визначати спосіб адресації до:

- джерела інформації, їх називають «операнди-джерело»;
- одержувача інформації, їх називають «операнди-призначення».

В загальному вигляді команда мікроконтролера серії MCS-51 у машинних кодах структурно має такий вигляд:

1-й байт (код команди)	2-й байт (розширення команди)	3-й байт (розширення команди)

Як вже відзначалося, команда в машинних кодах може мати розмірність один, два або три байта. Перший байт завжди містить код команди, який визначає функцію, що здійснюється при виконанні команди, а також розмірність команди (наявність другого та третього байтів).

Кожна команда мікропроцесора (мікроконтролера) виконується за певний час, який визначається архітектурою мікропроцесора та його схемотехнічними особливостями. Час виконання прийнято вимірювати в машинних циклах мікропроцесора. Всі команди мікроконтролера серії MCS-51 (залежно від команди і способу адресації) виконуються за один або два машинних цикли, винятком є команди множення і ділення, які виконуються за чотири машинних цикли.

При розгляді формату команд використовуються зарезервовані імена мікроконтролера та такі позначення:

<байт-джер.> – «байт-джерело» – джерело інформації з розмірністю даних один байт;
<байт-призн.> – «байт-призначення» – одержувач інформації з розмірністю даних один байт;
<біт-джер.> – «біт-джерело» – джерело інформації з розмірністю даних один біт;
<біт-призн.> – «біт-призначення» – одержувач інформації з розмірністю даних один біт;
<мітка> – «мітка» – символьне позначення адреси осередка пам’яті програм;

мц	–	«машинний цикл» – визначає час виконання команди в одиницях вимірювання «машинний цикл»;
bit	–	пряма адреса біта, що допускає пряму адресацію. Значення bit може набувати значення від 00h до FFh. Слід зазначити, якщо значення bit в діапазоні від 00h до 7Fh, то адресація здійснюється до бітів бітової зони внутрішньої пам'яті даних, а якщо в діапазоні від 80h до FFh, то адресація здійснюється до бітів, що розташовані в регістрах спеціальних функцій;
data	–	безпосередні дані розмірністю один байт;
data16	–	безпосередні дані розмірністю два байта;
direct	–	пряма адреса осередка внутрішньої пам'яті даних. Значення direct може набувати значення від 00h до FFh. Слід зазначити, якщо значення direct більш ніж 7Fh, то адресація здійснюється до регістрів спеціальних функцій;
label	–	адреса осередка пам'яті, що визначається міткою;
R16	–	один з 16-бітових регістрів. Як 16-бітовий регістр може бути вказівник даних – регістр DPTR або лічильник команд – регістр PC;
Ri	–	один з регістрів загального призначення (R0, R1) в поточному банку регістрів загального призначення, що може застосовуватися як вказівник;
Rn	–	один з регістрів загального призначення (R0, R1, …, R7) в поточному банку регістрів загального призначення. Структура зв’язків програмної моделі MCS-51 наведена рис. 3.1, з якої видно що більшість команд мікропроцесора використовують регістр – акумулятор.

3.2. Способи адресації

Як вже відзначалося, операнди можуть визначати спосіб адресації до джерела інформації та одержувача інформації.

Способи адресації операндів–джерел:

- регістрова адресація;
- пряма адресація;
- непряморегістрова адресація;
- безпосередня адресація;
- непряма адресація за сумою вмісту базового та індексного регістрів.

Способи адресації операндів–призначення:

- регістрова адресація;
- пряма адресація;
- непряморегістрова адресація.

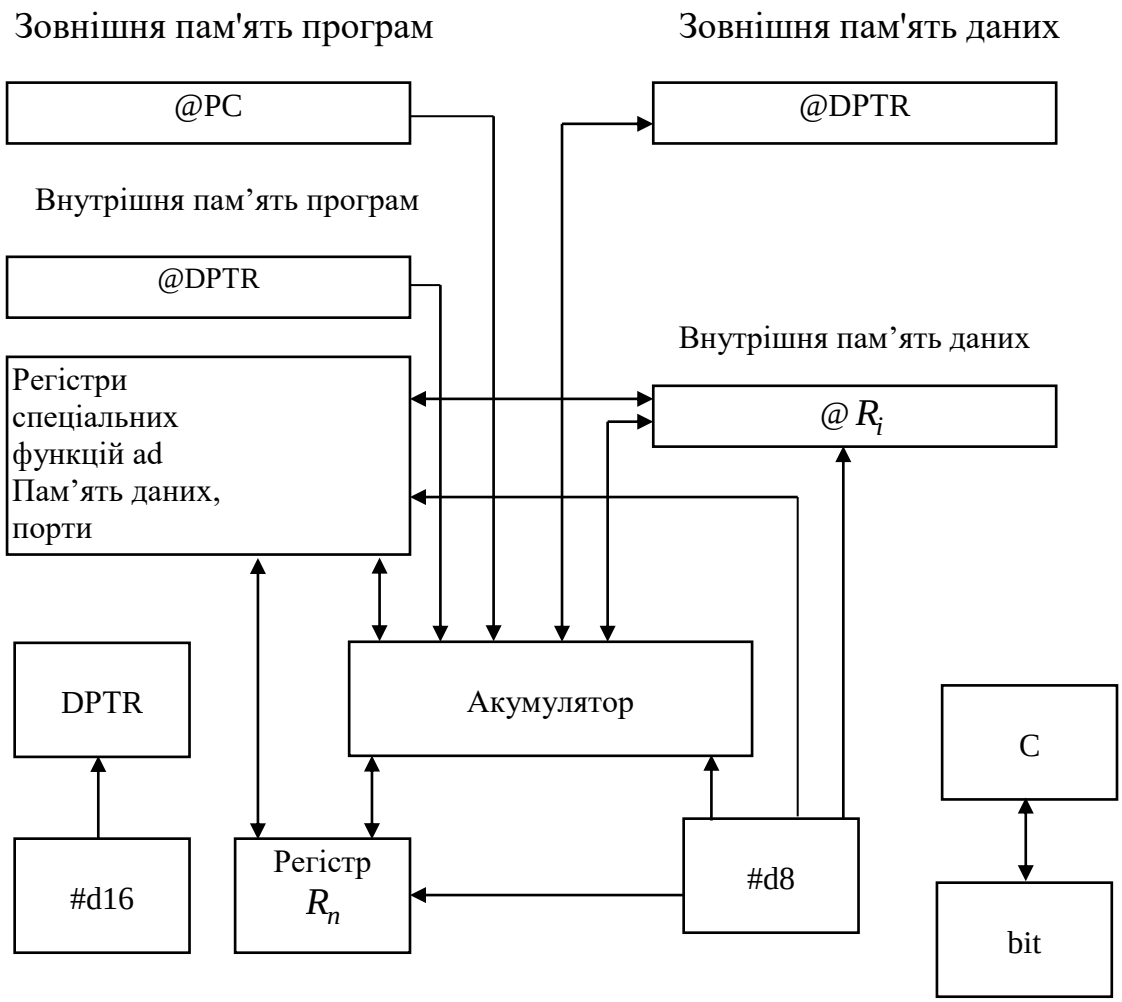


Рисунок 3.1 – Структура зв'язків програмної моделі MCS-51

Регістрова адресація

Регістрова адресація використовується для звернення до восьми регістрів загального призначення (поточного банку регістрів загального призначення). Слід зазначити, що звернення до цих регістрів може бути здійснено за допомогою прямої адресації і непряморегістрової адресації (як до звичайних осередків внутрішньої пам'яті даних).

Регістрова адресація також використовується для звернення до регістрової пари AB, регістрів A, B, DPTR і до прапорця переносу C. Використання регістрової адресації дозволяє отримати двобайтовий еквівалент трибайтових команд прямої адресації.

Пряма адресація

Пряма байтова адресація використовується для звернення до внутрішньої пам'яті даних і до регістрів спеціальних функцій.

Пряма бітова адресація використовується для звернення до 128 бітів у бітовій зоні внутрішньої пам'яті даних і до окремих бітів у регістрах спеціальних функцій.

Старший біт прямої адреси визначає звернення до бітової зони внутрішньої пам'яті даних або до окремих бітів у регістрах спеціальних функцій.

Непряморегістрова адресація

Непряморегістрова адресація використовується для звернення до осередків внутрішньої пам'яті даних. Як вказівник можуть використовуватися регістри R0, R1 поточного банку регістрів загального призначення.

При виконанні команд PUSH і POP (запис у стек і читання зі стека) як вказівник використовується регістр SP – вказівник стека.

Непряморегістрова адресація також використовується для звернення до осередків зовнішньої пам'яті даних. Можливі два способи організації непряморегістрової адресації до зовнішньої пам'яті даних. Використання першого або другого способу визначається схемним рішенням підключення зовнішньої пам'яті даних до мікроконтролера.

У першому варіанті – шина адреси зовнішньої пам'яті даних підключена до портів P0, P2, що забезпечує доступ до всього адресного простору зовнішньої пам'яті даних. У цьому випадку як вказівник може використовуватися регістр DPTR, який має розрядність 16 біт. Це дозволяє отримати доступ до будь-якого осередка зовнішньої пам'яті даних у межах 64 кбайт.

У другому варіанті – шина адреси зовнішньої пам'яті даних підключена таким чином: вісім молодших розрядів – до порту P0, а вісім старших розрядів – через будь-які лінії портів P1, P2, P3 (можливо з додатковим схемним рішенням, що забезпечує перемикання сторінок зовнішньої пам'яті). У цьому випадку зовнішня пам'ять даних має сторінкову організацію (розмір сторінки 256 байт) і як вказівник можуть використовуватися регістри R0, R1 поточного банку регістрів загального призначення.

Безпосередня адресація

Безпосередня адресація дозволяє вибрати з адресного простору пам'яті програм дані (константи), зазначені в командах явно.

Непряморегістрова адресація за сумою базового та індексного регістрів

Непряморегістрова адресація за сумою базового та індексного регістрів дозволяє здійснювати вибір даних (констант) з таблиць, розміщених у пам'яті програм.

Як базовий регістр може виступати регістр DPTR або лічильник команд PC, а як індексний регістр – акумулятор A.

Перш ніж розглядати команди мікроконтролера серії MCS-51 необхідно засвоїти способи адресації. Найбільш наочно це можна розглянути на прикладі команд пересилань. У загальному вигляді формат команди пересилання має такий вигляд:

MOV <байт-призн.>,<байт-джер.>

Способи адресації доцільно розглянути на прикладах. Приклади наведені у вигляді лістингу, де перше поле – адреса, за якою команда знаходиться в пам'яті програм, друге поле – машинний код команди, третє поле – мнемокод команди, четверте поле – операнди. У першому і другому полі значення наведені в шістнадцятковій формі. На початку і наприкінці прикладів додані команди NOP – «порожня операція» для більшої наочності розміщення машинних кодів у пам'яті програм.

Приклад 1

```
0000 00        NOP
0001 743E    MOV   A,#3Eh     ;Приклад 1.1
0003 7464    MOV   A,#01100100b   ;Приклад 1.2
0005 00        NOP
```

У прикладі 1.1 значення 3Eh пересилається в акумулятор А. У результаті виконання команди в акумуляторі буде значення 3Eh. У машинному коді ця команда подана двома байтами: перший – код команди (74h), другий – безпосередньо значення (3Eh), що пересилається в акумулятор. У пам'яті програм команда знаходиться в осередках з адресами 0001h і 0002h.

У прикладі 1.2 така сама команда, як і в прикладі 1.1, тому в машинному коді код команди має те саме значення (74h), а безпосереднє значення, що завантажується в акумулятор, має інше значення (64h). Команда займає також два байта і розташована в осередках пам'яті програм з адресами 0003h і 0004h.

Цей приклад ілюструє: другий операнд указаний у вигляді безпосередньої адресації (тобто задано безпосереднє значення) і є в пам'яті програм другим байтом команди.

Приклад 2

```
0000 00        NOP
0001 E549    MOV   A,49h     ;Приклад 2.1
0003 E52E   MOV   A,2Eh     ;Приклад 2.2
0005 00        NOP
```

У прикладі 2.1 вміст осередка внутрішньої пам'яті даних за адресою 49h пересилається в акумулятор А. В результаті виконання команди в акумуляторі буде значення, яке міститься в осередку внутрішньої пам'яті даних, адреса якої 49h. У машинному коді ця команда подана двома байтами: перший – код команди (E5h), другий – адреса осередка внутрішньої пам’яті даних (49h), з якої дані пересилаються в акумулятор. У пам'яті програм ця команда знаходиться в осередках з адресами 0001h і 0002h.

У прикладі 2.2 вміст осередка внутрішньої пам'яті даних за адресою 2Eh пересилається в акумулятор А. У результаті виконання команди в акумуляторі буде значення, яке міститься в осередку внутрішньої пам'яті даних, адреса якої 2Eh. У машинному коді ця команда подана двома байтами: перший – код команди (E5h), другий – адреса осередка внутрішньої пам’яті даних (2Eh), з якої дані пересилаються в акумулятор. У пам'яті програм ця команда знаходиться в осередках з адресами 0003h і 0004h.

Цей приклад ілюструє: другий операнд вказано у вигляді прямої адресації (тобто вказана пряма адреса осередка внутрішньої пам'яті даних) і є в пам'яті програм другим байтом команди.

Приклад 3

```
0000 00     NOP
0001 E8     MOV   A,R0      ;Приклад 3.1
0002 E9     MOV   A,R1      ;Приклад 3.2
0003 EF     MOV   A,R7      ;Приклад 3.3
0004 00     NOP
```

У прикладі 3.1 вміст регістра R0 пересилається в акумулятор А. У результаті виконання команди в акумуляторі буде значення, що міститься в регістрі R0. У прикладі 3.2 вміст регістра R1 пересилається в акумулятор А. У результаті виконання команди в акумуляторі буде значення, що міститься в регістрі R1. У прикладі 3.3 вміст регістра R7 пересилається в акумулятор А. У результаті виконання команди в акумуляторі буде значення, що міститься в регістрі R7.

У всіх прикладах (приклади 3.1, 3.2, 3.3) у машинному коді ця команда подана одним байтом. У двійковому вигляді машинні коди прикладів 3.1, 3.2, 3.3 матимуть вигляд:

1110 1000 – для прикладу 3.1;
1110 1001 – для прикладу 3.2;
1110 1111 – для прикладу 3.3.

Старші п'ять біт є кодом команди, а молодші три – адресою (номером) регістра.

Цей приклад ілюструє: другий операнд вказаний у вигляді регістрової адресації (тобто адреса джерела інформації вказана неявно) і знаходиться в пам'яті програм безпосередньо в коді команди. Цей спосіб адресації дозволяє отримати більш коротку команду і, як правило, виконання за менший час, що ефективно позначається як на обсязі програмного забезпечення, так і на часі виконання програми.

Приклад 4

```
0000 00      NOP
0001 7833  MOV   R0,#33h   ;Запис в R0 значення 33h
0003 7940  MOV   R1,#40h   ;Запис в R1 значення 40h
0005 7411  MOV   A,#11h   ;Запис в A значення 11h
0007 F6      MOV   @R0,A    ;Приклад 4.1
0008 08      INC   R0     ;Значення в R0 збіл. на 1
0009 F6      MOV   @R0,A    ;Приклад 4.2
000A 08      INC   R0     ;Значення в R0 збіл. на 1
000B F6      MOV   @R0,A    ;Приклад 4.3
000C F7      MOV   @R1,A    ;Приклад 4.4
000D 00      NOP
```

Для наочності спочатку в регістр R0 завантажено значення 33h, у регістр R1 – 40h, в акумулятор – 11h.

У прикладі 4.1 вміст акумулятора A пересилається в осередок внутрішньої пам'яті даних, адреса якої міститься в регістрі R0. У результаті виконання команди вміст акумулятора (11h) буде пересланий в осередок внутрішньої пам'яті даних за адресою 33h (тому що в регістрі R0 знаходиться значення 33h).

У прикладі 4.2 команда виконується таким самим чином, як і в прикладі 4.1, але в регістрі R0 знаходиться значення 34h, тому що після виконання команди INC R0 вміст регістра R0 збільшився на 1 (було 33h, стало 34h). У результаті виконання команди значення з акумулятора (11h) буде переслане в осередок внутрішньої пам'яті даних за адресою 34h.

У прикладі 4.3 команда виконується таким самим чином, як і в прикладі 4.2, але в регістрі R0 знаходиться значення 35h, тому що після виконання команди INC R0 вміст регістра R0 збільшився на 1 (було 34h, стало 35h). У результаті виконання команди значення з акумулятора (11h) буде переслане в осередок внутрішньої пам'яті даних за адресою 35h.

У прикладі 4.4 вміст акумулятора A пересилається в осередок внутрішньої пам'яті даних, адреса якої міститься в регістрі R1. У результаті виконання команди вміст акумулятора (11h) буде пересланий в осередок внутрішньої пам'яті даних за адресою 40h (тому що в регістрі R1 знаходиться значення 40h).

У всіх прикладах (приклади 4.1, 4.2, 4.3, 4.4) у машинному коді ця команда подана одним байтом. У двійковому вигляді машинні коди прикладів 4.1, 4.2, 4.3, 4.4 матимуть вигляд:

1111 0110 – для прикладу 4.1;
1111 0110 – для прикладу 4.2;
1111 0110 – для прикладу 4.3;
1111 0111 – для прикладу 4.4.

Старші сім бітів є кодом команди, а молодший біт – адресою (номером) регістра, що використовується як вказівник на осередок внутрішньої пам'яті даних.

Цей приклад ілюструє: перший операнд вказаний у вигляді непряморегістрової адресації. Даний спосіб називається непряморегістровий: непрямо – тому, що адреса одержувача інформації задана непрямо (адреса осередка внутрішньої пам'яті даних міститься в регістрі), і регістровий – тому, що вказівник задано неявно (номер регістра, що виступає в ролі покажчика, знаходиться в пам'яті програм безпосередньо в коді команди).

Цей спосіб адресації дозволяє отримати більш короткий код програми при роботі з масивами завдяки організації циклу.

Приклад 5

```
0000 00        NOP
0001 9001A0  MOV   DPTR,#01A0h ;Запис в DPTR знач. 1A0h
0004 7405      MOV   A,#05h   ;Запис в A значення 05h
0006 93        MOVC  A,@A+DPTR  ;Приклад 5.1
0007 7464      MOV   A,#64h   ;Запис в A значення 64h
0009 83        MOVC  A,@A+PC   ;Приклад 5.2
000D 00        NOP
```

Для наочності прикладу 5.1 спочатку в регістр DPTR завантажено значення 01A0h, в акумулятор – 05h.

У прикладі 5.1 вміст осередка пам'яті програм, адреса якої визначається як сума значень, що знаходяться в регістрі DPTR і акумуляторі A, пересилається в акумулятор. У результаті виконання команди вміст осередка пам'яті програм з адресою 01A5h (тому що 01A0h + 05h = 01A5h) буде пересланий в акумулятор.

Для наочності прикладу 5.2 спочатку в акумулятор завантажено значення 64h.

У прикладі 5.2 вміст осередка пам'яті програм, адреса якої визначається як сума значень, що знаходяться в лічильнику команд PC і акумуляторі A, пересилається в акумулятор. У результаті виконання команди вміст з осередка пам'яті програм за адресою 006Dh (тому що 0009h + 64h = 006Dh) буде пересланий в акумулятор.

Приклади 5.1 і 5.2 демонструють непряморегістрову адресацію за сумою базового та індексного регістрів. Як базовий може виступати регістр DPTR або лічильник команд PC, а як індексний регістр – акумулятор A.

Непряморегістрова адресація за сумою базового та індексного регістрів дозволяє здійснювати вибірку даних (констант) з таблиць, розташованих у пам'яті програм.

Приклад 6

```
0000 00      NOP
0001 785C  MOV   R0,#5Ch    ;Приклад 6.1
0003 A85C MOV   R0,5Ch    ;Приклад 6.2
0005 A65C MOV   @R0,5Ch    ;Приклад 6.3
0007 765C  MOV   @R0,#5Ch   ;Приклад 6.4
000D 00      NOP
```

Даний приклад демонструє синтаксис, що забезпечує різні способи адресації.

Приклад 6.1 – значення 5Ch пересилається в регістр R0.

Приклад 6.2 – вміст осередка внутрішньої пам'яті даних за адресою 5Ch пересилається в регістр R0.

Приклад 6.3 – вміст осередка внутрішньої пам'яті даних за адресою 5Ch пересилається в осередок пам'яті даних, адреса якої міститься в регістрі R0.

Приклад 6.4 – значення 5Ch пересилається в осередок внутрішньої пам'яті даних, адреса якої міститься в регістрі R0.

3.3. Команди мікроконтролера

Увесь набір команд мікроконтролера серії MCS-51 відповідно до функції (операції), що здійснюється при виконанні команди, можна умовно розподілити та подати структурно (рис. 3.2).

Тут слід звернути увагу, що цей розподіл вельми умовний, тому що деякі команди можна віднести як до однієї, так і до іншої групи. Одна команда не вписується в зазначену структуру – це команда «немає операції», або як її ще називають – «команда порожньої операції».

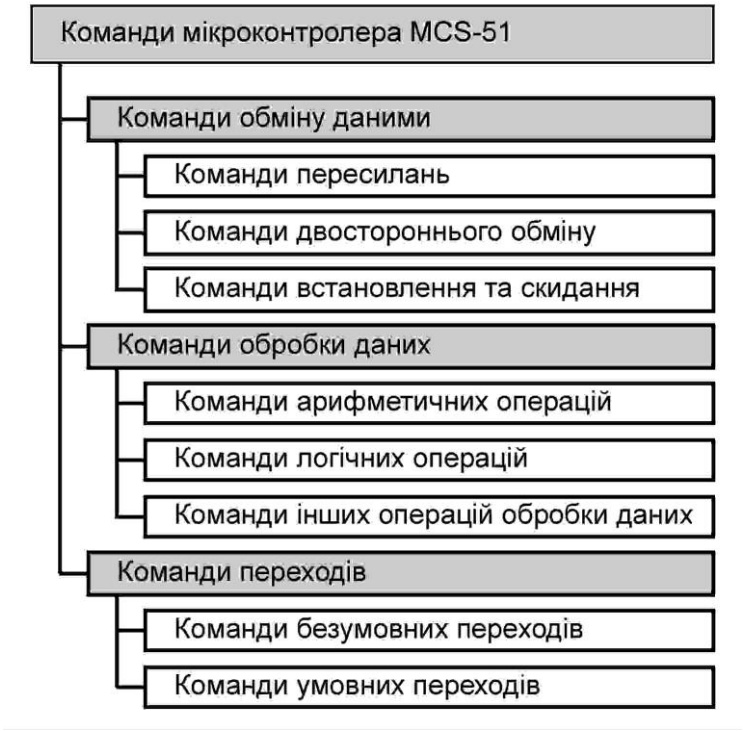


Рисунок 3.2 – Структура команд мікроконтролера серії MCS-51

Команда NOP

Команда «немає операції» – це така, яка не містить ніякої функції (операції). Не впливає на вміст регістрів, прапорців, осередків пам'яті даних.

```
NOP          ;1 байт, 1 мц
```

Виконання команди приводить тільки до зміни вмісту лічильника команд PC, (зчитування команди призводить до збільшення вмісту на 1) та до витрати часу, що становить 1 машинний цикл.

3.3.1. Команди обміну даними

У літературі команди обміну даними часто називають командами передачі даних, командами пересилань, командами читання та запису, командами загрузки і деякими іншими термінами. Залежно від процесора серед команд обміну даними можна виділити за функціональними ознаками ті чи інші групи команд. В системі команд мікроконтролера серії MCS-51 команди, яким властиві ознаки обміну даними, можна сформувати в такі групи:

- команди пересилань – команди, що здійснюють пересилання даних від джерела інформації до одержувача інформації;

- команди двостороннього обміну – команди, що здійснюють одночасно пересилання в двох напрямках від одержувача інформації до джерела інформації та від джерела інформації до одержувача інформації;
- команди встановлення та скидання – команди, що здійснюють завантаження (запис) 0 або 1, тобто вміст скидається в 0 або встановлюється в 1.

3.3.2. Команди пересилань

Команди пересилань призначені для забезпечення передачі даних між внутрішніми ресурсами мікроконтролера, так із зовнішніми складовими елементами контролера керування. Архітектура мікроконтролерів серії MCS-51 обумовлює обмін даними як між внутрішніми складовими частками, так із зовнішніми, причому, так само, як і з елементами (осередками) пам'яті. Тому фактично весь обмін даними будується на базі команд пересилань з пам'яттю. Група команд пересилань складається з таких команд:

- MOV – для роботи з внутрішньою пам'яттю даних;
- MOVX – для роботи із зовнішньою пам'яттю даних;
- MOVC – для роботи з пам'яттю програм;
- PUSH та POP – для роботи зі стековою пам’яттю;

Команди пересилань у загальному форматі мають такий вигляд:

MOV <байт-призн.>,<байт-джер.>
MOV <біт-призн.>,<біт-джер.>
MOV DPTR,#data16.
MOVX <байт-призн.>,<байт-джер.>
MOVC A,@A+<R16>
PUSH <байт-джер.>
POP <байт-призн.>

Команда MOV <байт-призн.>,<байт-джер.>

Команда «переслати змінну-байт» пересилає значення розмірністю один байт з «байта-джерела» в «байт-призначення», тобто перший операнд визначає, куди пересилаються дані, а другий – звідкіля. Вміст «байта-джерела» не змінюється. Ця команда на прапорці та інші регістри не впливає. Операнди джерела та призначення допускають п'ятнадцять комбінацій адресації:

MOV A,Rn ;1 байт, 1 мц

Вміст регістра Rn пересилається в регістр А.

```
MOV  Rn,A          ;1 байт, 1 мц
```

Вміст регістра А пересилається в регістр Rn.

```
MOV  A,direct      ;2 байти, 1 мц
```

Вміст осередка внутрішньої пам'яті даних, адреса якої вказана в другому операнді, пересилається в регістр А.

```
MOV  direct,A      ;2 байти, 1 мц
```

Вміст регістра А пересилається в осередок внутрішньої пам'яті даних, адреса якої вказана в першому операнді.

```
MOV  A,@Ri         ;1 байт, 1 мц
```

Вміст осередка внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, пересилається в регістр А.

```
MOV  @Ri,A         ;1 байт, 1 мц
```

Вміст регістра А пересилається в осередок внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri.

```
MOV  Rn,direct     ;2 байти, 2 мц
```

Вміст осередка внутрішньої пам'яті даних, адреса якої вказана в другому операнді, пересилається в регістр Rn.

```
MOV  direct,Rn     ;2 байти, 2 мц
```

Вміст регістра Rn пересилається в осередок внутрішньої пам'яті даних, адреса якої вказана в першому операнді.

```
MOV  direct,@Ri    ;2 байти, 2 мц
```

Вміст осередка внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, пересилається в осередок внутрішньої пам'яті даних, адреса якої вказана в першому операнді.

```
MOV  @Ri,direct    ;2 байти, 2 мц
```

Вміст осередка внутрішньої пам'яті даних за адресою, що вказана як другий операнд, пересилається в осередок внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri.

```
MOV  direct,direct ;3 байти, 2 мц
```

Вміст осередка внутрішньої пам'яті даних, адреса якої вказана в другому операнді, пересилається в осередок внутрішньої пам'яті даних, адреса якої вказана в першому операнді.

```
MOV  A,#data          ;2 байти, 1 мц
```

Значення (байт даних), яке вказане в другому операнді, завантажується (пересилається) в регістр A.

```
MOV  Rn,#data         ;2 байти, 1 мц
```

Значення (байт даних), яке вказане в другому операнді, завантажується (пересилається) в регістр Rn.

```
MOV  direct,#data     ;3 байти, 2 мц
```

Значення (байт даних), яке вказане в другому операнді, завантажується (пересилається) в осередок внутрішньої пам'яті даних, адреса якої визначається першим операндом.

```
MOV  @Ri,#data        ;2 байти, 1 мц
```

Значення (байт даних), яке вказане в другому операнді, завантажується (пересилається) в осередок внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri.

Команда MOV <біт-призн.>,<біт-джер.>

Команда «переслати змінну-біт» пересилає значення величиною один біт з «біта-джерела» в «біт-призначення», тобто перший операнд визначає, куди пересилаються дані, а другий – звідкіля. Вміст «біта-джерела» не змінюється. Одним із операндів має бути прапорець переносу C, а іншим може бути будь-який біт, що допускає пряму адресацію. Команда допускає дві комбінації адресації «байта-джерела» і «байта-призначення»:

```
MOV  C,bit            ;2 байти, 2 мц
```

Вміст біта, адреса якого вказана в другому операнді, пересилається на прапорець переносу C.

```
MOV  bit,C            ;2 байти, 2 мц
```

Вміст прапорця переносу C пересилається в біт, адреса якого вказана в першому операнді.

Команда MOV DPTR,#data16

Команда «завантажити вказівник даних 16-бітовою константою» завантажує вказівник даних DPTR 16-бітовою константою. Ця команда є єдиною, яка одночасно завантажує дані розмірністю 16 біт.

```
MOV  DPTR,#data16        ;3 байти, 2 мц
```

Значення, що вказане в другому операнді, завантажується (пересилається) у регістр вказівника даних DPTR.

Команда MOVX <байт-призн.>,<байт-джер.>

Команда «переслати в зовнішню пам'ять (зі зовнішньої пам'яті) даних» пересилає дані між акумулятором і осередком зовнішньої пам'яті даних. Архітектура мікроконтролера забезпечує обмін із зовнішньою пам'яттю даних тільки з використанням непряморегістрової адресації. Є два типи команд, які відрізняються тим, що забезпечують 8-бітову або 16-бітову непряморегістрову адресацію до зовнішньої пам'яті даних. Як вказівник осередка зовнішньої пам'яті даних у першому випадку використовується регістр R0 або R1, а в другому – регістр вказівника даних DPTR. Також слід зазначити, що звернення до зовнішньої пам'яті даних у першому випадку посторінково з розмірністю 256 байт, у другому – безперервне в усьому адресному просторі 64 кбайт. Операнди джерела та призначення допускають чотири комбінації адресації:

```
MOVX A,@DPTR          ;1 байт, 2 мц
```

Вміст осередка зовнішньої пам'яті даних, адреса якої визначається вмістом регістра DPTR, пересилається в регістр A.

```
MOVX A,@Ri            ;1 байт, 2 мц
```

Вміст осередка зовнішньої пам'яті даних, адреса якої визначається вмістом регістра Ri, пересилається в регістр A.

```
MOVX @DPTR,A          ;1 байт, 2 мц
```

Вміст регістра A пересилається в осередок зовнішньої пам'яті даних, адреса якої визначається вмістом регістра DPTR.

```
MOVX @Ri,A            ;1 байт, 2 мц
```

Вміст регістра A пересилається в осередок зовнішньої пам'яті даних, адреса якої визначається вмістом регістра Ri.

Команда MOVC A,@A+<R16>

Команда «переслати байт з пам'яті програм» пересилає в акумулятор байт коду або константи з осередка пам'яті програм. Адреса осередка пам'яті програм визначається (обчислюється) як сума 8-бітового вмісту акумулятора без знака і вмісту 16-бітового регістра, який вказано в другому операнді команди. Як 16-бітовий регістр може бути вказівник даних – регістр DPTR, або лічильник команд – регістр PC.

У разі, коли використовується лічильник команд PC, то його вміст збільшується до адреси наступної команди перед тим, як його вміст буде доданий з вмісту акумулятора. Додавання виконується так, що перенесення з молодших восьми біт може поширюватися на старші біти.

MOVC A,@DPTR+A ;1 байт, 2 мц

Вміст осередка пам'яті програм, адреса якої визначається сумою вмісту регістрів A та DPTR, пересилається в регістр A.

MOVC A,@PC+A ;1 байт, 2 мц

Вміст осередка пам'яті програм, адреса якої визначається сумою вмісту регістрів A та PC, пересилається в регістр A.

Команда PUSH <байт-джер.>

Команда «запис у стек» здійснює запис вмісту «байта-джерела» в стекову пам'ять. Команда виконується в два етапи: перший – вміст вказівника стека SP збільшується на 1, другий – вміст «байта-джерела» пересилається в осередок стека (внутрішньої пам'яті даних), адреса якої міститься в регістрі вказівника стека SP. Вміст «байта-джерела» залишається незмінним. Застосовується ця команда для тимчасового зберігання вмісту регістрів спеціальних функцій при вході в підпрограму, а також для передачі параметрів підпрограмам через стек.

PUSH direct ;2 байти, 2 мц

Вміст регістра вказівника стека SP збільшується на 1, після чого вміст осередка пам'яті даних, адреса якої вказана як операнд у вигляді прямої адреси, пересилається в осередок стека (внутрішньої пам'яті даних), адреса якої міститься в регістрі вказівника стека SP.

Примітка. Слід згадати, що пряма адресація до осередка внутрішньої пам'яті даних за адресою від 80h до FFh здійснює звернення до регістрів спеціальних функцій.

Команда POP <байт-призн.>

Команда «читання зі стека» здійснює читання зі стекової пам'яті вмісту осередка і пересилання в «байт-призначення». Команда виконується в два етапи: перший – вміст осередка стека (внутрішньої пам'яті даних), адреса якої міститься в регістрі вказівника стека SP, пересилається в «байт-призначення», другий – вміст вказівника стека SP зменшується на 1. Вміст осередка стекової пам'яті залишається незмінним. Застосовується ця команда для відновлення вмісту регістрів спеціальних функцій (що були тимчасово збережені за допомогою команди PUSH) при виході з підпрограми, а також у деяких випадках для передачі параметрів з підпрограм через стек.

POP direct ;2 байти, 2 мц

Вміст осередка стека (внутрішньої пам'яті даних), адреса якої міститься в регістрі вказівника стека SP, пересилається в осередок пам'яті даних, адреса якої вказана як операнд у вигляді прямої адреси. Після чого вміст вказівника стека SP зменшується на одиницю.

Примітка. Слід згадати, що пряма адресація до осередків внутрішньої пам'яті даних за адресою від 80h до FFh здійснює звернення до регістрів спеціальних функцій.

3.3.3. Команди двостороннього обміну

Команди двостороннього обміну в мікроконтролерах серії MCS-51 дозволяють здійснювати одночасний обмін вмісту регістра А з вмістом внутрішнього ресурсу контролера (регістр загального призначення, регістр спеціальних функцій, осередок внутрішньої пам’яті даних). Розмірність даних обміну (залежно від команди) становить один байт або одна тетрада. Група команд двостороннього обміну подана командами:

- XCH – для обміну даних з розмірністю один байт;
- XCHD – для обміну даних з розмірністю одна тетрада.

Команди двостороннього обміну в загальному форматі мають такий вигляд:

XCH A,<байт-джер.>

XCHD A,<байт-джер.>

Команда XCH A,<байт-джер.>

Ця команда здійснює обмін вмісту акумулятора з вмістом «байта-джерела». Операнди джерела допускають три комбінації адресації:

XCH A,Rn ;1 байт, 1 мц

Вміст регістра А обмінюється з вмістом регістра Rn, що вказаний у другому операнді, тобто після виконання команди регістр А має вміст регістра Rn до виконання команди, а регістр Rn має вміст регістра А до виконання команди

XCH A,direct ;2 байти, 1 мц

Вміст регістра А обмінюється із вмістом осередка пам'яті даних, пряма адреса якої вказана в другому операнді, тобто після виконання команди регістр А має вміст осередка пам'яті даних до виконання команди, а осередок пам’яті даних має вміст регістра А до виконання команди

XCH A,@Ri ;1 байт, 1 мц

Вміст регістра А обмінюється з вмістом осередка пам'яті даних, адреса якої міститься в регістрі Ri, тобто після виконання команди регістр А має вміст осередка пам'яті даних до виконання команди, а осередок пам’яті даних має вміст регістра А до виконання команди.

Команда XCHD A,<байт-джер.>

Команда «обмін вмісту тетрадою» виконує обмін вмісту молодшої тетради (біти 3 – 0) акумулятора з вмістом молодшої тетради (біти 3 – 0) осередка внутрішньої пам’яті даних, адреса якої міститься в регістрі Ri. На старші біти (біти 7 – 4) ця команда не впливає

XCHD A,@Ri ;1 байт, 1 мц

Вміст молодшої тетради (біти 3 – 0) акумулятора обмінюється з вмістом молодшої тетради (біти 3 – 0) осередка внутрішньої пам’яті даних, адреса якої міститься в регістрі Ri.

3.3.4. Команди встановлення та скидання

Команди встановлення та скидання – це такі, які модифікують вміст таким чином, що він приймає значення 1 або 0 відповідно. Команди скидання інколи називають командами очищення. До команд встановлення та скидання інколи застосовується загальний термін «команди встановлення»: з деталізацією «команди встановлення в 1» – для команд встановлення та «команди встановлення в 0» – для команд скидання.

Чому ж команди встановлення та скидання можливо віднести до команд обміну даними? Формально команду скидання можливо трактувати як «переслати значення 0», а команду встановлення – як «переслати значення 1».

Команди встановлення та скидання в загальному форматі мають такий вигляд:

SETB <біт-призн.>

CLR <біт-призн.>

CLR <байт-призн.>

Команда SETB C

Команда «встановити біт прапорця перенесу» встановлює біт прапорця переносу C в одиницю.

```
SETB C          ;1 байт, 1 мц
```

Вміст біта прапорця переносу C приймає значення одиниці.

Команда SETB <біт-призн.>

Команда «встановити біт» встановлює біт-призначення в одиницю.

```
SETB bit        ;2 байти, 1 мц
```

Вміст біта, пряма адреса якого визначається операндом, приймає значення одиниці.

Команда CLR C

Команда «скинути біт прапорця переносу» скидає біт прапорця переносу C в нуль.

```
CLR  C          ;1 байт, 1 мц
```

Вміст біта прапорця переносу C приймає значення нуль.

Команда CLR <біт-призн.>

Команда «скинути біт» скидає «біт-призначення» в нуль.

```
CLR  bit        ;2 байти, 1 мц
```

Вміст біта, пряма адреса якого визначається операндом, приймає значення нуль.

Команда CLR A

Команда «очистити акумулятор» обнуляє вміст акумулятора.

```
CLR  A          ;1 байт, 1 мц
```

Вміст акумулятора приймає значення нуль.

3.3.5. Команди обробки даних

Команди обробки даних – набір команд, що дозволяє здійснювати арифметичні, логічні та інші операції над даними.

3.3.5.1. Команди арифметичних операцій

Набір команд мікроконтролера дозволяє реалізувати над цілочисельними значеннями такі арифметичні операції, як додавання, віднімання, множення та ділення.

Команди арифметичних операцій в загальному форматі мають такий вигляд:

ADD A,<байт-джер.>
ADDC A,<байт-джер.>
SUBB A,<байт-джер.>
MUL AB
DIV AB

Команда ADD A,<байт-джер.>

Команда «складання» складає вміст акумулятора A з вмістом «байта-джерела», результат складання розміщується в акумуляторі. При появі перенесень з розрядів 7-го і 3-го встановлюються відповідно прапорців переносу C і додаткового переносу AC, в іншому випадку ці прапорці скидаються. При додаванні цілих чисел без знака прапорця переносу C вказує на появу переповнення. Прапорець переповнення OV встановлюється, якщо є перенесення з біта 6 і немає перенесення з біта 7, або є перенесення з біта 7 і немає – з біта 6, в іншому випадку прапорець OV скидається. При додаванні цілих чисел зі знаком прапорець OV вказує на від'ємну величину, отриману при підсумовуванні двох додатних операндів, або на додатну суму для двох від'ємних операндів. Операнди джерела допускають чотири комбінації адресації:

```
ADD  A,Rn          ;1 байт, 1 мц
```

Вміст регістра Rn додається до вмісту регістра A, сума пересилається в регістр A. Прапорці C, AC, OV приймають значення відповідно з результатом складання.

```
ADD  A,direct      ;2 байти, 1 мц
```

Вміст осередка внутрішньої пам'яті даних, адреса якої вказана в другому операнді, додається до вмісту регістра A, сума пересилається в регістр A. Прапорці C, AC, OV приймають значення відповідно до результату складання.

```
ADD  A,@Ri          ;1 байт, 1 мц
```

Вміст осередка внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, додається до вмісту регістра А, сума пересилається в регістр А. Прапорці С, АС, OV приймають значення відповідно до результату складання.

```
ADD  A,#data        ;2 байти, 1 мц
```

Значення, що вказане в другому операнді, додається до вмісту регістра А, сума пересилається в регістр А. Прапорці С, АС, OV приймають значення відповідно до результату складання.

Команда ADDC A,<байт-джер.>

Команда «складання з переносом» складає одночасно вміст акумулятора А, значення прапорця переносу С та вміст «байта-джерела», що визначається другим операндом, результат складання розміщується в акумуляторі. При виникненні перенесень з розрядів 7-го і 3-го встановлюються відповідно прапорців переносу С і додаткового переносу АС, в іншому випадку ці прапорці скидаються. При додаванні цілих чисел без знака прапорець переносу С вказує на появу переповнення. Прапорець переповнення OV встановлюється, якщо є перенесення з біта 6 і немає перенесення з біта 7, або є перенесення з біта 7 і немає – з біта 6, в іншому випадку прапорець OV скидається. При додаванні цілих чисел зі знаком прапорець OV вказує на від’ємну величину, отриману при підсумовуванні двох додатних операндів, або на додатну суму для двох від’ємних операндів.

По суті команда ADDC відрізняється від ADD тільки тим, що ще додається значення прапорця переносу С. Операнди джерела допускають чотири комбінації адресації:

```
ADDC A,Rn           ;1 байт, 1 мц
```

Одночасно складається вміст регістра Rn, регістра А та прапорця переносу С, сума пересилається в регістр А. Прапорці С, АС, OV приймають значення відповідно до результату складання.

```
ADDC A,direct       ;2 байти, 1 мц
```

Одночасно складається вміст осередка внутрішньої пам'яті даних, адреса якої вказана в другому операнді, регістра А та прапорця переносу С, сума пересилається в регістр А. Прапорці С, АС, OV приймають значення відповідно до результату складання.

ADDC A,@Ri ;1 байт, 1 мц

Одночасно складається вміст осередка внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, регістра А та прапорця переносу С, сума пересилається в регістр А. Прапорці С, АС, OV приймають значення відповідно до результату складання.

ADDC A,#data ;2 байти, 1 мц

Одночасно складається значення, що вказане в другому операнді, вміст регістра А та прапорця переносу С, сума пересилається в регістр А. Прапорці С, АС, OV приймають значення відповідно до результату складання.

Команда SUBB A,<байт-джер.>

Команда «віднімання з позичкою» віднімає вміст «байта-джерела» разом з прапорцем переносу С від вмісту акумулятора А, результат розташовується в регістрі А. Залежно від результату віднімання модифікуються прапорець переносу С, прапорець додаткового переносу АС та прапорець переповнення OV.

Прапорець переносу С встановлюється, якщо при відніманні для біта 7 необхідна позика, в іншому випадку – скидається. Прапорець додаткового переносу АС встановлюється, якщо позика необхідна для біта 3 і скидається в іншому випадку. Прапорець переповнення OV встановлюється, якщо позика необхідна для біта 6, але її немає для біта 7, або вона є для біта 7, але її немає для біта 6, в інших випадках – скидається.

При відніманні цілих чисел зі знаком прапорець OV вказує на невід’ємне число, яке утворюється при відніманні невід’ємної величини від додатної, або додатне значення, яке утворюється при відніманні додатного числа від невід’ємного. Операнд джерела допускає чотири режими адресації:

SUBB A,Rn ;1 байт, 1 мц

Вміст регістра Rn разом з прапорцем переносу С віднімається від вмісту регістра А, різниця пересилається в регістр А. Прапорці С, АС, OV приймають значення відповідно до результату віднімання.

SUBB A,direct ;2 байти, 1 мц

Вміст осередка внутрішньої пам'яті даних, адреса якої вказана в другому операнді, разом з прапорцем переносу С віднімається від вмісту регістра А, різниця пересилається в регістр А. Прапорці С, АС, OV приймають значення відповідно до результату віднімання.

SUBB A,@Ri ;1 байт, 1 мц

Вміст осередка внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, разом з прапорцем переносу C віднімається від вмісту регістра A, різниця пересилається в регістр A. Прапорці C, AC, OV приймають значення відповідно до результату віднімання.

SUBB A,#data ;2 байти, 1 мц

Значення, що вказане в другому операнді, разом з прапорцем переносу C віднімається від вмісту регістра A, різниця пересилається в регістр A. Прапорці C, AC, OV приймають значення відповідно до результату віднімання.

Команда MUL AB

Команда «множення» здійснює множення двох 8-бітових цілих чисел без знака, які знаходяться в регістрах A і B. Добуток розташовується наступним чином: старший байт в регістрі B, а молодший байт в регістрі A. Якщо добуток більше, ніж 255, то встановлюється прапорець переповнення OV, в іншому випадку він скидається. Прапорець переносу C завжди скидається. Ця команда має лише один вид, і слід звернути увагу на її синтаксис – написання регістрів «AB» здійснюється без коми та пробілу (як один цілий операнд).

MUL AB ;1 байт, 4 мц

Здійснює множення вмісту регістрів A та B. Добуток розподіляється в регістрах A (молодший байт) та B (старший байт). Прапорець OV приймає значення відповідно до результату множення, а прапорець C скидається.

Команда DIV AB

Команда «ділення» здійснює ділення 8-бітового цілого числа без знака, яке знаходиться в регістрі A, на 8-бітове ціле число без знака, яке знаходиться в регістрі B. Частка розташовується наступним чином: в регістрі A – ціла частина частки, а в регістрі B – залишок від ділення. Прапорці переносу C і переповнення OV скидаються. Якщо вміст регістра A менше вмісту регістра B, то прапорець додаткового переносу AC не скидається. Ця команда має лише один вид і слід звернути увагу на її синтаксис – написання регістрів «AB» здійснюється без коми та пробілу (як один цілий операнд).

DIV AB ;1 байт, 4 мц

Здійснює ділення вмісту регістра A на вміст регістра B. Результат ділення: в регістрі A – ціла частина частки, а в регістрі B – залишок від ділення. Прапорець

АС приймає значення відповідно до результату ділення, а прапорці С та OV скидаються.

Примітка. Ділення на нуль неможливе, оскільки регістр В містить нуль, то після команди DIV вміст регістрів А і В буде невизначеним, а це призведе до скидання прапорця переносу С та встановлення прапорця переповнення OV.

Команда DEC <байт-призн.>

Команда «декремент» зменшує на одиницю вміст «байта-призначення». У разі, коли вміст має початкове значення, що дорівнює 00h, виконання декременту дасть значення FFh. Операнди «байта-призначення» допускають чотири комбінації адресації:

DEC A ;1 байт, 1 мц

Зменшується на одиницю вміст регістра А.

DEC Rn ;1 байт, 1 мц

Зменшується на одиницю вміст регістра загального призначення Rn, який розглядається як операнд.

DEC direct ;2 байти, 1 мц

Зменшується на одиницю вміст осередка внутрішньої пам’яті даних, пряма адреса якої вказана як операнд.

DEC @Ri ;1 байт, 1 мц

Зменшується на одиницю вміст осередка внутрішньої пам’яті даних, адреса якої міститься в регістрі Ri, що вказаний як операнд.

Команда INC <байт-призн.>

Команда «інкремент» збільшує на одиницю вміст «байта-призначення». У разі, коли вміст має початкове значення, що дорівнює FFh, виконання інкременту дасть значення 00h. Операнди «байта-призначення» допускають чотири комбінації адресації:

INC A ;1 байт, 1 мц

Збільшується на одиницю вміст регістра А.

INC Rn ;1 байт, 1 мц

Збільшується на одиницю вміст регістра загального призначення Rn, який розглянутий як операнд.

```
INC  direct          ;2 байти, 1 мц
```

Збільшується на одиницю вміст осередка внутрішньої пам’яті даних, пряма адреса якої розглянута як операнд.

```
INC  @Ri             ; 1 байт, 1 мц
```

Збільшується на одиницю вміст осередка внутрішньої пам’яті даних, адреса якої міститься в регістрі Ri, що вказаний як операнд.

Команда INC DPTR

Команда «інкремент вказівника DPTR» збільшує на одиницю вміст 16-розрядного вказівника DPTR. У разі, коли вміст має початкове значення, що дорівнює FFFFh, виконання інкременту дасть значення 0000h. Команда має одну комбінацію адресації:

```
INC  DPTR            ;1 байт, 2 мц
```

Збільшується на одиницю вміст регістра DPTR.

3.3.5.2. Команди логічних операцій

Набір команд мікроконтролера дозволяє реалізувати виконання логічних функцій «НІ», «І», «АБО», «ВИКЛЮЧНЕ АБО».

Команди логічних операцій в загальному форматі мають такий вигляд:

```
ANL  <байт-призн.>,<байт-джер.>
ANL  C,<біт-джер.>
ANL  C,/<біт-джер.>
ORL  <байт-призн.>,<байт-джер.>
ORL  C,<біт-джер.>
ORL  C,/<біт-джер.>
XRL  <байт-призн.>,<байт-джер.>
CPL  A
CPL  C
CPL  <біт-джер.>
```

Команда ANL <байт-призн.>,<байт-джер.>

Команда «логічне І для змінних-байтів» виконує операцію порозрядного логічного «І» над вмістом «байта-призначення» та вмістом «байта-джерела» і поміщає результат у «байт-призначення». Ця операція не впливає на стан

прапорців. Операнди джерела та призначення допускають шість комбінацій адресації:

```
ANL  A,Rn          ;1 байт, 1 мц
```

Виконується порозрядно функція «логічне І» над вмістом регістра А та вмістом регістра Rn, результат розташовується в регістрі А.

```
ANL  A,direct      ;2 байти, 1 мц
```

Виконується порозрядно функція «логічне І» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої визначається другим операндом, результат розташовується в регістрі А.

```
ANL  A,@Ri         ;1 байт, 1 мц
```

Виконується порозрядно функція «логічне І» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri. Результат розташовується в регістрі А.

```
ANL  A,#data       ;2 байти, 1 мц
```

Виконується порозрядно функція «логічне І» над вмістом регістра А та значенням, що вказане в другому операнді. Результат розташовується в регістрі А.

```
ANL  direct,A      ;2 байти, 1 мц
```

Виконується порозрядно функція «логічне І» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої визначається першим операндом. Результат пересилається в осередок внутрішньої пам'яті даних, адреса якої визначається першим операндом.

```
ANL  direct,#data      ;3 байти, 2 мц
```

Виконується порозрядно функція «логічне І» над вмістом осередка внутрішньої пам'яті даних, адреса якої визначається першим операндом, та значенням, що вказане в другому операнді. Результат пересилається в осередок внутрішньої пам'яті даних, адреса якої визначається першим операндом.

Примітка. Якщо команда ANL застосовується для зміни вмісту порту, то значення, яке використовується як дані порту, буде зчитуватися із «засувки» порту, а не з виводів порту мікроконтролера.

Команда ANL C,<біт-джер.>

Команда «логічне І для змінних-бітів» виконує операцію логічного «І» над вмістом прапорця переносу С та вмістом «біта-джерела» і поміщає результат у прапорець переносу С. Існує один вид цієї команди:

```
ANL  C,bit          ;2 байти, 2 мц
```

Виконується функція «логічне І» над вмістом прапорця переносу С та вмістом біта, адреса якого визначається другим операндом. Результат пересилається у прапорець переносу С.

Команда ANL C, <біт-джер.>

Команда «логічне І для змінних-бітів» виконує операцію логічного «І» над вмістом прапорця переносу С та інверсним значенням вмісту «біта-джерела» і поміщає результат у прапорець переносу С. Існує один вид цієї команди:

```
ANL  C,bit          ;2 байти, 2 мц
```

Виконується функція «логічне І» над вмістом прапорця переносу С та інверсним значенням вмісту біта, адреса якого визначається другим операндом. Результат пересилається у прапорець переносу С.

Команда ORL <байт-призн.>,<байт-джер.>

Команда «логічне АБО для змінних-байтів» виконує операцію порозрядного логічного «АБО» над «байтом-призначення» та «байтом-джерела» і поміщає результат у «байт-призначення». Ця операція не впливає на стан прапорців. Операнди джерела та призначення допускають шість комбінацій адресації:

```
ORL  A,Rn           ;1 байт, 1 мц
```

Виконується порозрядно функція «логічне АБО» над вмістом регістра А та вмістом регістра Rn, результат розташовується в регістрі А.

```
ORL  A,direct       ;2 байти, 1 мц
```

Виконується порозрядно функція «логічне АБО» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої визначається другим операндом, результат розташовується в регістрі А.

```
ORL  A,@Ri          ;1 байт, 1 мц
```

Виконується порозрядно функція «логічне АБО» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri. Результат розташовується в регістрі А.

```
ORL  A,#data          ;2 байти, 1 мц
```

Виконується порозрядно функція «логічне АБО» над вмістом регістра А та значенням, що вказане в другому операнді. Результат розташовується в регістрі А.

```
ORL  direct,A         ;2 байти, 1 мц
```

Виконується порозрядно функція «логічне АБО» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої визначається першим операндом. Результат пересилається в осередок внутрішньої пам'яті даних, адреса якої визначається першим операндом.

```
ORL  direct,#data       ;3 байти, 2 мц
```

Виконується порозрядно функція «логічне АБО» над вмістом осередка внутрішньої пам'яті даних, адреса якої визначається першим операндом, та значенням, що вказане в другому операнді. Результат пересилається в осередок внутрішньої пам'яті даних, адреса якої визначається першим операндом.

Примітка. Якщо команда ORL застосовується для зміни вмісту порту, то значення, яке використовується як дані порту, буде зчитуватися із «засувки» порту, а не з виводів порту мікроконтролера.

Команда ORL C, <біт-джер.>

Команда «логічне АБО для змінних-бітів» виконує операцію логічного «АБО» над вмістом прапорця переносу С та вмістом «біта-джерела», та поміщає результат у прапорець переносу С. Існує один вид цієї команди:

```
ORL  C,bit          ;2 байти, 2 мц
```

Виконується функція «логічне АБО» над вмістом прапорця переносу С та вмістом біта, адреса якого визначається другим операндом. Результат пересилається на прапорець переносу С.

Команда ORL C,<біт-джер.>

Команда «логічне АБО для змінних-бітів» виконує операцію логічного «АБО» над вмістом прапорця переносу С та інверсним значенням вмісту «біта-джерела», та поміщає результат у прапорець переносу С. Існує один вид цієї команди:

```
ORL  /C,bit         ;2 байти, 2 мц
```

Виконується функція «логічне АБО» над вмістом прапорця переносу С та інверсним значенням вмісту біта, адреса якого визначається другим операндом. Результат пересилається в прапорець переносу С.

Команда XRL <байт-призн.>,<байт-джер.>

Команда «виключне АБО» для змінних-байтів виконує операцію порозрядного «виключного АБО» над «байтом-призначення» та «байтом-джерела», і поміщає результат у «байт-призначення». Ця операція не впливає на стан прапорців. Операнди джерела та призначення допускають шість комбінацій адресації:

```
XRL  A,Rn          ;1 байт, 1 мц
```

Виконується порозрядно функція «виключне АБО» над вмістом регістра А та вмістом регістра Rn, результат розташовується в регістрі А.

```
XRL  A,direct      ;2 байти, 1 мц
```

Виконується порозрядно функція «виключне АБО» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої визначається другим операндом, результат розташовується в регістрі А.

```
XRL  A,Ri          ;1 байт, 1 мц
```

Виконується порозрядно функція «виключне АБО» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri. Результат розташовується в регістрі А.

```
XRL  A,#data       ;2 байти, 1 мц
```

Виконується порозрядно функція «виключне АБО» над вмістом регістра А та значенням, що вказане в другому операнді. Результат розташовується в регістрі А.

```
XRL  direct,A      ;2 байти, 1 мц
```

Виконується порозрядно функція «виключне АБО» над вмістом регістра А та вмістом осередка внутрішньої пам'яті даних, адреса якої визначається першим операндом. Результат пересилається в осередок внутрішньої пам'яті даних, адреса якої визначається першим операндом.

```
XRL  direct,#data      ;3 байти, 2 мц
```

Виконується порозрядно функція «виключне АБО» над вмістом осередка внутрішньої пам'яті даних, адреса якої визначається першим операндом, та значенням, що вказане в другому операнді. Результат пересилається в осередок внутрішньої пам'яті даних, адреса якої визначається першим операндом.

Примітка. Якщо команда XRL застосовується для зміни вмісту порту, то значення, яке використовується як дані порту, буде зчитуватися із «засувки» порту, а не з виводів порту мікроконтролера.

Команда CPL A

Команда «інверсія акумулятора» інвертує (змінює значення на протилежне) кожен біт акумулятора. Фактично виконання цієї команди реалізує порозрядно функцію «логічне НІ» над вмістом регістра А. Регістр А одночасно є байтом-джерелом та байтом-призначення.

```
CPL  A          ;1 байт, 1 мц
```

Виконується порозрядно функція «логічне НІ» над вмістом регістра А, результат розташовується в регістрі А.

Команда CPL C

Команда «інверсія біта С» інвертує (змінює значення на протилежне) вміст прапорця переносу С. Фактично над вмістом прапорця переносу С здійснюється функція «логічне НІ». Прапорець переносу С одночасно є «бітом-джерелом» і «бітом-призначення».

```
CPL  C          ;1 байт, 1 мц
```

Виконується функція «логічне НІ» над вмістом прапорця переносу С, результат розташовується у прапорці переносу С.

Команда CPL <біт-джер.>

Команда «інверсія біта» інвертує (змінює значення на протилежне) вміст «біта-джерела». Фактично над вмістом «біта-джерела» здійснюється функція «логічне НІ». Цей біт одночасно є «бітом-джерелом» і «бітом-призначення».

```
CPL  bit        ;2 байти, 1 мц
```

Виконується функція «логічне НІ» над вмістом біта, адреса якого визначається операндом. Результат розташовується в цьому ж біті.

Примітка. Якщо ця команда застосовується для зміни інформації на виході порту, то значення зчитується із «засувки» порту, а не з виводів мікроконтролера.

3.3.5.3. Команди інших операцій обробки даних

В окрему групу команд обробки інформації можна виділити такі команди, що здійснюють операції зсуву та операції обміну. В мікроконтролерах серії MCS-51 виконання цих операцій можливо тільки над вмістом регістра А. Склад вищеозначених команд:

```
RL  A
RLC  A
```

RR A
RRC A
SWAP A

Команда RL A

Команда «циклічний зсув вмісту акумулятора вліво», здійснює зсув вмісту восьми біт акумулятора на один розряд вліво, при цьому вміст 7-го біта (розряду) акумулятора зсувається в нульовий біт (розряд) акумулятора.

```
RL  A          ;1 байт, 1 мц
```

Здійснюється зсув вмісту акумулятора на один розряд вліво, при цьому вміст 7-го розряду акумулятора зсувається в 0-й розряд акумулятора.

Команда RLC A

Команда «циклічний зсув вмісту акумулятора вліво через прапорець переносу», здійснює зсув вмісту восьми біт акумулятора на один розряд вліво, при цьому вміст 7-го біта (розряду) акумулятора зсувається у прапорець переносу С, а вміст прапорця переносу С зсувається в нульовий біт (розряд) акумулятора.

```
RLC  A         ;1 байт, 1 мц
```

Здійснюється зсув вмісту акумулятора на один розряд вліво, при цьому вміст 7-го розряду акумулятора зсувається у прапорець переносу С, а вміст прапорця переносу С в 0-й розряд акумулятора.

Команда RR A

Команда «циклічний зсув вмісту акумулятора вправо», здійснює зсув вмісту восьми біт акумулятора на один розряд вправо, при цьому вміст нульового біта (розряду) акумулятору зсувається в 7-й біт (розряд) акумулятора.

```
RR  A          ;1 байт, 1 мц
```

Здійснюється зсув вмісту акумулятора на один розряд вправо, при цьому вміст 0-го розряду акумулятора зсувається в 7-й розряд акумулятора.

Команда RRC A

Команда «циклічний зсув вмісту акумулятора вправо через прапорець перенесення», здійснює зсув вмісту восьми біт акумулятора на один розряд вправо, при цьому вміст нульового біта (розряду) акумулятора зсувається у прапорець переносу С, а вміст прапорця переносу С зсувається в 7-й біт (розряд) акумулятора.

```
RRC  A          ;1 байт, 1 мц
```

Здійснюється зсув вмісту акумулятора на один розряд вправо, при цьому вміст 0-го розряду акумулятора зсувається у прапорець переносу C, а вміст прапорця переносу C в 7-й розряд акумулятора.

Команда SWAP A

Команда «обмін тетрадами всередині акумулятора» здійснює обмін вмісту між старшою та молодшою тетрадами акумулятора. Ця команда може розглядатися так само, як і команда чотирьох циклічних зсувів (що можна досягти послідовним виконанням чотирьох команд RL A або чотирьох команд RR A).

```
SWAP A          ;1 байт, 1 мц
```

Здійснюється взаємний обмін вмісту старшої та молодшої тетради акумулятора, тобто здійснюється обмін вмістом відповідно розрядів 0-го та 4-го, 1-го та 5-го, 2-го та 6-го, 3-го та 7-го.

3.3.6. Команди переходів

Команди переходів – це такі, які дозволяють здійснювати переходи в програмі. Розробка програмного забезпечення неможлива без застосування команд умовного і безумовного переходів.

Команди безумовного переходу дозволяють здійснити перехід у деяку точку програми без перевірки будь-якої умови.

Команди умовного переходу дозволяють здійснити розгалуження в програмі. Якщо вказана в команді умова виконується, то здійснюється перехід у вказану точку програми, у разі, якщо умова не виконується, то буде задіяна наступна команда. Якщо точку переходу в команді умовного переходу помістити вище за алгоритмом, то за допомогою даної команди можна створити відповідний цикл у програмі.

Отже, робота команд умовного переходу і команд безумовного переходу відрізняється тільки тим, що в командах умовного переходу здійснюється перевірка умови переходу і безпосередньо перехід, якщо умова виконується.

Сам процес переходу здійснюється шляхом модифікації вмісту лічильника команд мікроконтролера. Нагадаємо, що лічильник команд являє собою регістр, що має адресу осередка пам'яті програм, в якій знаходиться поточна для виконання команда.

Як бачимо, у результаті виконання переходу значення лічильника команд змінюється таким чином, що воно відповідає адресі осередку пам'яті програм, де знаходиться команда, на яку виконується перехід. У початковому тексті програми для позначення точки переходу використовують мітку – це довільне символьне ім'я, яке в процесі компіляції програми перетворюється в значення, яке модифікує вміст лічильника команд.

3.3.6.1. Команди безумовних переходів

У системі команд мікроконтролерів серії MCS-51 є декілька видів безумовного переходу, які можна поділити на:

- команди переходу за вказаною міткою;
- команди переходу, що обчислюється;
- команди переходу на підпрограму;
- команди переходу (повернення) з підпрограми.

Команди переходу за вказаною міткою представлені так:

AJMP <мітка>
LJMP <мітка>
SJMP <мітка>

Команди переходу, що обчислюється, мають вигляд:

JMP @A+DPTR

Команди переходу на підпрограму відповідно:

ACALL <мітка>
LCALL <мітка>

Команди переходу (повернення) з підпрограми це:

RET
RETI

Команда LJMP <мітка>

Команда «довгий перехід» дозволяє здійснити перехід у будь-яку точку програми в межах всього адресного простору пам'яті програм мікроконтролера (64 кбайт). Точка переходу визначається міткою, що вказана в команді як операнд. Причому, в машинному коді операнд буде поданий у вигляді абсолютного значення адреси переходу, тобто, по суті, мітка являє собою значення адреси осередка пам'яті програм, де знаходиться команда, на яку здійснюється перехід. При виконанні цієї команди абсолютне значення адреси переходу (мітки) пересилається в лічильник команд мікроконтролера PC.

LJMP label ;3 байти, 2 мц

Значення адреси, що визначається міткою, яка вказана як операнд, завантажується (пересилається) в лічильник команд мікроконтролера PC.

Команда SJMP <мітка>

Команда «короткий перехід» дозволяє здійснити перехід у будь-яку точку програми, що розташована на відстані до 127 байт вище або нижче від розташування команди короткого переходу. Точка переходу визначається міткою, що вказана в команді як операнд. Причому, в машинному коді операнд буде мати значення зміщення (зі знаком +/– в «додатковому коді») відносно розташування поточної команди (відносно поточного значення лічильника команд). При виконанні цієї команди значення зміщення додається до поточного значення лічильника команд (після завершення зчитування команди) і пересилається до лічильника команд.

SJMP label ;2 байти, 2 мц

Значення зміщення, що визначається міткою, яка вказана як операнд, додається до поточного значення лічильника команд (після завершення зчитування команди) і пересилається до лічильника команд.

Примітка. Використання команди «короткий перехід» (у випадках, коли команда і мітка розташовані на відстані до 127 байт) дозволяє зменшити обсяг програмного забезпечення.

Команда AJMP <мітка>

Команда «абсолютний перехід» дозволяє здійснити перехід у будь-яку точку програми в межах однієї сторінки об'ємом 2 кбайта пам'яті програм. Точка переходу визначається міткою, що вказана в команді як операнд. Причому, в машинному коді операнд буде мати значення молодших одинадцяти розрядів абсолютного значення адреси переходу. Старші п’ять розрядів абсолютного значення адреси переходу беруться з поточного значення лічильника команд (після завершення зчитування команди). При виконанні цієї команди указані вище п’ять старших розрядів суміщаються з одинадцятьма молодшими розрядами і утворюють значення адреси переходу, яке пересилається до лічильника команд.

AJMP label ;2 байти, 2 мц

Одинадцять розрядів, що визначаються міткою, яка вказана як операнд, модифікують одинадцять молодших розрядів лічильника команд PC (після завершення зчитування команди).

Примітка. Практично ця команда не використання. Основна її мета – забезпечення сумісності і полегшення переходу від мікроконтролерів серії MCS-48 до мікроконтролерів серії MCS-51.

Команди переходу, що обчислюється

Команда JMP @A+DPTR

Команда «непрямий перехід», дозволяє здійснити перехід у будь-яку точку програми в межах всього адресного простору пам'яті програм мікроконтролера (64 кбайт). Точка переходу (адреса переходу) визначається побічно за сумою значень вмісту регістра DPTR і вмісту регістра A. Застосовується ця команда для утворення «багатовекторного» переходу за деяким значенням.

```
JMP  @A+DPTR          ;1 байт, 2 мц
```

Вміст регістра A додається до вмісту регістра DPTR, результат пересилається в лічильник команд PC мікроконтролера.

Команда LCALL <мітка>

Команда «довгий виклик підпрограми» дозволяє здійснити перехід на виконання підпрограми (виклик підпрограми), яка може розташовуватися в будь-якій точці в межах всього адресного простору пам'яті програм мікроконтролера (64 кбайт). Точка переходу визначається міткою (мітка є ім’ям підпрограми), яка вказана в команді як операнд. Причому, в машинному коді операнд може мати вигляд абсолютного значення адреси переходу, тобто, по суті, мітка являє собою значення адреси осередка пам'яті програм, в якій знаходиться перша команда підпрограми.

```
LCALL label           ;3 байти, 2 мц
```

Механізм виконання даної команди описується послідовним виконанням двох операцій. Перша – зберігання поточного значення лічильника команд мікроконтролера в стековій пам’яті. Друга – завантаження в лічильник команд мікроконтролера адреси осередка пам'яті програм, в якій знаходиться перша команда підпрограми, що викликається.

Зберігання поточного значення лічильника команд мікроконтролера (вміст після завершення зчитування команди) у стековій пам’яті відбувається таким чином. Вміст вказівника стека SP збільшується на 1 і відбувається пересилання старшого байта лічильника команд PC у стек (в осередок пам'яті даних, адреса якої знаходиться в регістрі SP). Потім вміст вказівника стека SP збільшується на 1

і відбувається пересилання молодшого байта лічильника команд PC у стек (в осередок пам'яті даних, адреса якої знаходиться в регістрі SP).

Завантаження в лічильник команд мікроконтролера адреси осередка пам'яті програм, в якій знаходиться перша команда підпрограми, що викликається, полягає в пересиланні коду, еквівалентного значенню мітки, безпосередньо в лічильник команд PC.

Команда ACALL <мітка>

Команда «абсолютний виклик підпрограми» дозволяє здійснити перехід на виконання підпрограми (виклик підпрограми), що розташована в межах однієї сторінки об'ємом 2 кбайт пам'яті програм. Точка переходу визначається міткою (мітка є ім’ям підпрограми), що вказана в команді як операнд. Причому, в машинному коді операнд буде мати значення молодших одинадцяти розрядів абсолютного значення адреси переходу. Старші п’ять розрядів абсолютного значення адреси переходу беруться з поточного значення лічильника команд (після завершення зчитування команди). При виконанні цієї команди вказані раніше п’ять старших розрядів суміщаються з одинадцятьма молодшими розрядами і утворюють значення адреси переходу, яке пересилається до лічильника команд.

ACALL label ;2 байти, 2 мц

Механізм виконання даної команди описується таким самим механізмом, як і механізм виконання команди LCALL, за виключенням процесу визначення адреси переходу. Процес визначення адреси переходу повністю співпадає з процесом, описаним в команді AJMP <мітка>.

Примітка. Практично ця команда не використовується, основна її мета – забезпечення сумісності і полегшення переходу від мікроконтролерів серії MCS-48 до мікроконтролерів серії MCS-51.

Команди переходу (повернення) з підпрограми

Команда RET

Команда «повернення з підпрограми» здійснює повернення з підпрограми в точку, що розташована безпосередньо за командою виклику підпрограми, тобто лічильник команд мікроконтролера модифікується таким чином, що містить адресу осередка пам'яті програм, яка є наступною за осередком, в якій розташована команда виклику підпрограми. Це досягається відновленням вмісту

лічильника команд значенням, що було збережене в стековій пам'яті при виконанні команди виклику підпрограми.

```
RET             ;1 байт, 2 мц
```

Механізм виконання цієї команди відбувається таким чином. Вміст із стека (з осередка пам'яті даних, адреса якої знаходиться в регістрі SP) пересилається в молодший байт лічильника команд, і вміст вказівника стека SP зменшується на одиницю. Потім вміст із стека (з осередка пам'яті даних, адреса якої знаходиться в регістрі SP) пересилається в старший байт лічильника команд, і вміст вказівника стека SP зменшується на одиницю.

Команда RETI

Команда «повернення з підпрограми переривання» здійснює повернення з підпрограми переривання в точку, що розташована безпосередньо за командою, на якій був здійснений перехід на підпрограму обробки переривання (це досягається відновленням вмісту лічильника команд значенням, що було збережене в стековій пам'яті при виконанні апаратної команди виклику підпрограми.). А також відновлює «логіку переривань», дозволяючи обробку інших переривань з рівнем пріоритету, який дорівнює пріоритету тільки що обробленого переривання.

```
RETI            ;1 байт, 2 мц
```

Механізм виконання цієї команди відбувається таким чином. Вміст із стека (з осередка пам'яті даних, адреса якої знаходиться в регістрі SP) пересилається в молодший байт лічильника команд, і вміст вказівника стека SP зменшується на одиницю. Потім вміст із стека (з осередка пам'яті даних, адреса якої знаходиться в регістрі SP) пересилається в старший байт лічильника команд, і вміст вказівника стека SP зменшується на одиницю. В завершення відновлюється «логіка переривань».

3.3.6.2. Команди умовних переходів

Виконання всіх команд умовного переходу здійснюється в два етапи. Перший – перевірка умови, що визначається командою, другий – перехід за вказаною міткою, якщо умова виконується, або перехід на виконання наступної команди, якщо умова не виконується.

Всі команди умовного переходу дозволяють здійснити перехід (при виконанні умови переходу) у будь-яку точку програми, розташовану в пам'яті

програм на відстані до 127 байт вище або нижче від розташування команди умовного переходу. Причому, в командах умовного переходу операнд, що є міткою, представляє абсолютне значення (зі знаком +/– в «додатковому коді»), яке додається до поточного вмісту лічильника команд (після завершення зчитування команди).

Команди умовного переходу представлені таким набором:

JB <біт-джер.>,<мітка>
JNB <біт-джер.>,<мітка>
JBC <біт-джер.>,<мітка>
JC <мітка>
JNC <мітка>
JZ <мітка>
JNZ <мітка>
CJNE <байт-призн.>,<байт-джер.>,<мітка>
DJNZ <байт-джер>,<мітка>

Більшість команд утворюють взаємопротилежні пари, тобто як умова переходу використовуються взаємопротилежні умови переходу. Наприклад: «перехід, якщо вміст акумулятора дорівнює нулю» і «перехід, якщо вміст акумулятора не дорівнює нулю», «перехід, якщо біт встановлений» і «перехід, якщо біт не встановлений».

Команда JB <біт-джер.>,<мітка>

Команда «перехід, якщо біт встановлений» здійснює перехід за міткою, якщо «біт-джерела» встановлено (дорівнює одиниці), в іншому випадку – виконується така команда.

```
JB  bit,label        ;3 байти, 2 мц
```

Якщо вміст біта, що зазначений прямою адресою в першому операнді, дорівнює одиниці – здійснюється перехід за міткою, що визначається другим операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною є команда JNB <біт-джер.>,<мітка>.

Команда JNB <біт-джер.>,<мітка>

Команда «перехід, якщо біт не встановлений» здійснює перехід за міткою, якщо «біт-джерела» не встановлено (дорівнює нулю), в іншому випадку – виконується така команда.

```
JNB  bit,label        ;3 байти, 2 мц
```

Якщо вміст біта, що зазначений прямою адресою в першому операнді, дорівнює нулю – здійснюється перехід за міткою, що визначається другим операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною є команда JB <біт-джер.>,<мітка>.

Команда JBC <біт-джер.>,<мітка>

Команда «перехід, якщо біт встановлений і його скидання» здійснює перехід за міткою, якщо «біт-джерела» встановлено (дорівнює одиниці), в іншому випадку – виконується наступна команда. В будь-якому випадку «біт-джерела» скидається (тобто приймає значення нуль).

JBC bit,label ;3 байти, 2 мц

Якщо вміст біта, що зазначений прямою адресою в першому операнді, дорівнює одиниці – здійснюється перехід за міткою, що визначається другим операндом, в іншому випадку – виконується наступна команда. В будь-якому випадку вказаний біт скидається – приймає нульове значення.

Команда не має взаємопротилежної команди.

Примітка. Якщо команда JBC застосовується для зміни вмісту порту, то значення, яке використовується як дані порту, буде зчитуватися із «засувки» порту, а не з виводів порту мікроконтролера.

Команда JC <мітка>

Команда «перехід, якщо біт переносу встановлений» здійснює перехід за міткою, якщо прапорець переносу C встановлено (дорівнює одиниці), в іншому випадку – виконується така команда.

JC label ;2 байти, 2 мц

Якщо вміст прапорця переносу C дорівнює одиниці – здійснюється перехід за міткою, що визначається операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною є команда JNC <мітка>.

Команда JNC <мітка>

Команда «перехід, якщо біт переносу не встановлений» здійснює перехід за міткою, якщо прапорець переносу C не встановлено (дорівнює нулю), в іншому випадку – виконується така команда:

JNC label ;2 байти, 2 мц

Якщо вміст прапорця переносу C дорівнює нулю – здійснюється перехід за міткою, що визначається операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною є команда JC <мітка>.

Команда JZ <мітка>

Команда «перехід, якщо вміст акумулятора дорівнює нулю» здійснює перехід за міткою, якщо вміст регістра A дорівнює нулю, в іншому випадку – виконується така команда:

```
JZ  label          ;2 байти, 2 мц
```

Якщо вміст регістра A дорівнює нулю – здійснюється перехід за міткою, що визначається операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною є команда JNZ <мітка>.

Команда JNZ <мітка>

Команда «перехід, якщо вміст акумулятора не дорівнює нулю» здійснює перехід за міткою, якщо вміст регістра A не дорівнює нулю, в іншому випадку – виконується така команда:

```
JNZ  label          ;2 байти, 2 мц
```

Якщо вміст регістра A не дорівнює нулю – здійснюється перехід за міткою, що визначається операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною є команда JZ <мітка>.

Команда CJNE <байт-призн.>,<байт-джер.>,<мітка>

Команда «порівняння і перехід, якщо не дорівнює» порівнює значення «байта-призначення» та «байта-джерела», і, якщо їх значення різні, то здійснюється перехід за міткою, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним вміст «байт-призначення» та «байт-джерела» і супроводжується модифікацією прапорця переносу C.

Прапорець переносу C встановлюється в одиницю, якщо значення цілого без знака «байта-призначення» менше, ніж значення цілого без знака «байта-джерела», в іншому випадку прапорець переносу C скидається в нуль.

Операнди джерела та призначення допускають чотири комбінації адресації:

```
CJNE A,direct,label       ;3 байти, 2 мц
```

Порівнюється вміст регістра А та вміст осередка внутрішньої пам’яті даних, пряма адреса якої визначається другим операндом. У разі, якщо їх значення різні, то здійснюється перехід за міткою, що визначається третім операндом, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним значення, що порівнюються і супроводжується модифікацією прапорця переносу С відповідно до результату порівняння.

CJNE A,#data,label ;3 байти, 2 мц

Порівнюється вміст регістра А та безпосереднє значення, що визначається другим операндом. У разі, якщо їх значення різні, то здійснюється перехід за міткою, що визначається третім операндом, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним значення, що порівнюються і супроводжується модифікацією прапорця переносу С відповідно до результату порівняння.

CJNE Rn,#data,label ;3 байти, 2 мц

Порівнюється вміст регістра загального призначення Rn, який вказаний як перший операнд, та безпосереднє значення, що визначається другим операндом. У разі, якщо їх значення різні, то здійснюється перехід за міткою, що визначається третім операндом, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним значення, що порівнюються і супроводжуються модифікацією прапорця переносу С відповідно до результату порівняння.

CJNE @Ri,#data,label ;3 байти, 2 мц

Порівнюється вміст осередка внутрішньої пям’яті даних, адреса якої міститься в регістрі Ri, який вказаний як перший операнд, та безпосереднє значення, що визначається другим операндом. У разі, якщо їх значення різні, то здійснюється перехід за міткою, що визначається третім операндом, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним значення, що порівнюється і супроводжується модифікацією прапорця переносу С відповідно до результату порівняння.

Команда не має взаємопротилежної команди.

За допомогою двох команд – CJNE та JC (або JNC) можливо здійснення розгалуження за ознаками «байт-призначення» дорівнює «байту-джерела», «байт-призначення» менше «байта-джерела» та «байт-призначення» більше «байта-джерела». Завдяки відповідного об’єднання гілок розгалуження не суворі

нерівності – «байт-призначення» менше або дорівнює «байту-джерела» та «байт-призначення» більше або дорівнює «байту-джерела».

Команда DJNZ <байт-призн.>,<мітка>

Команда «декремент і перехід, якщо не дорівнює нулю» зменшує на одиницю вміст «байта-призначення» і здійснює перехід за міткою у разі, якщо вміст «байта-призначення» не дорівнює нулю, в іншому випадку – виконується наступна команда. Операнди «байта-призначення» допускають дві комбінації адресації:

DJNZ Rn,label ;2 байти, 2 мц

Зменшується на одиницю вміст регістра загального призначення Rn, який вказаний як перший операнд. Якщо вміст цього регістра не дорівнює нулю, то здійснюється перехід за міткою, що вказана як другий операнд, в іншому випадку – виконується така команда:

DJNZ direct,label ;3 байти, 2 мц

Зменшується на одиницю вміст осередка внутрішньої пам’яті даних, пряма адреса якої визначається першим операндом. Якщо вміст цього осередка не дорівнює нулю, то здійснюється перехід за міткою, що вказана як другий операнд, в іншому випадку – виконується наступна команда.

Команда не має взаємопротилежної команди.

Примітка. Команду «декремент і перехід, якщо не дорівнює нулю» за ознаками функцій, що здійснюються при її виконанні, можливо віднести як до групи команд умовного переходу, так і до групи команд інкременту та декременту. Функціональне призначення і мнемонічні означення команд наведені в табл. 3.1.

Таблиця 3.1 – Мнемонічні позначення команд мікроконтролерів серії MCS-51 та їх функціональне призначення

Команда	Операнди	Функція команди
ACALL	<мітка>	Абсолютний виклик підпрограми
ADD	A,<байт-джер.>	Додавання
ADDC	A,<байт-джер.>	Додавання з переносом
AJMP	<мітка>	Абсолютний перехід
ANL	<байт-приз.>,<байт-джер.>	Логічне «І» для байтів
ANL	C,<біт-джер.>	Логічне «І» для бітів
CJNE	<байт-приз.>,<байт-джер.>,<мітка>	Порівняння і перехід, якщо не дорівнює
CLR	A	Скидання акумулятора
CLR	C	Скидання прапорця переносу C

Команда	Операнди	Функція команди
CLR	<біт-приз.>	Скидання біта
CPL	A	Інверсія акумулятора
CPL	C	Інверсія прапорця переносу C
CPL	<біт-приз.>	Інверсія біта
DA	A	Десяткова корекція
DEC	<байт-приз.>	Декремент
DIV	AB	Ділення
DJNZ	<байт-приз.>,<мітка>	Декремент і перехід, якщо не дорівнює нулю
INC	<байт-приз.>	Інкремент
INC	DPTR	Інкремент вказівника DPTR
JB	<біт-приз.>,<мітка>	Перехід, якщо біт встановлений
JBC	<біт-приз.>,<мітка>	Перехід, якщо біт встановлений зі скиданням
JC	<мітка>	Перехід, якщо прапорець переносу C встановлений
JMP	@A+DPTR	Непрямий перехід
JNB	<біт-приз.>,<мітка>	Перехід, якщо біт не встановлений
JNC	<мітка>	Перехід, якщо прапорець переносу C не встановлений
JNZ	<мітка>	Перехід, якщо вміст акумулятора не дорівнює нулю
JZ	<мітка>	Перехід, якщо вміст акумулятора дорівнює нулю
LCALL	<мітка>	Довгий виклик підпрограми
LJMP	<мітка>	Довгий перехід
MOV	<байт-приз.>,<байт-джер.>	Пересилання байта даних
MOV	<біт-приз.>,<біт-джер.>	Пересилання біта даних
MOV	DPTR,#<значення>	Завантаження вказівника даних DPTR
MOVC	A,@A+<R16>	Пересилання байта з пам'яті програм
MOVX	<байт-приз.>,<байт-джер.>	Пересилання в зовнішню пам'ять даних (із зовнішньої пам'яті даних)
MUL	AB	Множення
NOP		Немає операції
ORL	<байт-приз.>,<байт-джер.>	Логічне «АБО» для байтів
ORL	C,<біт-джер.>	Логічне «АБО» для бітів
POP	<байт-приз.>	Читання зі стека
PUSH	<байт-джер.>	Запис у стек
RET		Повернення з підпрограми
RETI		Повернення з підпрограми переривання
RL	A	Зсув вмісту акумулятора вліво
RLC	A	Зсув вмісту акумулятора вліво через прапорець переносу
RR	A	Зсув вмісту акумулятора вправо

Команда	Операнди	Функція команди
RRC	A	Зсув вмісту акумулятора вправо через прапорець переносу
SETB	<біт-приз.>	Встановлення біта
SJMP	<мітка>	Короткий перехід
SUBB	A,<байт-джер.>	Віднімання з позичкою
SWAP	A	Обмін тетрадами в акумуляторі
XCH	A,<байт-джер.>	Обмін вмісту акумулятора з байтом
XCHD	A,@<R8>	Обмін тетрадой
XRL	<байт-приз.>,<байт-джер.>	Логічне «ВИКЛЮЧАЮЧЕ АБО»

Контрольні питання

1. Скільки банків даних містить мікроконтролер MCS-51?
2. Де розташована стекова пам’ять у MCS-51?
3. Яку кількість розрядів містить покажчик стекової пам’яті в MCS-51?
4. Яку кількість регістрів містить нульовий банк у MCS-51?
5. Яка початкова адреса стекової пам’яті в MCS-51?
6. Який мінімальний обсяг пам’яті даних в MCS-51?
7. Який регістр MCS-51, крім акумулятора, використовується в командах множення і ділення MUL і DIV?
8. Яка команда пересилає інформацію із стека до MCS-51?
9. Яка команда пересилає інформацію з банка даних до акумулятора?
10. Яка команда пересилає константу до регістра, що знаходиться в банку даних?
11. Яка команда пересилає інформацію з осередка в осередок, що знаходяться в пам’яті даних?
12. Яка команда пересилає інформацію із зовнішньої пам’яті даних до MCS-51?
13. Яка команда виконує десяткову корекцію акумулятора?
14. Яка команда збільшує вміст акумулятора?
15. Яка команда виконує циклічний зсув вмісту акумулятора ліворуч?
16. Яка команда виконує скидання вмісту акумулятора?

4. ПОРТИ ВВЕДЕННЯ/ВИВЕДЕННЯ ІНФОРМАЦІЇ ВІД ЗОВНІШНІХ ПРИСТРОЇВ

Мікроконтролер серії MCS-51 має чотири порти введення/виведення інформації (рис. 4.1). Вони призначені для двонаправленого обміну інформацією із зовнішніми пристроями, такими як кнопки, датчики, виконавчі прилади, індикатори, а також із зовнішньою пам'яттю програм і даних. Крім цього, порти використовуються під час програмування внутрішньої пам'яті програм контролера.

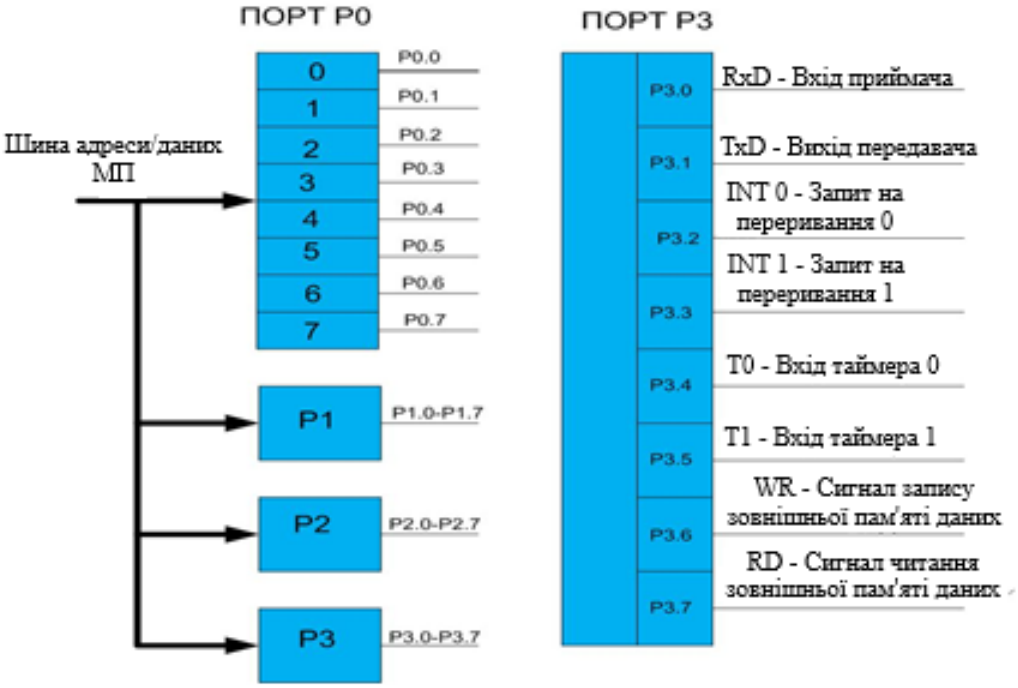


Рисунок 4.1 – Структурна схема портів мікроконтролера MCS-51

Функціональна схема ліній портів зображена на рис. 4.2 і 4.3. Базовою є модель порту P1. Вона містить керований тригер-засувку, виконаний у вигляді D-тригера, вхідний буфер B1, буфер B2 читання тригера-засувки і вихідний драйвер, виконаний на польовому транзисторі з ізольованим затвором і навантажений на резистор. Схему порту P0 доповнено мультиплексором MS (на рисунку зображений у вигляді перемикача), який дозволяє підключати до виходів порту молодший байт адресної шини або шину даних процесора. Крім цього, для підвищення швидкодії порту при роботі із зовнішньою пам'яттю вихідний ключ виконаний за двотранзисторною схемою. Схему порту P2 доповнено мультиплексором MS, який дозволяє підключати до виходів порту старший байт адресної шини процесора при роботі із зовнішньою пам'яттю. Схему порту P3 доповнено буферами підключення периферійних пристроїв.

Адресація до портів виконується так само, як до осередків пам'яті. Фізичні адреси портів такі:

P0-80h, при бітовій адресації 80h-87h;

P1-90h, при бітовій адресації 90h-97h;

P 2-A0h, при бітовій адресації A0h-A7h;

P 3-B0h, при бітовій адресації B0h-B7h.

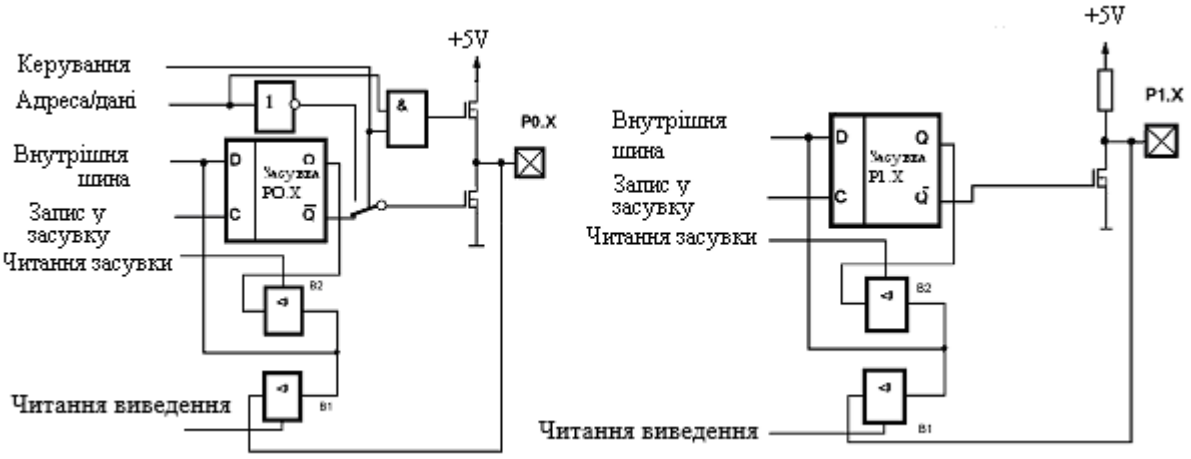


Рисунок 4.2 – Функціональні схеми каналів портів P0 і P1

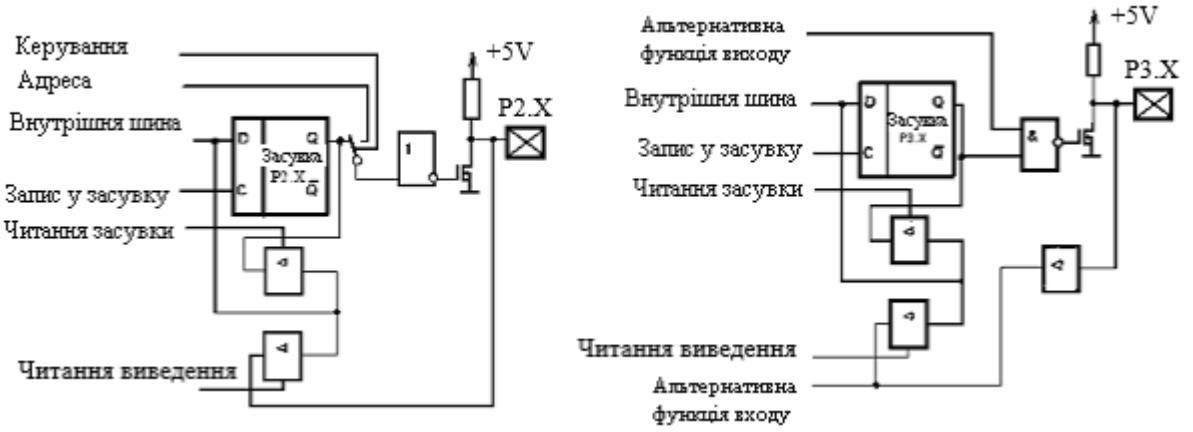


Рисунок 4.3 – Функціональні схеми каналів портів P2 і P3

Порт P0 є двонаправленим, а порти P1, 2P, P3 квазідвонаправленими. Кожна лінія портів може бути використана незалежно для введення або виведення інформації. Для організації введення інформації на якийсь лінії порту D-тригер регістра засувки порту встановлюють в одиницю. При цьому вихідний МОП-транзистор закривається і на виведенні мікросхеми встановлюється високий рівень напруги (тобто 1). При подачі напруги живлення на мікроконтролер формується сигнал скидання (RESET), при цьому в регістри-засувки порту автоматично записуються одиниці, і тим самим всі лінії порту програмуються на введення.

При обміні інформацією із зовнішньою пам'яттю даних і програм інформація на портах встановлюється процесором автоматично. При цьому через порт P0 здійснюється видача молодшого байта адреси і прийняття або видача даних, а на порт P2 видається старший байт адреси. У тригері-засувці порту P0 при обміні даними записуються всі одиниці.

У режимі програмування порт P0 використовується як шина даних, через порт P1 подається молодший байт адреси, через порт P2 – старші розряди адреси і керуючі сигнали. Розряди порту P3, крім функцій введення/виведення, мають альтернативні функції, які наведені в табл. 4.1.

Таблиця 4.1 – Альтернативні функції порт P3

Сигнал	Розряд порту	Призначення
$\overline{RD}$	P3.7	Читання. Активний сигнал низького рівня формується апаратно при зверненні до ЗПД
$\overline{WR}$	P3.6	Читання. Активний сигнал низького рівня формується апаратно при зверненні до ЗПД
T1	P3.5	Вхід таймера лічильника 1 або тест-вхід
T0	P3.4	Вхід таймера лічильника 0 або тест-вхід
$\overline{INT1}$	P3.3	Вхід запиту переривання 1. Сприймається сигнал низького рівня або зріз
$\overline{INT0}$	P3.2	Вхід запиту переривання 1. Сприймається сигнал низького рівня або зріз
TxD	P3.1	Вихід передавача послідовного порту в режимі УСАПП. Вихід синхронізації в режимі регістра, що зсуває
RxD	P3.0	Вхід приймача послідовного порту в режимі УАСПП. Введення/виведення даних у режимі регістра, що зсуває

Навантажувальна здатність порту P0 – два TTL входу, портів P1, P2, P3 – один TTL вхід. Для приєднання потужного навантаження необхідно використовувати транзисторні підсилювачі (рис. 4.3).

Операцію прийняття сигналів від зовнішнього пристрою контролером прийнято називати введенням. Для налаштування порту або окремих бітів порту на введення у тригери-засувки відповідних розрядів необхідно занести одиницю.

Прочитати стан портів можна командою

```
MOV A, P0 ; занесення в акумулятор значення порту P0.
```

Крім цього, з розрядами портів можливі бітові операції:

```
MOV C, P1.1  ; пересилання розряду порту P1.1 у перенос.
```

Оскільки в модернізованому мікроконтролері 1T8051 збережена повністю система команд MCS 8051, то збережено і кількість портів, їх найменування та

квазідвонаправлена структура. Запис і читання регістра керування портом здійснюються по-різному.

Запис у регістр керування портом встановлює логічне значення тригера-засувки вихідного порту, а читання здійснюється з вихідного контакту порту. Всі лінії портів, крім P2.0, можуть бути налаштовані індивідуально на чотири режими введення/виведення. Ці чотири режими є: квазідвонаправлений (структура порту MCS 8051), двотактний, тільки введення і відкритий стік.

Режими роботи портів встановлюються за допомогою регістрів спеціальних функцій PxM1 і PxM2 (Px – лінія порту). Стан бітів портів PxM1.n PxM2.n для різних режимів введення/виведення: 00 – квазідвонаправлений, 01 – двотактний, 10 – тільки введення (високоімпедансний), 11 – відкритий стік.

Всі лінії портів введення/виведення можуть бути запрограмовані на використання сигналів рівня TTL або входів тригерів Шмітта, вибором відповідних бітів у регістрі PxS. Вхід з тригером Шмітта забезпечує краще заглушення перешкод.

Всі лінії введення/виведення також мають можливість вибору швидкості наростання фронтів за допомогою бітів-регістрів керування PxSR. У вихідних режимах швидкість зростання повільна.

У квазідвонаправленій структурі введення/виведення є три польових транзистори (рис. 4.4), які забезпечують підтягування виходу порту до потенціалу джерела живлення.

Коли мікропроцесор на вихідний контакт порту видає високий логічний рівень, транзистор T4 забезпечує «дуже слабке» підтягування виведення порту до потенціалу джерела живлення. Цей режим забезпечує дуже малий струм навантаження. При цьому можна управляти затвором польового транзистора.

Другий режим «слабкого» підтягування забезпечується транзистором T3. У цьому випадку на лінії порту встановлений високий рівень (1) і при цьому забезпечується більший струм навантаження, наприклад, вихідний сигнал сенсора управляє світлодіодом оптрона (рис. 4.5).

На рис. 4.5 наведена схема підключення сенсорів з безконтактним виведенням і з введенням типу «сухий контакт». Сенсори з безконтактними виходами бувають двох видів. З вихідним транзистором NPN-типу для підключення до систем із загальним плюсом (рис. 4.5, *а*), при цьому транзистор комутує мінусовий потенціал, і з транзистором PNP-типу – для підключення до систем із загальним мінусом (рис. 4.5, *б*), при цьому транзистор комутує плюсовий потенціал.

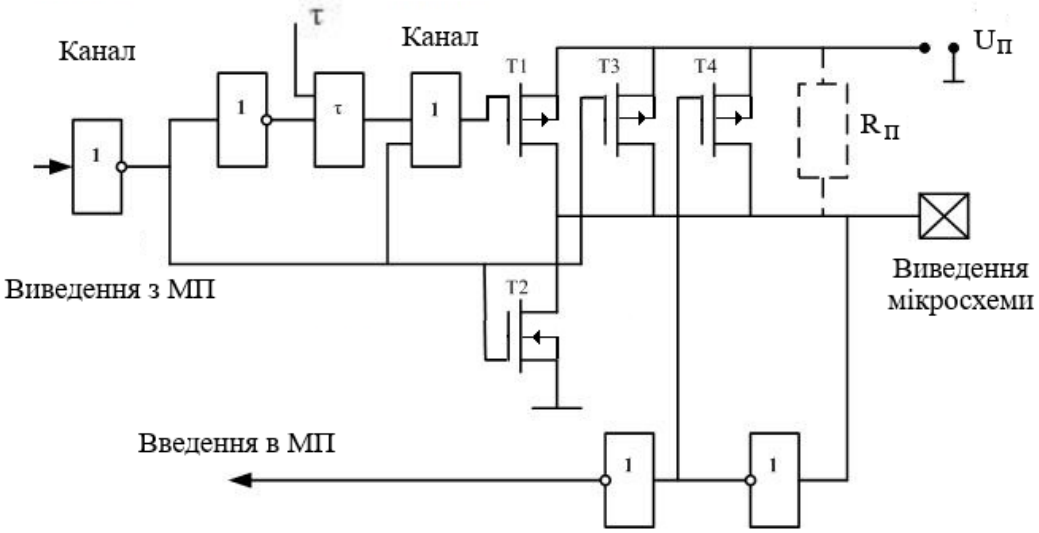
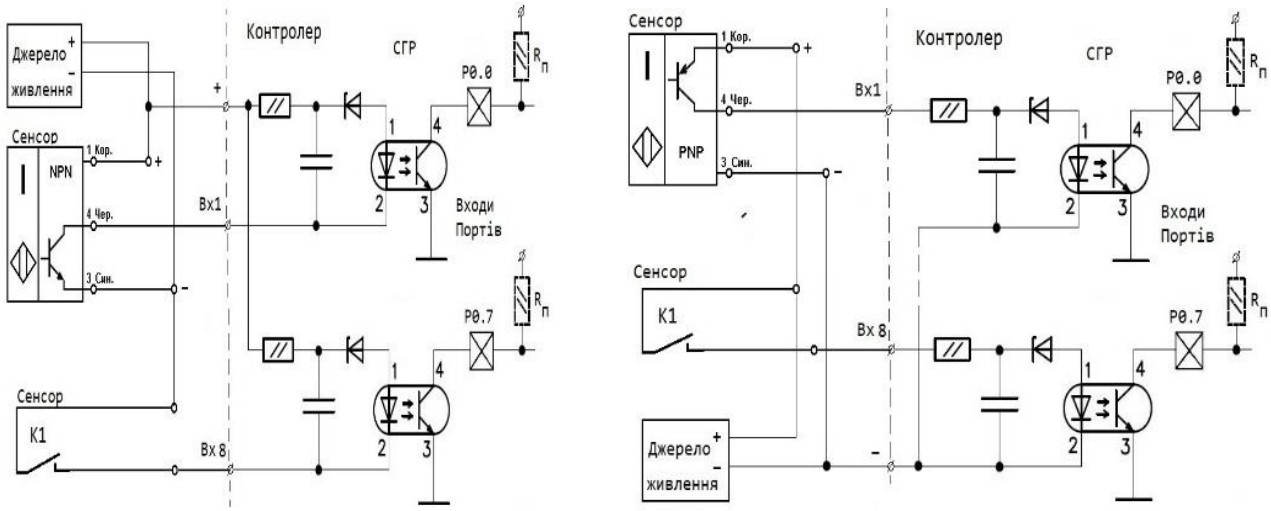


Рисунок 4.4 – Функціональна схема квазідвонаправленого
режиму введення/виведення лінії порту мікроконтролера 1Т80-51



а) *б*)

Рисунок 4.5 – Функціональна схема введення інформації від сенсорів

Якщо під час введення інформації від зовнішніх пристроїв на вихідні контакти порту виведена логічна одиниця, а зовнішній пристрій тягне вихідний потенціал порту до нуля, рис. 4.5, *а* і *б*, (при протіканні струму світлодіода відкривається фототранзистор оптрона і замикає вивід мікроконтролера на землю. При цьому режим «слабкого» підтягування відключається, і залишається тільки «дуже слабкий» режим підтягування. У цьому випадку струм навантаження більший, ніж один вхід TTL (декілька світлодіодів оптронів).

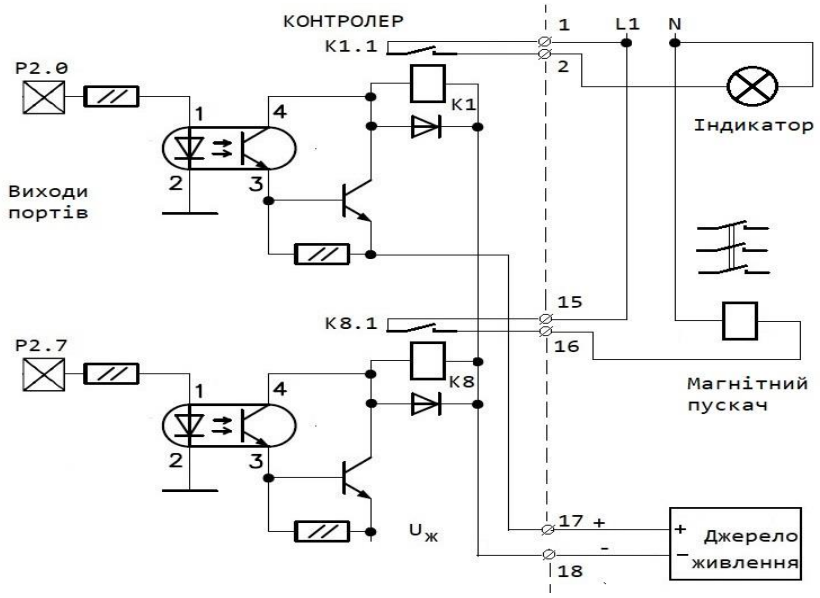


Рисунок 4.6 – Функціональна схема виведення інформації з порту

У режимі квазідвонаправленого виведення порту, при зміні стану тригера-засувки порту з логічного нуля на логічну одиницю для забезпечення високої крутизни фронтів при перемиканні вмикається режим «сильного підтягування» (рис. 4.6), при цьому на час двох тактів мікропроцесора з використанням сигналу τ, вмикається польовий транзистор Т2 (рис. 4.4), щоб швидко змінити вихід порту з низького потенціалу на високий. «Слабкий» і «дуже слабкий» режими підтягування продовжують залишатися на рівні потенціалу виведення порту.

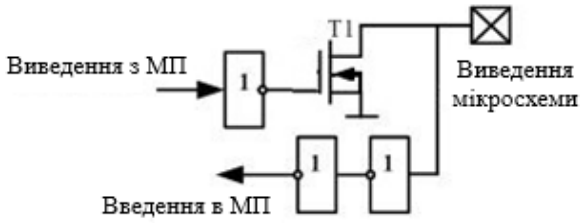


Рисунок 4.7 – Функціональна схема режиму «відкритий стік»

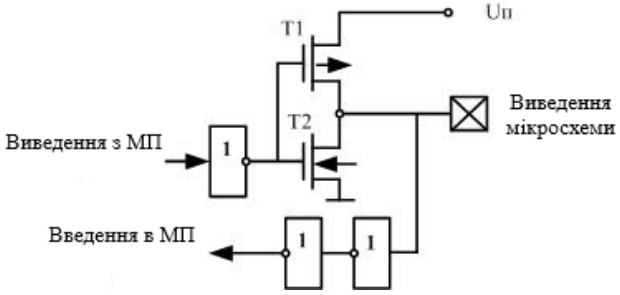


Рисунок 4.8 – Функціональна схема двотактного режиму

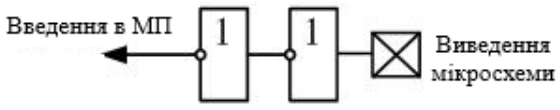


Рисунок 4.9 – Функціональна схема режиму «тільки введення»

Двотактний режим (Push-Pull, рис. 4.8) має ту саму структуру з низькою напругою, що і квазідвонаправлений режим, але забезпечує безперервний двотактний режим, коли засувка порту записується логічною одиницею (1).

Зазвичай двотактний режим використовується як вихідний контакт, коли вихідна лінія порту потребує великого струму навантаження.

У режимі «тільки введення» (рис. 4.9) лінія порту підключається до джерела напруги через резистор.

Режим з відкритим стоком (рис. 4.7) відключає всі транзистори високої напруги (рис. 4.4), і управляє низьким рівнем напруги на виході порту тільки тоді, коли засувка порту задається логічним нулем (0). Якщо вихід засувки порту встановлений в одиницю, то він працює тільки в режимі введення.

Команди передачі керування дозволяють за значеннями окремих бітів порту змінювати послідовність виконання програми, наприклад:

```
JB P2.0, M1  ; перехід на мітку M1, якщо розряд P2.0 дорівнює одиниці
JNB P0.0, M2 ; перехід на мітку M2, якщо розряд P0.0 дорівнює нулю.
```

Операцію передачі сигналів від мікроконтролера до зовнішнього пристрою називають виведенням. У цьому випадку на розрядах портів мікроконтролер установлює потрібні логічні рівні сигналів, і за таким способом управляє зовнішніми пристроями. При виконанні операції виведення в порт нове значення записується в засувку у фазі S6P2 останнього машинного циклу команди. Однак новий вміст засувки виводиться безпосередньо на вихідний контакт тільки у фазі S1P1 наступного машинного циклу. Виведення інформації в порт можна виконати командами пересилань, наприклад:

```
MOV P1, A ; пересилання в порт P1 вмісту акумулятора
```

Якщо порт є одночасно операндом і місцем призначення результату, то процесор автоматично реалізує спеціальний режим, який називається «читання–модифікація–запис». Цей режим опитування припускає введення сигналів не із зовнішніх виводів порту, а з його регістра-засувки, що дозволяє виключити неправильне зчитування раніше виведеної інформації, оскільки передбачається модифікування вмісту тригера-засувки, а не вхідної інформації. Цей режим звертання до портів реалізований у таких командах:

ANL – логічне І, наприклад ANL P1, A;

ORL – логічне АБО, наприклад ORL P2, A;

XRL – виключне АБО, наприклад XRL P3, A;

JBC – перехід, якщо в адресованому біті – одиниця, і наступне скидання біта, наприклад JBC P1.1,LABEL;

CPL – інверсія біта, наприклад CPL P3.3;

INC – інкремент порту, наприклад INC P2;

DEC – декремент порту, наприклад DEC P2;

DJNZ – декремент порту і перехід, якщо його вміст не дорівнює нулю, наприклад DJNZ P3,LABEL;

MOV PX.Y,C – передача біта переносу в біт Y порту X;

SETB PX.Y – встановлення біта Y порту X;

CLR PX.Y – скидання біта Y порту X.

Приклади роботи з портами.

Порти в мікроконтролері призначені для введення вхідних сигналів і реалізації функцій дискретного керування. Фактично, задача дискретного керування об'єктами з використанням програмувальних контролерів полягає у введенні вхідних сигналів, в обчисленні логічної функції, що зв’язує вхідні і вихідні сигнали та виведення результату обчислення на виходи контролера. У першій главі розглянуто задачу дискретного керування, яке можна записати у вигляді комбінаційних і послідовнісних автоматів. Задача умови для комбінаційного автомата можна зобразити у вигляді таблиці істинності. Розглянемо задачу побудови комбінаційного автомата на прикладі розробки дешифратора для семисегментного індикатора. Схему підключення семисегментного індикатора наведено на рис. 4.10. Вхідні сигнали надходять на розряди порту P1.0-P1.3, керування індикатором з порту 2 здійснюється через буфер-підсилювач.

Сформуємо в пам'яті програм контролера масив значень байта виводу згідно з табл. 4.2 (мітка BEGDIM), які потрібно вивести на індикатор для формування шістнадцяткових цифр від 0 до F. Текст підпрограми, яка реалізує дешифратор, наведено нижче.

```
DESH:    MOV DPTR,#BEGDIM; заносимо в DPTR адресу першого
                                ; елемента масиву
         MOV A,P1        ; уводимо вхідні сигнали
         ANL A,#0FH      ; маскуємо старші розряди вхідних
              ; сигналів, одержуємо зсув в таблиці для
           ; даної комбінації
         MOVC A,@A+DPTR; заносимо в акумулятор значення
              ; елемента масиву, що відповідає
              ; вхідному коду
         MOV P2,A        ; виводимо значення в порт
         RET                  ; виходимо з підпрограми
BEGDIM: DB 3Fh, 06h, 5Bh, 4Fh, ; 0, 1, 2, 3
```

```
DB 66h, 6Dh, 7Dh, 07h, ; 4, 5, 6, 7
DB 7Fh, 6Fh, 77h, 7Ch, ; 8, 9, A, B
DB 39h, 5Eh, 79h,71h ; C, D, E, F
```

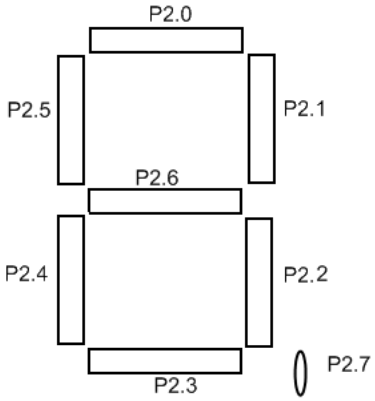


Рисунок 4.10 – Схема підключення семисегментного індикатора до мікроконтролера

У даній програмі вхідне значення коду, утвореного вхідними сигналами, перетвориться в зсув для масиву вихідних значень і виводиться у вихідний порт.

Такий спосіб реалізації комбінаційного автомата має такі обмеження: вхідні сигнали повинні бути підключені на один порт і підряд (інакше прийдеться вирішувати завдання зведення різних входів в одне вхідне слово); при досить великій кількості входів (7 і більше) і неповному використанні вихідних комбінацій нераціонально використовується пам'ять програм. Тому пропонується інший спосіб реалізації, оснований на програмній реалізації логічних кон’юнктивно диз'юнктивних рівнянь.

Як і в попередньому випадку, функціонування автомата задається у вигляді таблиці істинності, наприклад табл. 4.3.

Розв'язок. Запишемо таблицю істинності для дешифратора.

Таблиця 4.2 – Істинності для дешифратора

Входи				Виходи								Символ
1	2	3	4	5	6	7	8	9	10	11	12	13
P1.3	P1.2	P1.1	P1.0	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	
0	0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	0	0	0	0	1	1	0	1
0	0	1	0	0	1	0	1	1	0	1	1	2
0	0	1	1	0	1	0	0	1	1	1	1	3
0	1	0	0	0	1	1	0	0	1	1	0	4
0	1	0	1	0	1	1	0	1	1	0	1	5
0	1	1	0	0	1	1	1	1	1	0	1	6

0	1	1	1	0	0	0	0	0	1	1	1	7
1	0	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	0	1	1	0	1	1	1	1	9
1	0	1	0	0	1	1	1	0	1	1	1	A
1	0	1	1	0	1	1	1	1	1	0	0	B
1	1	0	0	0	0	1	1	1	0	0	1	C
1	1	0	1	0	1	0	1	1	1	1	0	D
1	1	1	0	0	1	1	1	1	0	0	1	E
1	1	1	1	0	1	1	1	0	0	0	1	F

Таблиця 4.3 – Стан бітів портів

Входи				Вихід
P1.2	P1.4	P2.1	P2.3	P1.7
0	1	1	0	1
1	0	1	0	1
1	1	0	0	1

Запишемо логічне рівняння для виходу згідно з наведеною таблицею.

P1.7=!P1.2 & P1.4 & P2.1 & P2.3 +P1.2 & !P1.4 & P2.1 & !P2.3+P1.2 & !P1.4 & P2.1 & !P2.3

Схеми алгоритму програмної реалізації логічних функцій «І» та «АБО» наведені на рис. 4.11 та 4.12.

Програмна реалізація для цієї функції буде така:

```
    JB P1.2,M1
    JNB P1.4,M1
    JNB P2.1,M1
    JNB P2.3,M1
    SETB P1.7
    LJMP EXIT
M1:         ; перевірка наступної умови.
```

Слід звернути увагу, що команди умовного переходу при виконанні умови «порушують» послідовне виконання програми. Тому для того щоб програма була зручною, пропонується використовувати команди, які перевірять інверсну умову. Тоді послідовне виконання таких команд і буде відповідати виконанню вихідної умови логічної функції.

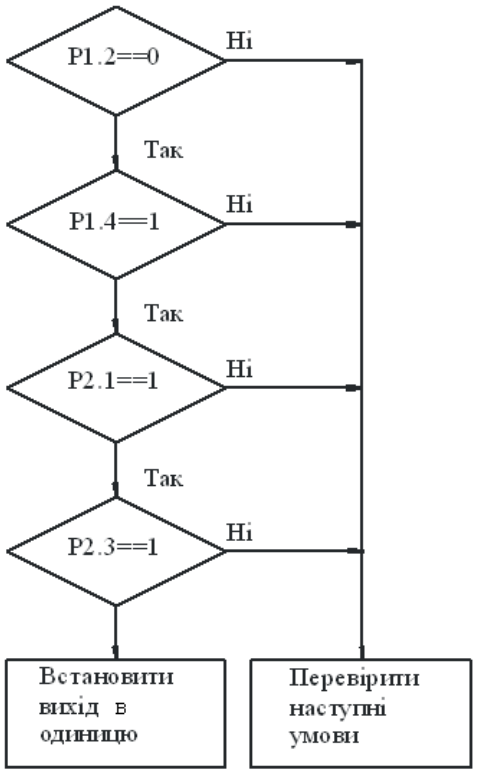


Рисунок 4.11 – Схема алгоритму програмної реалізації логічної функції «І»

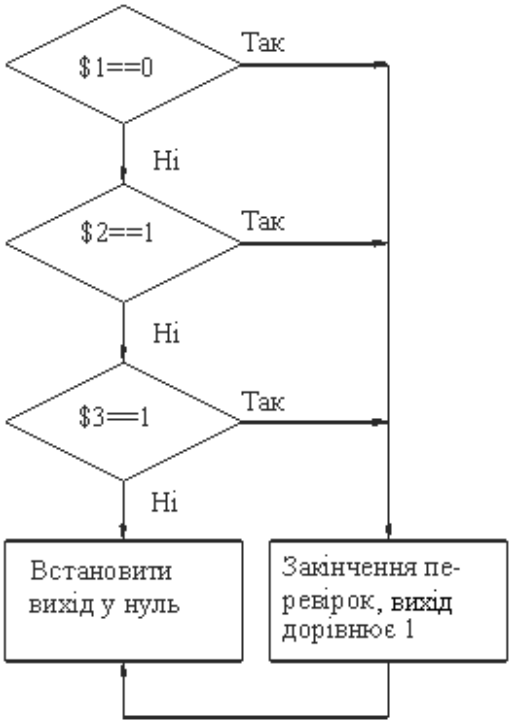


Рисунок 4.12 – Схема алгоритму програмної реалізації логічної функції «АБО»

У цьому алгоритмі під &1, &2 маються на увазі логічні функції «І» наведеного рівняння. Повна програма логічного рівняння, що відповідає таблиці істинності, така:

```
        JB  P1.2,M1; Перевірка умови першого рядка таблиці
        JNB P1.4,M1
        JNB P2.1,M1
        JNB P2.3,M1
        SETB P1.7  ; встановлення виходу в одиницю
        LJMP EXIT ; і вихід
M1:
        JNB P1.2,M2; Перевірка умови другого рядка таблиці
        JB  P1.4,M2
        JNB P2.1,M2
        JB  P2.3,M2
        SETB P1.7  ; встановлення виходу в одиницю
        LJMP EXIT ; і вихід
M2:   JNB P1.2,M3; Перевірка умови третього рядка таблиці
```

```
        JB  P1.4,M3
        JNB P2.1,M3
        JB  P2.3,M3
        SETB P1.77  ; встановлення виходу в одиницю
        LJMP EXIT  ; і вихід
M3:   CLR  P1.7   ; скинути вихід у нуль – не виконалася ні жодна з умов
EXIT:
```

Таким чином реалізується формальний перехід від таблиці істинності до програми керування.

Контрольні питання

1. Який порт MCS-51, крім порту P0, використовується при звертанні до зовнішньої пам’яті?
2. Який з портів MCS-51 має подвійне призначення?
3. В якому порті MCS-51 розташовані сигнали читання та запису інформації в зовнішню пам’ять даних?
4. Яким чином вхід паралельного порту налаштовується на введення інформації?
5. Яка навантажувальна здатність нульового порту?
6. Скільки паралельних портів має мікроконтролер MCS-51?
7. Яка розрядність паралельних портів мікроконтролера MCS-51?

5. ОРГАНІЗАЦІЯ ПЕРЕРИВАНЬ У МІКРОПРОЦЕСОРНИХ СИСТЕМАХ

Логіку роботи режиму переривань зручно розглянути на прикладі читання лекції. Викладач викладає лекцію (мікропроцесор виконує програму). Якщо студенту щось не зрозуміло, він піднімає руку (виставляє сигнал-запит на переривання), щоб професор дав роз'яснення з тих або інших питань. Викладач запам'ятовує, в якому місці перервана лекція (мікропроцесор запам'ятовує у стековій пам'яті адресу, на якій перервана програма), і після цього відповідає на питання студента (мікропроцесор виконує підпрограму обслуговування цього запиту). Після відповідей на питання студента (після виконання підпрограми мікропроцесором) викладач продовжує лекцію з того місця, де він її перервав (мікропроцесор вертається до виконання основної програми).

Таким чином, термін «переривання» відноситься до функціонування мікропроцесора (МП). При звичайному функціонуванні МП на шину адреси виставляє вміст лічильника команд. Функції, які виконує МП, залежать від типу команди. Якщо виконуються команди пересилання інформації, арифметичні, логічні, то вміст лічильника команд (PC) збільшується відомим способом – зростає на одиницю, якщо виконується однобайтова команда, на два, якщо виконується двобайтова команда, на три, якщо виконується трибайтова команда. Новий вміст лічильника команд і є тепер адресою наступної команди в пам'яті програм.

Під час виконання команд умовних і безумовних переходів (LJMP) вміст PC модифікується, як правило, другим і третім байтом цих команд.

І тільки під час виконання команд виклику підпрограм LCALL вміст лічильника команд збільшується на два (PC+3), при виконанні команди ACALL вміст лічильника команд збільшується на одиницю (PC+2) і запам'ятовується в стеку, а потім уже в лічильник команд завантажується адреса підпрограми, яка перебуває в другому і третьому байті команд LCALL або ACALL.

Таким чином, у стеку запам'ятовується адреса команди, за якою була викликана підпрограма. Після закінчення підпрограми ця адреса буде відновлена в лічильнику команд (PC). Реагування мікропроцесора на запити переривань полягає в переході від виконання поточної програми до виконання підпрограми обслуговування переривання, відповідної до даного запиту.

У МК51 це реалізується в такий спосіб. При виставлянні сигналу-запиту на переривання від одного із джерел переривань завершується виконання поточної команди, блокується вибірка команд із пам'яті програм наступної команди. Далі

контролер переривання на шину даних виставляє код операції команди LCALL за адресою підпрограми, відповідної даному запиту. Завдяки цій команді МП поточне значення лічильника команд запам'ятовує в стеку, а в лічильник команд заноситься другий і третій байти команди LCALL. Таким чином, звичайний хід виконання програми примусово переривається й мікропроцесор переходить до виконання підпрограми. Саме на цьому принципі й організовано переривання в мікроконтролері MCS-51.

Функціональна схема взаємодії МП із периферійними пристроями в MCS-51 наведена на рис. 5.1.

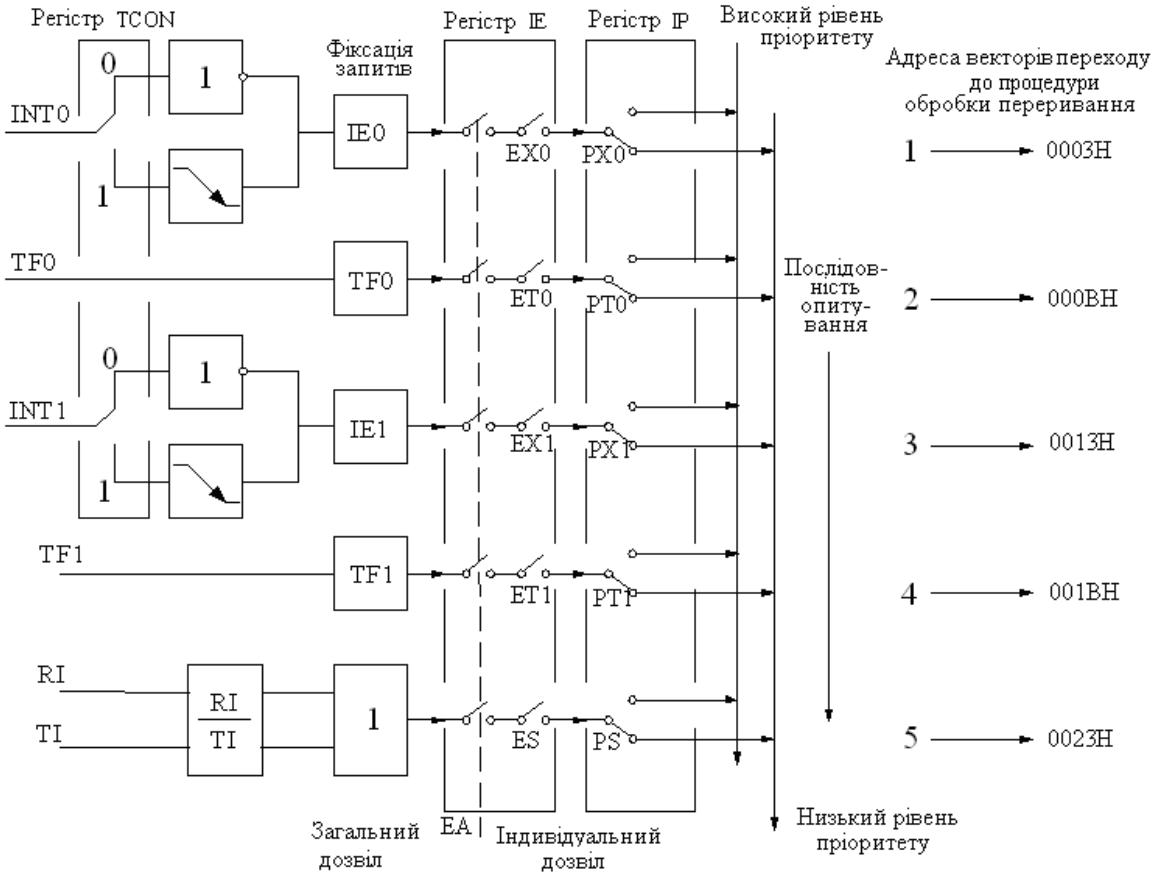


Рисунок 5.1 – Функціональна схема системи переривань MCS-51

Роботу мікропроцесора можуть переривати як внутрішні периферійні пристрої мікроконтролера – таймер 0 (T0), таймер 1 (T1), послідовний порт (SBUFF), так і зовнішні пристрої, які перебувають за межами мікроконтролера (усілякі датчики, пристрої захисту і т.ін.). Для зовнішніх пристроїв у МК51 передбачено два входи від сигналів-запитів переривань ($\overline{INT0}$ і $\overline{INT1}$).

Таймери T0 і T1 формують сигнали-запити на переривання прапорцями переповнення TF0 і TF1 у регістрі керування TCON (TCON.5, TCON.7) і прапорцями TI, RI у регістрі керування послідовним портом SCON (SCON1, SCON0).

Спрощена функціональна схема системи переривань зображена на рис. 5.1, де показано п'ять сигналів запитів переривань, $\overline{INT0}$ $\overline{INT1}$ – зовнішні сигнали запитів переривань, TF0, TF1, SBUFF – сигнали запитів переривань від внутрішніх периферійних пристроїв (таймери T0 і T1 і послідовний порт). Причому зовнішні сигнали запитів переривань можуть сприйматися контролером як за рівнем, так і за перепадом сигналу (зрізу) з «1» в «0». Вибір режиму визначається станами бітів регістра керування TCON (біти TCON.0 і TCON.3) табл. 6.4.

Якщо зазначені біти встановлені в одиницю, то активним є перехід з одиниці в нуль, при нульових значеннях цих бітів активним є низький рівень сигналів $\overline{INT0}$ або $\overline{INT1}$.

Оскільки в МК є п'ять запитів переривань, отже, є й п'ять команд LCALL з адресами відповідних підпрограм.

Причому код операції команди LCALL і адреси переходів, які відповідні кожному запиту, фіксовані для кожного джерела і формуються в контролері переривання відповідно табл. 5.1.

Таблиця 5.1 – Адреси підпрограм відповідних запитів переривання

Запит переривання	Адреса підпрограми
Зовнішній запит $\overline{INT0}$	0003h
Запит від таймера T/30	000Bh
Зовнішній запит $\overline{INT1}$	0013h
Таймер лічильник T/31	001Bh
Послідовний порт	0023h

Як можна побачити з табл. 5.1, інтервал між адресами підпрограм становить 8 осередків пам'яті програм.

Типову структуру підпрограми обслуговування переривання можна уявити так. На початку будь-якої підпрограми в стеку зберігається проміжна інформація основної програми (вміст акумулятора і регістрів загального призначення та ін.). Здійснюється це командами PUSH.

```
PUSH A          ; завантаження в стек вмісту акумулятора
PUSH PSW        ; завантаження в стек вмісту регістра стану програми
Тіло підпрограми
POP PSW         ; вивантаження з вершини стеку в регістр стану
POP A           ; вивантаження з вершини стеку в акумулятор
RETI            ; повернення на адресу перерваної програми.
```

Потім виконується сама підпрограма обслуговування переривання (наприклад, опитування стану датчиків у системі керування). Наприкінці підпрограми за допомогою команд POP відновлюється інформація в регістрах мікропроцесора. Після виконання останньої команди POP у вершині стеку буде перебувати адреса перерваної програми. Кількість команд PUSH і POP повинна бути однаковою і визначається самим розробником підпрограми, оскільки від цього залежить час обслуговування переривання. Будь-яка підпрограма обслуговування переривання завершується командою RETI. Завдяки цій команді з вершини стеку пересилається в лічильник команд (PC) адреса перерваної програми, крім того, нею скидається прапорець запиту переривання, відповідний цій підпрограми.

Аналіз структури підпрограми показує, що розмістити підпрограму обслуговування переривання у восьми осередках пам'яті програм далеко не завжди вдається. Тому, як правило, за адресою підпрограм розміщається команда безумовного переходу ljmp (адреса переходу), яка вказує адресу переходу на область пам'яті, де розміщена основна підпрограма обслуговування переривання.

Кожний із запитів переривання має режим дозволу на обслуговування або заборону з використанням відповідних бітів регістра керування перериваннями IE (табл. 5.2).

Установлення відповідного біта регістра IE (SETB IE.x) в одиницю (еквівалентно замиканню перемикача на рис. 5.1) дозволяє обслуговування цього запиту, скидання в нуль (еквівалентно розмиканню перемикача) забороняє обслуговування. Крім того, скидання біта IE.7 (CLR IE.7) забороняє обслуговування всіх запитів переривання.

З одночасною появою декількох запитів переривання в МК51 передбачено двоступеневий механізм пріоритетів переривань. Перший ступень механізму пріоритетів, забезпечується станами бітів регістра пріоритетів IP (табл. 5.3).

Встановлення відповідного біта регістра IP в одиницю (еквівалентно верхньому положенню перемикача PX) привласнює «високий» рівень, а скидання в нуль (еквівалентно нижньому положенню перемикача PX) – «низький».

Другий ступень реалізовано у послідовності опитування запитів переривання, яка встановлює фіксовану черговість обслуговування. При цьому запиті $\overline{\text{INT0}}$ присвоюється найвищий пріоритет, а послідовному порту (запити RI, TI) – найнижчий (рис. 5.1).

Таблиця 5.2 – Організація регістра дозволу переривань IE

Символічне позначення	Позиційне позначення	Назва і призначення
EA	IE.7	Біт дозволу переривань. Програмно скидається для заборони всіх переривань і встановлюється для дозволу переривань залежно від стану бітів IE.4- IE.0
	IE.6	Не використовується
	IE.5	Не використовується
ES	IE.4	Біт керування перериваннями послідовного порту. Програмно встановлюється (скидається) для дозволу (заборони) переривань за прапорцями TI і RL
ET1	IE.3	Біт дозволу переривання від таймера 1. Програмно встановлюється і скидається для дозволу (заборони) переривань від таймера (лічильника) 1
EX1	IE.2	Біт дозволу зовнішнього переривання 1. Програмно встановлюється (скидається) для дозволу (заборони) апаратного переривання $\overline{\text{INT1}}$
ET0	IE.1	Біт дозволу переривання від таймера 0. Програмно встановлюється (скидається) для дозволу (заборони) переривань від таймера (лічильника) 0
EX0	IE.0	Біт дозволу зовнішнього переривання 0. Програмно встановлюється (скидається) для дозволу (заборони) апаратного переривання $\overline{\text{INT0}}$

Таблиця 5.3 – Організація регістра пріоритетів переривань IP

Символічне позначення	Позиційне позначення	Назва і призначення
	IP.7	Не використовується
	IP.6	Не використовується
	IP.5	Не використовується
PS	IP.4	Біт керування пріоритетом переривань послідовного порту. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету за допомогою переривання послідовного порту
PT1	IP.3	Біт керування пріоритетом переривання від таймера 1. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету за допомогою переривання від таймера (лічильника)1
PX1	IP.2	Біт керування пріоритетом зовнішнього переривання 1. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету апаратному перериванню 1
PT0	IP.1	Біт керування пріоритетом переривання від таймера 0. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету за допомогою переривання від таймера (лічильника) 0
PX0	IP.0	Біт керування пріоритетом зовнішнього переривання 0. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету апаратному перериванню $\overline{\text{INT0}}$

Алгоритм функціонування системи обслуговування запитів переривання наведено на рис. 5.2. Цей алгоритм відображає послідовність дій мікропроцесора при обслуговуванні переривань. Програма з вищим пріоритетом може перервати виконання програми з нижчим пріоритетом. Таку організацію роботи системи переривань називають режимом вкладених переривань (рис. 5.3). З рисунка можна побачити, що основну програму на адресі 0007h переривається підпрограма, що відповідає запиту T1.

Потім виконання цієї підпрограми на адресі 1002h переривається підпрограма з більш високим пріоритетом, що відповідає запиту $\overline{\text{INT1}}$, яку на адресі 1303h переривається підпрограма з найвищим пріоритетом, відповідна до запиту $\overline{\text{INT0}}$.

Як можна побачити з рисунка, спочатку завершується виконання підпрограми, відповідної до запиту $\overline{\text{INT0}}$, потім мікропроцесор перейде до завершення підпрограми, що відповідає запиту T1, і тільки після завершення цієї підпрограми МП переходить до виконання основної програми.

На рис. 5.3 також зображено вміст стеку на момент виконання підпрограми, відповідної запиту $\overline{\text{INT0}}$, де можна побачити, що всі адреси перерваних підпрограм і адреса переривання основної програми запам'ятовуються в стеку.

Принцип організації переривань модернізованого мікроконтролера 1T8051 повністю збігається із системою переривання MCS-8051, але має розширене число запитів переривань від периферійних пристроїв. Залежно від моделі 1T8051 має до 18 джерел переривань з чотирма рівнями пріоритетів. У табл. 5.4 наведений перелік запитів на переривання від периферійних пристроїв для моделей 1T8051 (ML-51, ML-54, ML-56).

Із табл. 5.4 видно, що перелік запитів переривань значно збільшився. Повністю збереглися запити переривань від таймерів T0 і T1, і додатково введені переривання ще від трьох таймерів. Збереглися також переривання від зовнішніх пристроїв і послідовного порту, і також уведені в перелік переривання додаткових послідовних інтерфейсів. Також додатково введені переривання від аналого-цифрових перетворювачів (АЦП) і широтно-імпульсних модуляторів (ШІМ) для керування об’єктами, які мають безперервні характеристики.

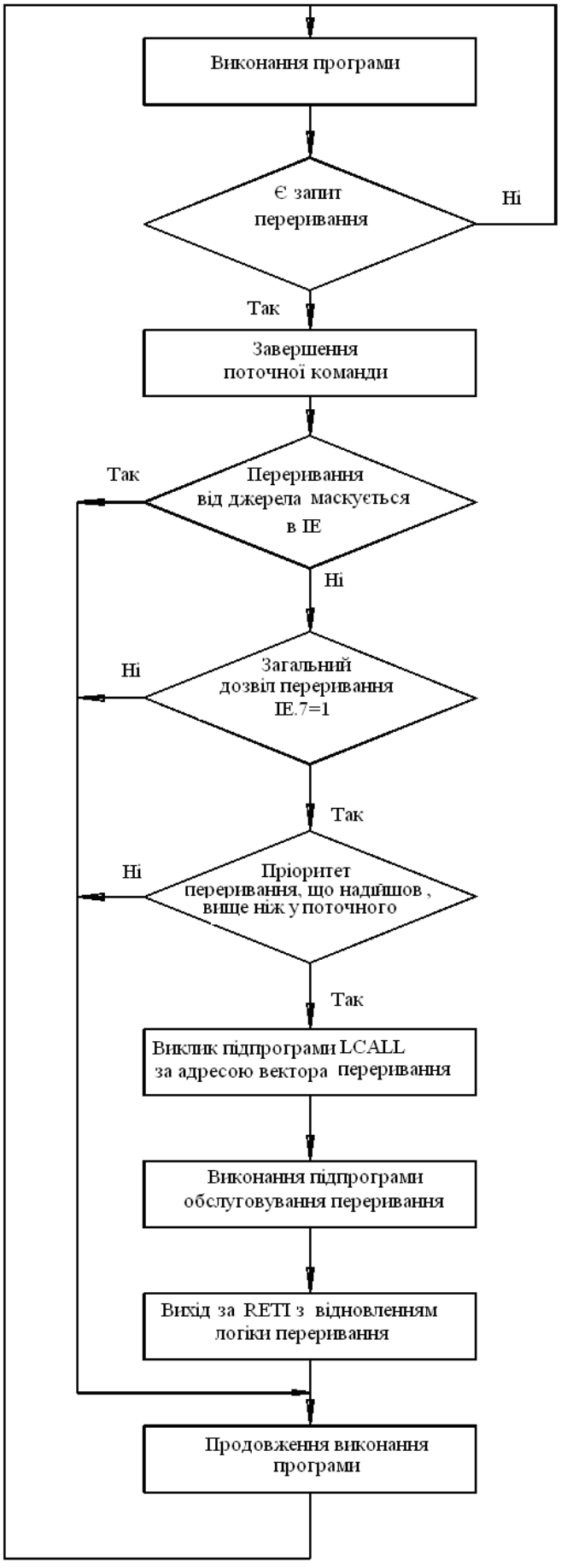


Рисунок 5.2 – Алгоритм обробки переривань у MCS-51

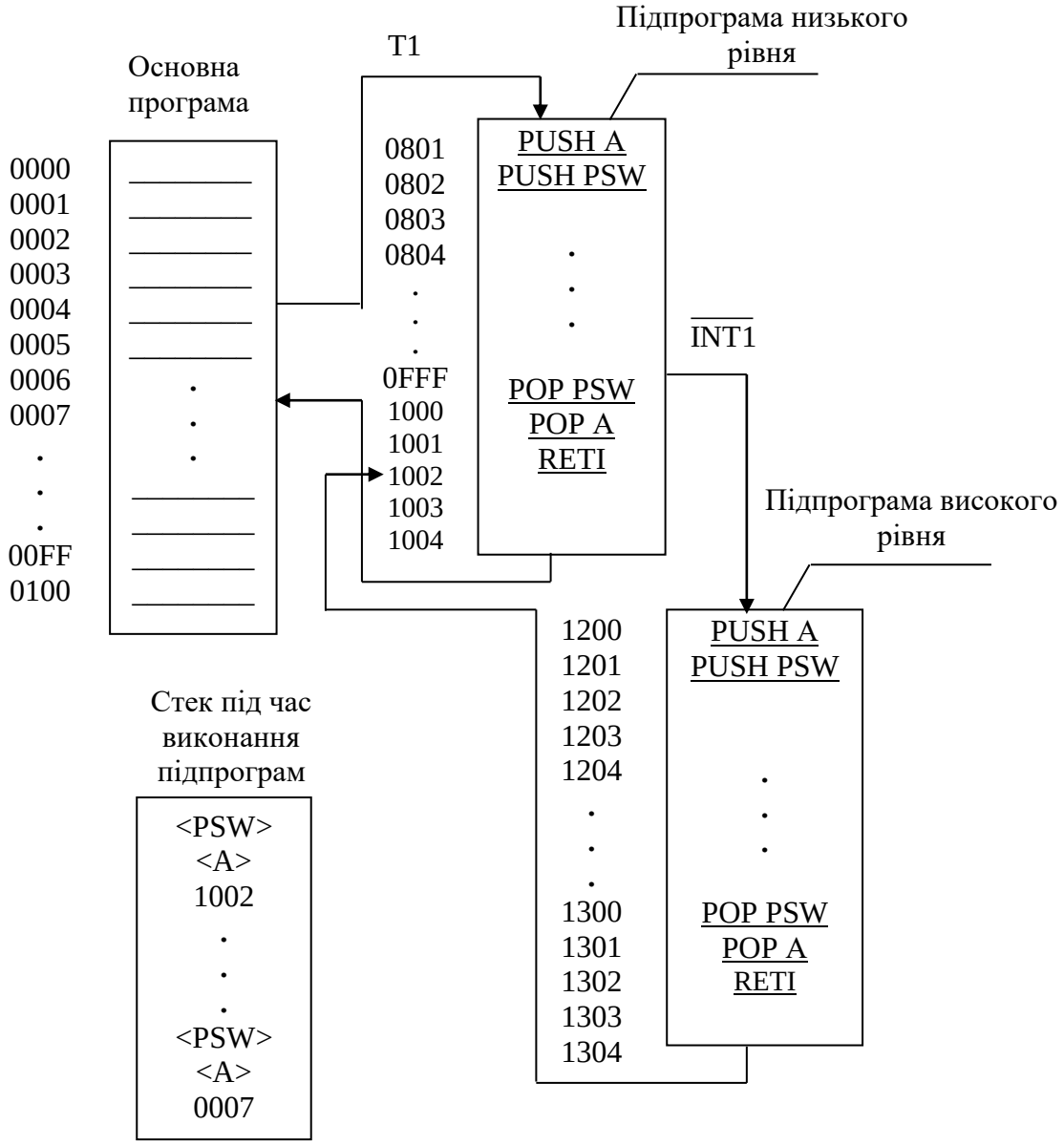


Рисунок 5.3. – Режим вкладених переривань

Таблиця 5.4 – Перелік запитів на переривання від периферійних пристроїв для моделей 1T8051

Source Запит переривання	Vektor Addess Адреса вектора	Vektor Number Номер вектора	Source Запит переривання	Vektor Addess Адреса вектора	Vektor Number Номер вектора
Reset Скидання	0000h	-	Serial port 1 interrupt Переривання послідовного порту 1	007Bh	15
External interrupt 0 Зовнішній запит переривання 0	0003h	0	Timer 3 overflow Переривання таймера 3	0083h	16
Timer 0 overflow Переривання таймера 0	000Bh	1	Self Wake-up Timer interrupt Переривання таймера пробудження	008Bh	17

External interrupt 1 Зовнішній запит переривання 1	0013h	2	CPU Hard Fault interrupt Вихід за межі пам’яті	0093h	18
Timer 1 overflow Переривання таймера 1	001Bh	3	SMC 0 interrupt Переривання послідовного порту 2	009Bh	19
Serial port 0 interrupt Переривання послідовного порту 0	0023h	4	PDMA 0 interrupt Прямий доступ до пам’яті 0	00A3h	20
Timer 2 event Подія від таймера 2	002Bh	5	PDMA 1 interrupt Прямий доступ до пам’яті 1	00ABh	21
P/W0 status/timer-out interrupt Переривання від лічильника I2C 0	0033h	6	SPI 1 interrupt Переривання від SPI інтерфейсу 1	00B3h	22
Pin interrupt Переривання від входів портів	003Bh	7	ACMP interrupt Переривання від компараторів	00BBh	23
Brown-out detection interrupt Переривання щодо зниження напруги	0043h	8	P/W 1 status/timer-out interrupt Переривання від лічильника I2C 1	00C3h	24
SPI 0 interrupt Переривання від SPI інтерфейсу 0	004Bh	9	PWM 123 interrupt Переривання від ШІМ 123	00CBh	25
WDT interrupt Переривання від сторожового таймера	0053h	10	Touch Key interrupt Переривання від контролю дотику	00D3h	26
ADC interrupt Переривання від АЦП	005Bh	11	SMC 1 interrupt Переривання послідовного порту 3	00DBh	27
Input capture interrupt Переривання щодо захвату	0063h	12	PDMA 2 interrupt Прямий доступ до пам’яті 2	00E3h	28
PWM 0 interrupt Переривання від ШІМ 0	006Bh	13	PDMA 3 interrupt Прямий доступ до пам’яті 3	00EBh	29
Fault Brake 0 interrupt Переривання від гальмування ШІМ	0073h	14	RTC interrupt Годинник реального часу	00F3h	30

Контрольні питання

1. Скільки сигналів переривань обробляється в MCS-51 і скільки в 1T8051?
2. Скільки сигналів переривань обробляється від зовнішніх пристроїв в МК-51?
3. Який регістр MCS-51 призначено для керування режимом переривань?
4. Який регістр MCS-51 призначено для керування пріоритетом запитів переривань?
5. Який сигнал дозволяє переривання від послідовного порту?
6. Який сигнал дозволяє переривання від таймера 0?
7. Який сигнал забороняє чи дозволяє обслуговування переривань від усіх пристроїв?
8. Який сигнал встановлює найвищий пріоритет запиту від таймера 0?
9. Який сигнал встановлює найвищий пріоритет запиту від послідовного порту?
10. Який сигнал встановлює найвищий пріоритет запиту від зовнішнього пристрою 1?

6. ОРГАНІЗАЦІЯ ТИМЧАСОВИХ ЗАТРИМОК У МІКРОПРОЦЕСОРНИХ СИСТЕМАХ

Реалізація закону керування в мікропроцесорних системах (МПС) здійснюється за допомогою впливу на об'єкт як апаратно (формування усіляких сигналів, наприклад, увімкнення реле, тиристора), так і програмно. При цьому сигнали керування формуються програмою, яка виконується мікропроцесором. Для прикладу розглянемо роботу звичайного автодорожнього світлофора. Час горіння зеленого й червоного світла – у межах 40 – 150 секунд залежно від інтенсивності руху, жовтого – 3 – 7 секунд. Якщо світлофор обладнано кнопкою для увімкнення зеленого світла або звуковим сигналом для незрячих, то зелене світло має горіти від 50 до 100 секунд. Із цього прикладу можна зробити висновок, що в системі керування світлофором необхідно формувати кілька різних тимчасових інтервалів. Вони залежать від результатів подій у системі. Натиснута пішоходом кнопка – завершився часовий інтервал горіння червоного або жовтого світла і т.п.

Мікроконтролер відрізняється від мікропроцесора тим, що апаратні функції керування реалізуються периферійними пристроями, розміщеними на тому самому кристалі, що й мікропроцесор (таймери, послідовний порт, АЦП, контролер переривань і т.п.). При цьому реалізація апаратного і програмного впливів на об'єкт здійснюється паралельно, як правило, цей час називають системним. Системний час створюється на основі деякого елементарного інтервалу, що формується тактовим генератором мікропроцесора, і являє собою сукупність різних тимчасових інтервалів, яку називають тимчасовою сіткою. Тимчасова сітка, як правило, формується таймерами.

При цьому в системі формуються сигнали початку $(t_н)$, закінчення $(t_о)$ інтервалу й, власне, величина тимчасового інтервалу $(t_з)$, рис. 6.1.

У базовій моделі MCS-51 для формування тимчасових інтервалів є два таймери, які являють собою регістри, що працюють у рахункових режимах.

Принцип роботи таймера розглянемо на прикладі посудини, яка наповнюється із крапельниці (рис.6.2).

Краплі починають надходити в посудину, якщо відкритий кран.

Під час наповнення посудини доверху відбувається відкриття дна, рідина виливається, посудина стає порожньою, дно закривається і система приходить у вихідний стан (роботу механізму відкривання й закривання дна посудини для спрощення не розглядаємо).

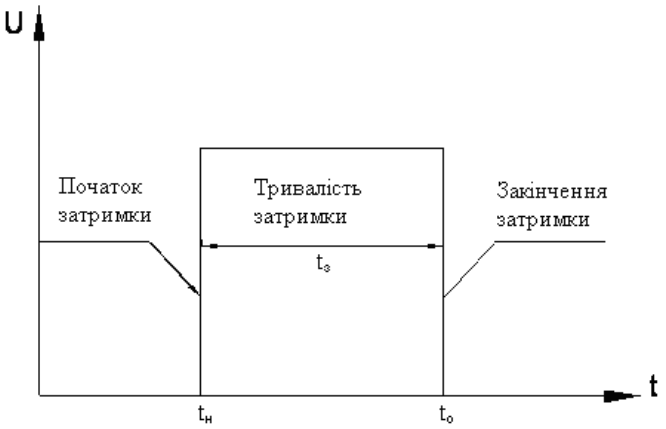


Рисунок 6.1 – Графічне зображення тимчасового інтервалу

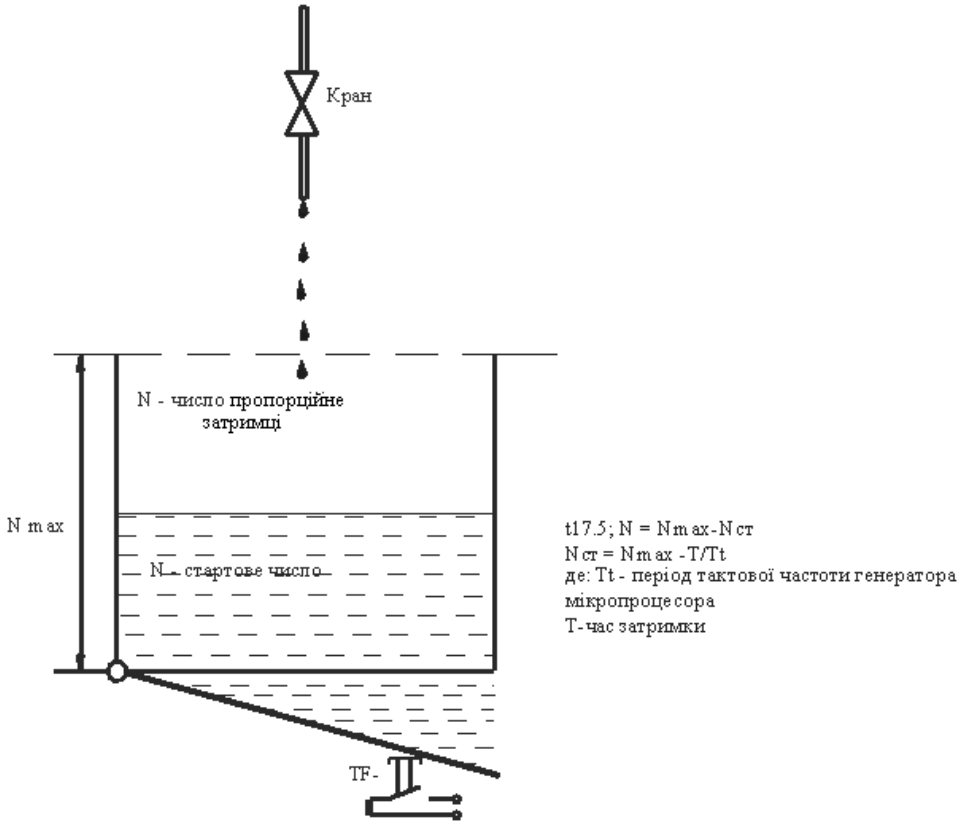


Рисунок 6.2 – Принцип роботи таймера

Якщо краплі починають капати в порожню посудину, то для її заповнення необхідно N_{max} крапель. Якщо ж попередньо в посудину налити N (стартове число) крапель і потім відкрити кран, то для її заповнення знадобиться менше число крапель. При постійній швидкості руху краплі час наповнення ємності доверху так само буде менший.

Таким чином, завдяки зміні стартового числа крапель можна змінювати час заповнення посудини доверху.

Отже, початку затримки відповідає відкриття крана для руху крапель, закінчення затримки фіксується замиканням контакту при заповненні посудини доверху і відкриванні її дна. Величина тимчасового інтервалу пропорційна числу крапель, яке необхідно накапати в посудину, щоб вона наповнилася доверху.

У базовій моделі MCS-51 використовуються два 16-розрядних лічильники, таймер 0 і таймер 1, кожний з яких працює незалежно в чотирьох режимах 0, 1, 2, 3.

Логічне функціонування таймера розглянуто на рис. 6.3, а графіки тимчасового аналізу роботи наведено на рис. 6.4.

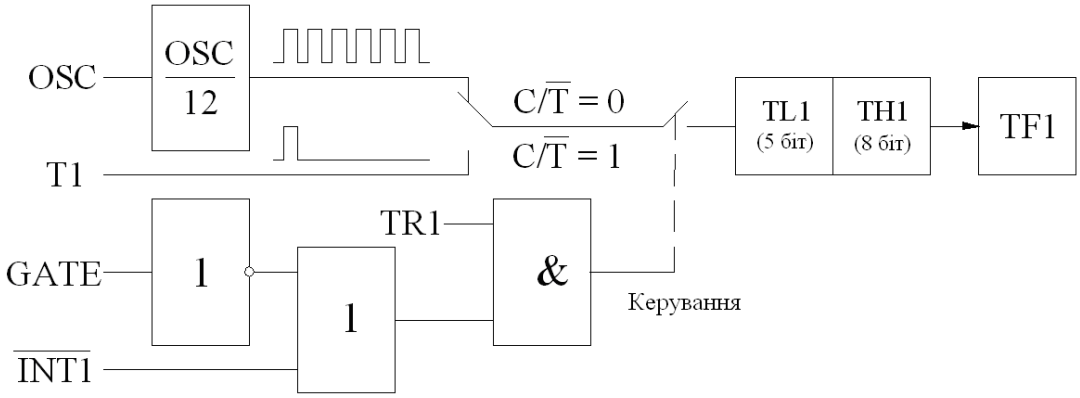


Рисунок 6.3 – Функціональна схема таймера в режимі 0

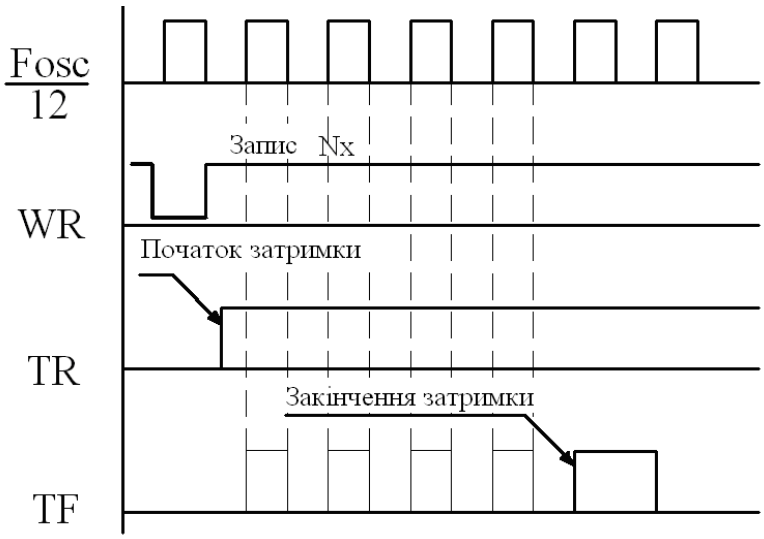


Рисунок 6.4 – Графік тимчасового аналізу роботи таймера в режимі 0

Кожен таймер може бути запрограмований, власне, у свій режим, у цьому випадку відбувається підрахунок імпульсів тактового генератора МП (підрахунок крапель), а також у режим лічильника зовнішніх подій (тобто є два види крапельниць).

Для керування організацією взаємодії таймерів із системою переривання й установлення режимів їх роботи використовується два регістри спеціальних функцій. Регістр режиму TMOD і регістр керування TCON.

Вибір режиму роботи кожного таймера здійснюється шляхом пересилання в регістр режиму TMOD керуючого слова, яке складається відповідно до табл. 6.1.

У режимі таймера вміст регістра лічильника збільшується на одиницю (інкрементується) у кожному машинному циклі МП, при цьому частота рахування складає $F_{osc}/12$ (при частоті кварцового резонатора 12 МГц частота рахування складає 1 МГц).

Таблиця 6.1 – Призначення бітів регістрів TMOD

Ім'я біта	Номер біта	Функція
GATE1	TMOD.7	Біт керування таймером 1. При GATE1=1 таймер 1 працює завжди при TR1=1. При GATE1=0 для роботи необхідно дотримуватися умови TR1=1 і $\overline{INT1}$=1
Л/$\overline{Т}$1	TMOD.6	Біт вибору типу подій для таймера 1. При Л$\overline{Т}$1 = 1 він працює як лічильник, при Л$\overline{Т}$1 = 0 – як таймер
М1.1	TMOD.5	Біт 1 визначення режиму роботи таймера 1
М1.0	TMOD.4	Біт 0 визначення режиму роботи таймера 1
GATE0	TMOD.3	Біт керування таймером 0. При GATE0 = 1 таймер 0 працює завжди при TR0=1. При GATE0 = 0 для роботи необхідна умова TR0=1 і $\overline{INT0}$ = 1
Л/$\overline{Т}$0	TMOD.2	Біт вибору типу подій при таймері 0. При Л/$\overline{Т}$0 = 1 він працює як лічильник, при Л/$\overline{Т}$0 = 0 – як таймер
М1.0	TMOD.1	Біт 1 визначення режиму роботи таймера 0
М0.0	TMOD.0	Біт 0 визначення режиму роботи таймера 0

У режимі лічильника інкрементування регістра лічильника (TH, TL) здійснюється при переході вхідного сигналу ($\overline{Т}$0 або $\overline{Т}$1) з «1» в «0».

Входи аналізуються у фазі S5P2 кожного машинного циклу. Інкрементування вмісту регістра лічильника здійснюється в циклі, що слідує за тим, у якому було виявлено перехід з «1» в «0». Таким чином, на розпізнавання переходу в цьому режимі необхідно два машинних цикли роботи мікропроцесора, частота обчислення в цьому випадку складе $F_{osc}/24$ (частота рахунку 500 кГц).

Вибір типу підраховуваних подій визначається станами бітів TMOD.2 і TMOD.6 регістра режиму TMOD. При одиничному стані цих бітів обчислювальні регістри працюють як лічильники, при нульовому стані – як таймери.

Особливістю регістра режиму TMOD є неможливість керування окремими бітами. Таким чином, для встановлення режиму роботи таймера або обчислювального регістра необхідно в регістр TMOD переслати керуюче слово, яке складається відповідно до вибраного режиму згідно з табл. 6.2.

Таблиця 6.2 – Структура керуючого слова регістра TMOD

TMOD.7	TMOD.6	TMOD.5	TMOD.4	TMOD.3	TMOD.2	TMOD.1	TMOD.0
GATE1	Л/$\overline{\text{Т}}$	M1	M0	GATE0	Л/$\overline{\text{Т}}$	M1	M0
Таймер 1					Таймер 0		

Біти TMOD.0 і TMOD.1 регістра визначають режим роботи таймера 0, а TMOD.4 і TMOD.5 – режими роботи таймера 1 (табл. 6.3).

Таблиця 6.3 – Вибір режимів роботи таймера

M1	M0	Режим роботи
0	0	**Режим 0**. TH_x як 8-розрядний таймер/лічильник. TL_x як 5-розрядний переддільник.
0	1	**Режим 1.** TH_x і TL_x з'єднанні послідовно і являють собою 16-розрядний таймер/лічильник.
1	0	**Режим 2.** TL_x являє собою 8-розрядний таймер/лічильник з автоперезавантаженням значення із TH_x.
1	1	**Режим 3.** TL0 як 8-розрядний таймер/лічильник, який керується бітами регістра TCON таймера 0, TH0 як 8-розрядний таймер/лічильник, який керується бітами регістра TCON таймера 1. Таймер 1 не працює.

Наприклад, таймер 0 потрібно встановити в режим лічильника зовнішніх подій, а таймер 1 – у режим таймера. Тоді біт TMOD.2 має бути встановлений в 1, а TMOD.6 в 0. У таймері 0 і в таймері 1 використовуємо конфігурацію 0, тобто біти TMOD.0, TMOD.1, TMOD.4 TMOD.5 перебувають у стані 0. Біти TMOD.3, TMOD.7 (сигнали GATE0 і GATE1) встановлюємо такими, що дорівнюють нулю.

Тоді для програмування режиму роботи таймерів у регістр TMOD занесемо керуюче слово

MOV TMOD, #04h (00000100B).

Стан бітів TMOD.7 і TMOD.3 впливає на те, як здійснюється керування початком рахунку регістрів TH і TL (відкривання крана в крапельниці). При одиничному стані зазначених бітів запуск таймера (початок обчислення) здійснюється бітами запуску таймера TR0 або TR1 (біти SETB TCON.6 або SETB TCON.4) регістра керування TCON (табл. 6.4.).

Таблиця 6.4 – Призначення бітів регістра TCON

Ім'я біта	Номер біта	Функція
TF1	TCON.7	Прапорець переповнення таймера 1. Установлюється при переході обчислювального регістра таймера зі стану FFh у стан 00h. Тобто обнулюється при передачі керування на процедуру обробки переривання.
TR1	TCON.6	Біт запуску таймера 1. При TR1=1 обчислення дозволено. Установлюється і скидається програмно.
TF0	TCON.5	Прапорець переповнення таймера 0. Установлюється при переході обчислювального регістра таймера зі стану FFh у стан 00h. Тобто обнулюється при передачі керування на процедуру обробки переривання.
TR0	TCON.4	Біт запуску таймера 0. При TR0=1 обчислення дозволено.
IE1	TCON.3	Прапорець запиту зовнішнього переривання за входом $\overline{\text{INT1}}$. Установлюється апаратно при виявленні запиту за входом $\overline{\text{INT1}}$. Скидання залежить від стану біта IT1.
IT1	TCON.2	Біт керування типом сигналу переривання $\overline{\text{INT1}}$.При IT1=1 – визначається за зрізом (перехід 1-0). Скидання IE1 здійснюється апаратно (автоматично). При IT1=0 – визначається при низькому рівні сигналу на вході $\overline{\text{INT1}}$.
IE0	TCON.1	Скидання IE1 здійснюється програмно.
IT0	TCON.0	Прапорець запиту переривання за входом $\overline{\text{INT0}}$ (аналогічно IE1). Біт керування типом сигналу переривання $\overline{\text{INT0}}$ (аналогічно IT1).

При нульовому стані бітів TMOD7 і TMOD.3 запуск таймера здійснюється попередньою установкою TR0 або TR1 і наявності сигналу $\overline{\text{INT0}}$ або $\overline{\text{INT1}}$, які є інверсними (активним нулем), у цьому випадку при переході сигналу на вході $\overline{\text{INT0}}$ або $\overline{\text{INT1}}$з 0 в 1 відбувається запуск таймера, а при переході з 1 в 0 обчислення зупиняється, рис. 6.5.

Таким чином, вміст лічильника прямо пропорційний тривалості імпульсу сигналу $\overline{\text{INT0}}$або $\overline{\text{INT1}}$, тобто в цьому режимі за допомогою таймера можна вимірювати тривалість імпульсів, які подаються на входи $\overline{\text{INT0}}$або $\overline{\text{INT1}}$.

Після того, як лічильник повністю заповнено (посудина наповнилася доверху) стан «усі одиниці» (FFFFh) переходить у стан «усі нулі» (0000h) (відкрилося дно посудини), формується сигнал закінчення обчислення TF0 (замкнувся контакт при відкриванні дна посудини) або TF1, які використовуються як запити на переривання від таймерів T0 або T1. Якщо вважати тривалість машинного циклу 1 мкс при частоті тактування T_t 12 МГц, а максимальне число імпульсів $N_{\max}$ (число крапель, що розміщуються в посудині), яке може порахувати

обчислювальний регістр, складає FFFFh (65536 десятковий еквівалент), то максимальний часовий інтервал на одному таймері (на початку обчислення посудина пуста) складе 65,536 мс.

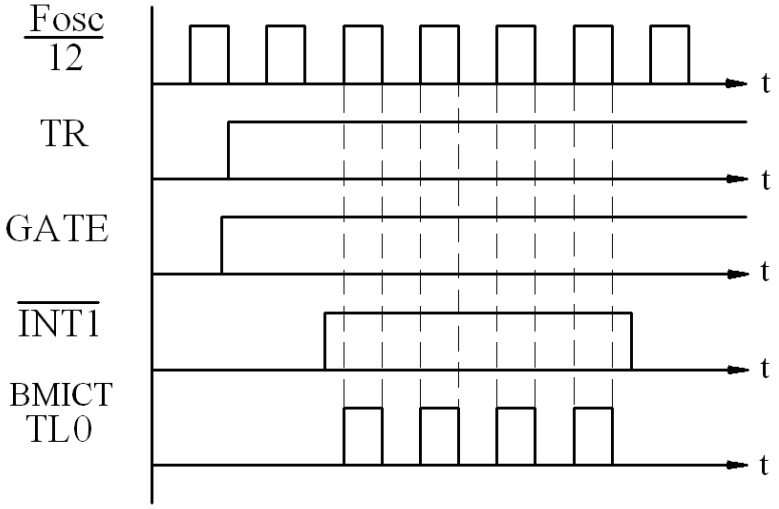


Рисунок 6.5 – Графік тимчасового аналізу роботи таймера в режимах 0 і 1

Регулювати величину затримки, що формується таймером, можна зміною стартового числа N_{CT}, яке попередньо записується в обчислювальний регістр (попереднє накапування рідини в посудину). Тоді тривалість затримки, оскільки

$$N_з = N_{\max} - N_{CT},$$

а $N_з = \frac{T_з}{T_t}$, де $N_з$ – число, яке пропорційне величині часовій затримки, як

$$N_{CT} = N_{\max} - \frac{T_з}{T_t},$$

$$\frac{T_з}{T_t} = N_{\max} - N_{CT},$$

$$T_з = \left(N_{max} - N_{CT}\right)T_t.$$

Наприклад, потрібно сформувати часову затримку тривалістю $T_з = 1$ мс, тоді

$$N_{CT} = N_{\max} - \frac{T_з}{T_t} = 65536 - \frac{1\cdot 10^{-3}}{1\cdot 10^{-6}} = 65536 - 1000 = 64536.$$

Це число попередньо необхідно записати в обчислювальний регістр таймера, а потім встановити біт запуску таймера (TR0 або TR1).

У режимі 0 обчислювальний регістр містить 13 розрядів, де регістр TH працює як 8-розрядний лічильник, а регістр TL – як 5-бітовий переддільник.

У режимі 1 обчислювальний регістр містить 16 розрядів, тобто TH має 8 розрядів і TL – 8 розрядів.

Режим 2 можна назвати режимом програмувального генератора, рис. 6.6.

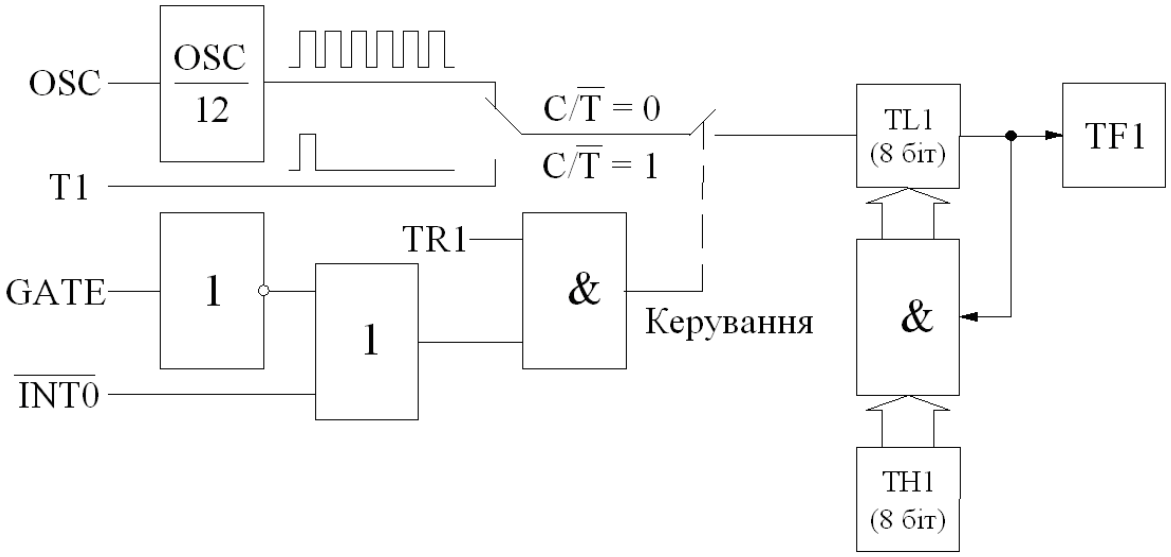


Рисунок 6.6 – Функціональна схема таймера в режимі 2

При цьому регістр TL працює як 8-розрядний обчислювальний регістр, а в TH завантажує число, яке визначає частоту імпульсів на виході таймера.

Після переповнення регістра TL регістр TH автоматично перезавантажується в регістр TL, при цьому вміст регістра TH зберігається. На графіку тимчасового аналізу (рис. 6.7) для прикладу показано, що в регістр TH попередньо занесене число EAH (250 у десятковому еквіваленті, тобто для заповнення посудини доверху необхідно всього 5 крапель) і через кожні 5 імпульсів на виході таймера буде формуватися імпульс, частота повторення якого в розглянутому прикладі

$$F = 1\ \text{МГц} / 5 = 200\ \text{кГц}.$$

Таким чином, змінюючи вміст регістра TH (MOV TH, #n), можна змінювати частоту імпульсів на виході обчислювального регістра.

У режимі 3 таймер 1 заблокований еквівалентно (TR1=0), а таймер 0 працює як два 8-розрядні обчислювальні регістри, причому регістр TL0 керується бітами таймера 0, а TH0 – бітами таймера 1 (рис. 6.8).

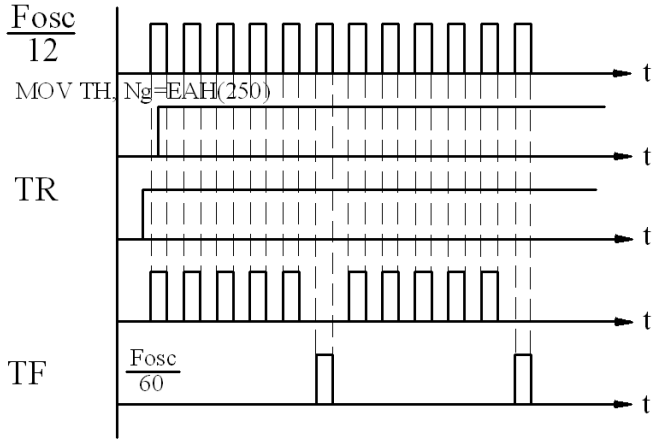


Рисунок 6.7 – Графік тимчасового аналізу роботи таймера в режимі 2

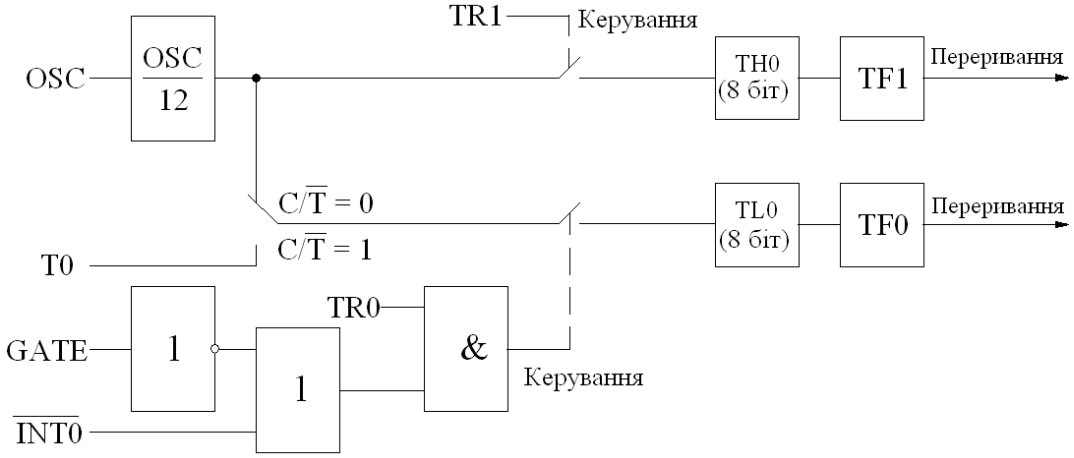


Рисунок 6.8 – Функціональна схема роботи таймера в режимі 3

Взаємодія таймера із системою переривання визначає регістр TCON, в якому установлена бітова адресація. Призначення бітів цього регістра наведено в табл. 6.4. Біти TCON.7 і TCON.5 фіксують переповнення таймера і використовуються як сигнали запиту переривання від таймера до мікропроцесора. Біти TCON.4 і TCON.6 керують запуском таймерів за командою SETB TCON.4 або SETB TCON.6. Біти TCON.3 і TCON.1 фіксують зовнішні сигнали запитів переривання. Біти TCON.2 і TCON.0 керують системою скидання сигналів $\overline{\text{INT0}}$ і $\overline{\text{INT1}}$. При одиничному стані біта TCON.2 аналіз сигналу $\overline{\text{INT1}}$ виконується за зрізом, а скидання автоматично (апаратно). При нульовому стані біта TCON.2 аналіз запиту сигналу $\overline{\text{INT1}}$ здійснюється за низьким рівнем сигналу на вході, а скидання сигналу $\overline{\text{INT1}}$ виконується програмно після закінчення підпрограми обслуговування переривання.

У режимі 0 обчислювальний регістр містить 13 розрядів, де регістр TH працює як 8-розрядний лічильник, а регістр TL – як 5-бітовий переддільник.

У режимі 1 обчислювальний регістр містить 16 розрядів, тобто вісім розрядів TH і вісім TL.

Режим 2 можна назвати режимом програмного генератора. При цьому регістр TL працює як 8-розрядний обчислювальний регістр, а в TH завантажується число, пропорційне до частоти імпульсів на виході таймера.

Після переповнення регістра TL вміст регістра TH автоматично перезавантажується в регістр TL, при цьому вміст регістра TH зберігається.

Таким чином, змінюючи вміст регістра TH (MOV TH, #n), можна змінювати частоту імпульсів на виході обчислювального регістра.

У режимі 3 (рис. 6.8) таймер 1 заблоковано еквівалентно (TR1=0), а таймер 0 працює як два 8-розрядні рахункові регістри, причому регістр TL0 керується бітами таймера 0, а THO – бітами таймера 1.

Розглянемо приклад організації часової затримки за секунду на одному із таймерів. Оскільки максимальна величина затримки в одному таймері складає 65 мс, то спочатку зробимо затримку 50 мс, а потім створимо програмний лічильник з коефіцієнтом 20 (50x20=1000 мс). Розрахуємо число, яке необхідно завантажити в таймер T0 для отримання затримки 50 мс. Згідно з наведеним раніше виразам $Ncm = 65536 - 50000 = 15536$ переведемо у шістнадцяткову систему Ncm = 3cb0h, та далі наступним етапом визначимо число, яке необхідно занести в регістр TMOD для забезпечення необхідного режиму. Аналіз свідчить, що це число дорівнює #01h.

```
 mov tmod,#01h; програмування режиму роботи таймера T0
 mov th0,#3ch ; заносимо в таймер 0 стартове число, яке
mov   tl0,#0b0h ; визначає величину затримки 50 мс,
setb tcon.4 ; запуск таймера (встановлення біта TR0 у регістрі TCON)
setb ie.7          ; дозвіл переривань
setb ie.1           ; дозвіл переривань від таймера 0
mov prescal,#20 ; коефіцієнт множення затримки 50 мс,
m0 pushA  (підпрограма переривань
mov th0,#3ch ; перезавантаження таймера (50 мс)
mov tl0,#0b0h ;
djnz prescal,m0; перевірка затримки одна секунда,
pop acc ; відновлюємо вміст акумулятора,
reti; вихід з підпрограми переривання.
 end.
```

Контрольні питання

1. Скільки таймерів містить MCS-51?
2. Скільки режимів роботи має таймер MCS-51?
3. Який сигнал MCS-51 формує початок затримки?
4. Який сигнал формує закінчення затримки?
5. Яке максимальне число можна завантажити в таймер?
6. Яку максимальну затримку можна організувати на одному таймері?
7. Наведіть вміст таймера в процесі роботи.
8. Який регістр забезпечує встановлення режимів роботи таймера?
9. Під дією яких сигналів змінюється вміст таймера?
10. Скільки розрядів використовується при роботі таймера в режимі 0?

7. ПРИКЛАДИ ВВЕДЕННЯ І ВИДЕННЯ ІНФОРМАЦІЇ В ДИСКРЕТНИХ СИСТЕМАХ

7.1. Особливості контролю дискретних датчиків на реальних об’єктах керування

У реальних системах керування майже завжди присутні всілякі контактні дискретні датчики, кнопки, органи блокування і т.ін. Схеми підключення датчиків до МК51 наведено на рис 7.1, *а*. Ввід інформації із цих датчиків має свої особливості.

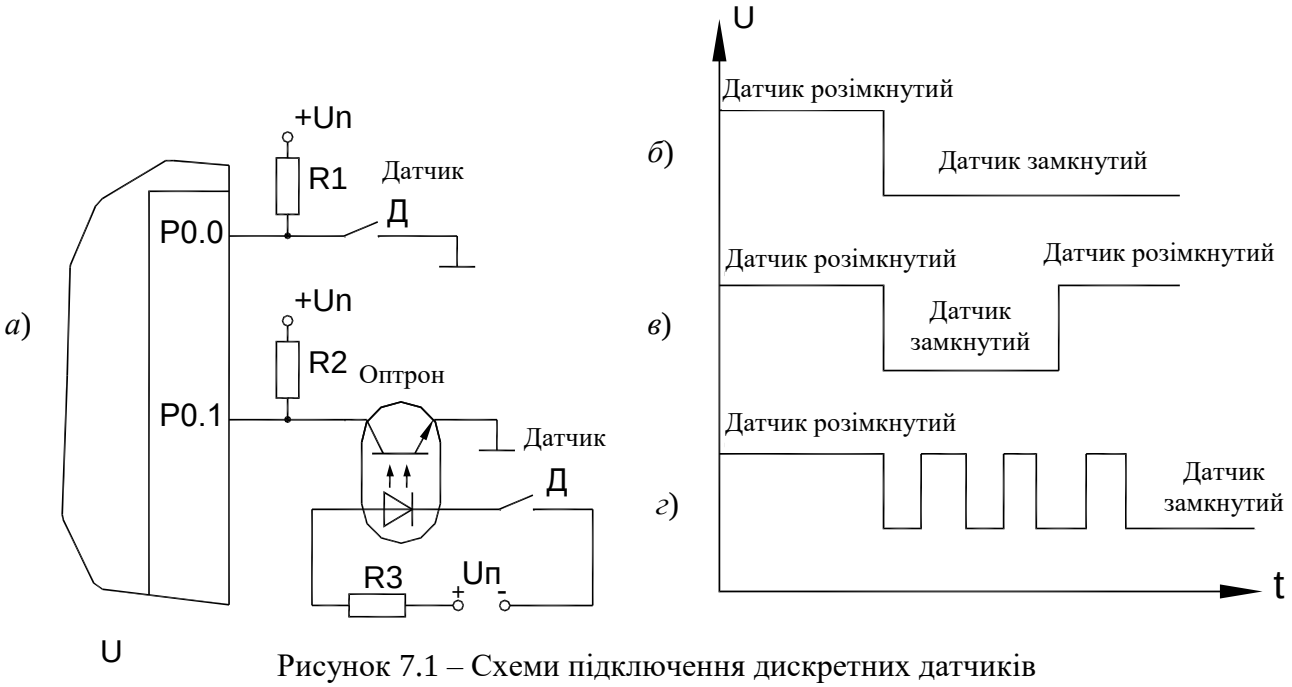


Рисунок 7.1 – Схеми підключення дискретних датчиків

По-перше, сигнал при спрацьовуванні може бути статичним, тобто при спрацьовуванні датчика формується тільки перехід з 1 в 0, як зображено на рис. 7.1, *б*. Але може формуватися й імпульсний сигнал (рис. 7.1, *в*). Крім того, при спрацьовуванні таких датчиків виникає деренчання контакту. При замиканні контакту виникає відскік контактів і в систему може бути введена випадкова послідовність нулів і одиниць (рис. 7.1, *г*).

Приномму датчик може бути приєднаний до входів T0 або T1 порту P3 або до входів портів P0…P3.

При приєднанні до входів T0 або T1 здійснюється контроль тривалості спрацьовування датчика з використанням таймерів T0 або T1 у режимі 0 або режимі 1.

Контроль стану датчика при приєднанні до входів портів P0…P2 і сигналі, як зображено на рис. 7.1, *а*, на виході датчика можна здійснити всього однією

командою, наприклад, JB P1.4, мітка; мітка переводить мікропроцесор на виконання підпрограми, яка виконує команди, що відповідають розімкненому стану контакту. Схему алгоритму опитування стану датчика наведено на рис. 7.2, *а*.

Якщо спостерігається деренчання контакту, то усунути його можна шляхом введення в схему алгоритму лічильника заданого числа станів, що збігаються (рис. 7.2, *б*), або шляхом введення тимчасової затримки (рис. 7.2, *в*).

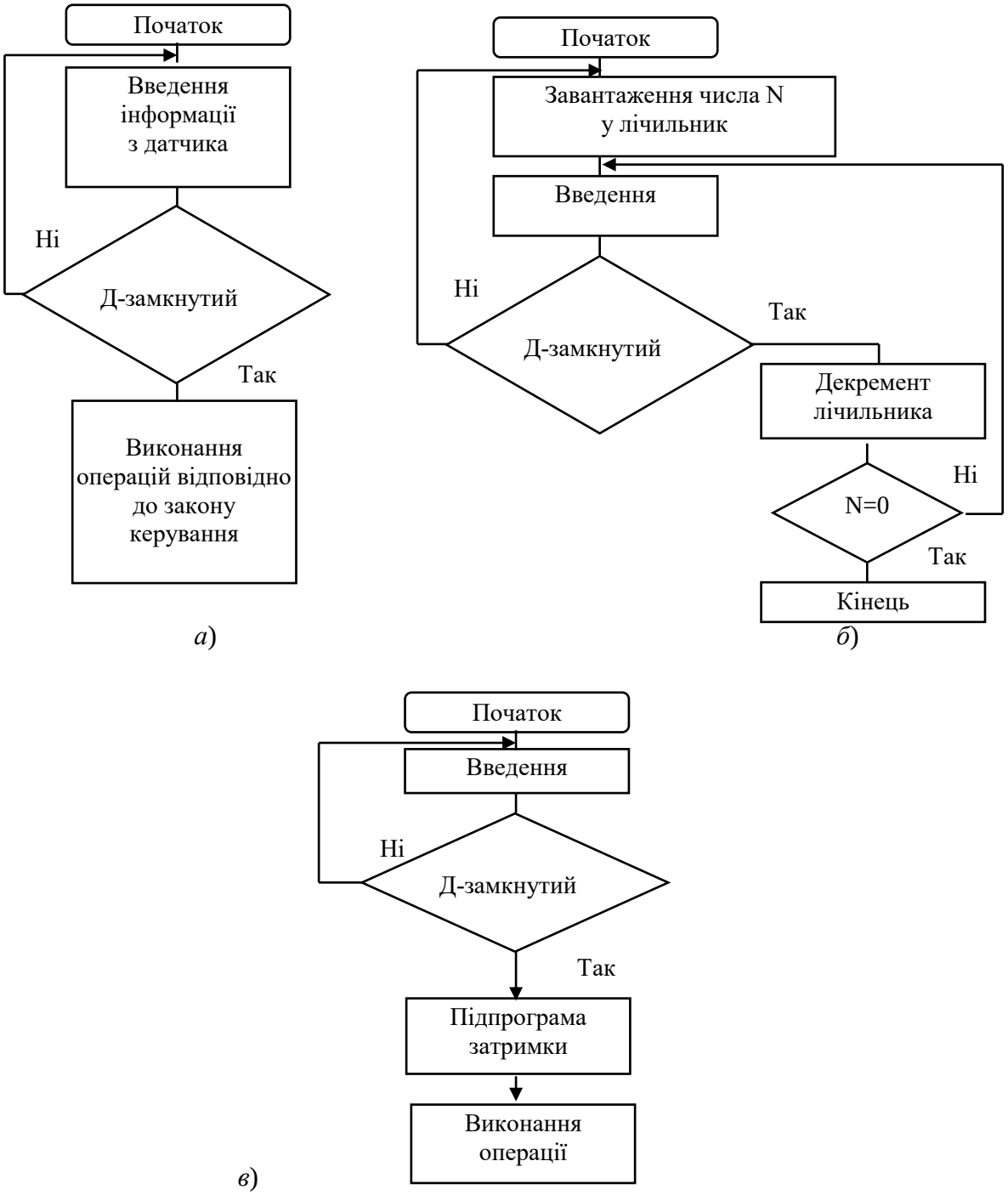


Рисунок 7.2 – Схеми алгоритму аналізу інформації з дискретного датчика за допомогою опитування датчика – *а*); використання лічильника – *б*); та відповідно часової затримки – *в*)

7.2. Приклад виконання програмної частини курсового проєкту

Як приклад об'єкта керування буде розглянута технологічна установка для отримання рідких сумішей шляхом змішування вихідних компонентів у необхідних пропорціях. Установки такого типу застосовуються для виготовлення будівельних сумішей, лакофарбових покриттів та ін. Технологічна схема для суміші, яка складається з двох компонентів, зображена на рис. 7.3.

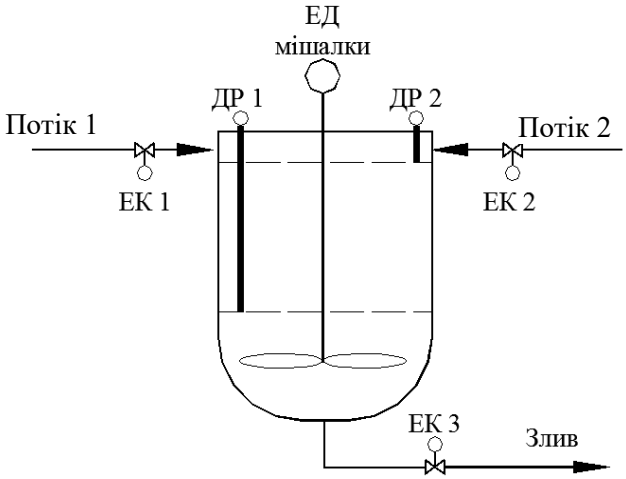


Рисунок 7.3 – Технологічна схема приготування рідких сумішей

У міксерну ємність вихідні компоненти подаються почергово через електроклапани ЕК1 для першого потоку і ЕК2 – для другого. Дозування компонентів виконується за допомогою датчиків рівня ДР1 і ДР2. Привід мішалки обладнаний електродвигуном ЕД. Для зливу вихідного компонента призначений клапан ЕК3. Процес приготування рідкої суміші та її злив відбувається за часом.

Узагальнений алгоритм технологічного процесу зводиться до наступного: за сигналом від кнопки «пуск» система вмикає електроклапан ЕК1та заливає компонент потоку 1 до моменту спрацьовування датчика ДР1, після чого електроклапан ЕК1 закривається; далі відкривається електроклапан ЕК2 та подається компонент потоку 2 до моменту спрацьовування датчика ДР2, після чого електроклапан ЕК2 закривається; вмикається ЕД мішалки на 4 хвилини; після вимикання ЕД мішалки відкривається зливний клапан ЕК3 на 1,5 хвилини.

Якщо в момент пуску ДР1 показує, що компонент потоку 1 є а іншого компонента немає, то необхідно зробити заливку потоку 2, і далі виконати перемішування і злив готового продукту. У разі початку роботи при повній ємності слід виконати перемішування та злив, відповідно до алгоритму, наведеному в попередньому абзаці.

Наведений приклад являє собою словесний опис об'єкта керування. Така робота завжди виконується на початку проектування.

При виконанні курсового проекту словесний опис об'єкта керування надається в завданні.

На другому етапі проектування розробляється алгоритм керування. Задачу логічного керування найкраще описувати алгоритмами роботи автоматів з пам'яттю. Такі алгоритми зручніше наводити у вигляді спрямованого графа-станів. У цьому разі вершини графа позначають фізичні стани, в яких може перебувати об'єкт керування або система, а на дугах вказуються логічні умови переходів між цими станами. При використанні моделі автомата «Міллі» на дугах також наводяться і вихідні сигнали.

При програмних методах реалізація алгоритму, поданого у вигляді графа, можливості використання вхідних і вихідних сигналів значно розширюються порівняно з апаратною реалізацією. Як вхідні змінні можуть використовуватися: дискретні і аналогові сигнали, числові змінні, програмні лічильники та таймери. У разі вихідних змінних можуть бути дискретні або аналогові сигнали, числові значення, програмні лічильники або таймери. З вхідними змінними можуть виконуватися арифметичні або логічні операції, а також операції порівняння (більше, менше, дорівнює). За допомогою таймерів – визначення кінця затримки. Вихідними можуть бути бітові та числові змінні, лічильники та таймери. З ними виконуються операції встановлення, скидання, присвоєння значень, інкремент, декремент або скидання в нуль. З таймерами – запуск та зупинки.

Приклад графа для наведеного словесного опису розглянутий на рис. 7.4. Ідентифікація вхідних та вихідних змінних, функцій керування часовими затримками наведена в табл. 7.1. У таблиці також показані адреси портів для підключення датчиків і виконавчих механізмів.

Таблиця 7.1 – Ідентифікація вхідних і вихідних змінних

Вхідні параметри об'єкта керування	Ім'я змінної	Ідентифікатор змінної	Порт МК (адреса)
Електродвигун мішалки, ЕД	electricstirrermotor	ESM	P2.0 (0A0h)
Електроклапани: ЕК1 (потік 1) ЕК2 (потік 2) ЕК 3 (злив)	 electrovalve 1, electrovalve2, electrovalve3	 EV1, EV2, EV3	 P2.1 (0A1h) P2.2 (0A2h) P2.3 (0A3h)
Кнопка «пуск»	start	start	P1.0 (90h)
Датчики рівня: ДР1 ДР2	 levelsensor 1 levelsensor2	 LS1 LS2	 P1.1 (91h) P1.2 (92h)

Затримки часу – змішування– 4 хв. – злив – 1.5 хв.	timedelay1	Tm1	 240 с 90 с

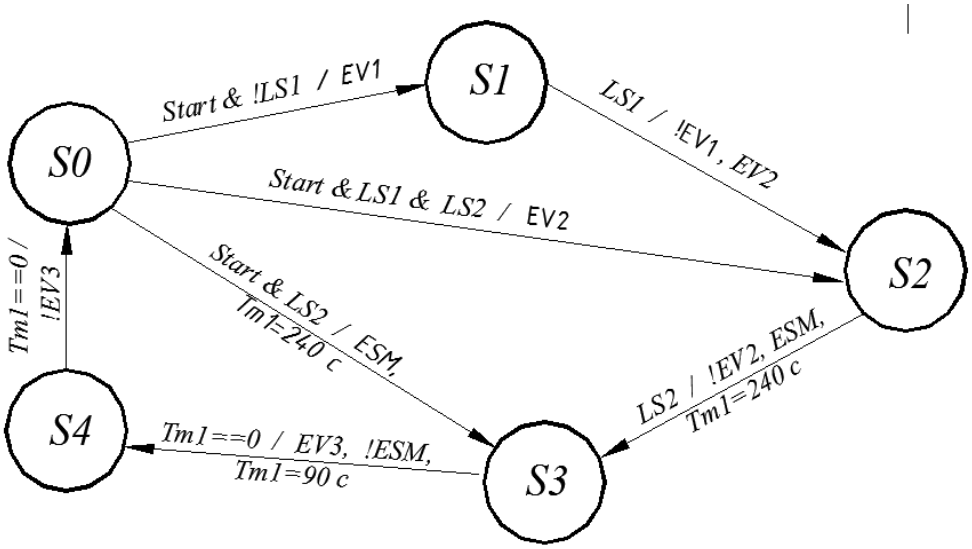


Рисунок 7.4 – Граф керування мішалкою

Опис станів

S0 – початковий стан. Очікування початку роботи. З цього стану можливо три дуги-переходи:

S0 – S1 – змішувач порожній і треба залити компонент 1;

S0 – S2 – в змішувачі є компонент 1 треба залити компонент 2;

S0 – S3 – ємність повна і потрібно виконати змішування компонентів. Умови цих переходів визначається сигналами від кнопки «пуск» і датчиків рівня ДР1 і ДР2 відповідно змінні Start, *LS1* і *LS2*. Вихідними сигналами є бітові змінні *EV1, EV2* – відкривання/закривання клапанів компонентів 1 і 2 та *ESM* – керування двигуном мішалки:

S1 – стан заповнення міксера компонентом 1;

S1 – S2 після спрацьовування датчика рівня ДР1 (*LS1*) закривається ЕК1 (*!EV1*) і відкривається ЕК2 (*EV2*) для потоку 2;

S2 – стан заповнення міксера компонентом 2;

S2 – S3 після спрацьовування датчика рівня ДР2 (*LS2*) відключаються ЕК2 (*!EV2*), вмикається ЕД мішалки (*ESM*) і відбувається формування першої затримки часу (*Tm1*=240);

S3 – стан змішування;

S3 – S4 після закінчення затримки 240 с (*Tm1*=0) відбувається перехід до зливу готової суміші. На цій дузі вимикається двигун мішалки (*!ESM*),

вмикається ЕК3 (*EV3*) і відбувається формування другої затримки часу (*Tm1*=90);

S4 – стан зливу готової суміші;

S4 – S0 після закінчення затримки часу 90 с (*Tm1*=0) вимикається ЕК3 (!*EV3*), і здійснюється перехід у початковий стан.

Програму, що реалізує алгоритм керування, записаний у вигляді графа, слід оформляти як підпрограму. Такий підхід дозволяє легко організувати багатозадачність роботи мікроконтролера, де кожен об'єкт керується від окремої підпрограми. Така підпрограма викликається або циклічно, або за допомогою лічильника часу виходячи з умов необхідної швидкодії системи.

Для зберігання значення стану графа формується окрема змінна, наприклад StateG1. Її значення модифікується при виконанні переходу з попереднього стану в наступний. Фактично це маркер поточного стану. Для наведеного прикладу його кодування зручно виконати у вигляді послідовності чисел: S0 – 00; S1 – 01; S2 – 02; S3 – 03; S4 – 04. У стартовій частині програми перед першим викликом підпрограми змінної StateG1 слід привласнити значення 00 (StateG1=00). Місце її розташування в ОЗУ мікроконтролера задається директивою асемблера EQU, наприклад StateG1 EQU 40h.

При викликані підпрограми її робота складається з двох етапів:

- пошук поточного стану;
- логічний аналіз умов переходів із поточного стану і формування вихідних змінних та функцій.

Знаходження поточного стану легко виконується за допомогою команди порівняння і переходу (cjne). Для цього в акумулятор переміщається значення номеру поточного стану StateG1 і по черзі порівнюється з кодами 00, 01, 02, 03, 04. Далі у знайденому стані виконується аналіз його дуг-переходів. Якщо ні одна із умов не виконується, то проводиться перехід в кінець підпрограми і далі вихід з неї.

У разі виконання умови переходу формуються вихідні змінні і функції, а також модифікується змінна StateG1 кодом стану, в який має відбутися перехід. Вихід з підпрограми завершує поточний цикл.

У наведеному прикладі розглядається однозадачний варіант з циклічним викликом підпрограми, яка обслуговує об'єктний граф. Крім цього, програма також містить в собі частину ініціалізації периферійних пристроїв і підпрограму обслуговування переривання від таймера.

```
        ;****************** ТЕКСТ ПРОГРАМИ *****************
;*****визначення адресів змінних
        Start       equ   90h   ; бітова адреса кнопки «пуск» (порт P1.0)
        LS1         equ   91h   ; бітова адреса ДР1 (порт P1.1),
        LS2         equ   92h   ; бітова адреса ДР2 (порт P1.2)
        ESM         equ   0A0h ; бітова адреса ЕД мішалки (порт P2.0)
        EV1         equ   0A1h ; бітова адреса ЕК1 (порт P2.1)
        EV2         equ   0A2h ; бітова адреса ЕК2 (порт P2.2)
        EV3         equ   0A3h ; бітова адреса ЕК3 (порт P2.3)
        StateG1     equ   40h   ; адреса осередка для зберігання коду стану
;*****додаткові змінні
        Tm1         equ   41h   ; адреса осередка для затримки часу
        Divider     equ   43h   ; адреса осередка перед дільника

        sjmp  M0    ; перехід на початок програми
              ORG 0bh    ; адреса вектора переривання таймера 0
        ljmp  Tim_0       ; перехід на підпрограму обслуговування
                          ; переривання таймера 0
              ORG 30h    ; початкова адреса трансляції програми
;***** обнуління розрядів, що керують зовнішніми механізмами
M0:   clr     ESM  ; P2.0 – ЕД мішалки
      clr     EV1  ; P2.1 – ЕК 1
      clr     EV2  ; P2.2 – ЕК 2
      clr     EV2  ; P2.3 – ЕК 3
;*****початкове встановлення регістрів керування МК і таймера 0
      mov   sp,#70h      ; встановлення початкової адреси стекової пам'яті
      mov   tmod,#01h  ; встановлення режиму 1 для таймера 0
      mov   th0,#3ch   } ; занесення в таймер 0 числа 3cb0h, яке визначає
      mov   tl0,#0b0h  } ; період відмітчика часу 50 мс
      setb tcon.4        ; запуск таймера (встановлення біта TR у регістрі TCON)
      setb ie.1          ; дозвіл переривань від таймера 0
      setb ie.7          ; загальний дозвіл переривань
;***** завдання значень змінним, які потребують початкову установку
      mov   Divider,#20 ; завантаження переддільника для затримки 1 с
      mov   stateG1, #00;присвоювання нуля змінній поточного номера стану
      ;*****ОСНОВНА ПРОГРАМА*****
;***** виклики підпрогам об'єктних графів
graph_program:
      lcall   graph_1      ; виклик підпрограми об'єктного графа 1
      ; виклики підпрограм для інших графів
      sjmp  graph_program; повернення на початок циклу
;***** підпрограма для graph_1
graph_1:
      mov   a,stateG1     ; пересилка в акумулятор номера поточного стану
;***** перевірка на знаходження в стані S0
```

```
        cjne   a,#0,S1_S2  ; якщо номер поточного стану не дорівнює 00,
                           ; то виконується перехід на перевірку номера стану S1
;*****аналіз дуги S0–S1 (Start & !LS1 / EV1)
        jnb Start,S0_S2 ; якщо логічна умова для першої дуги не виконується,
        jbLS1,S0_S2   ;  то треба перейти на аналіз другої дуги S0–S2
;***** формування сигналів керування
        setb   EV1         ; відкривання клапана ЕК1
        mov    stateG1,#01 ; змінній stateG1=01 (стан S1)
        sjmp   ExitG1      ; перехід на вихід із підпрограми
;*****;аналіз дуги S0–S2 (Start & LS1 & !LS2 / EV2)
S0_S2:jnb   Start,S0_S3
        jnb    LS1,S0_S3  ; перехід на аналіз дуги S0–S3
        jb     LS2,S0_S3
;***** формування сигналів керування
        setb   EV2         ; відкривання клапана ЕК2
        mov    stateG1,#01 ; змінній stateG1=01 (стан S1)
        sjmp   ExitG1      ; вихід із підпрограми
;***** аналіз дуги S0–S3 (Start & LS1 @ LS2 / ESM, Tm1=240 с)
S0_S3:jnb   Start,ExitG1 ; перевірка логічної умови та вихід із підпрограми,
        jnb    LS1,ExitG1 ; якщо вона не виконується
        jnb    LS2,ExitG1
;*****формування вихідних сигналів керування та змінних
        setb   ESM         ; ввімкнути ЕД мішалки
        mov    Tm1,#0F0h   ; 240 с
        mov    stateG1,#01 ; змінній stateG1=01
        sjmp   ExitG1      ; перехід на вихід із підпрограми
;***** перевірка на знаходження в стані S1
S1_S2:cjne  a,#01,S2_S3
;***** аналіз дуги S1–S2 (LS1 / !EV1, EV2)
        jnb    LS1, ExitG1; перехід на вихід із підпрограми
        clr    EV1         ; закриття ЕК1
;***** формування вихідних сигналів керування та змінних
        setb   EV2         ; закриття ЕК2
        mov    stateG1,#02 ; змінній stateG1=02
        sjmp   ExitG1      ; перехід на вихід із підпрограми
;***** перевірка на знаходження в стані S2
S2_S3: cjne a,#02,S3_S4
;***** аналіз дуги S2-S3 (LS2 / !EV2, EMS, Tm1=240 с)
        jnb    LS2, ExitG1; перехід на вихід із підпрограми
        clr    EV2         ; закриття ЕК1
        setb   ESM         ; відкриття ESM
;***** формування вихідних сигналів керування та змінних
        mov    Tm1,#240    ; запуск затримки часу 240 с (4 хв)
        mov    stateG1,#03 ; змінній stateG1=03
        sjmp   ExitG1      ; перехід на вихід із підпрограми
```

```
;***** перевірка на знаходження в стані S3
S3_S4: cjne a,#03,S4_S0
;***** аналіз дуги S3-S4 (Tm1==0 / EV3, !EMS, Tm1=90 с)
       mov   a,Tm1       ; перевірка закінчення затримки 240 с
       cjne  a,#0, ExitG1;
;***** формування вихідних сигналів керування та змінних
       clr   ESM         ; вимикання ЕД мішалки
       setb  EV3         ; відкриття клапана зливуЕК3
       mov   Tm1,#90     ; запуск затримки часу 90 с (1,5 хв)
       mov   stateG1,#04 ; змінній stateG1=04
       sjmp  ExitG1      ; перехід на вихід із підпрограми
;***** перевірка на знаходження в стані S3
S4_S0:cjne  a,#04,M00   ; перехід, якщо виник збій при виконанні програми
;***** аналіз дуги S4-S0 (Tm1==0 / !EV3)
       mov   a,Tm1       ; перевірка закінчення затримки часу 90 с
       cjne  a,#0, ExitG1;
;***** формування вихідних сигналів керування та змінних
       clr   EV3         ; закриття клапана зливу ЕК3
       mov   stateG1,#00 ; змінній stateG1=00
;***** вихід із підпрограми graph_1
ExitG1: ret
M00: ljmp M0             ; перехід на початок програми
;*********************************************************
Tim_0:      ; підпрограма обслуговування переривання від таймера 0
       push acc          ; зберігання вмісту акумулятора, PSW
       push PSW
       mov th0,#3ch      ; перезавантаження таймера 0
       mov tl0,#0b0h ;
       djnz  Divider,m1  ; формування міток часу 1 с
       mov   Divider,#20 ; перезавантаження перед дільника
; ***** декрементування затримок
       mov   a, Tm1      ; пересилання змінної Tm1 в акумулятор
       jz, m1            ; перевірка закінчення затримки, якщо ні
       decTm1            ; декремент змінної Tm1,
m1:    pop PSW           ; відновлення вміст PSW та акумулятора
       popacc
       reti              ; вихід із підпрограми переривання

       End               ; кінець програми
```

Курсовий проєкт виконується в середовищі розробки MCStudio. Для налагодження програми та демонстрації її роботи викладачу слід використовувати «Середовище оточення» пакета, в якому присутня картинка технологічного процесу. Стан датчиків та кнопок керування вводиться за

допомогою елемента «кнопка», а виконавчих пристроїв – за допомогою елемента «індикатор».

Принцип роботи контролера розроблений на основі схем введення інформації із сенсорів та виведення інформації з портів, що наведено в розділі 4.

7.3. Система керування кроковим двигуном

Кроковий двигун (КД) є традиційним виконавчим пристроєм багатьох електронних приладів і систем. Він являє собою безколекторний двигун постійного струму з фіксованими положеннями вала. КД призначений у першу чергу для точного позиціювання вала без застосування систем зворотного зв'язку. Обмотки КД є частиною статора. На роторі розташовано постійний магніт або у разі зі змінним магнітним опором – зубчастий блок з магнітом'якого матеріалу. Усі комутації залежать від зовнішніх схем. Відносно контролера, то відмінність між ними відсутня. На двигунах з постійними магнітами зазвичай є дві незалежні обмотки.

Крокові двигуни мають широкий діапазон кутових розділових здатностей. Більш громіздкі мотори, звичайно з постійними магнітами, обертаються на 90 ° за крок, у той час як прецизійні двигуни можуть мати роздільність 1,8 або 0,72 ° за крок.

Для правильного керування біполярним кроковим двигуном необхідна електрична схема, яка має виконувати функції старту, зупинки, реверсу й зміни швидкості. Кроковий двигун має послідовність цифрових перемикань під час руху. Обертове магнітне поле забезпечується відповідними перемиканнями напруг на обмотках. Далі за цим полем буде обертатися ротор, з'єднаний за допомогою редуктора з вихідним валом двигуна.

Потужність крокових двигунів знаходиться у межах від одиниць ватів до одного кіловата. Кроковий двигун має не менш двох положень стійкої рівноваги ротора в межах одного обертання. Напруга живлення обмоток керування кроковим двигуном являє собою послідовність однополярних або двополярних прямокутних імпульсів, що надходять від електронного комутатора (К). Результуючий кут відповідає числу перемикань комутатора, а частота обертання двигуна – частоті перемикань електронного комутатора.

Комутатор повинен забезпечувати зміну полярності струму в обмотках. Він являє собою два транзисторних мости, в діагональ яких приєднані статорні обмотки.

На рис. 7.5 зображено положення ротора крокового двигуна залежно від комутації обмоток. Послідовно, комутуючи струм в обмотках відповідно до діаграм, наведених на рис. 7.6, можна змусити обертатися вектор магнітного поля, а за ним і ротор у прямій або зворотній послідовності. При такому керуванні двигун має чотири стійких стани за один оберт ротора.

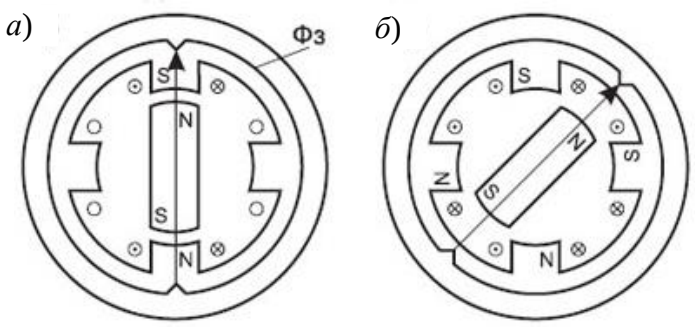


Рисунок 7.5 – Положення ротора при кроковому – *а*) та напівкроковому – *б*) керуванні КД

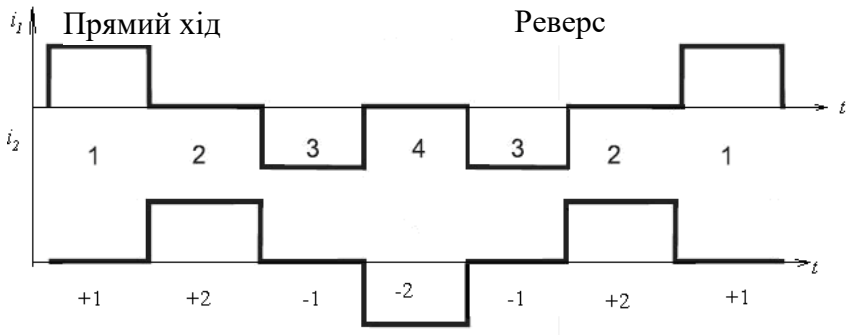


Рисунок 7.6 – Діаграма прямого і зворотного ходу крокового двигуна

Одночасне вмикання двох обмоток приводить до орієнтування вектора магнітного поля із кроком 45 ° щодо вертикальної осі. Таке керування називається напівкроковим. Напівкроковий режим дозволяє вдвічі підвищити точність позиціювання крокових двигунів. Послідовність комутації обмоток двигуна в напівкроковому режимі наведена на рис. 7.7. При комутації почергово вмикаються одна обмотка, а за нею дві разом і т.ін. При такому керуванні двигун має вісім стійких станів.

У крокових приводах електрична енергія перетворюється в механічну у вигляді обертального руху. Тому рівняння руху записується як рівняння рівноваги всіх моментів. Для більшості робочих механізмів і машин, що мають постійний момент інерції, цей вираз має такий вигляд:

$$M - M_C = J\frac{\omega - \omega_0}{\Delta t}, \qquad (7.1)$$

де M – обертаючий момент електродвигуна, Дж; M_C – статичний момент робочого механізму, зведений до вала двигуна, Дж; J – момент інерції системи, зведений до вала двигуна, Дж·с^2; α – кут повороту, рад.; t – час, с;

$$\frac{\omega-\omega_0}{\Delta t} \text{ – кутове прискорення, } 1/\text{с}^2; \tag{7.2}$$

$$\omega=\frac{\alpha-\alpha_0}{\Delta t} \text{ – кутова швидкість, } 1/\text{с}^2. \tag{7.3}$$

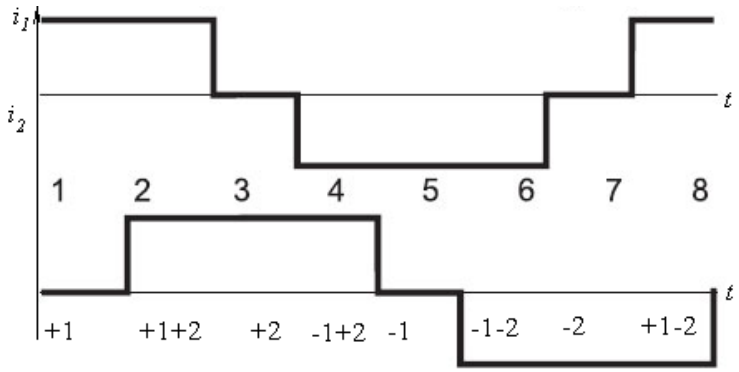


Рисунок 7.7 – Діаграми напівкрокового керування кроковим двигуном

Наявність моменту інерції сприяє тому, що для розгону або гальмування двигуна необхідний час, інакше вектор положення ротора почне відставати або випереджати вектор поля, і двигун втратить синхронізм, що призведе або до пропуску кроків, або зупинки двигуна.

Розглянемо приклади розрахунків розгону й гальмування крокового двигуна. Двигун з моментом 1500 Дж повинен обертати об’єкт керування зі швидкістю 60 об/хв. Момент інерції об’єкта – 500 Дж. Керування двигуном здійснюється в напівкроковому режимі. Необхідно визначити час виходу двигуна на задану швидкість, а також кількість кроків для розгону і тривалість кожного кроку. Розрахунки виконуються без урахування моменту опору обертанню.

Розв’язування. Оскільки початкова швидкість дорівнює нулю, то $\omega'=0$. Тоді вираз (7.1) можна записати так:

$$M=J\frac{\omega}{\Delta t}; \tag{7.4}$$

$$\Delta t=\omega\frac{J}{M}; \tag{7.5}$$

$$\omega=\frac{60\cdot 2\pi}{60}=6,28\ \frac{\text{рад}}{\text{с}}. \tag{7.6}$$

Звідки

$$\Delta t = 6,28\frac{500}{1500} = 2,1 \text{ с.}$$

Визначимо, скільки кроків має зробити двигун до повного розгону.

Перед початком розгону $\alpha = 0$, тоді

$$M = J\frac{2\alpha}{\Delta t^2}, \tag{7.7}$$

звідки

$$\alpha = \frac{\Delta t^2}{2}\cdot\frac{M}{J} = \frac{2,1^2}{2}\cdot\frac{1500}{500} = 6,62 \text{ рад.}$$

У напівкроковому режимі КД робить вісім кроків за один оберт. Тоді один крок – це поворот на кут

$$S = \frac{2\pi}{8} = \frac{\pi}{4} \text{ рад.}$$

Виходить, що до кінця розгону двигун зробить $\frac{6,62\cdot 4}{\pi} = 8,4$ кроків.

Отже, тривалість першого кроку згідно з виразом (7.7)

$$t_1 = \sqrt{2\cdot\alpha_1\cdot\frac{J}{M}} = \sqrt{2\cdot\frac{\pi}{4}\cdot\frac{500}{1500}} = 0,72 \text{ с,}$$

а тривалість другого кроку визначимо як різницю часу повороту на два кроки мінус поворот на один крок, а третього – як різницю часу повороту на три й два кроки тощо. Розрахунки значень зручніше за все виконувати в програмі EXEL (табл. 7.2).

$$t_2 = \sqrt{2\cdot 2\cdot\alpha_1\cdot\frac{J}{M}} - \sqrt{2\cdot\alpha_1\cdot\frac{J}{M}} = 0,3 \text{ с;}$$

$$t_3 = \sqrt{3\cdot 2\cdot\alpha_1\cdot\frac{J}{M}} - \sqrt{2\cdot 2\cdot\alpha_1\cdot\frac{J}{M}} = 0,23 \text{ с;}$$

. . . .

$$t_9 = \sqrt{9\cdot 2\cdot\alpha_1\cdot\frac{J}{M}} - \sqrt{8\cdot 2\cdot\alpha_1\cdot\frac{J}{M}} = 0,124 \text{ с.}$$

Тривалість кроку для швидкості обертання 60 об/хв обчислюємо так:

$$t_s = \frac{S}{\omega} = \frac{\pi}{4\cdot 6,28} = 0,125 \text{ с.}$$

Таблиця 7.2 – Таблиця тривалості імпульсів для розгону двигуна

№	1	2	3	4	5	6	7	8	9
T	0,72	0,3	0,23	0,19	0,17	0,15	0,14	0,13	0,12

Керування кроковим двигуном у мікропроцесорній системі складається із частин. Перша оформляється у вигляді підпрограми з передачею параметрів. Як параметри задається напрямок руху, кількість кроків. Підпрограма розраховує кількість кроків розгону і гальмування двигуна, а також кількість кроків з номінальною швидкістю. Алгоритм підпрограми, яка відображає керування двигуном згідно з прикладом розрахунків, наведений на рис. 7.8.

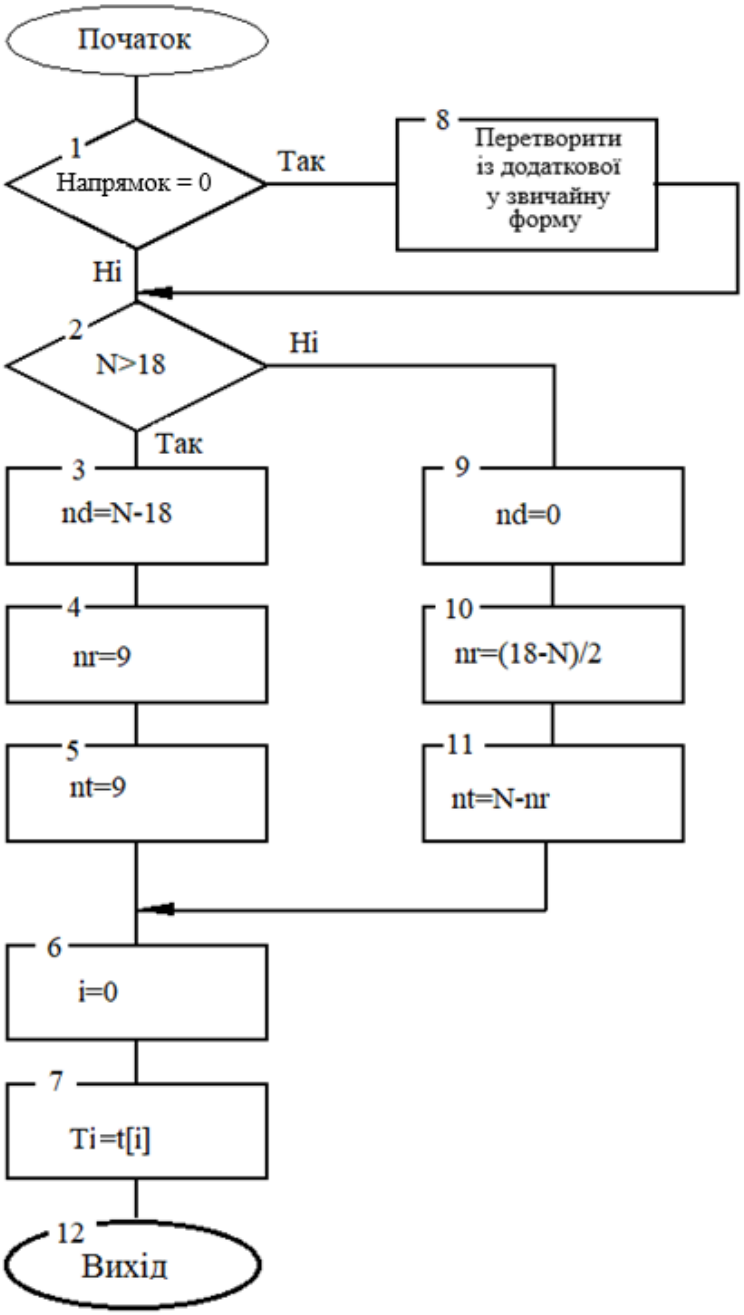


Рисунок 7.8 – Алгоритм підпрограми ініціалізації руху

Друга частина розміщається в підпрограмі обслуговування переривань від таймера. У ній за відмітником часу 10 мс формуються імпульси керування струмами в обмотках. Алгоритм відпрацьовує спочатку розгін, потім роботу на номінальній швидкості, а після цього гальмування двигуна. Для визначення тривалості імпульсу під час розгону і гальмування програма використовує табл. 7.2, записану в пам'яті програм. Алгоритм роботи цієї частини наведено на рис. 7.9. Значення на виході порту керування мостами формує підпрограма «крок», для цього вона використовує таблицю активізації виходів для кожного положення ротора. Алгоритм цієї підпрограми розглянутий на рис. 7.10.

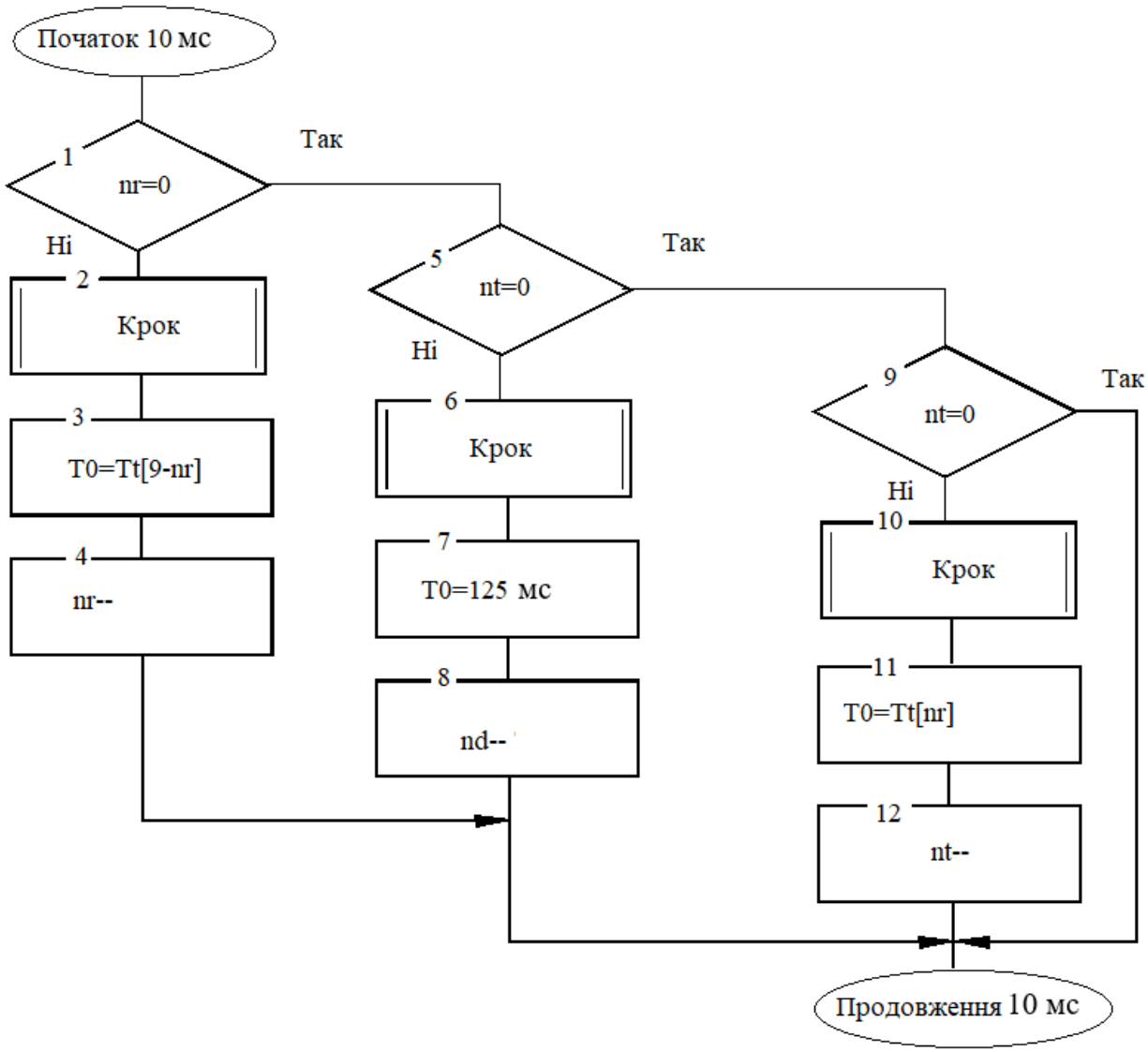


Рисунок 7.9 – Алгоритм відпрацьовування руху крокового двигуна

Схему підключення крокового двигуна до мікроконтролера зображено на рис. 7.11. Вона складається із двох транзисторних мостів, у діагональ яких приєднані статорні обмотки крокового двигуна ОВ1 і ОВ2. Керування мостами здійснюється чотирма молодшими бітами порту Р0. Коли подається одиниця на

P0.0, то відкривається транзистор Q3, а за ним і Q2, і через першу обмотку проходить струм позитивної полярності. Під час подачі одиниці на P0.1 відкривається транзистор Q4, а за ним і Q1, і через першу обмотку проходить струм негативної полярності. Другий міст працює аналогічно.

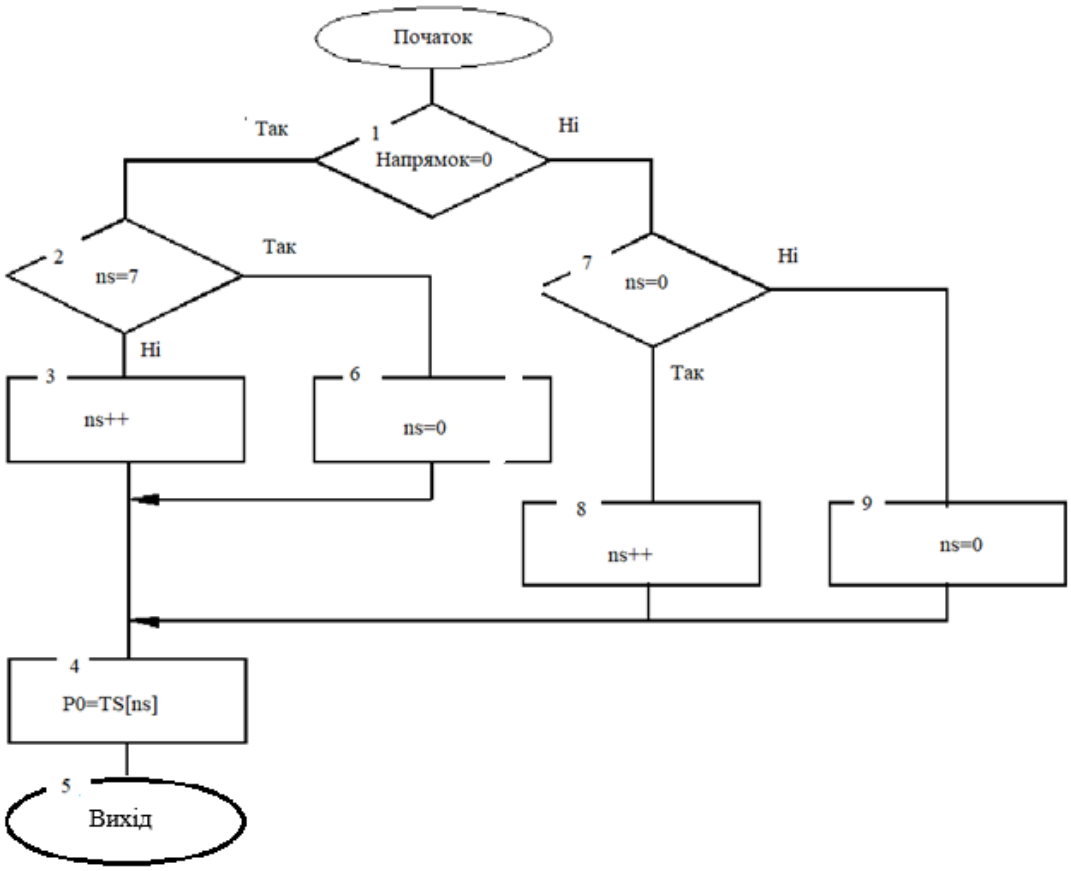


Рисунок 7.10 – Алгоритм підпрограми формування сигналів керування мостами

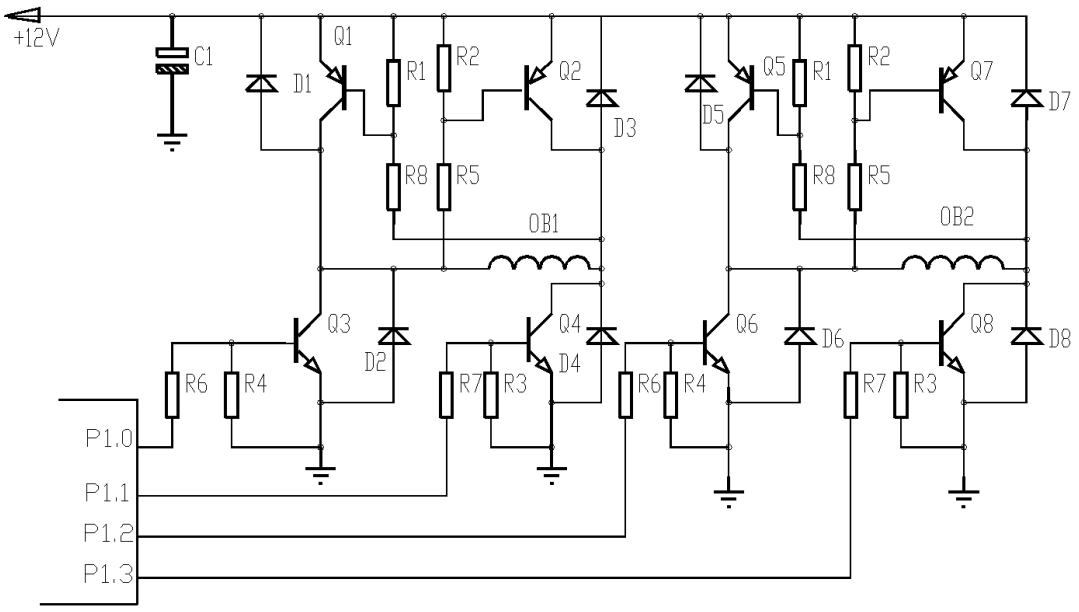


Рисунок 7.11 – Схема підключення крокового двигуна до мікроконтролера

Програма керування крокoвим двигуном наведена нижче. Підпрограма Step_ini формує вихідні дані для прямого і реверсивного керування крокoвим двигуном від 2 до 127 кроків з розгоном і гальмуванням.

```
N EQU 20h; Кількість кроків і напрямок
nr EQU 30h; Кількість кроків розгону
nd EQU 31h; Кількість кроків руху
nt EQU 32h; Кількість кроків гальмування
ns EQU 33h; поточне положення ротора
tim EQU 34h; тривалість імпульсу
sr EQU 35h; номер кроку розгону гальмування
fp EQU 08h; прапорець непарної кількості кроків розгону гальмування
```

Наведений фрагмент є описом призначення символьних змінних і вказівкою, де зберігаються значення проміжних і кінцевих змінних.

```
        ljmp main
        org 0bh; вектор оброблювача від таймера
        ljmp Tim_0
main:   mov TMOD,#01h; ініціалізація таймера
        mov TL0,#0f0h; налаштування відмітника на 10 мс
        mov TH0,#0d8h;
        setb tcon.4  ; запуск таймера
        setb ie.7   ; дозвіл переривань
        setb ie.1
        mov P1,#0
```

З периферійних пристроїв у програмі використовується таймер 0 і порт P1. При ініціалізації системи таймер 0 налаштовується на режим 1 з періодом переповнення 10 мс, у 0 установлюються порти виводу і початкові значення змінних.

```
.   .   .
        mov N,#8h
        lcall Step_ini
.   .   .
```

Вихідні дані, напрямки руху і кількість кроків передаються в підпрограму через осередка 20 h, символьне позначення N.

```
Step_ini:   ; підпрограма ініціалізації руху
        mov sr,#0 ; номер кроку розгону гальмування
        mov a,N
        jnb Acc.7,m1
        cpl a ; перетворення додаткового формату у звичайний
        inc a
        mov N,a
```

```
                setb 07h; зберігання прапорця негативного напрямку

m1:             clr c
                subb a,#18
                jc m3  ; якщо менше 18 на м3
                mov nd,A  ; кількість кроків рівномірного руху
                mov nr,#9 ; кількість кроків розгону
                mov nt,#9 ; кількість кроків гальмування
                ret
m3:             mov nd,#0 ; якщо <18, кількість кроків рівномірного руху дорівнює 0
                mov a,N
                anl a,#7fh
                clr c
                rrc a
                jnc m12
                setb fp; установлюємо позначення, що кроків гальмування менше,
                ніж розгону
 m12:           mov nt,a ; кількість кроків розгону
                mov a,N
                anl a,#7fh
                clr c
                subb a,nt
                mov nr,a ;кількість кроків гальмування
                        ret
```

Результатом виконання цієї підпрограми є кількість кроків розгону (nr), рівномірного ходу (nd) і гальмування (nt).

```
;**************************************
;оброблювач переривань від таймера
;**************************************
Tim_0:          push acc
                push PSW
                mov TL0,#0f0h
                mov TH0,#0d8h; наступна оцінка через 10 мс
                mov a,tim ; час кроку закінчено?
                jz raz
                dec tim ; ні
                ljmp ex
raz:            mov a,nr ; розгін закінчено?
                jz m4   ;так
                lcall step ; формуємо сигнали керування на мости
                clr c
                mov a,sr ; визначаємо покажчик на тривалість імпульсу
                mov DPTR,#razgon
                movc a,@a+DPTR
                mov tim,a ; установлює тривалість імпульсу
```

```
        inc sr  ; номер наступного кроку
        dec nr
        ljmp ex
m4:     mov a,nd; рівномірний рух закінчено?
        jz m5  ; так
        lcall step
        mov tim,#12 ; тривалість імпульсу фіксована
        dec nd   ; номер наступного кроку
        ljmp ex
m5:     mov a,nt  ; гальмування закінчено?
        jz ex
        jnb fp,m13
        dec sr
        clr fp
m13:    lcall step
        mov a,sr ; визначаємо покажчик на тривалість імпульсу
        dec a
        mov DPTR,#razgon
        movc a,@a+DPTR
        mov tim,a ; установлює тривалість імпульсу
        dec nt ; номер наступного кроку
        dec sr
ex:     pop PSW
        pop Acc
        reti
```

В оброблювач переривання програма заходить кожні 10 мс, змінна tim є програмним таймером тривалості імпульсу кроку. При розгоні і гальмуванні в неї заносяться значення з таблиці razgon, при рівномірному русі туди заноситься число 12, яке відповідає тривалості кроку 120 мс.

```
;**********************************
;Підпрограма формування значення сигналів для мостів
;**********************************
step:  mov DPTR,#faza
        mov a,ns
        jb 07h,m6
        cjne a,#7,m7
        mov ns,#0ffh
m7:     inc ns
m9:     mov a,ns
        movc a,@a+DPTR
        mov P1,a
        ret
```

```
m6:   cjne a,#0,m8
      mov ns,#8
m8:   dec ns
      ljmp m9
faza:
      db 01h,05h,04h,06h,02h,0Ah,08h,09h ; значення бітів порту P0 для 8
                                         ; положень ротора
      razgon:
      db 72,30,23,19,17,15,14,13,12; значення тривалості кроку при розгоні
      END
```

Підпрограма step формує сигнали керування мостами. Змінна ns зберігає положення ротора. Оскільки крокові двигуни використовуються без датчиків зворотного зв'язку за положенням, то алгоритм задання початкової системи координат виглядає так. У системі, як правило, є дискретний кінцевий датчик початку координат. При подачі живлення, якщо датчик не спрацював, кроковий двигун включається на рух до початку координат на мінімальній швидкості для перших восьми кроків. Таким чином, тривалість імпульсу дорівнює першому значенню з таблиці razgon. Це дозволяє засинхронізувати положення вектора поля і ротора. При спрацьовуванні кінцевого вимикача процес зупиняється і фіксується початок координат. Мікропроцесорна система керування дозволяє максимально використовувати можливості крокового приводу. Вона забезпечує гнучке керування, особливо в перехідних режимах, зберігаючи точність позиціювання і виконуючи будь-які програми відпрацьовування напрямку і швидкості ходу.

Контрольні питання

1. Чим відрізняються сигнали контролю стану дискретних датчиків?
2. Поясніть алгоритм деренчання контактних датчиків?
3. Поясніть алгоритм контролю часових затримок?
4. З якою метою введення сигналів з датчиків здійснюється через оптрони?
5. Поясніть принцип роботи крокового двигуна?
6. Назвіть режими роботи крокових двигунів?
7. Поясніть алгоритм формування сигналів руху крокового двигуна?

8. ОРГАНІЗАЦІЯ ПЕРЕДАЧІ ІНФОРМАЦІЇ В ПОСЛІДОВНОМУ ФОРМАТІ В МІКРОПРОЦЕСОРНИХ СИСТЕМАХ (МПС)

Сучасні МПС мають багаторівневу структуру, де виникає завдання обміну інформацією не тільки між різними рівнями, але й між окремими мікроконтролерами на одному рівні. Крім того, у системах, як правило, передбачається зв'язок з персональним комп'ютером (ПК), який виконує реєстрацію (моніторинг) основних параметрів процесу керування. При цьому всі мікроконтролери об'єднані в локальну керуючо-обчислювальну мережу, через яку й здійснюється обмін інформацією між мікроконтролерами.

8.1. Принципи передачі інформації з послідовного каналу зв'язку

Обмін даними між пристроями у мікроконтролері відбувається по паралельній шині. Вона складається із провідників, які з'єднують пристрої. При цьому по кожному провіднику передається окремий біт, за таким способом шина даних являє собою вісім провідників, на кожний з яких подається від пристроїв значення бітів від D0 до D7.

Такий спосіб обміну виправдовує себе, коли обмін відбувається на невеликі відстані, наприклад, усередині одного пристрою або мікросхеми. Якщо є необхідність передавати інформацію між пристроями на значні відстані, такий спосіб неефективний через значну вартість багатопровідникових шин. Тому обмін з віддаленими пристроями, як правило, здійснюється за дво- або трипровідниковою схемою. При цьому використовується так званий спосіб тимчасового ущільнення каналу. Його сутність розглянуто на рис. 8.1.

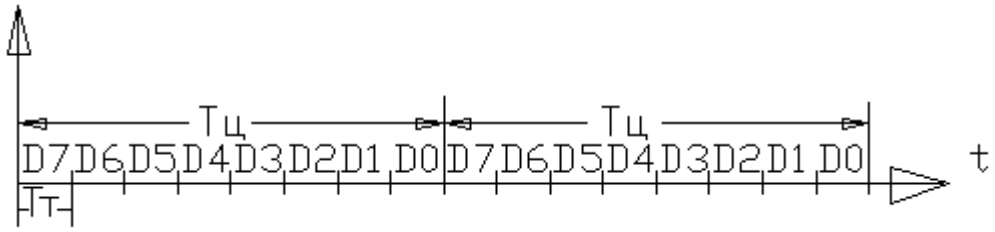


Рисунок 8.1 – Часове ущільнення каналу зв'язку

У цьому разі інформація передається по двом проводам – по сигнальному відносно загального.

На сигнальний провід подається послідовно в часі стан розрядів шини даних, починаючи з D0 по D7. Значення кожного розряду втримується на сигнальному проводі протягом часу Тт. Таким чином, відбувається перетворення даних з паралельного формату в послідовний, що здійснюється за допомогою

зсувного регістра й тактового генератора. Структурна схема цього пристрою зображена на рис. 8.2.

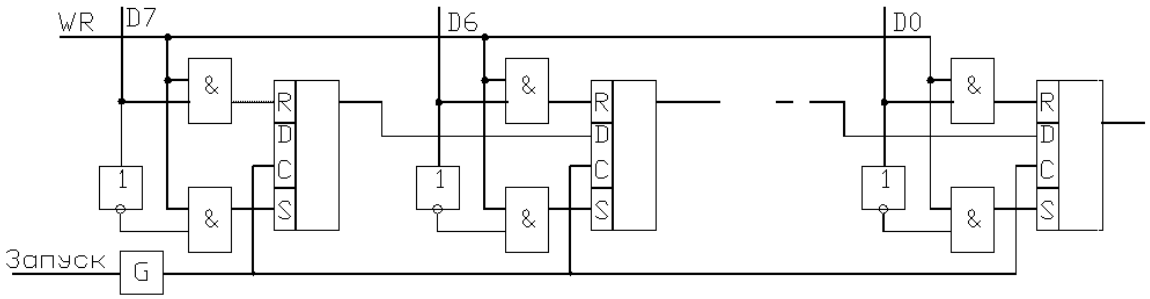


Рисунок 8.2 – Схема зсувного регістра з паралельним занесенням даних

Зсувний регістр виконується на D-тригерах, схем занесення інформації в тригери (D0-D7) (паралельний формат) і генератора синхроімпульсів G.

Після занесення інформації в тригери запускається генератор. На останньому тригері послідовно в часі формується значення D0, на другому – D1 і т.ін.

Час утримання значень бітів на виході визначається періодом генератора.

При такому способі передачі виникають такі проблеми: синхронізація прийнятих даних за тактами, байтами і повідомленнями. Синхронізація за тактами передбачає, що фіксація інформації на приймальній стороні відбувається в моменти часу, коли на виході лінії зв'язку встановлюється значення переданого біта (після закінчення перехідного процесу). Найбільш простим способом такої синхронізації є використання ще одного сигнального провідника для синхронізації приймального пристрою. Структуру такої передачі зображено на рис. 8.3.

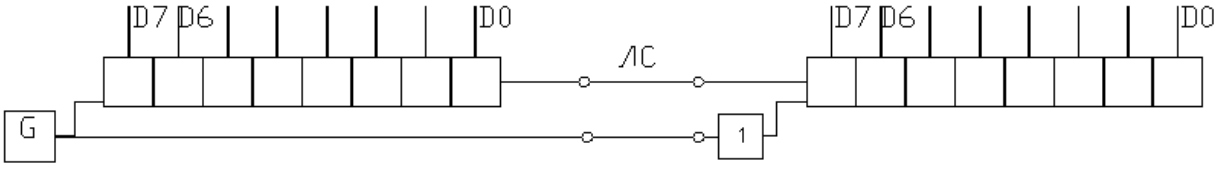


Рисунок 8.3 – Передача інформації з додатковим сигналом синхронізації

Такий спосіб дозволяє передавати інформацію з максимальними швидкостями. Однак, наявність додаткового провідника, обмежує область застосування цього способу передачі. Він використовується частіше для обміну інформацією з периферійними пристроями в межах одного апарата, наприклад, для зв'язку з АЦП, ЦАП, ЖК – індикаторами, флеш-пам'яттю і т.ін.

Графік часового аналізу цього способу передачі інформації зображений на рис. 8.4.

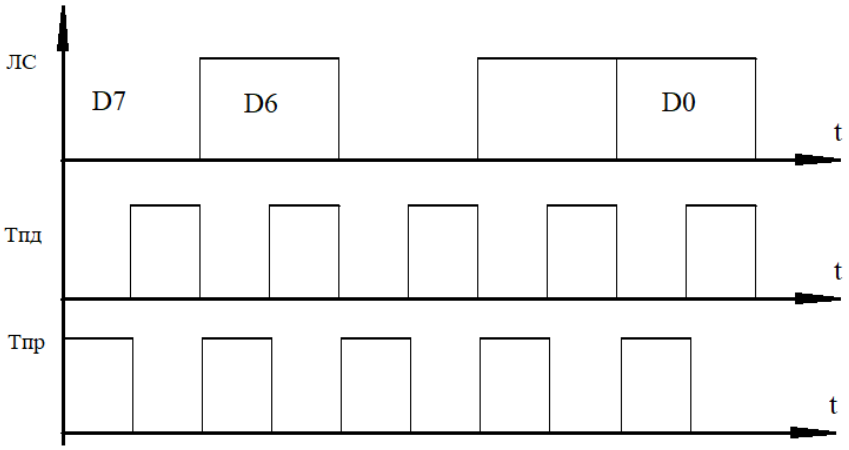


Рисунок 8.4 – Графік часового аналізу обміну інформацією при використанні додаткового тактового сигналу

Інвертування тактового сигналу приймача приводить до того, що підтвердження значення в ЛС відбувається у середині бітового інтервалу при значенні сигналу, що вже встановилося.

У більш досконалих системах приймач синхронізується від власного генератора, але, тому що генератори розділені, їх необхідно синхронізувати. Для цього в схему вводиться додатковий пристрій, який синхронізує фазу приймального генератора за фронтами і зрізами, які виділяються із прийнятого сигналу. Структурна схема такої системи передачі наведена на рис. 8.5.

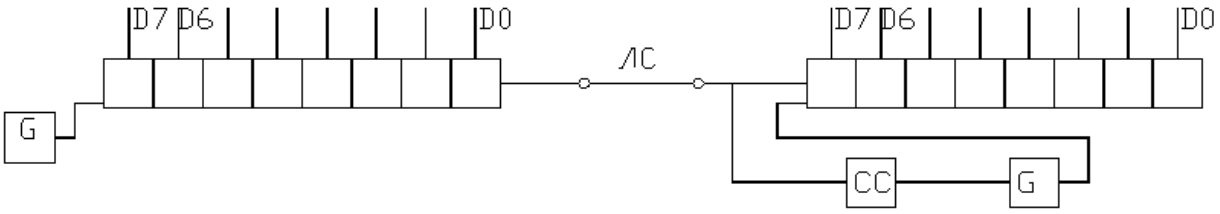


Рисунок 8.5 – Обмін інформацією з незалежним тактуванням

8.2. Послідовний інтерфейс у мікроконтролері MCS51

Для обміну інформацією із зовнішніми пристроями в послідовному форматі до складу мікроконтролера входить послідовний інтерфейс, або універсальний синхронно–асинхронний приймач-передавач (УСАПП). До його складу належать: зсувні приймальні і передавальні регістри, буфери приймача і передавача, а також схеми тактування передавача й синхронізації приймача.

Наявність додаткових буферів дозволяє суміщати операції зчитування вже прийнятого байта з прийманням наступного.

Послідовний інтерфейс може працювати за чотирма режимами і налаштовуватися на відповідний за допомогою запису керуючого слова в регістр SCON. Призначення бітів регістра керування наведено в табл. 8.1.

Для обміну інформацією із зовнішніми пристроями УСАПП використовують сигнали Rxd і Txd – альтернативне призначення бітів порту P3: розряди P3.0 і P3.1 відповідно. Залежно від режиму роботи УСАПП призначення цих розрядів різне.

Послідовний інтерфейс має символічне ім'я SBUF. Для пересилання байта інформації необхідно передати цей байт у буфер прийомо-передавача послідовного інтерфейсу командою MOV SBUF, A; (Передача вмісту акумулятора в буфер послідовного обміну.). Після цього байт автоматично пересилається в зсувний регістр передавача і відразу починає побітно виштовхувати байт в лінію зв'язку. Після передачі останнього біта встановлюється в одиницю тригер TI «запит на переривання передавача» (рис. 8.6), який підключений до системи переривання мікроконтролера та інформує систему про те, що буфер передавача порожній.

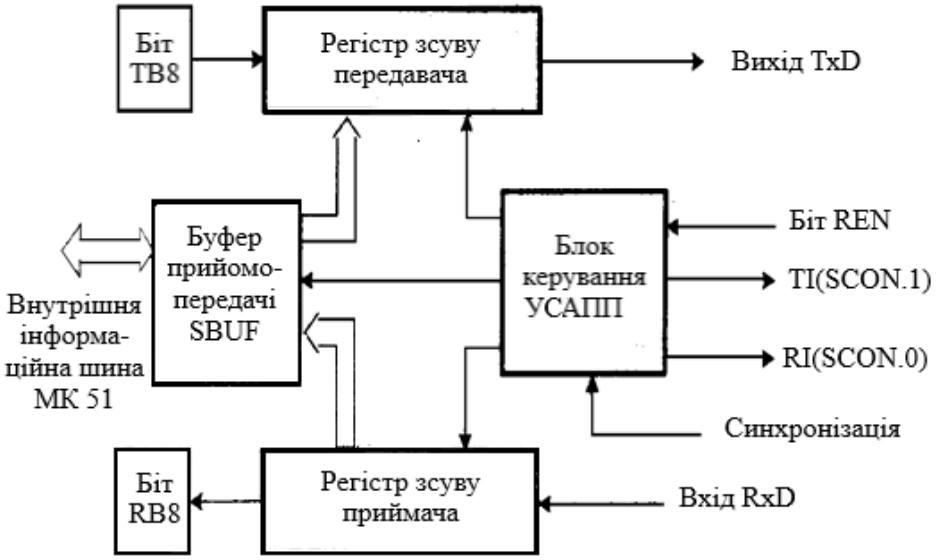


Рисунок 8.6 – Структурна схема УСАПП

Інформація з лінії зв'язку приймається побітно через вхід RxD. Після прийому останнього біта повідомлення встановлюється в одиницю тригер «готовність приймача» RI (рис. 8.6) та інформує систему про те, що буфер приймача повний і інформацію з буфера необхідно забрати. Якщо байт з буфера не буде прочитаний, то він буде втрачений.

Таблиця 8.1 – Організація регістра керування SCON

Символічне позначення	Позиційне позначення	Назва й призначення бітів
SM0	SCON.7	Біти керування режимом послідовного порту
SM1	SCON.6	Програмно встановлюються і скидаються. SM0=0, SM1=0 – режим розширення введення/виведення (зсувний регістр); SM0=0, SM1=1 – 8-бітний послідовний асинхронний порт, швидкість передачі визначається частотою таймера (лічильника) 1; SM0=1, SM1=0 – 9-бітний послідовний асинхронний порт, швидкість передачі: fgq/64 або fgq/32; SM0=1, SM1=1 – 9-бітний послідовний асинхронний порт, швидкість передачі визначається частотою таймера (лічильника) 1
SM2	SCON.5	Біт 2 керування режимом послідовного порту. Програмно встановлюється для заборони приймання кадрів з нульовим восьмим бітом
REN	SCON.4	Біт керування з дозволом приймання. Програмно встановлюється і скидається для дозволу (заборони) приймання послідовних даних
TB8	SCON.3	Восьмий біт при передачі даних. Програмно встановлюється й скидається для задання восьмого біта в режимі 9-бітного послідовного порту
RB8	SCON.2	Восьмий біт при прийманні даних. Апаратно встановлюється і скидається залежно від значення восьмого біта в режимі 9-бітного послідовного порту
TI	SCON.1	Прапорець переривання при закінченні передачі. Установлюється апаратно після передачі байта, скидається програмно після обслуговування переривання
R1	SCON.0	Прапорець переривання при закінченні приймання. Установлюється апаратно після приймання байта, скидається програмно після обслуговування переривання

8.3. Режим роботи 0

Цей режим виконує завдання синхронного обміну. Під час роботи в цьому режимі застосовується додатковий провід тактування для синхронізації приймача зовнішнього пристрою. Схему підключення зовнішнього пристрою зображено на рис. 8.7.

Передача і прийом інформації здійснюється через вхід Rxd, імпульси синхронізації – через Txd. Обміну підлягають 8-бітові дані. Мікроконтролер у цьому режимі є ведучим. Він ініціює обмін, здійснюючи початкову передачу даних. Якщо це потрібно, то передає запит на прийняття інформації, після чого переходить у режим приймання. Потім формує тактові імпульси для зовнішнього пристрою й ухвалює призначену для нього інформацію. Швидкість обміну при

цьому фіксована і дорівнює 1/12 Fosc. Для ініціювання передачі необхідно занести до регістра SCON у розряди MSO, MS1 нулі та записати в SBUF перший байт повідомлення. Наступний байт може бути записаний в SBUF тільки після передачі байта. Прапорець закінчення передачі перебуває в біті SCON.1 (TI), формується апаратно після закінчення передачі байта. Він може викликати переривання за адресою 23h. Після закінчення обробки переривання прапорець необхідно скинути програмно. Допускається опитування прапорця без використання переривання. Після встановлення прапорця TI в SBUF можна записати наступний біт повідомлення. Якщо було передано останнє байт-повідомлення, то досить скинути прапорець TI і більше він формуватися не буде. Для поновлення передачі необхідно в SBUF записати перший байт-повідомлення і процес передачі повторюється.

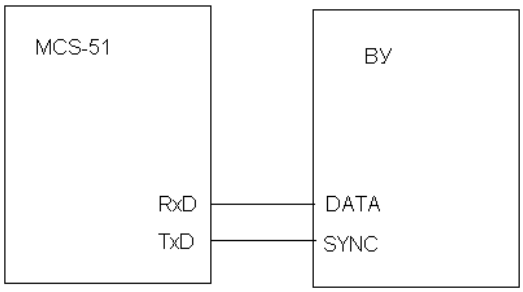


Рисунок 8.7 – Схема підключення периферійних пристроїв
до контролера в режимі роботи 0

Для переведення УСАПП у режим прийому потрібно встановити в SCON.4 одиницю.

Режим роботи 0 призначений для обміну інформацією з пристроями, що розташовані на платі контролера, при дотриманні правил перешкодозахисту та мінімальній довжині провідників. Він забезпечує максимальну швидкість обміну по послідовному каналу.

8.4. Режим роботи 1

Режим асинхронного обміну застосовується найчастіше за інші. Це пояснюється тим, що формат обміну даними у цьому режимі такий самий, як і у послідовному інтерфейсі персональних комп'ютерів (COM).

Структуру байтового повідомлення в режимі роботи 1 зображено на рис. 8.8. У повідомленні передаються 8-бітові дані, які доповнюються двома службовими бітами: старт біт (передається на початку) і стоп біт (передається наприкінці повідомлення).

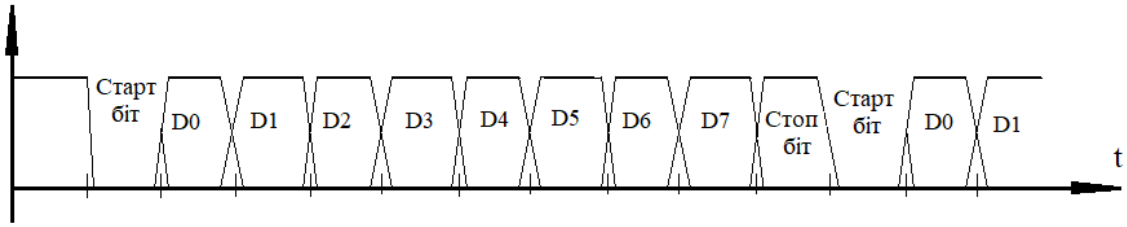


Рисунок 8.8 – Структура байтового повідомлення в режимі роботи 1

Передача даних здійснюється через вихід Txd. Прийом даних – через вхід Rxd (рис. 8.9).

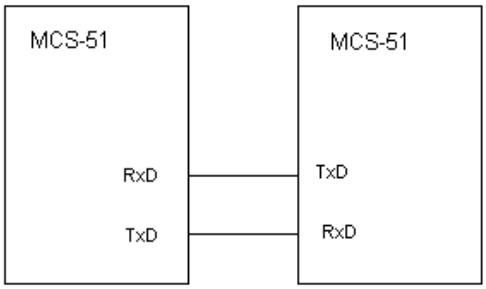


Рисунок 8.9 – Схема підключення двох контролерів у режимі роботи 1

Якщо передача в режимі роботи 1 відсутня, то вихід Txd перебуває в стані 1. Після занесення даних в SBUF передавача, до лінії зв'язку передається старт біт, за ним вісім бітів даних, а потім стоп біт. Старт біт передається нульовим, а стоп біт одиничним рівнем. Таким чином, навіть, якщо серед даних передаються всі 0 або всі 1, то у повідомленні можна виділити початок і кінець передачі байта.

Структурну схему УСАПП у режимі роботи 1 зображено на рис. 8.10.

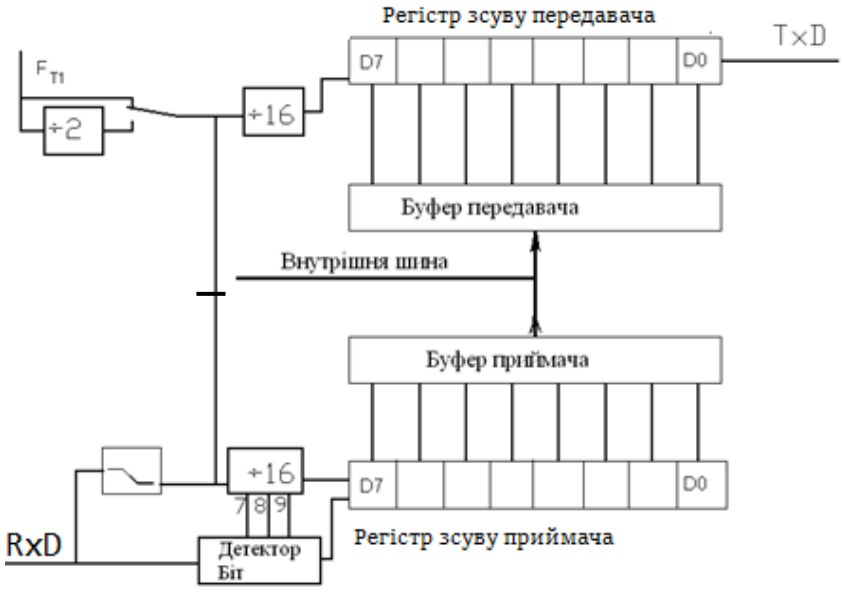


Рисунок 8.10 – Структурна схема УСАПП у режимі роботи 1

Тактування прийому й передачі в режимі 1 здійснюється від таймера 1, що працює в режимі 2 (автоматичне перезавантаження). Швидкість передачі визначається значенням константи, що завантажується в TH1.

У режимі роботи 1 відсутній сигнал зовнішнього тактування приймача. Імпульси синхронізації приймача формуються з тактової частоти таймера 1 разом зі схемами детектора спаду, детектора бітів і керованого дільника на 16.

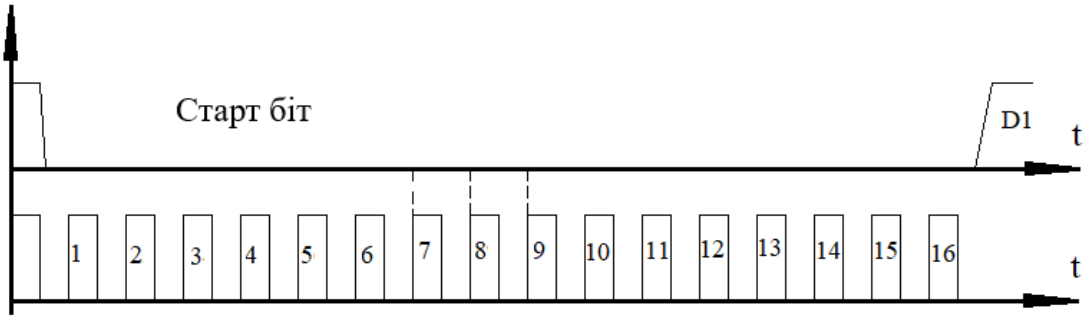


Рисунок 8.11 – Фрагмент перевірки старт біта в середині інтервалу для захисту від імпульсної перешкоди

Прийом починається під час виявлення детектором зміни сигналу на вході приймача з 1 в 0. При цьому скидається в 0 дільник на 16 приймача. На сьомому, восьмому і дев’ятому такті дільника опитується значення входу приймача RxD (рис. 8.11) і, якщо за трьома вимірами більше одиниць, то старт біт не зачитується, йде його подальший пошук. Таким чином відбувається налаштування фази тактування приймача за сигналами з лінії зв'язку. Налаштування фази забезпечує підтвердження значення прийнятих даних у середині тактового інтервалу. А мажоритарне опитування (за трьома значеннями на інтервалі) – захист від імпульсних перешкод.

При прийомі, крім контролю старт-біта, здійснюється контроль стоп-біта. Якщо він не дорівнює 1 – байт не зачитується і дані губляться.

Як правило, обмін даними між контролерами виконується декількома байтами. Цю послідовність байтів називають повідомленням. Для визначення початку повідомлення використовують або не застосовуване в оригінальне значення першого байта, або так званий тайм-аут, коли протягом певного часу передача даних перед пересиланням чергового повідомлення не відбувається.

Як правило, обмін між пристроями в послідовному форматі відбувається за алгоритмом запит – відповідь, або передача – підтвердження. При цьому ініціатором обміну є ведучий контролер або Master. Другий контролер називають веденим або Slave. Завдання на програму обміну інформацією між контролерами можна сформулювати так: необхідно передати через послідовний

інтерфейс масив даних, розміщених за адресою Adr1, довжиною в L байт; прийняти у відповідь масив довжиною L і записати його за адресою Adr2.

На рис. 8.12 наведено схему алгоритму і програму, що виконує такий обмін без використання переривань.

Приклад програми:

```
        Mass equ 30h; початок переданого масиву
        Recivmas equ 50h ; початок прийнятого масиву
        Flag equ 0h ; ознака закінчення обміну даними
        Razmer equ 2h ; R2
        Ukaz equ 0h ; R0
;OSNOVNAYA PROGRAMMA
main: mov TMOD,#20h; reg 2 taim 1
        mov PCON,#80h;SMOD=1
mov TL1,#0fdh
        mov TH1,#0fdh; частота обміну 19,2 кбіт
        setb TCON.6; режим роботи USART
        setb SCON.6
        setb SCON.7
        setb SCON.3
        mov Ukaz,#mass+1
        mov Razmer,#9; довжина масиву
                setb Flag
;PEREDACHA
                mov SBUF, Mass; передача першого байта
m1t:            jnb SCON.1,m1t
m2t:            mov SBUF,@R0
                inc r0
m2:             clr SCON.1
                djnz r2,m1t; контроль кінця масиву передачі
m1:
;PRIEM
                mov Ukaz,#recivmas
                mov Razmer,#10; довжина масиву прийому
                setb SCON.4
m1r:            jnb SCON.0,m1r
reciw:          mov @R0,SBUF; зчитування прийнятого байта
                inc R0
                clr SCON.0
                djnz r2,m1r; контроль кінця прийому
m2r:            ljmp m2r
        END
```

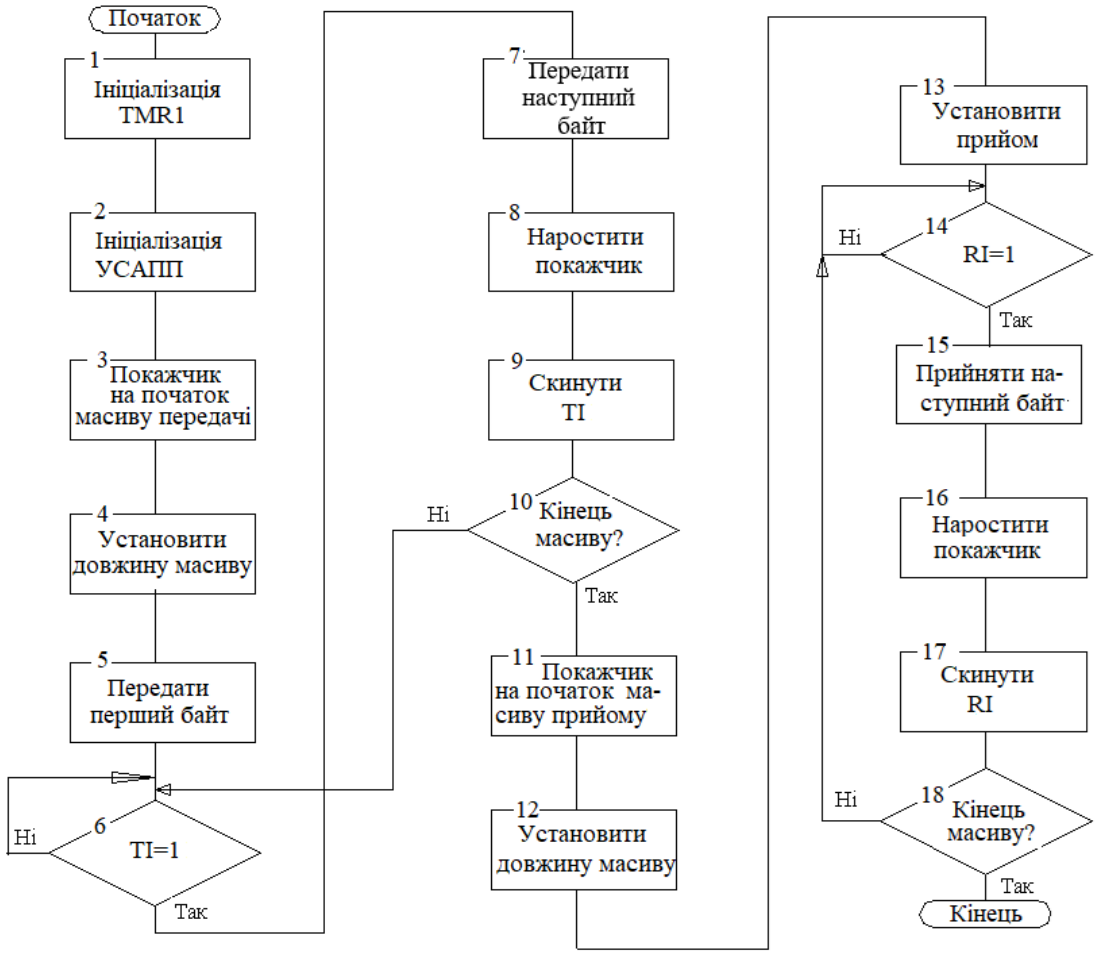


Рисунок 8.12 – Схема алгоритму обміну повідомленнями
між контролерами в режимі 1

На початку програми задається другий режим таймера 1, а також заноситься константа в TH1, що забезпечує швидкість обміну 19,2 кбіт/с.

Потім налаштовується УСАПП на перший режим роботи, а також заноситься перший байт масиву в SBUFF. Контроль передачі байта здійснюється шляхом програмного опитування біта TI в SCON. Після закінчення передачі наступного байта цей біт установлюється в одиницю. В SBUFF заноситься значення наступного елемента масиву, нарощується покажчик масиву, скидається в 0 біт TI, декрементується й перевіряється на нуль лічильник елементів масиву. Якщо останній елемент масиву передано, програма переходить на прийом відповіді на запит, якщо він був не останнім, то вертається на контроль біта TI. Після закінчення передачі масиву УСАПП переводиться в режим прийому, для цього біт REN (SCON.4) встановлюється в 1. Потім формується значення покажчика масиву для прийому, встановлюється його довжина і відбувається опитування й контроль біта RI (SCON.0). Одиниця в цьому біті означає, що приймач закінчив прийом чергового байта, тепер необхідно вичитати буфер, щоб звільнити його для прийняття наступного байта. Після зчитування з SBUFF значення заноситься в масив прийому,

нарощується покажчик елементів масиву, скидається біт RI, декрементується і перевіряється на нуль лічильник елементів масиву. Якщо останній байт масиву прийнято, біт REN скидається в 0, прийом припиняється і програма переходить на продовження виконання завдання керування.

8.5. Режим роботи 2

У цьому режимі, крім старт і стоп біта та восьми бітів даних, додається дев'ятий біт, значення якого можна задавати програмно. Він використовується для підвищення вірогідності прийому інформації (для передачі значення біта контролю парності), або для передачі ознаки початку повідомлення в багатоконтролерних мережах, де першим байтом передається номер адресата. Частота прийому/передачі може дорівнювати 1/32 або 1/64 частоти резонатора залежно від значення SMOD (PCON.7).

8.6. Режим роботи 3

Режим роботи 3 збігається з режимом роботи 2 за форматом повідомлення, але швидкість обміну може задаватися таймером 1, як і в режимі 1.

Контрольні питання

1. Яку кількість бітів містить байтове повідомлення УСАПП МК51 в режимі 1?
2. Яку кількість бітів містить байтове повідомлення УСАПП МК51 в режимі 2?
3. Який біт регістра SCON УСАПП МК51 дозволяє роботу контролера в мережевому режимі?
4. Який сигнал УСАПП МК51 використовується в системі переривань при передачі інформації?
5. Який сигнал УСАПП МК51 використовується в системі переривань при прийомі інформації?
6. Скільки режимів роботи має послідовний порт МК51?
7. Яку кількість бітів містить байтове повідомлення УСАПП МК51 в режимі 0?
8. Який перехід у лінії зв’язку характеризує початок байтове повідомлення?
9. Який перехід у лінії зв’язку характеризує завершення байтове повідомлення?
10. Який біт регістра SCON УСАПП МК51 дозволяє прийом інформації?

9. ПОСЛІДОВНІ ШИННІ СИСТЕМИ (КАНАЛИ ЗВ'ЯЗКУ)

9.1. Загальні поняття про електронні шинні системи

У сфері електроніки під шинними системами розуміють зв’язок між більш ніж двома електронними компонентами або системами задля передачі інформації між ними через спільний канал зв’язку.

У перекладі на німецьку «шинна система» буде «Bussystem», де «Bus» перекладається також як «автобус». Таким чином, можна припустити, що це слово було запозичено зі сфери транспортної техніки, де автобус для пасажирів є транспортним засобом, який везе їх за єдиним маршрутом, хоча, як правило, вони підсаджуються в різних точках шляху і мають різні пункти призначення.

Середовище передачі даних в електронних шинах часто має гальванічний характер, тобто воно складається зі з’єднань, що проводять електричний струм, до яких підключені елементи системи. Також існують та використовуються інші середовища передачі даних – оптичні з’єднання (наприклад, за допомогою оптичного волокна) та безпровідне з’єднання.

У цьому розділі мова буде йти про неоптичні лінії передачі даних.

Таким чином, згідно з визначенням, до однієї шини можуть бути приєднанні більш ніж два об’єкти, тобто існує декілька джерел передачі даних. Звідки одразу зрозуміло, що однією з основних особливостей шини має бути обробка конфліктів на лінії зв’язку або ж за необхідності стратегії, які можна із самого початку ліквідувати під час передачі інформації.

Електронні шинні системи поділяються на паралельні та послідовні.

У паралельних шинних перенесення даних відбувається паралельно, тобто одночасно за декількома паралельними лініями зв’язку. Паралельні з’єднувальні шини часто розглядають як шини в комп’ютерних чи мікропроцесорних системах для з’єднання декількох електронних блоків, що знаходяться на різних платах.

У промисловій та непромисловій сферах використовується велика кількість різноманітних паралельних шин. Як приклад з промислової сфери можна назвати шину VME, що використовується у вимірювальній техніці та техніці автоматизації.

До характерних особливостей паралельних шинних з’єднань відносяться:

- висока швидкість передачі даних;
- мала довжина шин;
- значні високі витрати на процеси з’єднань через велику кількість ліній зв’язку.

У шинних з’єднаннях послідовного обміну передача даних відбувається послідовно в часі, тобто у формі один за одним переданих бітів через один канал зв’язку.

До характерних особливостей послідовно з’єднувальних шин відносяться:

- більш низька у порівнянні з паралельними шинами такої самої тактової частоти швидкість обміну даними;
- більш малі витрати на процеси з’єднань через меншу кількість ліній зв’язку;
- більша просторова протяжність шини, яка може коливатись від декількох десятків метрів до кількох кілометрів;
- необхідні паралельно-послідовні та послідовно-паралельні перетворювачі даних, оскільки інформація в передавачі та приймачі, як правило, і видається, і приймається в паралельній формі.

Більше того, шини послідовного обміну розрізняються за сферою їх використання. Ми можемо спостерігати в оточуючому нас середовищі, наприклад, в офісі «нормальні» шини, для яких не існує окремого терміна. Відомий приклад – USB (Universal Serial Bus – універсальна послідовна шина) зі сфери персональних комп’ютерів.

Системам, що позначаються спеціальним терміном «польові шини», властива більш висока надійність, особливо це стосується електромагнітної сумісності для жорстких умов – використання в промисловості або в транспортних засобах. Перш за все, за допомогою польових шин було розроблено велику кількість різноманітних систем, що поширені також дуже по-різному. Як приклад можна навести одну з найпоширених систем CAN (Controller Area Network). Вона застосовується в мільйонах транспортних засобів, а також у великій кількості інших сфер.

9.2. Різновиди послідовних шинних систем

Кількість шинних систем, пов’язаних з керуванням у сфері офісної техніки, не дуже велика. До них належать USB (Universal Serial Bus), Ethernet , а також IEEE 1394 «Firewire».

На відміну від офісних, існує велика кількість різноманітних систем у сфері промисловості. У цьому разі потрібно розрізняти, так звані, патентовані та «відкриті» шини. Патентовані системи мають різноманітні характеристики, залежно від специфіки роботи фірми, і вони, як правило, не розголошуються.

«Відкритою» називають таку шинну систему, до специфікацій якої відкрито вільний доступ, і прилади, що підключаються до шини, можуть бути змінені та допрацьовані будь-якою фірмою. Однак, стосовно ступеня відкритості існують чіткі відмінності. Наприклад, щоб мати право використовувати бренд з шини деяких систем, необхідно платити за ліцензію, або треба перебувати в певних групах, проводити тести на сумісність.

Ще одна явна відмінність відносно відкритості системи стосується різних комунікаційних рівнів шинної системи, що будуть розглядатися в наступному розділі. Таким чином, існують шини, які відкриті тільки на певних сеансових рівнях, на інших рівнях знаходяться патентовані елементи. Класичним прикладом можуть бути шинні системи в транспортних засобах, таких як CAN, де обидва нижні рівні комунікації (так званий фізичний рівень та рівень біт-передачі) відповідають міжнародним стандартам, однак, на рівні прикладного протоколу (тобто значення переданих даних) виробники автомобілів часто використовують свої особисті закриті специфікації.

Вже мова йде про те, що для розуміння шинних систем дуже корисно знати їх загальну класифікацію за ISO/OSI (базова модель для систем зв’язку). Ця модель включає в себе сім рівнів, які виконують різні функції в шинних системах, що тут розглядаються. Нижній рівень має характеристики середовища передачі. Другий рівень (рівень біт-передачі) містить процеси зображення бітів та доступу до шини, включаючи рішення або попередження комунікаційних колізій.

Рівні четвертий – шостий у шинних системах, що розглядаються, відображені слабо і через свою простоту до уваги не приймаються. Рівень сьомий (прикладний рівень або протокол прикладної програми) є важливим. Варто відзначити, що приписування окремих специфікацій, процесів або протоколів до певних рівнів не завжди можливо та потрібно. Не дивлячись на це, багаторівнева модель допомагає все структурувати та зрозуміти.

На прикладі використання CAN у транспортних засобах вже було показано, що в шинних системах різні рівні можуть бути стандартизованими або нестандартизованими. Дуже важливо враховувати це, коли мова йде про одиничні шини.

Часто мета відкритої шинної системи полягає в тому, щоб різні виробники мали змогу розробляти устаткування для даної системи, а прилади від них працювали на тій самій шині. При цьому варто чітко розрізняти, чи є ці прилади «сумісними» або «мають можливість взаємодіяти». В цьому випадку під

сумісністю розуміють можливість приладів не заважати комунікації в шині, а брати в ній активну участь. Функціональна сумісність потребує загальних специфікацій значень, тобто інтерпретації переданих даних. У цьому відношенні функціональна сумісність передбачає специфікатори на всіх реалізованих шинною системою комунікаційних рівнях, зокрема, на верхніх рівнях протоколу (прикладних рівнях).

Через широкий спектр вимог та умов використання виникла велика різноманітність промислових шинних систем. Сьогодні відомо більш ніж 50 шин з відкритим протоколом, частково з підваріантами застосування, що потребують особливої безпеки, та тільки декілька шин спеціально розроблені для забезпечення безпеки критично важливих додатків. Крім того, відомо понад 15 відкритих промислових систем Ethernet, що базуються на комунікаційних системах.

Щоб мати уяву про різноманіття шин, які використовуються у сфері промисловості, наведемо деякі з них: ARCNET, AS-Interface (Aktuator-Sensor-Interface), BACnet (Building Automation and Control Network), Bitbus, byteflight, CAN (Controller Area Network), CC-Link, EIB (European Installation Bus), FIP (Factory Instrumentation Protocol), Flexray, HART (Highway Addressable Remote Transducer), INTERBUS, LIN (Local Interconnecting Network), LON (Local Operating Network), Modbus, MOST (Media Oriented System Transfer), PROFIBUS (mehrere Varianten), Safetybus-P, SERCOS, SwiftNet. І це ще не весь перелік.

Розглянемо декілька прикладів промислових систем Ethernet, тобто систем, які базуються на різноманітних модифікаціях Ethernet, що були вдосконалені та адаптовані до використання в промисловому середовищі: EtherCAT, Ethernet/IP, Ethernet Powerlink, Profinet, SERCOS III, VARAN.

Звісно, виникає питання, яка шинна система найбільш відповідає тим чи іншим вимогам, та які системи мають розширення. У системі [GRU2001] детально розглянуто засіб для порівняння, оцінки та вибору послідовної шини.

У наступних часткових розділах стисло наведені деякі приклади різноманітних систем зв’язку, які, без сумніву, мають відношення до мікрокомп’ютерів чи мікроконтролерів.

У наступних підпунктах будуть розглянуті деякі приклади абсолютно різноманітних комунікаційних систем, які, звісно, безпосередньо пов’язані з мікрокомп’ютерною та мікропроцесорною технікою.

9.3. Окремі приклади інтерфейсів та послідовних шин

Далі як приклад будуть наведені інтерфейс RS232, система RS485, локальна шина I2C, а також польові шини CAN і Modbus.

Зв'язок між комунікаційною системою та мікропроцесором або мікроконтролером здійснюється за допомогою відповідного контролера та трансивера. Шинні контролери генерують та обробляють логічні фронти імпульсу та послідовність бітів, у той час трансивери як безпосередньо інтерфейс до комунікаційних ліній надають та сприймають сигнальні фронти імпульсу та рівні біта безпосередньо у вигляді сигналів на лінії.

З одного боку, зв’язкові контролери можуть приєднуватися користувачем до мікроконтролерів як окремі модулі за допомогою адресних шин, ліній передачі даних та шини керування. З іншого – зараз багато мікроконтролерів вже обладнано вбудованими зв’язковими контролерами, що, звісно, суттєво полегшує роботу розробника.

Таким чином, на сьогодні доступно багато модифікацій мікропроцесорів 8051 вже з вбудованими комунікаційними контролерами.

9.3.1. Інтерфейс RS232

Інтерфейс зв’язку RS232 являє собою EIA-Norm (Electronic Industries Association). Тут мова йде не про шинну систему, а про зв’язковий інтерфейс виключно між двома користувачами, тобто про двоточкове з’єднання або однорангове з’єднання «ЕВМ-ЕВМ» (оверлейна комп’ютерна мережа).

9.3.1.1. Визначення логічного рівня

Визначення логічного рівня відносять до електричних властивостей інтерфейсу RS232. Варіант такого інтерфейсу розглянуто на прикладі прямого імпульсу (рис. 9.1).

Типове значення рівня лінії передачі даних складає +12V для логічного нуля та –12 V для логічної одиниці. Середні значення знаходяться в інтервалі +3V – логічний нуль, –3V – логічна одиниця, та максимально допустимі значення – +15V та –15V відповідно.

Таким чином, біт даних буде розшифровано як логічну одиницю, якщо рівень U менший за –3V, і як логічний нуль при рівні U більший за +3V. Навпаки, сигнали, що сповіщають та керують, є активними, коли напруга більша за +3V, та неактивними, коли рівень U менший за –3V.

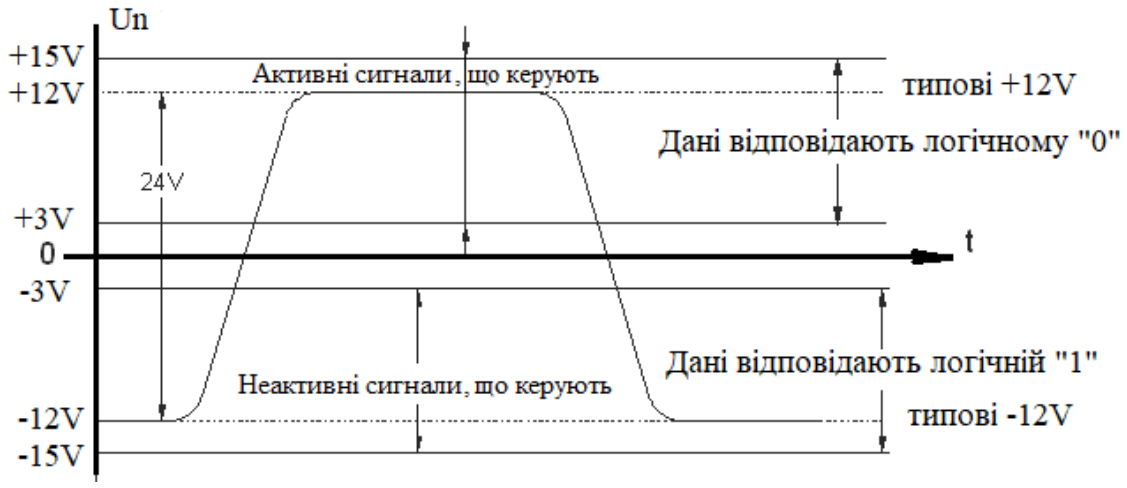


Рисунок 9.1 – Визначення логічного рівня RS232 для ліній передачі даних та керуючих сигналів

9.3.1.2. Розведення контактів та сигналів

Як роз'ємне з'єднання RS232 використовують, наприклад, штекерний підроз'єм D. Його вигляд зверху зображено на рис. 9.2.

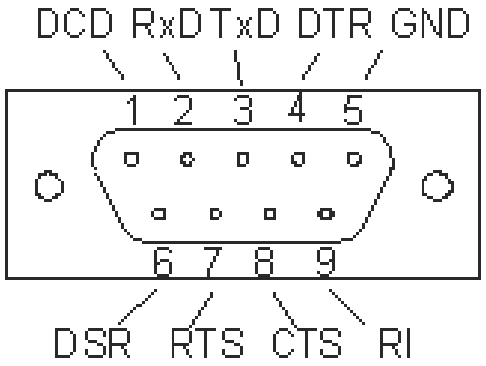


Рисунок 9.2 – Дев'ятиконтактний штекерний підроз'єм D інтерфейсу з призначенням контактів виведення для інтерфейсу RS232

Магістралі, які використовуються для передачі даних, можуть бути змонтовані відповідно до розпайки виводів, що демонструється вище (рис. 9.2).

Призначення сигналів інтерфейсу RS232 і контактів виведення наведені на рис. 9.3, де зображені найбільш важливі сигнали.

На рис. 9.4 зображено просте двопровідне з'єднання між двома терміналами на основі RS232.

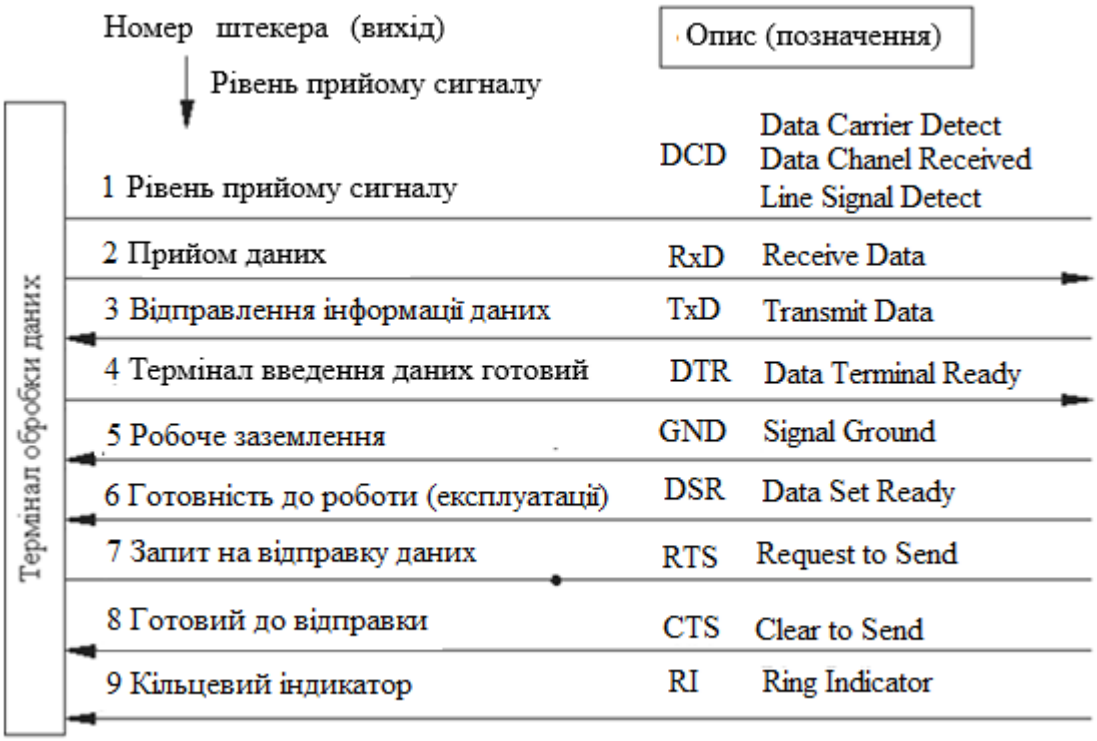


Рисунок 9.3 – Призначення сигналів інтерфейсу RS232

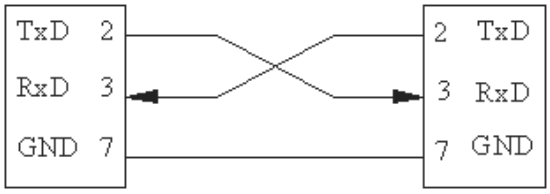


Рисунок 9.4 – Мінімальна конфігурація з’єднання RS232 між двома терміналами

9.3.1.3. Визначення довжини кабелю

На магістралях інтерфейсу RS232 сигнали є біполярними. Логічний рівень може коливатись від +3V до +15V та відповідно від –3V до –15V. Завдяки цьому може бути забезпечена швидкість передачі даних 19200 бодів на відстань приблизно 15 метрів, якщо паразитний зв'язок інших приладів не дуже сильний. Цього можна запобігти, якщо використовувати екрановані кабелі. Двічі екранований кабель з маленькою ємністю дає можливість використовувати в два та навіть і в три рази більшої довжини кабелю RS232 шляхом зниження спотворення форми імпульсу та значного зниження паразитних зв’язків.

9.3.2. Інтерфейс RS485

З’єднання за допомогою EIA RS485 дає можливість підключення до шини та комунікаційної лінії зв’язку до 32 користувачів.

Інтерфейс RS485 задає, так само як і EIA-Standard, тільки електричні або, кажучи іншими словами, фізичні умови. Його було розроблено для того, щоб мати можливість приєднати більшу кількість користувачів до шини.

Для передачі логічного рівня використовується метод різниці напруг. Всі прилади, які є у передавача та приймача даних, приєднуються до єдиної двопровідної лінії передач, клеми A і B, це означає, що вони ввімкнені паралельно (рис. 9.5). Графіки зміни сигналів при передачі 0 і 1 наведені на рис. 9.6.

У цьому інтерфейсі можливо підключення 32 користувачів (як передавачів, так і приймачів даних) до одного двопровідного шинного кабелю з симетричним перенесенням даних. Для забезпечення симетрії провідники закручуються з фіксованим кроком. Такий кабель ще називають «вита пара». У такому кабелю електромагнітні перешкоди будуть наводити відповідний сигнал однаковий для провідника A і провідника B. При цьому напруга на провіднику A буде $U_{a'} = U_a + U_{\text{п}}$, на провіднику B $U_{в'} = U_в + U_{\text{п}}$. Оскільки на вході приймача завжди діє різниця напруг U_a і $U_в$, то корисний сигнал на вході приймача визначається як $U_{вх} = U_{a'} - U_{в'} = U_a + U_{\text{п}} - U_в - U_{\text{п}} = U_a - U_в$. Завдяки симетричній конструкції інтерфейсу досягається високий рівень захищеності від перешкод.

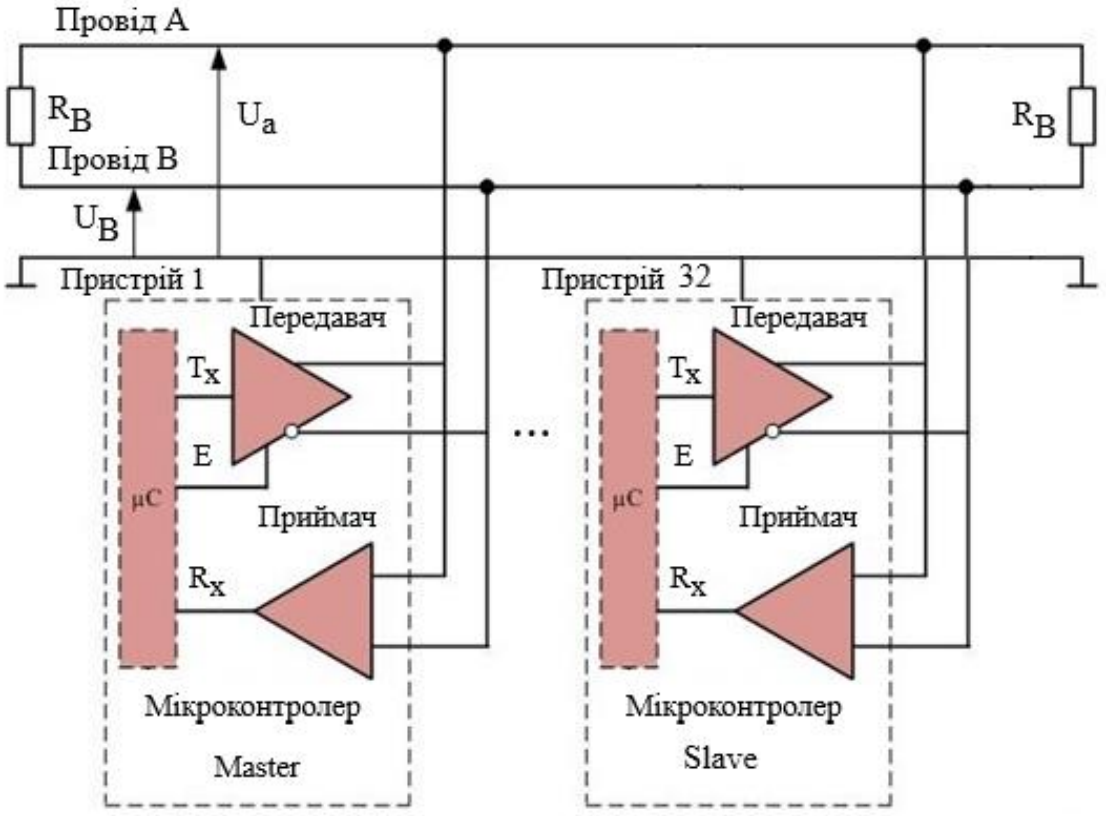


Рисунок 9.5 – Вигляд шинної системи з інтерфейсом RS485

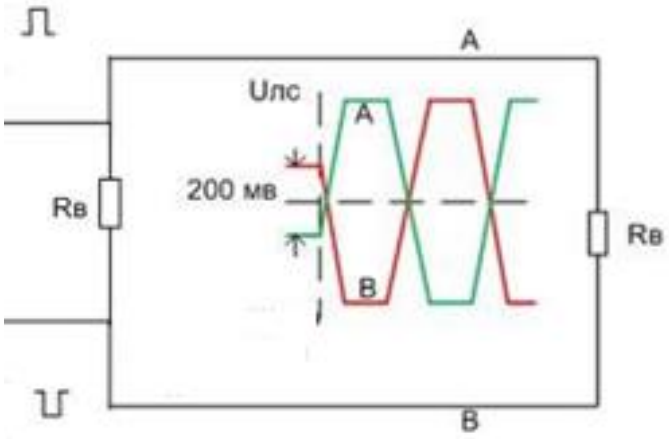
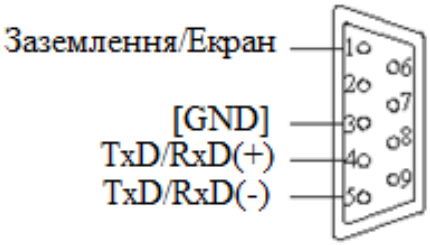


Рисунок 9.6 – Графік зміни сигналів при передачі 0 та 1

Часто для з’єднання RS485 використовується 9-контактний штекерний роз’єм. На рис. 9.7 зображено розташування виводів та призначення сигналів такого інтерфейсу.



Вивід	Функція
1	Заземлення/Екран
2	-
3	GND (З'єднання не обов'язкове)
4	TxD/RxD(+) Відправлені/Отримані дані(+)
5	TxD/RxD(-) Відправлені/Отримані дані(-)
6	-
7	-
8	-
9	-

Рисунок 9.7 – Розташування виводів та сигналів інтерфейсу RS485

Основні характеристики з’єднання RS485: максимальна довжина лінії передач 1200 м (при зменшеній швидкості обміну даними 100 кбіт/с) та максимальна швидкість передачі інформації 10 Мбіт/с (при дуже маленькій довжині магістралі).

EIA-Norm визначає спочатку тільки електричні (фізичні) характеристики інтерфейсу, коли на одну шинну лінію передач приєднується декілька користувачів (тут до 32). Проблему комунікаційного конфлікту можна

розв’язати за допомогою відповідного протоколу обміну даними. Тому з’єднання використовується завжди разом з протоколом обміну. Протокол може бути як власної розробки, так і стандартизований. Це означає, що описуючи з’єднання, реквізити RS485 вказують тільки на фізичні характеристики (фізичний рівень 1). Додатково потрібні дані про протокол, що використовується (фізичний рівень 2 та вище). Оскільки з’єднання RS485 приєднується до шини, то за його допомогою можуть бути реалізовані також і польові шинні системи. Таким чином, наприклад, раніше реалізовувалися частини протоколу CAN на основі фізичних з’єднань RS485. Як приклад польових шин, що частково використовують RS485, можна назвати ще PROFIBUS, MODBUS та INTERBUS.

9.3.3. CAN (Controller Area Network – мережа контролерів)

9.3.3.1. Основні відомості

Відтоді, як з 1980 р. електроніка почала займати все більше місця в автомобільній галузі, підвищувалась потреба в комунікації між окремими керуючими блоками. Звичайна проводка вже не відповідала цим вимогам. Компанія Bosh, як великий виробник керуючих блоків, дуже рано виявила потребу у винаході абсолютно нової, спеціально призначеної для використання в автопромисловості, шинної системи. Таким чином з’явилась мережа контролерів CAN (Controller Area Network). Важливими завданнями при її виробництві були надійність, низька вартість, ефективний протокол і можливість функціонування за наявності одночасно кількох ведучих пристроїв. У кінці вісімдесятих з’явився перший CAN-контролер, розроблений фірмою Intel. Протокол CAN було регламентовано у міжнародному стандарті ISO 11898.

Тим часом CAN затвердився як стандарт в організації мереж для автомобілів і почав використовуватися майже всіма заводами–виробниками транспортних засобів. В автоматиці, де часто переважають такі самі вимоги до шинної системи, як і в автомобільній галузі, CAN широко застосовується з початку дев’яностих. На базі CAN (мережевий рівень 1+2) було створено багато відкритих протоколів на рівні сім з частково власними групами користувачів в автоматиці, до того ж у цій галузі тоді можна було зустріти і протоколи, що належали певним фірмам. Міжнародна організація користувачів та виробників CAN in Automation (CiA) – CAN в автоматизації – виробляє в усьому світі основані на CAN програми та пристрої, особливо в автомобільній промисловості. CiA виконує функції нормування, маркетингу та розвитку CAN. Відповідно до даних CiA в усьому світі виробляється та відповідно застосовується більше міліарда CAN-вузлів.

9.3.3.2. Протоколи застосування на базі CAN

CAN-інтерфейс був запропонований німецькою компанією Bosch в 1980 р, основна мета якої – заміна всіх джгутів в автомобілі двома проводами (які кручені парою).

Основна відмінність CAN-шини від інших польових полягає в тому, що передане по шині повідомлення не містить адреси приймача пристрою призначення, а містить ідентифікатор повідомлення. Таким чином, передане повідомлення може бути одночасно прочитано декількома приймачами, яким потрібна ця інформація (повідомлення). Приймачі налаштовують свої внутрішні фільтри повідомлень на відповідні ідентифікатори.

Фізичне середовище, певна для CAN у стандарті ISO11898 – це диференційно–керована двопровідна шина із загальним зворотним проводом. Два провідники, згідно з полярністю щодо заземлюючого проводу, називаються CANH і CANL. На вході приймального пристрою діє диференціальна напруга між провідниками CANH і CANL. Шина CAN закінчується на обох кінцях резисторами з рекомендованим значенням хвильового опору 124 Ом.

Максимальна довжина лінії становить 1 км (для малих швидкостей пере ених вузлі] хема «мон

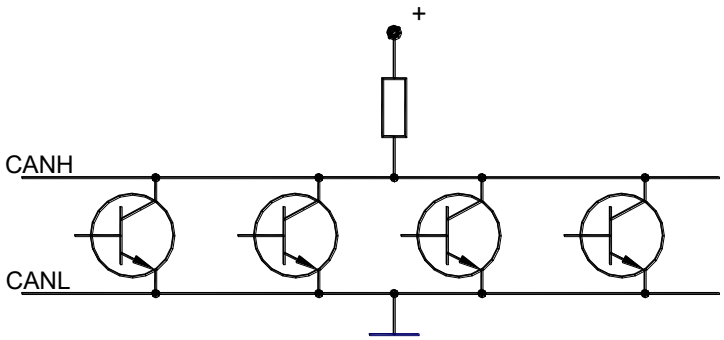


Рисунок 9.8 – Загальний вигляд схеми і принцип роботи «монтажне І»

Кожен з транзисторів, зображених на рис. 9.8, являє підключення вузла на шину CAN. Кожен вузол може передавати логічну одиницю (еквівалентно закриттю транзистора), при цьому на проводі шини CANH з'являються «+» напруги джерела живлення або логічна одиниця, відбувається відкриття транзистора, що призводить до заземлення проводу CANH, а це еквівалентно сигналу логічного нуля. Ця схема має назву «монтажне І», оскільки для

формування на провіднику CANH логічної одиниці всі модулі, підключені до лінії, повинні передавати логічну одиницю. Для формування логічного нуля на шині достатня наявність одного вузла, який передав би логічний нуль.

Через це значення нуль називається домінантним (Dominant) бітом на шині CAN, а значення одиниця – рецесивним (Recessive) бітом (рис. 9.9).

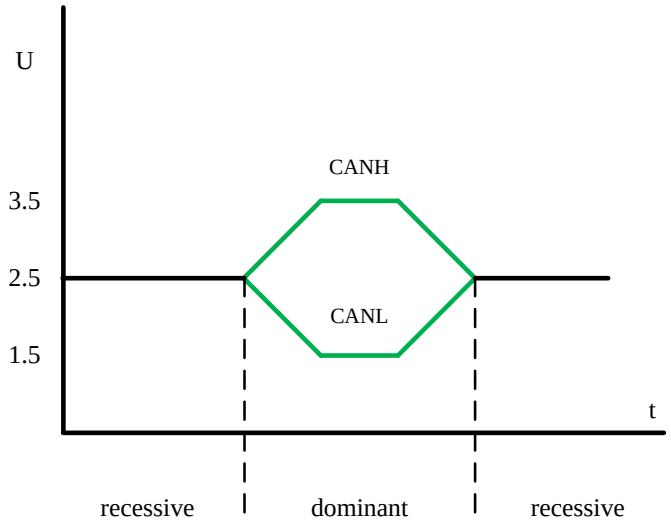


Рисунок 9.9 – Форма сигналів на CAN-шині

Передача даних по CAN-шині здійснюється за допомогою повідомлень (фреймів), встановлених стандартом CAN 2.0. B (рис. 9.10).

Якщо шина вільна, то в лінії зв'язку встановлений одиничний сигнал. Повідомлення даних розділяється на поля (рис. 9.10, *а*). Поле початку повідомлення характеризується старт-бітом (SOF-Start of frame, перехід з 1–0), потім розташоване поле арбітражу, яке містить 12-32 біта. Ідентифікатор повідомлення містить 11 або 29 бітів і один біт RTR, який визначає це повідомлення як повідомлення даних (RTR = 0) або кадр запиту даних (RTR = 1).

Контрольні шість бітів (Control Field), чотири з яких визначають кількість байт даних, що передаються в посилці (до восьми байт даних), один біт розширеного кадру (IDE) використовуються для вказівки розширеного ідентифікатора, один біт R0 резервний. Поле контрольної суми CRC Field містить 16 бітів, 15 бітів з яких складають контрольну суму, а один біт використовується як розділовий CRC Del. Поле підтвердження складається з двох бітів. Перший біт ACKSlot, який під час передачі формується як recessive, а пристрій, який приймає кадр без помилок, підтверджують правильність прийому установкою цього біта в стан dominant.

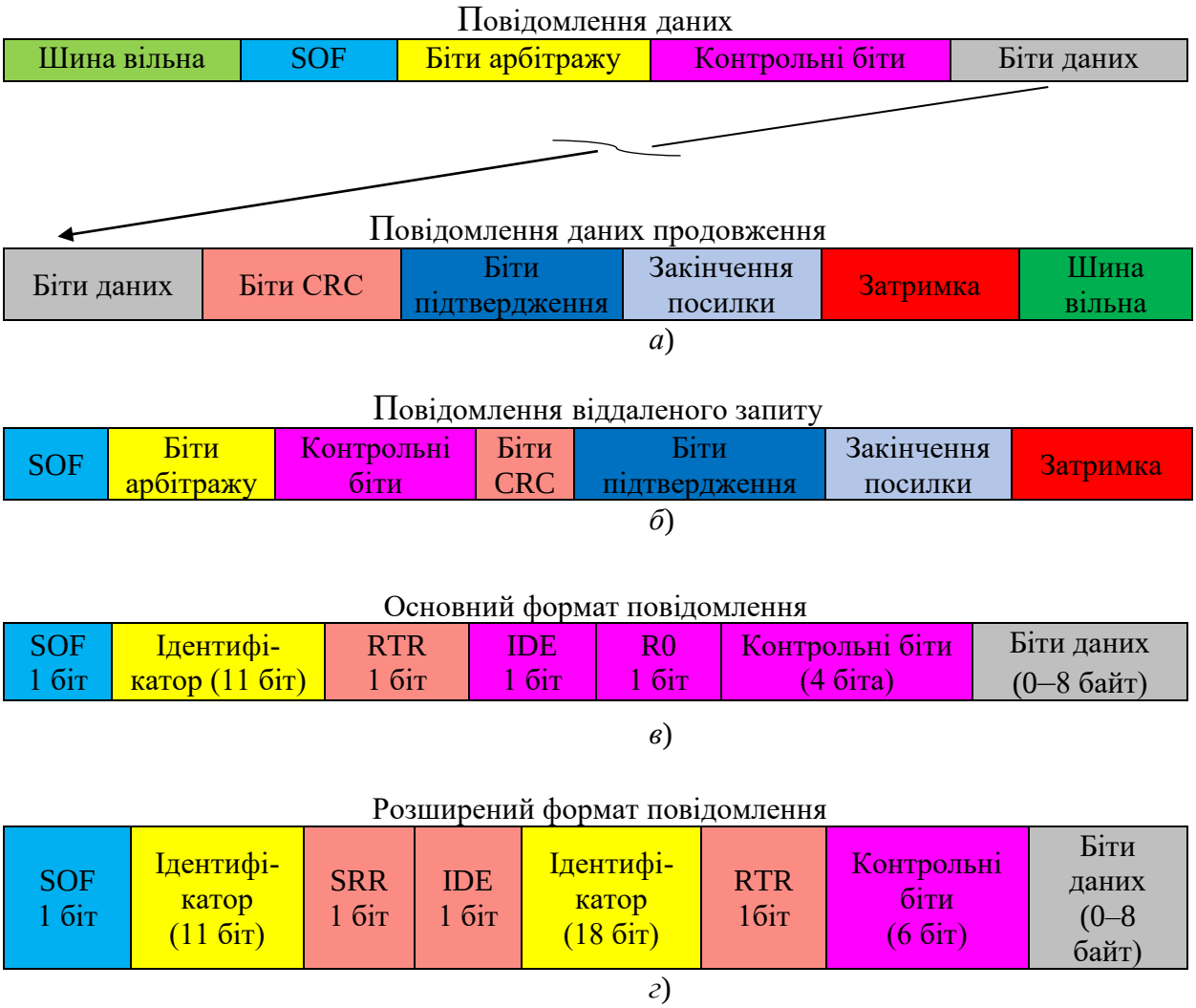


Рисунок 9.10 – Структура і формат посилок у CAN-шині

Другий розділовий біт ACKDel завжди зберігає значення recessive. Закінчується кадр полем закінчення кадру End of Frame (7 бітів). Всі біти мають стан recessive. Пауза між кадрами складається як мінімум з трьох одиничних бітів.

Таким чином, основний формат повідомлення (рис. 9.10, *в*.) містить ідентифікатор з 11 бітів, розширений формат (рис. 9.10, *г*) з 29 бітів. Біт RTR, що визначає тип повідомлення (дані або запит даних) і в стандартному, і в розширеному форматі, розташовується відразу за ідентифікатором. Розширений формат повідомлення підтверджується бітом SRR. Стандартне повідомлення з 11-бітовим ідентифікатором містить довжину 108 біт, повідомлення з розширеним ідентифікатором містить 131 біт, повідомлення запиту даних, яке має той самий ідентифікатор, тільки біт RTR в цьому повідомленні встановлено в 1. Оскільки даних в цьому повідомленні немає, то довжина такого повідомлення дорівнює 44 бітам. За даними повідомленнями розташовуються

однакові блоки бітів: контрольна сума, біти підтвердження правильного прийому повідомлення, біти закінчення повідомлення.

Якщо шина вільна, будь-який пристрій може почати передачу повідомлення. Ідентифікатор визначає пріоритет повідомлення. При одночасній передачі повідомлень одним або декількома пристроями кожен пристрій зчитує зміст стану шини і переданого біта ідентифікатора і, порівнюючи їх, вирішує конфліктну ситуацію.

Наприклад, можна задати, що отримує доступ до каналу той пристрій, який передає 0 в ідентифікаторі в порівнянні зі станом шини або навпаки. Якщо рівні однакові, передача триває.

Засіб керування доступом відповідальний за визначення, яким з вузлів у мережі надана шина для передачі повідомлення.

Домінуючий біт на шині відповідно до логіки роботи «монтажного І» завжди головніше будь-якого числа рецесивних бітів, переданих іншими вузлами. Отже, вузол, який робить спробу передачі, збереже контроль щодо шини тільки в тому разі, коли повідомлення, передане ним, має наймолодший ідентифікатор серед усіх повідомлень, що передаються одночасно в шину. Цей механізм і є метою доступу в CAN.

Розглянемо приклад, коли три вузли одночасно намагаються передати свої повідомлення (рис. 9.11).

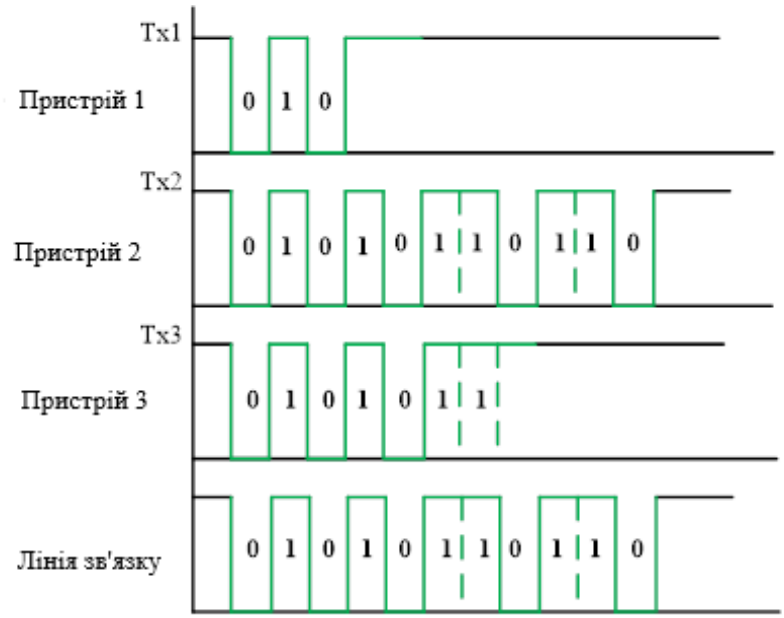


Рисунок 9.11 – Графік часового аналізу боротьби щодо переваги доступу до CAN-шини

Перший біт у всіх повідомленнях – це перехід з 1 в 0. Другий, третій і четвертий біти у всіх пристроїв однакові і всі продовжують передавати. На

п'ятому біті пристрій 1 передає одиничний біт, а пристрої 2 і 3 передають нулі. У цьому разі пристрій 1 апаратно відключається від шини.

Пристрої 2 і 3 продовжують передачу повідомлень до восьмого біта, коли пристрій 3 передає одиницю, а пристрій 2 – нуль. У цьому разі пристрій 3 відключається від шини, а пристрій 2 продовжує передавати своє повідомлення. Аналіз свідчить, що повідомлення, які передаються вузлами 1 та 3, мають більш низький пріоритет (більш високе значення ідентифікатора), отже, вузол 2 виграє боротьбу за канал і продовжує передачу. Таким чином при передачі CAN-пристроями в лінію зв'язку різних сигналів RS тригер скидається і блокує передачу даних у того пристрою, який передає одиничний сигнал. Важливо відзначити, що боротьба за канал відбувається без порушення передачі пристрою з високим пріоритетом. Наприклад, якщо одночасно починається передача кадру запиту даних (remote frame) і кадру даних (data frame) з одним і тим самим ідентифікатором, то доступ до каналу зв'язку отримає кадр даних, оскільки біт RTR (Remote Transmission Request) у кадрі запиту має значення (recessive), а в кадрі даних – (dominant). Приклад схемного рішення боротьби за канал наведений на рис. 9.12.

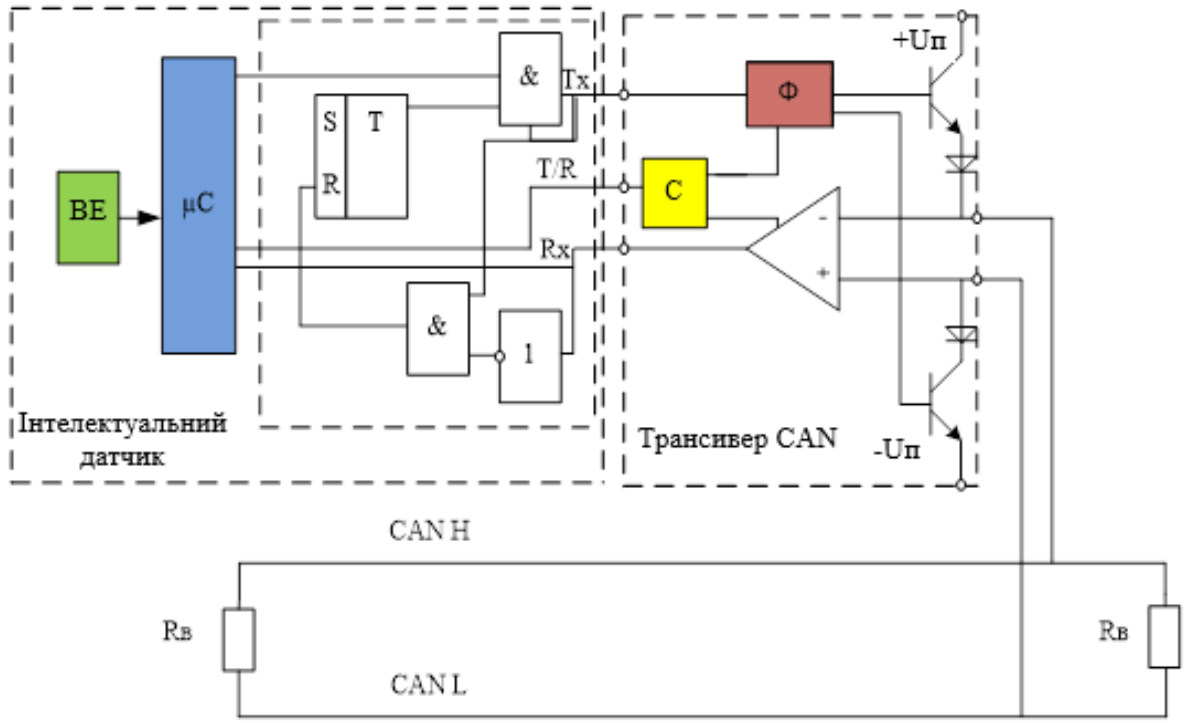


Рисунок 9.12 – Функціональна схема підключення інтелектуального датчика до CAN-шини

У CAN-шині передане повідомлення може бути прийнято декількома пристроями. При цьому пристрої можуть підключатися або виключатися з мережі без зміни програмного і апаратного забезпечення вузлів системи.

Алгоритм роботи CAN-шини сертифікований у вигляді стандарту CAN 2.0 B, що забезпечило виготовлення CAN-контролерів у вигляді окремих мікросхем для реалізації всіх функцій CAN-інтерфейсу на апаратному рівні.

Наприклад, найбільшого поширення набули мікроконтролери, які мають на кристалі декілька послідовних інтерфейсів, таких як для міжплатних комунікацій (I2C, SPI), так і таких як RS485 і CAN

Функціонально мікроконтролер складається з мікропроцесора і CAN-пристрою. Мікропроцесор через порти введення/виведення зв'язаний з датчиками, а через SPI-інтерфейс – з CAN-пристроєм.

Фільтрація всіх повідомлень у CAN-контролері здійснюється шляхом налаштуванням фільтрів і масок. CAN-шина містить кілька рівнів протоколу передачі даних:

– рівень прикладної програми – приймає відфільтровані повідомлення і запитує інформацію з необхідного ідентифікатора;

– рівень повідомлення – здійснює фільтрацію повідомлень і визначає їх статус;

– транспортний рівень – визначає спосіб передачі повідомлення по фізичному каналу. Цей рівень відповідає за бітову синхронізацію, розподіл повідомлення по кадрах, доступ до каналу, підтвердження повідомлення і розпізнавання помилок;

– фізичний рівень – забезпечує спосіб передачі бітів.

Бітове представлення, яке використовується для передачі повідомлень у шини CAN-NRZ кодування (без повернення до нуля), гарантує максимальну ефективність у бітовому кодуванні. Тому, якщо в переданому повідомленні присутні поспіль наступні один за одним однакові біти, то для забезпечення синхронізації після п'яти послідовних рівних бітів передавальний вузол вставляє біт з інвертованим значенням. На приймальній стороні ці біти наповнення видаляються перед обробкою повідомлення.

У CAN-шині кожен модуль повинен бути здатний взаємодіяти на швидкості 20 кбіт/с. Інші рекомендовані швидкості передачі інформації в бітах – 1 Мбіт/с, 800, 500, 250, 125, 50 і 10 кбіт/с. Рекомендовані довжини шини для кожної з цих швидкостей в бодах – 25, 50, 100, 250, 500, 1000, 2500 і 5000 м відповідно (рис. 9.13).

Рівень передачі даних (DLL) можна поділити на дві частини: засіб керування доступом (Medium Access Control) і керування логічним зв'язком (Logical Link Control).

CAN-пристрій, який виявляє помилку в передачі, здатний перервати передачу і послати прапорець помилки. Це запобігає інші вузли від прийняття помилкового повідомлення і запускає спробу повторної передачі відправником. CAN забезпечує механізм визначення спорадичних і систематичних помилок. Це здійснюється відповідно до статистичної оцінки помилок, виявлених у межах кожного вузла з метою виявлення дефектів у конкретному пристрої. При накопиченні помилок будь-яким пристроєм, то він відключається від мережі.

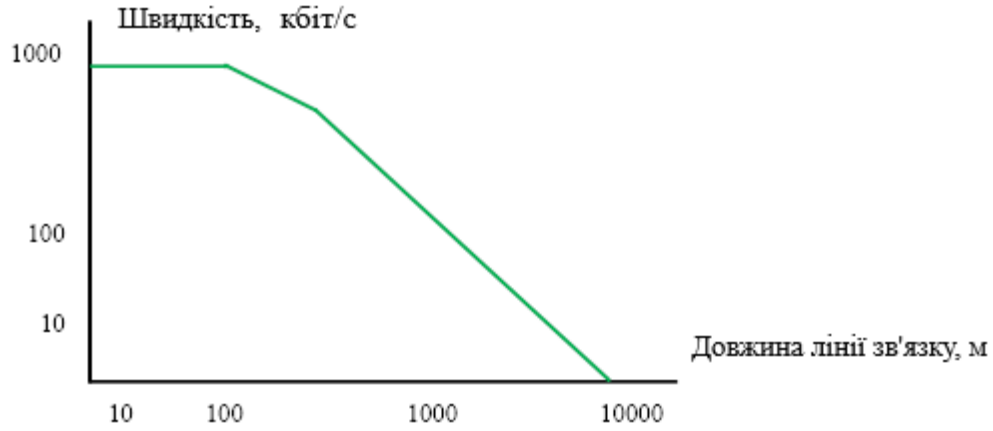


Рисунок 9.13 – Залежність швидкості передачі інформації по CAN-шині залежно від довжини лінії зв'язку

Рівень керування логічним зв'язком (LLC) гарантує, що більш високі рівні в мережі забезпечуються інтерфейсом, який є незалежним від основних рівнів. Іншими словами, цей підрівень повинен гарантувати, що фізичний рівень і засіб керування доступом (MAC) прозорі для верхніх рівнів. Практично це означає, що керування логічним зв'язком (LLC) відповідальне за виявлення і виправлення помилок при обміні даними між пристроями, при якому забезпечується безпомилковий механізм передачі та прийому даних.

Кадр помилок генерується будь-яким вузлом, який виявив помилку. Він складається з двох частин.

Перша частина розглядається, як об'єднання прапорців помилок від різних вузлів. Друга частина – біти роздільників. Тип поля кадру помилок залежить від статусу вузла, який розпізнав помилку. Якщо передавальний вузол сам виявляє помилку на шині, тоді він перериває передачу поточного повідомлення і передає на шину повідомлення про помилки, що складається з шести послідовних «dominant» бітів, таке повідомлення називають прапорцем помилки (рис. 9.14, *а*).

Всі пристрої, що підключені до шини, розпізнають помилку і передають нею свої прапорці помилок. Таким чином, прапорець помилки буде складатися від 6 – 12 бітів, що генеруються одним або декількома вузлами.

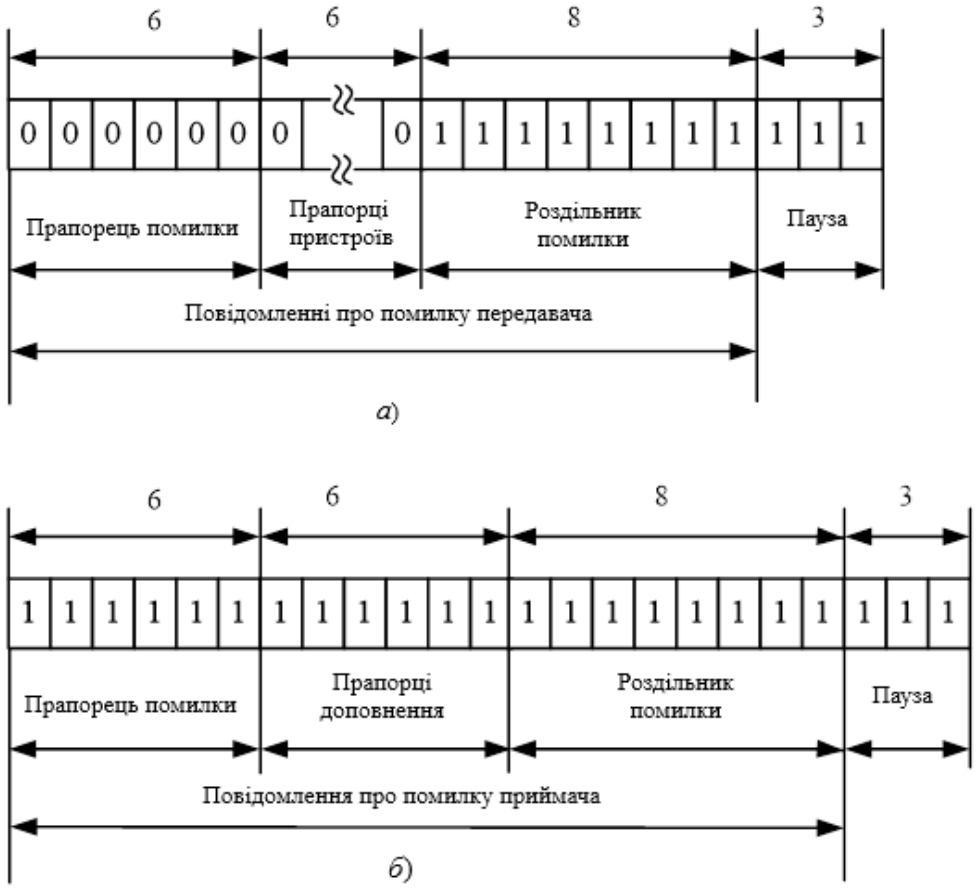


Рисунок 9.14 – Формати повідомлень про помилки

Повідомлення про помилку закінчується полем роздільника з 8 «recessive» бітів, яке називають кадром помилки. Після завершення кадру помилок шина переходить у нормальний режим, і вузол повторює спробу знову передати перерване повідомлення.

Якщо приймаючий вузол визначає помилку на шині, то він формує пасивний прапорець помилки з шести «recessive» бітів, доповнює ще 6 бітів заповнення і формує поле роздільника з 8 «recessive» бітів (рис. 9.14, *б*).

Передача пасивного кадру помилок не порушує роботу інших вузлів, підключених до шини. Після передачі кадру помилок вузол повинен чекати шість послідовних «recessive» бітів, щоб знову підключитися до шини.

Специфікації ISO/OSI рівня 2 недостатньо для обміну корисною інформацією між кількома користувачами та її інтерпретації. Потрібні подальші інструкції, наприклад: які ідентифікатори об’єкта (тобто, пріоритети), до яких сигналів застосовуються, і яку структуру та значення вони мають в полі даних переданих бітів. Ці та подальші інструкції визначаються на більш високих рівнях

протоколу. Вони розроблюються самостійно, або використовують вже існуючі стандартизовані протоколи. В транспортних засобах, як правило, використовуються розроблені виробником закриті протоколи застосування. В інших сферах застосування, перш за все в автоматиці, більш раціонально використовувати стандартизовані та відкриті протоколи застосування, тому що з їх допомогою можна зекономити витрати на вдосконалення і без будь-яких складностей експлуатувати пристрої різних виробників на загальній шині.

На основі специфікації рівня 2 від CAN було розроблено багато відкритих стандартів рівня застосування. Основні відкриті протоколи у сфері автоматизації та інших споріднених сферах – CANopen та DeviceNet.

Протокол CANopen дуже поширений та застосовується в багатьох інших сферах окрім автоматизації.

На рис. 9.15 зображено базовий концепт комунікації пристрою з інтерфейсом CANopen.

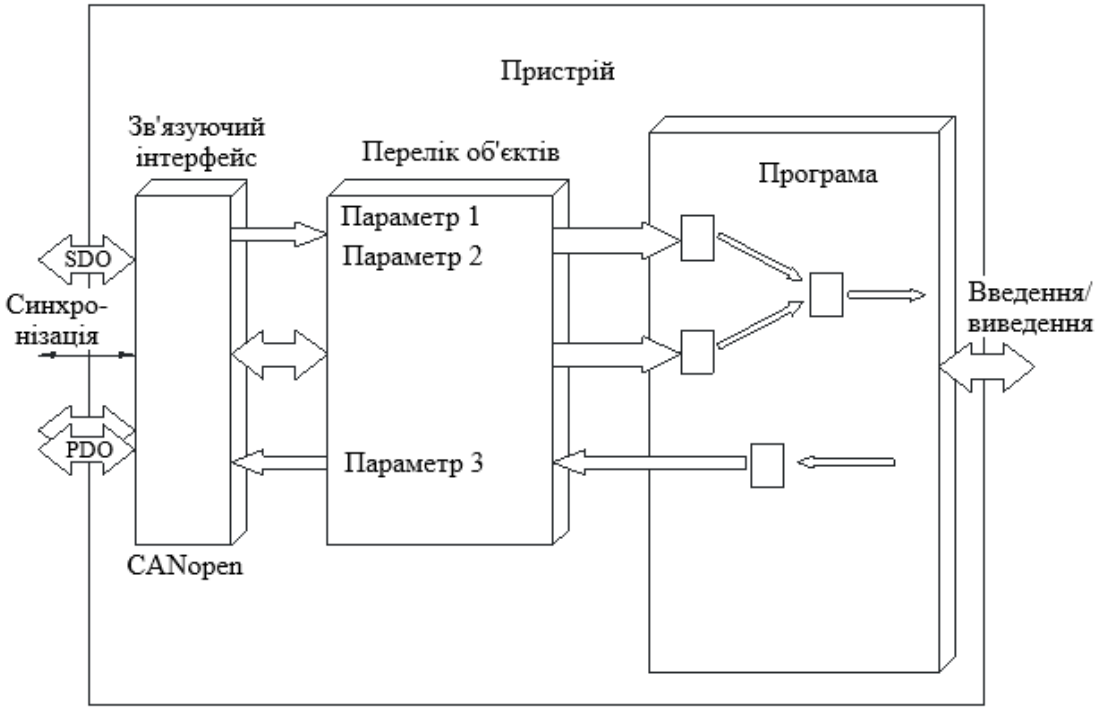


Рисунок 9.15 – Концепція комунікації пристрою CANopen

Протокол CANopen виділяє два основних класи сигналів: Service Data Objects (SDO) з низьким пріоритетом та Process Data Objects (PDO) з високим пріоритетом. Пріоритет, тобто номер ідентифікатора об’єкта, що використовується, вказується в специфікації CANopen для різноманітних пристроїв та застосувань. Разом з цим існують ще спеціальні сигнали з вже визначеними ідентифікаторами, як, наприклад, сигнал синхронізації (SYNC).

Сигнали SDO використовують для передачі великого набору даних, як, наприклад, файлів або наборів параметрів. При цьому дані можуть сегментуватись, тобто поділятись на декілька сигналів. PDO використовують для швидкої передачі невеликих наборів даних з високим пріоритетом сигналу.

В пристрої існує так званий словник об'єктів. Це список, в якому містяться інструкції для структури, значення та інтерпретації переданих корисних даних. Тепер ідентифікатори об'єкта переданих сигналів (SDO, PDO та ін.) можуть використовуватись як індекс у записах словника об'єктів, і завдяки цьому поля даних можуть бути інтерпретовані відповідно до збережених інструкцій. Це звісно стосується не тільки прийнятих сигналів. Також сигнали, котрі мають бути надіслані, кодуються відповідно до попередньо внесених інструкцій. Це дуже великий та універсальний концепт комунікації.

Специфікації CAN open існують для більшої кількості пристроїв різного типу, але основні механізми комунікації (SDO, PDO та ін.) завжди однакові.

9.3.3.3. Основні характеристики CAN

У табл. 9.1 ще раз зібрані основні характеристики мережі CAN.

Таблиця 9.1 – Основні характеристики CAN

Походження	Розроблена компаніями Bosch і Intel для застосування у сфері автоматизації (головним чином), стандартизована в ISO 11898
Ієрархія	Система з кількома ведучими пристроями
Топологія	Лінія (шина)
Швидкість передачі даних	До 1 Мбіт/с
Шинне середовище	ISO 11898, оптоволокно та ін.
Максимальна довжина шини	40 м при 1 Мбіт/с, до 2000 м при швидкості 25 кбіт/с з одним повторювачем сигналу
Доступ до середовища	CSMA/CA (недеструктивний арбітраж)
Кількість вузлів шини	ISO 11898: не вказана кількість; стандартно приблизно 30, розширені драйвери (100 і більше), доступні повторювачі
Адресація	Об'єктно-орієнтована (за ідентифікатором)
Довжина сигналу	Дані от 0 до 8 байтів, загальна – приблизно130 біт
Ефективність кадру даних	0 – 53 % для довжини кадру даних 0 – 8 байтів
Пріоритети	2032 при 11 ідентифікаторах біта, $2032 \cdot 2^{18}$ при 29 бітах ідентифікатора (розширена мережа CAN)
Механізми виявлення та попередження помилок	Контроль за допомогою циклічного надлишкового кода, визначення розрядного рівня, заповнення бітами, визначення довжини інформаційного кадру, обробка помилок в активному режимі, відстань Хемінга
Застосування	Автомобілі (легкові, вантажні, автобуси), промислова та будівельна автоматизація, ткацькі верстати, медичне обладнання, верстати та ін.

9.3.4. Протокол Modbus

Протокол Modbus також є шинною системою, що застосовується в промисловій автоматизації, котра підтримується більшою кількістю фірм. Як середовище передачі даних використовується екранована двопровідна лінія. Устаткуванням для передачі виступає оптимізований інтерфейс RS485 зі швидкістю передачі 1 Мбіт/с. Спосіб отримання доступу до шини Modbus Plus – передача маркера. Топологія являє собою фізичну лінію з максимальною загальною протяжністю 450 м. При використанні повторювачів сигналу кількість користувачів може бути збільшена до 64. Протяжність у такому випадку збільшується до 1800 м. Є можливість створити так звану «двокабельну мережу», завдяки якій користувачі мережі пов'язані між собою подвійною витою парою. За допомогою цього підвищується захищеність від відмов через пошкодження кабелю або помилок за бітами – користувачі постійно контролюють обидві лінії. У подібній техніці кабельного з'єднання повторювачі сигналів мають підключатись завжди між однаковими користувачами.

За допомогою спеціальних повторювачів топологія може бути розширена, зіркоподібна або деревоподібна.

Відповідно до еталонної моделі ISO/OSI у Modbus Plus реалізовано разом з рівнями 1 і 2 також рівень 3. Мережі можуть з'єднуватися шляхом мостів. У такому випадку кожна мережа має свій власний маркер та функціонує незалежно від інших мереж. За допомогою реалізації рівня 3 сигнали можуть передаватись через один або декілька мостів. Міст є активним користувачем шини з двома незалежними одна від одної адресами (рис. 9.16).

Якщо від одної мережі до іншої передається сигнал, то міст отримує відповідний сигнал, зберігає його та передає далі, як тільки сигнал отримав маркер іншої мережі. Таким чином можна посилати сигнали через чотири мости. Передача сигналу через різні мережі має задаватись прикладною програмою. Шляхом раціонального використання мостів можна знизити кількість користувачів мережею і значно підвищити потужність та захищеність від відмов кожної мережі окремо та відповідно до всієї системи в цілому. Адреси користувачів не залежать від їх фізичного положення в мережі, до того ж в одній логічній кільцевій схемі не може бути двох однакових адрес.

Якщо два користувачі в одній кільцевій схемі мають однакові адреси, це передається прикладній програмі.

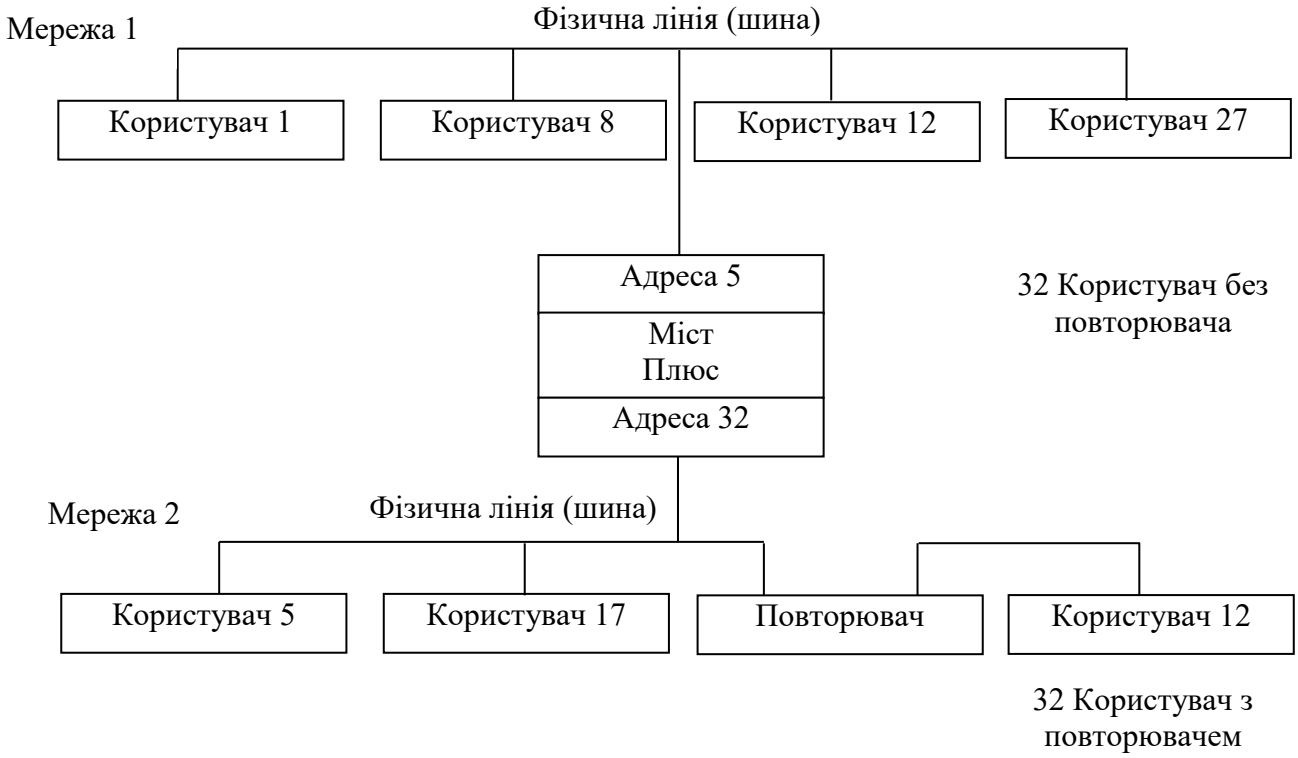


Рисунок 9.16 – Фізична та логічна структура Modbus

Коли інсталюється мережа, кожен користувач складає список користувачів. Кругообіг маркера починається з користувача, що має найнижчу адресу, та передається завжди далі на наступну більш високу за значенням адресу. Користувач з найвищою адресою передає маркер користувачу з найнижчою адресою назад. Під час функціонування користувачі можуть вмикатися або вимикатися. Коли один користувач лишає мережу, генерується новий маркер. Процес ініціалізації триває до 100 мс.

Якщо підключається новий користувач, його буде внесено до списку адрес інших користувачів вже через 15 с (найгірший випадок, зазвичай 5 с). Якщо користувач отримує маркер, то в нього є можливість підключатися до шини. Він може передавати сигнали довжиною максимум 512 біт на одного користувача. Є можливість так зконфігурувати користувача, що він зможе послати в цілому 8 кбіт різним користувачам. На сигнал завжди має надійти відповідь про його отримання. Якщо від відправника не прийнятий сигнал підтвердження, той самий сигнал буде повторюватись. Якщо під час другої спроби сигнал підтвердження відправника не надходить, він передає повідомлення про помилку, яке може бути запитано додатком.

При передачі маркера одночасно може бути надіслано до 512 біт широкомовлених повідомлень. Вони знаходяться у фреймі маркера і зберігаються

в глобальній базі даних з інформацією про їх відправника. У кожному логічному колі є власна база даних, до якої мають доступ тільки користувачі кола.

Після передачі маркера користувач слідкує за шиною. Якщо від отримувача маркера не надходить жодної реакції, то маркер передається вдруге. Якщо отримувач і після цього не реагує, то мережа заново ініціалізується, починається новий кругообіг маркера. Разом з цими механізмами попередження помилок передається також 16-бітний циклічний надлишковий код.

Додатково до модифікації Modbus існує також Modbus TCP/IP на основі оптимізованого з'єднання RS485. Тут як фізичне середовище передачі використовується Ethernet. Застосовується встановлений на нього протокол передачі TCP/IP. Таким чином виникає апаратна сумісність зі стандартними системами Ethernet. Застосування стеків протоколу TCP/IP в програмному забезпеченні сприяє зменшенню витрат на його вдосконалення. У сфері шарів користувачів протоколу також потрібні спеціальні реалізації Modbus.

Вперше Modbus почав використовуватися для передачі інформації від датчиків.

Протокол передбачає роботу одного ведучого пристрою (майстра) і декількох (до 247) ведених пристроїв (slave).

На практиці кількість ведених пристроїв завжди обмежена, наприклад, для одного сегмента інтерфейсу RS485 кількість пристроїв, підключених до сегменту обмежена, числом 32.

Ініціювати початок сеансу зв'язку може тільки ведучий пристрій (майстер). Сеанс зв'язку містить один запит, одну відповідь або один циркулярний запит провідного пристрою до ведених. Відповідь у цьому випадку не відправляється. Особливістю протоколу Modbus є те, що частина параметрів протоколу не змінюється. До них відносяться формат кадру повідомлення, послідовність кадрів, обробка помилок, комунікаційні параметри і команди.

Такі параметри, як швидкість передачі, фізичне середовище передачі, перевірка на парність або непарність, число стопових біт, режим передачі ASCI1 або RTU визначаються користувачем.

Кадр повідомлення (рис. 9.17) містить адресу одержувача, яку операцію має виконати одержувач, команду, дані для виконання команди і метод контролю достовірності передачі інформації.

Ведений пристрій отримує повідомлення, перевіряє його на правильність передачі, виконує команду і формує відповідь на повідомлення, яке було надіслане і має точно такий формат.

Адреса одержувача 1 байт	Команда 1 байт	Дані 0–252 байта	CRC контрольна сума 2 байти

Рисунок 9.17 – Кадр повідомлення протоколу Modbus

Протокол Modbus передбачає два режими роботи. Режим ASCI1, коли для кодування інформації використовується символ 0-9, A-F, і режим RTU (Remote Terminal Unit), коли інформація кодується в двійковій системі. Довжина кодової комбінації містить 8 біт. Виходячи з цього в режимі ASCI1 дані подані як 7-бітові числа, в RTU – 8-бітові.

Стандарт Modbus як фізичний рівень рекомендує використовувати інтерфейс RS485 з магістральним трипровідним кабелем в екрані. Два проводи витої пари використовуються для лінії зв'язку, третій провід – земля, що з'єднує виводи земля всіх пристроїв. Цей провід заземлюється в одній точці біля ведучого пристрою. Відповідно до стандарту на кінцях магістрального проводу обов'язково, незалежно від швидкості передачі, встановлюються термінальні резистори величиною $R_т = 150$ Ом і потужністю 0,5 Вт.

Резистори, які погождують волновий опір ліні зв’язку ($R_c = 450...650$ Ом), встановлюються тільки в одному місці.

Всі пристрої в мережі повинні забезпечувати швидкості обміну 9600 і 19200 біт/с, можливі швидкості передачі 1200, 2400, 4800, ..., 38400 біт/с, а також 65 і 115 кбіт/с. Допустимі коливання швидкості передачі не повинні перевищувати 1 %.

При цьому має підтримуватися стійкий прийом інформації при коливаннях швидкості передачі до 2 %.

Стандарт Modbus передбачає навіть колірну розмітку жил кабелю:

– жовтий – для позитивного проводу, який приєднується до «+» джерела;

– коричневий – для проводу, по якому передається логічний нуль;

– сірий – заземлення.

Повідомлення відповідно до стандарту містить 11 біт (рис. 9.18).

Старт біт	0	1	2	3	4	5	6	7	Біт парності	Стоп біт

а)

Старт біт	0	1	2	3	4	5	6	7	Стоп біт	Стоп біт

б)

Рисунок 9.18 – Структура повідомлення Modbus

Починається передача повідомлення завжди зі старт-біта (перехід з одиниці в нуль), рис. 9.18, *а*, потім передаються 8 біт, починаючи з молодшого біта, після чого встановлюється біт-парності і стоп-біт. Взагалі перевірка може проводитися як на парність, так і на непарність. Якщо в даному байті міститься непарне число одиниць, біт-парності встановлюється в одиницю. При цьому відбувається контроль парності. При парному числі одиниць у переданому байті біт парності встановлюється в нуль, при цьому відбувається контроль непарності.

За відсутності контролю парності в посилці замість молодшого розряду встановлюється другий стоп біт (рис. 9.18, *б*).

Повідомлення в Modbus передаються повідомленнями або фреймами. Послідовність передачі байтів у фреймі наведена на рис. 9.17.

Першим передається байт адреси, потім байт команди, в якому вказується, які функції має виконати ведений пристрій. За байтом команди йдуть байти даних, число яких може змінюватися від 0 до 252. Наприкінці фрейма слідують два байти контрольної суми, призначених для перевірки правильності прийнятого фрейма. Таким чином, довжина фрейма є змінною величиною, але не більше 256 байт.

Передача фреймів здійснюється із забезпеченням тимчасової синхронізації. Графік часового аналізу передачі фреймів наведено на рис. 9.19. Фрейми передаються через інтервали часу $T_{\text{п}} = 3{,}5\ t_{б}$, де $t_{б}$ час передачі одного байта. При швидкості передачі 9600 біт/с тривалість одного біта $t_{біт} = 0{,}1$ мс, тоді як $t_{б} = t_{біт} 11 = 1{,}14$ мс, а час паузи $T_{\text{п}} = 3{,}5\ t_{б} = 4$ мс. Початок передачі визначається пристроєм Modbus, який отримав команду на передачу даних.

При цьому пристрій виявляє наявність паузи $T_{\text{п}}$ і починає передачу фрейму. Закінчення передачі фрейма контролюється виявленням затримки після передачі контрольної суми, яка: $t_{з} = 1{,}5\ t_{б} = 1{,}7$ мс.

Якщо після закінчення часу і до закінчення паузи починається передача будь-яким пристроєм, то цей кадр вважається помилковим, і це повідомлення передається повторно і помилка фіксується у відповіді на запит.

Адреси пристроїв у Modbus визначені від 1 до 247. Адреси 248 – 255 зарезервовані.

Провідний пристрій не має адреси, оскільки ведений пристрій може передавати інформацію тільки ведучому.

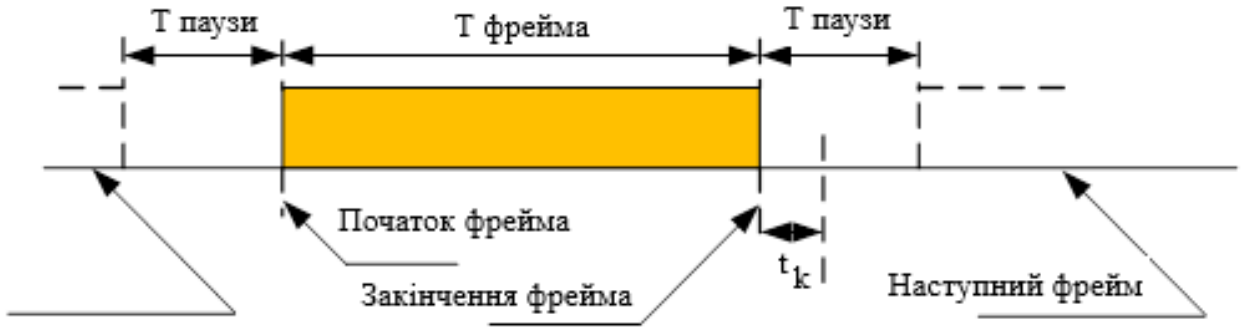


Рисунок 9.19 – Графік часового аналізу синхронізації передачі фреймів у протоколі Modbus

При широкомовному режимі встановлюється нульова адреса і при цьому всі ведені приймають інформацію. Коди команд поділяються на встановлені стандартом, визначені користувачем і зарезервовані.

Стандарт передбачає коди команд від 1 до 127. Коди, визначені користувачем, знаходяться в межах 65 – 72 і 100 – 110. Коди від 128 до 255 зарезервовані як коди помилок при формуванні відповідних повідомлень веденими пристроями.

Повний перелік команд протоколу Modbus наведено в стандарті.

Основними є команди читання і запису. При читанні мають бути вказані номер датчика, номер регістра або адреси осередків пам'яті, а при записі – вигляд змінюваного параметра, інтервал зміни вимірюваної величини і т.ін.

При передачі коду команди від ведучого до веденого пристрою, в полі даних при читанні необхідно вказувати початкову адресу входів, назва регістрів, адреси осередків пам'яті, кількість входів, регістрів і осередків пам'яті, в яких зберігаються значення вимірюваних параметрів.

Послідовність алгоритму формування контрольної суми CRC при передачі повідомлення така:

– у двобайтний регістр CRC завантажується число FFFFh (табл. 9.2);

– в пам'ять завантажується число, яке подається поліномом циклічного коду;

– перший байт повідомлення складається по модулю 2 (виключне АБО) з регістром CRC, результат залишається в регістрі CRC, зсувається у бік молодшого біта на один біт, при цьому старший біт регістра CRC заповнюється нулем (табл. 9.2).

Якщо при цьому молодший біт регістра CRC дорівнює нулю, то виконується наступний зсув на один біт. Якщо молодший розряд регістра CRC

дорівнює одиниці, то виконується операція додавання по модулю 2 вмісту регістра CRC і числа, яке відповідає поліному.

Як видно з таблиці, після передачі першого байта в регістрі CRC зберігається число 9BC2h. Такий принцип формування контрольної суми здійснюється при передачі всіх байт повідомлення. Сформована контрольна сума у вигляді двох байтів передається в кінці повідомлення. На приймальній стороні виконується зворотне перетворення, і отримане число порівнюється з числом CRC в повідомленні.

Таблиця 9.2 – Алгоритм перетворення регістром контрольної суми (CRC) при передачі одного байта

1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	Поліном
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Початкове CRC
0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1-й байт
1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	CRC_1+1 байт
0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	0	1-й зсув
0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	2-й зсув
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	Поліном
1	0	1	1	1	1	1	0	1	1	0	0	1	1	1	0	CRC_2+ поліном
0	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	3-й зсув
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	Поліном
1	1	1	1	1	1	1	0	0	1	1	0	0	1	1	0	CRC_3 +поліном
0	1	1	1	1	1	1	1	0	0	1	1	0	0	1	1	4-й зсув
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	Поліном
1	1	1	1	1	1	1	0	0	0	1	1	0	0	1	0	CRC_4+ поліном
0	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1	5-й зсув
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	Поліном
1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	0	CRC_5+ поліном
0	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	6-й зсув
0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	0	7-й зсув
0	0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	8-й зсув
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	Поліном
1	0	0	1	1	1	1	0	1	1	0	0	0	0	1	0	CRC_8+ поліном
9				B				C				2				

Контрольні питання

1. Скільки об’єктів можна підключити до шини?
2. Перелічити особливості послідовних шин.
3. Які різновиди польових шин ви знаєте?
4. Інтерфейс RS232. Назвати приклади його використання та логічні рівні.
5. Скільки користувачів можна підключити до інтерфейсу RS485?
6. Призначення шини I^2C.
7. Скільки ліній необхідно для роботи шини I^2C?
8. Як кодуються умови початку та кінця передачі по шині I^2C?
9. Як виконується арбітраж CAN-шини?
10. Які протоколи зв’язку використовують CAN-шину?
11. Яким чином отримують доступ до шини Modbus?
12. Яку топологію може мати шина Modbus?

10. ПРАКТИЧНА ЧАСТИНА

10.1. Правила запису програми на мові «Асемблер»

Оригінальний текст програми на мові «Асемблер» має певний формат. Кожна команда і директива є рядком:

МІТКА ОПЕРАЦІЯ Операнд(и) КОМЕНТАРІ

Поля можуть відділятися один від одного довільним числом пробілів і табуляцією.

Мітка. В полі мітки розміщується символічне ім'я осередка пам'яті, в якій зберігається вказана команда або операнд. Мітка являє собою буквено-цифрову комбінацію, що починається з букви. Використовуються тільки букви латинського алфавіту. «Асемблер» MCS-51 допускає використання в мітках символу підкреслення (_). Мітки ставляться в будь-якій колонці, якщо ім'я закінчується двокрапкою. Якщо двокрапка не використовується, мітка повинна починатися з першої колонки. Великі й маленькі букви вважаються різними.

Операція. В полі операції записується мнемонічне позначення команди або директиви асемблера, яке є скороченням (абревіатурою) повного англійського найменування виконуваної дії. Наприклад: MOV – move – перемістити, JMP – jump – перейти, DB – define byte – визначити байт.

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

Операнд може бути заданий константою або у вигляді його адреси (прямої або непрямої). Константа представляється числом (MOV A, # 15) або символічним ім'ям (ADDC A, # OPER2) з обов'язковим префіксом константи (#). Прямий доступ операнда може бути заданий мнемонічним позначенням (IN A, P1), числом (INC 40), символічним ім'ям (MOV A, MEMORY). Вказівкою на непряму адресацію служить префікс @. У командах передачі керування операндом може бути число (LCALL 0135h), мітка (JMP LABEL), непряма адреса (JMP @A) або вираз (JMP $ – 2, де $ – поточний зміст лічильника команд).

Операнди, символічні імена і мітки, які використовуються, мають бути визначені, а числа наведені із зазначенням системи числення, для чого використовується суфікс (буква, що стоїть після числа): B – для двійкової, Q – для вісімкової, D – для десяткової і h – для шістнадцяткової. Число без суфікса за замовчуванням вважається десятковим.

«Асемблер» MCS-51 допускає використання виразів у полі операндів, значення яких обчислюються в процесі трансляції. Вираз являє собою сукупність символічних імен і чисел, пов'язаних операторами асемблера. Оператори асемблера забезпечують виконання арифметичних ("+" – додавання, "–" – віднімання, "·" – множення, ":" – ціле ділення, MOD – ділення по модулю) і логічних (OR – АБО, AND – І, XOR – виключення АБО, NOT – заперечення) операцій у форматі 2-байтних слів. Наприклад, запис SUBB A, # ((NOT 27) +1) еквівалентний запису SUBB A, # 0E5h і забезпечує віднімання від вмісту акумулятора числа – 27, наданого в додатковому коді. Широко використовуються також оператори LOW і HIGH, що дозволяють обчислити молодший і старший байти 2-байтного операнда.

Коментарі завжди починаються зі символа (;) і служать для запису пояснень до тексту програми. Коментарі ігноруються транслятором і тому не впливають ні на процес трансляції, ні на вихідний код.

Директиви транслятора дозволяють дати символічне визначення змінним, резервують і ініціалізують простір пам'яті, визначають розташування згенерованого об'єктного коду в пам'яті.

EQU призначає символічному імені числове значення або ім'я регістра.

Приклад:

Sensor_level EQU 80h ;призначає символу Sensor_level адресу біта порту P0.0

SET призначає символічне ім'я числовим значенням або регістра. Ім'я може бути згодом змінено за допомогою директиви SET.

DB заносить у пам'ять програм байтову константу.

Приклад:

DB 20,0Fh,35h

DW ініціалізує пам'ять значенням слова.

Приклад:

DW 0457h,0F345h,789h

ORG змінює значення асемблерного лічильника адреси поточного сегмента програми.

Приклад:

ORG 100h; встановлює лічильник адреси в значення 100h.

END повідомляє про кінець трансльованого модуля.

10.2. Лабораторна робота 1

Ознайомлення із середовищем розробки і налагодження програмного забезпечення для контролерів серії MCS-51.

Мета роботи: вивчити технологію налагодження програмного забезпечення MCS-51, команди пересилань із прямою адресацією та арифметичних і логічних операцій з прямою адресацією.

Загальні відомості. Програма для мікроконтролера являє собою послідовність машинних інструкцій (кодів операцій), розглянутих у розділі «система команд мікроконтролера». Однак написання програми у вигляді послідовності двійкових значень кодів операцій досить трудомісткий процес і погано піддається редагуванню. Тому для автоматизації праці програмістів розроблені транслятори з мов різних рівнів. Мовою найнижчого рівня є «Асемблер». Фактично – це запис команд у вигляді мнемокоду (текстового опису) машинних інструкцій процесора плюс підтримка різних директив транслятора. Розробка програми для мікроконтролера мовою «Асемблер» полягає в такому:

1. Складання тексту програми у вигляді мнемокоду процесора.
2. Трансляція в машинні інструкції процесора.
3. Налагодження програми на програмному симуляторі мікроконтролера.
4. Запис налагодженої програми в мікроконтролер.
5. Перевірка функціонування програми на реальному об'єкті.

Ця лабораторна робота буде виконуватися в інтегрованій системі MCStudio. Вона являє собою інтегровані в одну оболонку такі компоненти: текстовий редактор, компілятор, програмний симулятор мікроконтролера, формувач моделі зовнішнього оточення контролера. Прикладна програма може бути виконана як у вигляді одного файлу (модуля), так і у декількох модулів. Крім цього, наприклад, у процесі трансляції, налагодження і т.ін. система створює додаткові службові файли. Для керування цією сукупністю файлів використовується концепція проєкту.

Методика виконання

1. Проект створюється з головного меню MCStudio послідовним вибором пунктів Файл – Створити – Проєкт. Для вказівки місця розташування проекту, імені проекту й модулів, типу процесора заповнюється форма, наведена на рис. 10.1.

У результаті на екрані відкриваються вікна: «Проєкт», «Ресурси» і вікно редактори тексту програми із прописаним шляхом до вихідного модуля програми (рис. 10.2).

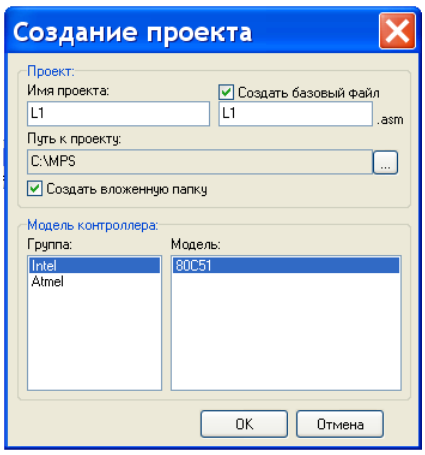


Рисунок 10.1 – Вікно створення проєкту

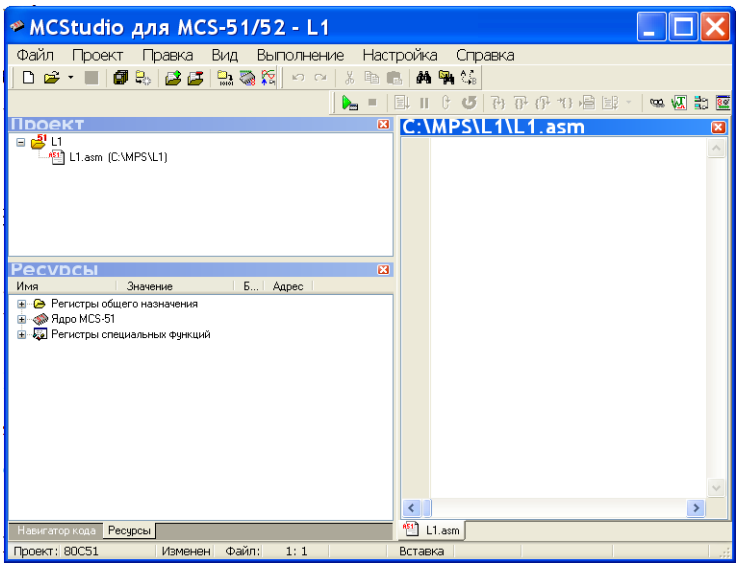


Рисунок 10.2 – Екран MCStudio після створення проєкту

1. У текстовому редакторі набираємо текст програми, наведений у лабораторній роботі. При наборі тексту необхідно дотримуватися таких правил:

– перші 8 позицій у рядку – це поле міток;

– за ними розташовується поле команд;

– за полем команд розташовується поле операндів;

– за полем операндів може розташовуватися поле коментарів, які відокремлюються від інших полів крапкою з комою.

2. За описом команд, наведених в «Робочому зошиті № 1», вивчаємо їх роботу.

3. Компілюємо програму. Для цього натискаємо комбінацію клавіш Ctrl+F9 або кнопку на панелі інструментів  . У результаті компіляції у вікні текстового редактора додається дві колонки ліворуч, рис. 10.3. Перша колонка містить адресу, за якою розташовуються машинні коди мікроконтролера. Друга колонка – це значення машинних кодів.

Якщо в програмі були синтаксичні помилки, компілятор виявляє і видає повідомлення про характер помилки у вікні повідомлень (рис. 10.4). Якщо у вікні повідомлень на помилку клацнути лівою кнопкою миші, то курсор буде поміщено у рядок програми, де була виявлена помилка.

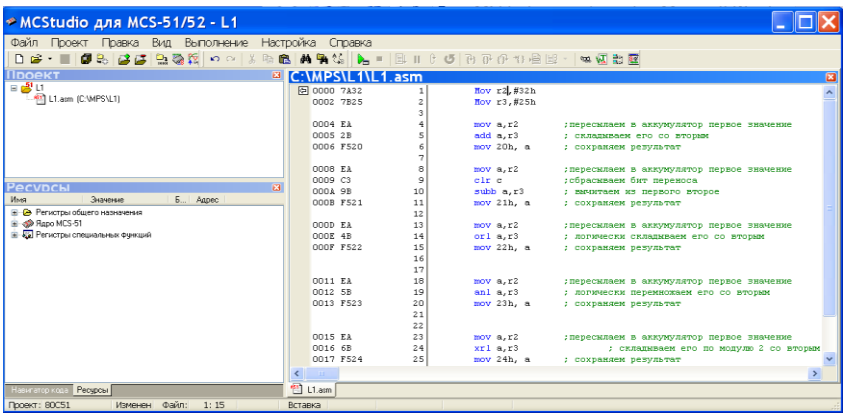


Рисунок 10.3 – Результат компіляції програми

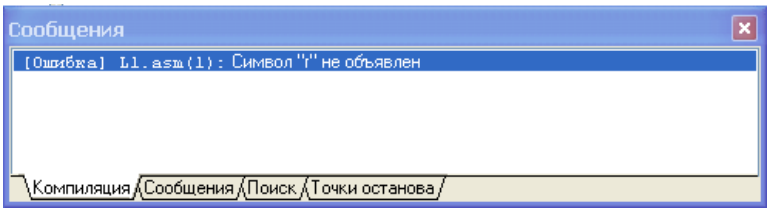


Рисунок 10.4 – Вигляд вікна повідомлень про помилку

4. Виконуємо симуляцію відкомпільованої програми в покроковому режимі. Симуляція завжди починається з натискання кнопки  або клавіші F9. Після натискання синя смуга в тексті програми вказує, яка команда буде виконуватися на наступному кроці. Черговий крок програми при покроковому налагодженні виконується натисканням клавіші F7. Покрокове налагодження необхідне, для того, щоб переконатися, чи однаково виконують команди контролер і програміст.

5. Результати виконання команд переглядаємо у вікні відображення дерева ресурсів мікроконтролера (рис. 10.5).

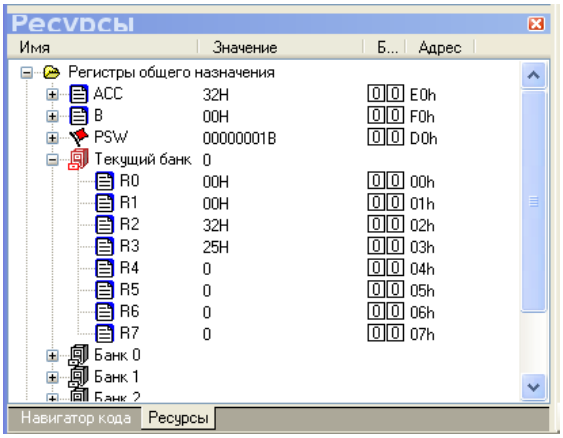


Рисунок 10.5 – Дерево ресурсів контролера на шостому кроці виконання програми

6. Підставляємо в програму значення згідно з варіантом, за результатами симуляції підготовляємо звіт лабораторної роботи.

Примітка: після закінчення симуляції натисніть клавіші Ctrl+ F2, щоб вийти з режиму симуляції. Без цього ви не зможете закрити програму MCStudio та зберегти свої дані.

Теоретичні відомості. Нижче наведені фрагменти вікон робочого середовища Mcstudio і пояснення щодо виконання команд пересилань із прямою адресацією.

Пересилання константи (рис. 10.6).

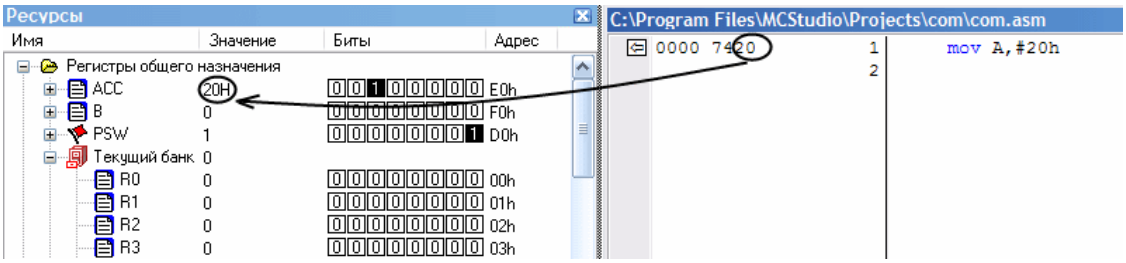


Рисунок 10.6 – Виконання команди mov A,# 20h

Команда MOV A,# d виконує пересилання константи, яке зберігається у другому байті команди, в акумулятор.

Пряма адресація (рис. 10.7).

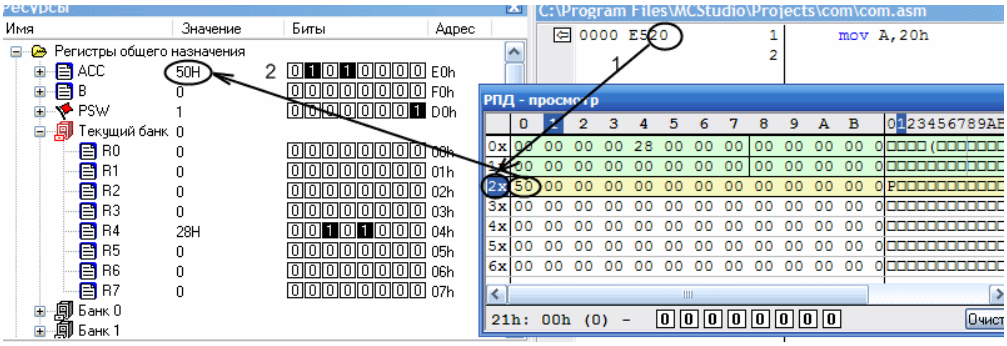


Рисунок 10.7 – Виконання команди mov A,20h

Команда MOV A,ad виконує пересилання даних із внутрішньої пам'яті в акумулятор. Адреса осередка пам'яті перебуває в другому байті команди (рис. 10.8).

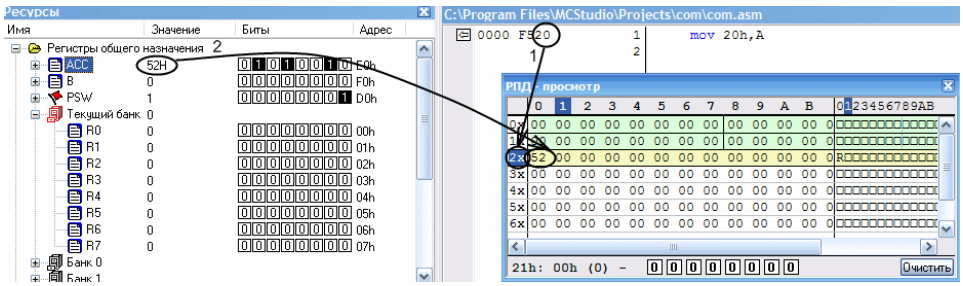


Рисунок 10.8 – Виконання команди mov 20h,A

Команда MOV ad, A виконує пересилання даних з акумулятора у внутрішню пам'ять. Адреса осередка пам'яті перебуває в другому байті команди.

Пряма регістрова адресація (рис. 10.9).

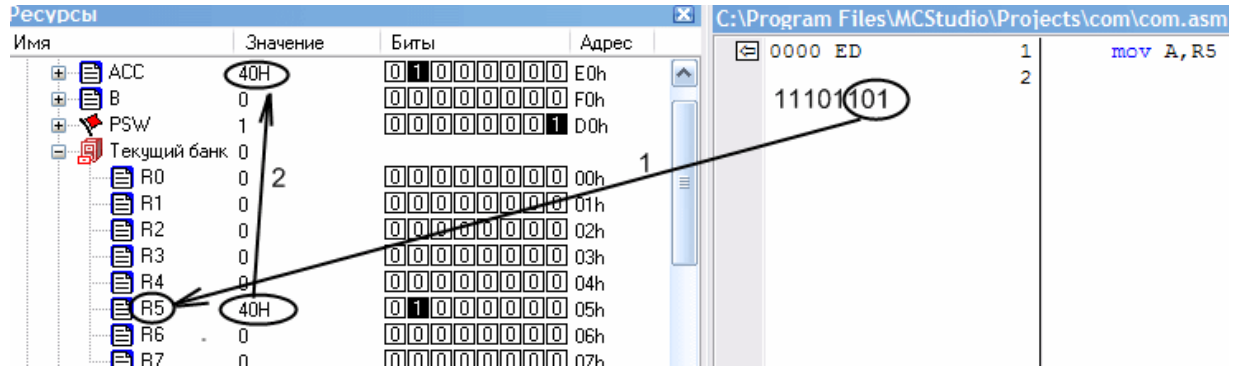


Рисунок 10.9 – Виконання команди mov A,R5

Команда MOV A,Rn виконує пересилання значення регістра поточного банка в акумулятор. Адреса регістра зазначена в молодших трьох бітах коду операції (рис. 10.10).

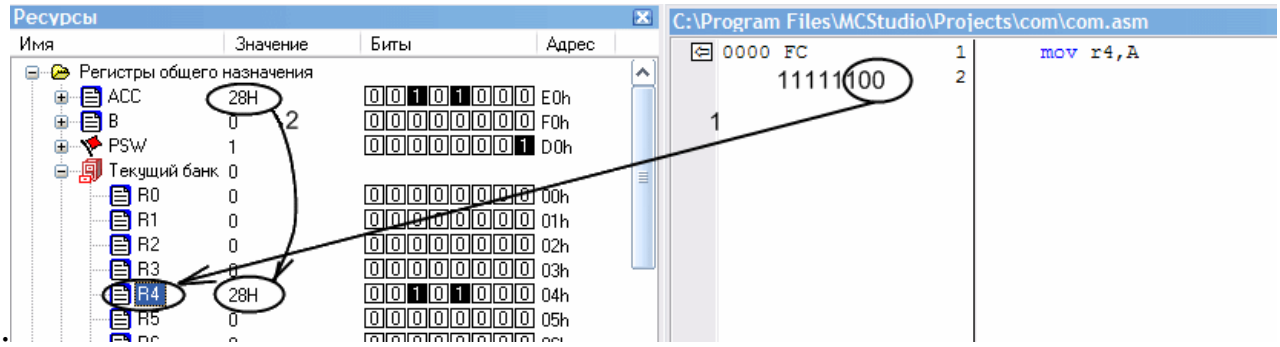


Рисунок 10.10 – Виконання команди mov R4, A

Команда MOV Rn,A виконує пересилання значення акумулятора в регістр поточного банка. Адреса регістра зазначена в молодших трьох бітах коду операції.

Завдання лабораторної роботи. Написати програму роботи мікроконтролера МК51, яка виконує арифметичні і логічні дії над двома числами, поміщеними в регістри R2 і R3, і результати записує у внутрішню пам'ять даних за адресою 20h.

Текст програми мовою «Асемблер»

```
Mov r2,#32h        ; занесення в регістри вихідних значень
Mov r3,#25h

mov a,r2      ; пересилаємо в акумулятор перше значення
add a,r3      ; складаємо його із другим
```

```
mov 20h, a   ; зберігаємо результат
mov a,r2     ; пересилаємо в акумулятор перше значення
clr c        ; скидаємо біт переносу
subb a,r3    ; віднімаємо від першого друге
mov 21h, a   ; зберігаємо результат

mov a,r2     ; пересилаємо в акумулятор перше значення
orl a,r3     ; логічно складаємо його із другим
mov 22h, a   ; зберігаємо результат

mov a,r2     ; пересилаємо в акумулятор перше значення
anl a,r3     ; логічно перемножуємо його із другим
mov 23h, a   ; зберігаємо результат

mov a,r2     ; пересилаємо в акумулятор перше значення
xrl a,r3     ; складаємо його за модулем 2 із другим
mov 24h, a   ; зберігаємо результат
end          ; кінець програми.
```

Вихідні дані для програми:

– перше значення дорівнює номеру за списком, помноженому на 5;

– друге значення дорівнює номеру за списком, помноженому на 7.

Звіт повинен містити текст програми і результати виконання кожної операції.

10.3. Лабораторна робота 2

Дослідження внутрішньої і зовнішньої пам'яті даних і пам'яті програм

Мета роботи: закріпити знання студентів при роботі з налагоджувачем, вивчення команд пересилань із непрямою адресацією, укажчиків та роботи з масивами.

Теоретичні відомості. Пам'ять даних необхідна для прийняття, зберігання і видачі інформації, використовуваної в процесі її виконання. Вона програм призначена для зберігання машинних кодів та констант, зберігання і видачі інформації, використовуваної в процесі виконання програми.

Для звертання до зовнішньої пам'яті даних використовується команда MOVX. Є два типи команд, що відрізняються забезпеченням 8-бітової або 16-бітової непрямої адреси до зовнішнього ОЗП даних. У першому випадку необхідно використовувати регістри R0 і R1 банків регістрів, вміст яких забезпечує 8-бітову адресу, яка мультиплексується з даними порту P0. У другому випадку при виконанні вищевказаної команди покажчик даних DPTR генерує 16-бітову адресу.

Команда MOVC A,@A+dptr здійснює звертання до пам'яті програм і пересилає з неї байт кода в акумулятор. Адреса зчитуваного байта обчислюється як сума 8-бітового вихідного вмісту акумулятора без знака і вмісту DPTR. Тому в лабораторній роботі перед звертанням до ПП рекомендується скинути вміст акумулятора.

При асемблерному програмуванні мікроконтролерів групи МК51 здійснювати бітову адресацію можна, указавши адресу біта або безпосередньо сам біт. Наприклад, запис JNB A.4,M1 або JNB E4h,M1 (адреса четвертого біта акумулятора) буде сприйматися налагоджувачем однаково.

Нижче наведені фрагменти вікон робочого середовища MCStudio і пояснення до виконання команд пересилання із непрямою адресацією.

Непряма адресація (рис. 10.11).

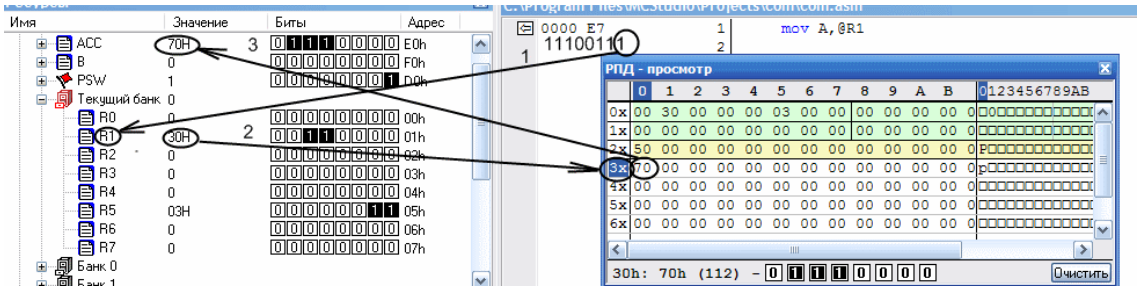


Рисунок 10.11 – Виконання команди mov A,@R1

Команда MOV A,@Ri виконує пересилання в акумулятор вмісту осередка внутрішньої пам'яті даних (рис. 10.12). Адреса регістра (стрілка 1) вказується в молодшому біті кода операції. Адреса осередка (стрілка 2) перебуває в регістрі-покажчику. Регістром-покажчиком можуть виступати тільки регістри R0 і R1.

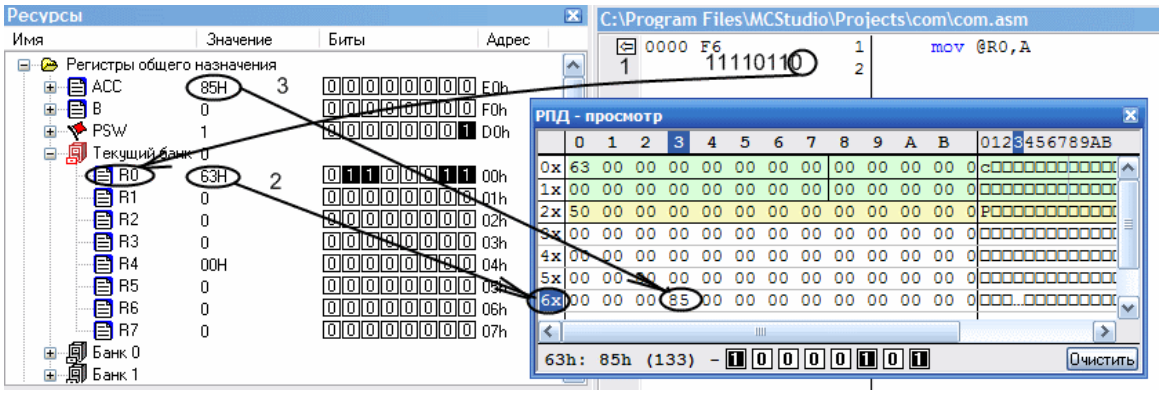


Рисунок 10.12 – Виконання команди mov @r0,A

Команда MOV @Ri,A виконує пересилання вмісту акумулятора в осередок внутрішньої пам'яті даних. Адреса осередка перебуває в регістрі-покажчику. Регістром-покажчиком можуть виступати тільки регістри R0 і R1.

Приклад завдання. Написати програму для мікроконтролера MCS-51, яка складає за модулем два елементи двох масивів довжиною 10 байтів. Перший масив перебуває у внутрішній пам'яті даних контролера, починаючи за адресою 60h. Другий – у пам'яті програм контролера, починаючи за адресою 40h. Результат додавання за модулем масивів записується починаючи з 50h зовнішньої пам'яті. При роботі із зовнішньою пам'яттю використовується однобайтова адресація (рис. 10.13).

Примітка. При написанні програми задається лише один масив, що перебуває в пам'яті програм, починаючи за адресою 40h. Другий масив можна задати безпосередньо в процесі роботи з налагоджувачем, записавши елементи масиву в заданій області внутрішньої пам'яті даних (у цьому разі – за адресою 60h) і зберігши їх в окремому файлі. Результат виконання програми значення вихідного масиву можна побачити у вікні зовнішньої пам'яті даних (ЗПД). Для цього в головному меню необхідно вибрати пункт Вид – Внутрішня пам'ять даних.

Текст програми мовою «Асемблер»

```
 org 00h
 MOV DPTR,#M3 ; завантаження покажчика даних адресою масиву, що перебуває в ПП
 MOV R0,#60h      ; початкова адреса масиву, що перебуває у ВПД
 MOV R1,#50h      ; початкова адреса масиву в зовнішній ПД
 MOV R3,#0Ah      ; кількість елементів масивів
M2:  CLR A              ; скидання вмісту акумулятора
 MOVC A,@A+dptr  ; додавання за модулем два n-х елементів масивів
 XRL A,@R0       ; і пересилання результату до зовнішньої пам’яті
 MOVX @R1,A
 INC R1          ; інкремент адреси масивів
 INC R0
 INC dptr
 DJNZ R3,M2      ; перевірка масивів на їхнє закінчення
M1: LJMP M1      ;програмна пастка в кінці програми
      org 40h
M3:  DB 01h,12h,43h,36h,0feh,37h,63h
      DB 80h,0ach,91h          ; завдання масиву в пам'яті програм
End
```

Звіт має містити текст програми, текст індивідуального завдання і блок-схему алгоритму, результат виконання програми.

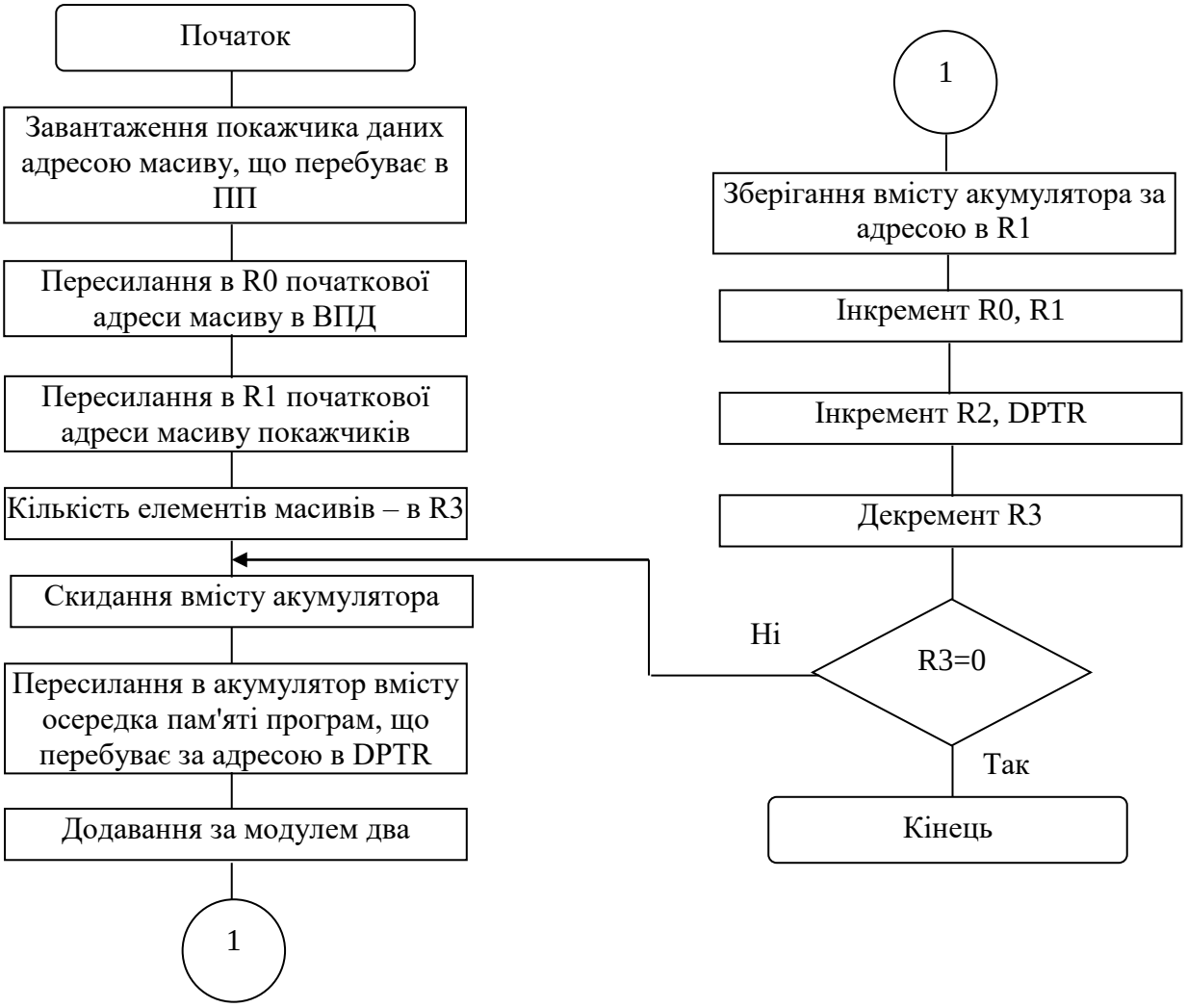


Рисунок 10.13 – Схема алгоритму програми складання за модулем двох масивів

Індивідуальні завдання

Варіант 1. Написати програму, яка обчислює двобайтову суму елементів масиву, розташованого в пам'яті програм за адресою 50h. Довжина вхідного масиву – 12 однобайтових елементів.

Варіант 2. Написати програму, яка виконує множення елементів масиву, розташованого в пам'яті програм, за адресою 80h на 25, і коли двобайтовий добуток зберігає у вигляді масиву у внутрішній пам'яті даних за адресою 30h. Довжина вхідного масиву – 8 однобайтових елементів.

Варіант 3. Написати програму, яка виконує сортування елементів масиву, розташованого в пам'яті програм за адресою 70h. Парні елементи зберігаються у внутрішній пам'яті даних за адресою 30h, а непарні – в зовнішній пам'яті даних за адресою 50h. Довжина вхідного масиву – 10 однобайтових елементів.

Варіант 4. Написати програму, яка виконує ділення елементів масиву, розташованого в пам'яті програм за адресою 80h, на 3. Ціле від ділення

зберігається у внутрішній пам'яті даних за адресою 50h, а залишок – у зовнішній пам'яті даних за адресою 20h. Довжина вхідного масиву – 15 однобайтових елементів.

Варіант 5. Написати програму, яка виконує додавання за модулем 2 елементів масиву, розташованого в пам'яті програм за адресою 20h з вмістом осередка пам'яті за адресою 55h. Довжина вхідного масиву – 12 однобайтових елементів. Результат розмістити за адресою 30h у внутрішній пам’яті даних.

Варіант 6. Написати програму, яка виконує додавання елементів масиву, розташованого в пам'яті програм за адресою 100h з вмістом осередка пам'яті за адресою 25h, і результат зберігає у вигляді масиву у внутрішній пам'яті за адресою 30h. Довжина вхідного масиву – 10 однобайтових елементів.

Варіант 7. Написати програму, яка виконує сортування елементів масиву, розташованого в пам'яті програм за адресою 50h. Негативні елементи зберігаються у внутрішній пам'яті даних за адресою 20h, а позитивні – у зовнішній пам'яті даних за адресою 60h. Довжина вхідного масиву – 10 однобайтових елементів. Ознакою від’ємного числа є 1 в D7.

Варіант 8. Написати програму, яка виконує множення елементів масиву, розташованого в пам'яті програм за адресою 80h, на 5. Результат зберігається у внутрішній пам'яті даних за адресою 50h. Довжина вхідного масиву – 15 однобайтових елементів.

Варіант 9. Написати програму, яка виконує логічне додавання елементів масиву, розташованого в пам'яті програм за адресою 30h із константою A5h. Довжина вхідного масиву **–** 9 однобайтових елементів. Результат розмістити за адресою 40h у зовнішній пам’яті даних.

Варіант 10. Написати програму, яка виконує сортування елементів масиву, розташованого в пам'яті програм за адресою 50h. Елементи, що більше 30h, зберігаються у внутрішній пам'яті даних за адресою 20h, інші – в зовнішній пам'яті даних за адресою 60h. Довжина вхідного масиву – 10 однобайтових елементів.

10.4. Лабораторна робота 3

Написання програми керування комбінаційним автоматом, алгоритм роботи якого заданий таблицею істинності

Мета роботи: вивчити методи використання теорії цифрових автоматів для формалізації написання програм комбінаційних автоматів табличним способом.

Вихідні дані. Вхідні сигнали для семисегментного індикатора надходять на розряди порту P1.0-P1.3, а керування індикатором здійснюється з порту 2 через буферний підсилювач.

Треба написати програму перекодування двійкового значення, яке надходить з порту P1, у значення, що відповідають цифрам шістнадцяткового алфавіту, котрі відображаються на семисегментному індикаторі. Підключення виходів порту P2 до індикатора зображено на рис. 4.10.

Теоретичні відомості. Дешифратори будуються на основі масивів вихідних сигналів, розташованих у пам'яті програм. Звертання до потрібного елемента масиву здійснюється шляхом завдання зсуву відносно початкової адреси масиву.

Розв'язування: Запишемо таблицю істинності для дешифратора (табл. 10.1).

Таблиця 10.1 – Значення істинності для дешифратора

Входи				Виходи								Символ
P1.3	P1.2	P1.1	P1.0	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	
1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	0	0	0	0	1	1	0	1
0	0	1	0	0	1	0	1	1	0	1	1	2
0	0	1	1	0	1	0	0	1	1	1	1	3
0	1	0	0	0	1	1	0	0	1	1	0	4
0	1	0	1	0	1	1	0	1	1	0	1	5
0	1	1	0	0	1	1	1	1	1	0	1	6
0	1	1	1	0	0	0	0	0	1	1	1	7
1	0	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	0	1	1	0	1	1	1	1	9
1	0	1	0	0	1	1	1	0	1	1	1	A
1	0	1	1	0	1	1	1	1	1	0	0	B
1	1	0	0	0	0	1	1	1	0	0	1	C
1	1	0	1	0	1	0	1	1	1	1	0	D
1	1	1	0	0	1	1	1	1	0	0	1	E
1	1	1	1	0	1	1	1	0	0	0	1	F

Сформуємо в пам'яті програм контролера масив значень байта виводу згідно з табл. 10.1 (мітка BEGDIM), які потрібно вивести на індикатор для формування

шістнадцяткових цифр від 0 до F. Текст підпрограми, що реалізує дешифратор, наведений нижче.

```
DECOD:   MOV DPTR,#BEGDIM ; заносимо в DPTR адресу першого
                          ; елемента масиву
         MOV A,P1         ; вводимо вхідні сигнали
         ANL A,#0Fh       ; маскуємо старші розряди вхідних
                          ; сигналів, одержуємо індекс в
                          ; таблиці для даної комбінації
         MOVC A,@A+DPTR   ; заносимо в акумулятор значення
                          ; елемента масиву, відповідного
                          ; вхідному коду
         MOV P2,A         ; виводимо значення в порт
         RET              ; виходимо з підпрограми
BEGDIM:  DB 3Fh, 06h, 5Bh, 4Fh  ; 0, 1, 2, 3
         DB 66h, 6Dh, 7Dh, 07h  ; 4, 5, 6, 7
         DB 7Fh, 6Fh, 77h, 7Ch  ; 8, 9, A, B
         DB 39h, 5Eh, 79h,71h   ; C, D, E, F
```

У даній підпрограмі вхідне значення коду, утвореного вхідними сигналами, перетвориться в індекс для масиву вихідних значень. Потім за його допомогою зчитуємо з таблиці значення і виводимо у вихідний порт. В основній програмі необхідно організувати циклічний виклик підпрограми для оновлення значення індикатора в часі.

Методика виконання:

1. Заносимо програму у вікно редактора програми.

2. Контроль над роботою програми здійснюємо за допомогою редактора оточення. Для цього натискаємо кнопку  панелі інструментів. Після входу в редакторі створюємо середовище оточення й у вікні «Оточення контролера» налаштовуємо виходи порту P2 на вихід (рис. 10.14).

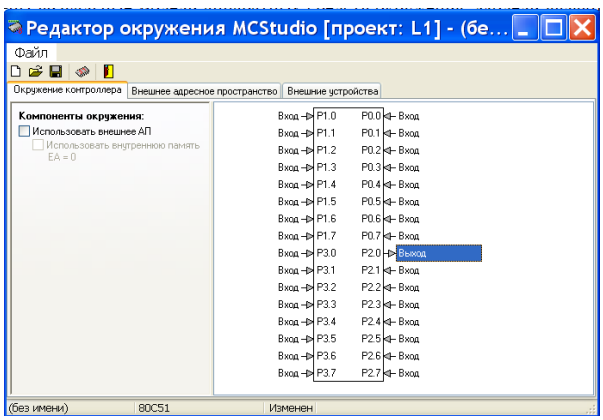


Рисунок 10.14 – Вікно налогодження оточення контролера

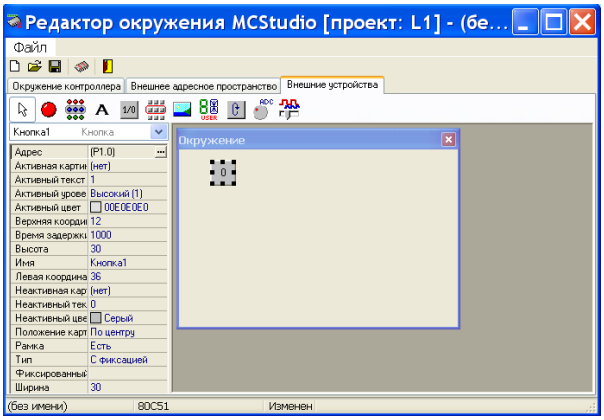


Рисунок 10.15 – Вікно підключення зовнішніх пристроїв до контролера

3. Переходимо на вкладку «Зовнішні пристрої» і встановлюємо у вікні оточення чотири кнопки і семисегментний індикатор зі списку елементів.

4. Активізуючи по черзі кожний елемент, заповнюємо форму підключення його до контролера. Наприклад клацнувши кнопку в колонці, що з'явилася ліворуч, натискуємо кнопку в полі адреси, у вікні, що з'явилося, вказуємо, що вона підключена до виводу контролера, і в сусідній колонці до біта 0 порту P1 (рис. 10.15).

5. Після підключення кнопок і індикатора виходимо з редактора оточення, зберігши файл оточення.

6. Після виходу з редактора в робочому вікні з'явиться вікно «Оточення». Якщо тепер перейти в режим налагодження, то за допомогою підключених кнопок можна задавати логічні сигнали на підключених входах, а індикатор буде відображати значення сигналів на виходах.

Звіт має містити текст індивідуального завдання і текст програми.

Завдання

Варіант 1. Написати програму, яка перетворить код на входах P0.0-P0.2 у символи від 3 до D.

Варіант 2. Написати програму, яка перетворить код на входах P0.4-P0.6 у символи 0, 2, 4, 6, 8, A, 3, E.

Варіант 3. Написати програму, яка перетворить код на входах P1.0-P1.2 у символи 1, 3, 5, 7, 9, B, D, F.

Варіант 4. Написати програму, яка перетворить код на входах P1.0-P1.2 у символи 1, 1, 2, 2, 3, 3, 4, 4.

Варіант 5. Написати програму, яка перетворить код на входах P1.0-P1.2 у символи 2, 5, 8, A, C, D, E, F.

Варіант 6. Написати програму, яка перетворить код на входах P0.0-P0.2 у символи 1, 3, 5, 7, 9, B, A, D.

Варіант 7. Написати програму, яка перетворить код на входах P0.3-P0.5 у символи 2, 6, A, E, 0, 4, 8, C.

Варіант 8. Написати програму, яка перетворить код на входах P0.2-P2.4 у символи 1, 4, 7, A, D, 0, 3, 6.

Варіант 9. Написати програму, яка перетворить код на входах P0.4-P0.6 у символи 0, 3, 6, 9, C, F, 2, 5.

Варіант 10. Написати програму, яка перетворить код на входах P0.5-P0.7 у символи 5, 9, C, 0, 4, 8, C, 0.

Програмний дешифратор виконати у вигляді підпрограми з параметрами, що задаються**.**

10.5. Лабораторна робота 4

Написання програми комбінаційного автомата за його таблицею істинності

Мета роботи: вивчити методи розв'язку логічних рівнянь для бітових змінних і засвоїти формальні методи написання програми для комбінаційних автоматів.

Теоретичні відомості. Спосіб реалізації комбінаційного автомата, розглянутий у третій лабораторній роботі, має такі обмеження: вхідні сигнали мають подаватися на один порт і підряд (інакше необхідно буде збирати інформацію з різних входів в одне вхідне слово); при досить великій кількості входів (7 і більше) і неповному використанні вихідних комбінацій нераціонально використовується пам'ять програм. Тому пропонується інший спосіб розв'язку задачі керування, оснований на програмній реалізації логічних кон’юнктивно диз'юнктивних рівнянь.

Як і в попередньому випадку, функціонування автомата задається у вигляді таблиці істинності, наприклад, табл. 10.2.

Таблиця 10.2 – Значення істинності логічних функцій

Входи				Вихід
P1.2	P1.4	P2.1	P2.3	P1.7
0	1	1	0	1
1	0	1	0	1
1	1	0	0	1

Запишемо логічне рівняння для виходу згідно з наведеною таблицею.

P1.7=!P1.2 & P1.4 & P2.1 & P2.3 +P1.2 & !P1.4 & P2.1 & !P2.3+P1.2 & !P1.4 & P2.1 & !P2.3

Програмна реалізація для першої функції «І» рівняння має вигляд:

```
JB P1.2,M1
JNB P1.4,M1
JNB P2.1,M1
JNB P2.3,M1
SETB P1.7
LJMP EXIT
M1: * * * ;перевірка наступної умови
```

Повна програма логічного рівняння відповідного до таблиці істинності має такий вигляд:

```
Loop: JB P1.2,M1; перевірка умови першого рядка таблиці
JNB P1.4,M1
JNB P2.1,M1
```

```
JNB P2.3,M1
SETB P1.7; установлення виходу в одиницю
LJMP Loop; і вихід
M1:
JNB P1.2,M2; ; перевірка умови другого рядка таблиці
JB P1.4,M2
JNB P2.1,M2
JB P2.3,M2
SETB P1.7; установлення виходу в одиницю
LJMP Loop; і вихід
M2:
JNB P1.2,M3; перевірка умови третього рядка таблиці
JB P1.4,M3
JNB P2.1,M3
JB P2.3,M3
SETB P1.7; установлення виходу в одиницю
LJMP Loop; і вихід
M3: CLR P1.7; скинути вихід у нуль – не виконалася жодна з умов
LJMP Loop; і вихід
```

Методика виконання:

1. Заносимо програму у вікно редактора.

2. Роботу програми проконтролюємо за допомогою редактора оточення. Для цього підключаємо на входи відповідні кнопки, а на виходи – світлодіоди.

3. Правильність появи вихідних сигналів перевірили під час симуляції виконання в покроковому режимі, остаточна перевірка – в автоматичному режимі. Для переходу на симуляцію в автоматичному режимі необхідно після включення повторно натиснути клавішу F9.

Завдання

Написати програму, яка реалізує функцію комбінаційного автомата згідно з варіантом, заданим у табл. 10.3. Програму налаштувати за допомогою «Середовища оточення».

Звіт має містити текст програми, вхідні дані та блок-схему алгоритму.

Таблиця 10.3 – Варіанти завдань комбінаційних автоматів

Варіант 1	**Комбінація натиснення клавіш**				**Стан портів виводу**			
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	0	0	0	1	0	0	1
Кодова комбінація 2	0	1	0	0	0	1	1	0
Варіант 2								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	0	0	1	1	1	0	0
Кодова комбінація 2	0	1	1	0	0	1	1	0

Варіант 3	**Комбінація натиснення клавіш**				**Стан портів виводу**			
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	1	0	0	1	0	0	1
Кодова комбінація 2	0	1	0	0	0	1	1	0
Варіант 4								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	0	0	0	1	1	1	1
Кодова комбінація 2	0	1	0	0	0	1	1	0
Варіант 5								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	0	0	1	1	0	0	1
Кодова комбінація 2	0	1	0	1	0	1	0	0
Варіант 6								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	0	0	1	1	0	0	1
Кодова комбінація 2	0	1	1	0	0	1	1	0
Варіант 7								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	0	0	0	1	1	0	1
Кодова комбінація 2	1	1	0	0	0	1	1	0
Варіант 8								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	1	0	1	1	1	0	0
Кодова комбінація 2	0	1	1	0	0	1	1	0
Варіант 9								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	1	1	0	1	0	0	1
Кодова комбінація 2	0	1	1	0	0	0	1	0
Варіант 10								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	0	0	1	1	0	0	1
Кодова комбінація 2	0	1	0	1	0	1	1	0
Варіант 11								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	0	0	1	1	0	0	1
Кодова комбінація 2	0	1	0	0	0	0	1	1
Варіант 12								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 1	1	1	0	1	1	0	0	1
Кодова комбінація 2	0	1	1	0	0	0	1	0
Варіант 13								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 3	0	0	1	0	1	1	1	0
Кодова комбінація 4	0	1	0	1	0	1	1	1
Варіант 14								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 3	0	1	1	0	1	1	1	0
Кодова комбінація 4	1	0	0	1	0	1	1	1

Варіант 15	Комбінація натиснення клавіш				Стан портів виводу			
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 3	1	0	1	0	1	1	1	0
Кодова комбінація 4	0	0	0	1	0	0	0	1
Варіант 16								
Порти МК	P0.0	P1.2	P1.4	P1.5	P2.1	P2.2	P1.3	P2.3
Кодова комбінація 3	0	0	1	1	0	1	1	0
Кодова комбінація 4	0	0	0	1	1	1	1	1

10.6. Лабораторна робота 5

Дослідження тимчасових затримок у МП системах

Мета роботи: навчитися писати програми та вміти розраховувати значення змінних для програмних тимчасових затримок.

Розрахунки часу затримки. За основу береться тривалість одного машинного циклу контролера. При тактовій частоті процесора 12 МГц тривалість машинного циклу 1 мкс. Загальна тривалість обчислюється за такою формулою

$$Тзд=1\ ц+1\ ц+2\ ц+1\ ц\cdot<r3>+1\ ц\cdot<r3>+2\ ц\cdot<r2>\cdot<r3>+2\ ц\cdot<r3>+2=$$
$$=1+1+2+71+71+2\cdot37\cdot71+2\cdot71+2=5544\ мкс.$$

Примітка: 1 ц – внутрішній програмний цикл, 2 ц – зовнішній програмний цикл.

Теоретичні відомості. Тимчасові затримки в мікропроцесорних системах легше всього організувати через програмні лічильники. Як лічильники зручно використовувати регістри R0 – R7. При цьому виходимо з того, що команди виконуються за кінцевий час. Вхідними даними для розрахунків є частота кварцу мікроконтролера. У середовищі MCStudio вона встановлюється в меню Виконання – Опції симуляції. Програми формування затримок зручно оформляти у вигляді підпрограми з передачею параметрів. Параметр – значення регістрів, що забезпечують задану тимчасову затримку.

У прикладі наведено програму тимчасової затримки для двобайтового лічильника.

Приклад:

```
Mov r2,#25h; завантаження молодшого байта затримки внутрішнього
             програмного циклу
Mov r3,#47 ; завантаження старшого байта затримки внутрішнього програмного
             циклу
Lcall zd1; виклик підпрограми затримки зовнішнього програмного циклу
M1:       ajmp M1; нескінченний цикл
```

```
zd1:    mov a,r2; пересилання в акумулятор кількості внутрішніх циклів,
                  значення яких вказано в регістрі r3
mov     r4,a; пересилання з акумулятора в регістр r4 кількості внутрішніх
              циклів, значення яких вказано в регістрі r3
m2:     djnz r4,h,m2; декремент значення молодшого байта затримки
                      зовнішнього програмного циклу, кількість яких вказано
                      в регістрах r2 і r3
        djnz r3,zd1; декремент значення старшого байта затримки
                     зовнішнього програмного циклу, кількість яких вказано в
                     регістрі r3
        ret     ;вихід з підпрограми зовнішнього програмного циклу
```

Методика виконання

1. Контроль затримки затримки виконується за покажчиком часу, що перебуває у вікні «Ресурси» (пункт «Ядро MCS-51» (рис. 10.17).

2. Встановлення заданої частоти генератора виконується через пункт меню Виконання – Опції симуляції (рис. 10.18).

3. Для контролю виконання програми в автоматичному режимі використовувати точку зупинки, встановлену на мітці M1. Для задання точки зупинки необхідно маркер помістити на потрібний рядок програми й натиснути клавішу F5. Значення часу виконання можна побачити у вікні Таймінга, рис. 10.16. Після першої зупинки програми на вікні Таймінга натискаємо кнопку Запуск, і після повторної зупинки бачимо кількість циклів та час виконання фрагмента коду.

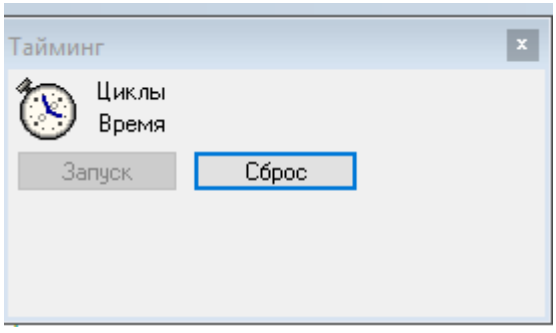


Рисунок 10.16 – Вікно Таймінга

4. Задається точка зупинки. Програма запускається на симуляцію в автоматичному режимі після переривання виконання (у точці зупинки), час виконання не повинен відрізнятися від встановленого в завданні більш ніж на 5 мкс.

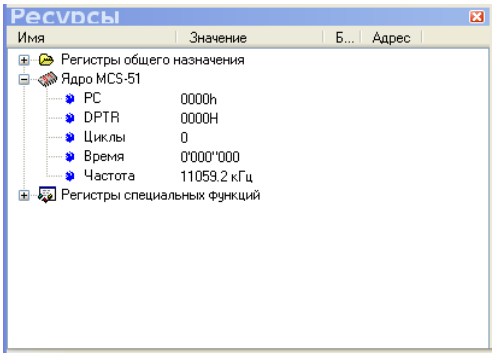


Рисунок 10.17 – Вікно ядра MCS-51

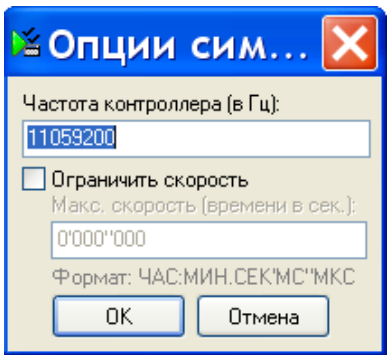


Рисунок 10.18 – Вікно встановлення частоти генератора мікроконтролера

5. Значення регістрів визначається за формулою, що наведена вище. Для цього треба задати значення старшого байта затримки, що дорівнює 255, і за рівнянням знайти значення молодшого байта. Якщо отримане значення перевищує число 255, то додаємо ще один цикл у програмі. З урахуванням цього коректуємо формулу і знаходимо потрібні значення.

Варіанти завдань

Час затримки повинен дорівнювати номеру за списком, що помножений на 27 мс.

Звіт має містити текст програми, вхідні дані і блок-схему алгоритму, отримане реальне значення затримки.

10.7. Лабораторна робота 6

Побудова апаратного відмітника часу з використанням таймера

Мета роботи: вивчити принципи написання оброблювачів переривання, навчитися розраховувати значення константи, що завантажується в таймер за заданим періодом формування міток часу, навчитися працювати у програмі налагоджувача при використанні переривань, спостерігати за таймером у режимі відмітника часу.

Теоретичні відомості. Відмітник часу – пристрій, який формує переривання через фіксовані проміжки часу.

Робота з таймером у режимі відмітника часу з використанням переривання складається з декількох етапів.

Спочатку необхідно встановити таймер у режим 1 (16-розрядний таймер). Режим задається за допомогою занесення керуючого слова в регістр TMOD. За режим TMR0 відповідають біти D1, D0.

Для формування необхідного коефіцієнта ділення заносимо в регістри TH0, TL0 число, що дорівнює різниці 65535 і значенню необхідного коефіцієнта ділення.

Для дозволу роботи TMR0 установлюємо в регістрі TCON.4 логічну «1».

Для дозволу проходження переривання від TMR0 у регістрі IE.1 і IE.7 необхідно встановити «1».

Для організації тимчасових затримок з використанням відмітника часу використовуємо програмно організовані лічильники, які розташовані в пам'яті даних. Для запуску затримки в осередок заноситься число. Значення числа визначається як результат ділення необхідної тимчасової затримки на період відмітника часу. Контроль закінчення затримки виконується перевіркою на нуль значення в програмному лічильнику. Декрементування лічильників проводиться в оброблювачі переривання, якщо лічильник не дорівнює нулю.

Текст програми мовою «Асемблер», що реалізує миготіння світлодіода порту P1.0, яка здійснює керування двома затримками. Лічильники затримок організовано в 31 і 32 осередках пам'яті даних.

```
        ljmp m0          ; обходимо вектор переривання
        org 0bh          ; вектор переривання від таймера TMR0
        push acc
        mov th0,#3ch     ; налагодження таймера на період в 50 мс
        mov tl0,#0b0h
        djnz 30h,m2      ; організація постдільника
        mov 30h, #20
        mov a,31h
        jz m3
        dec a            ; організація першої затримки
        mov 31h,a
m3:     mov a,32h
        dec a            ; організація другої затримки
        mov 32h,a
m2:     pop acc
        reti                    ; повернення з переривання
m0:     mov sp,#70h
        mov tmod,#1h     ; встановлення таймера 0 у режим 1
        mov th0,#0ffh
        mov tl0,#0e0h
        setb tcon.4
        setb ie.7        ; дозвіл роботи TMR0 і проходження
        setb ie.1        ; переривання від TMR0
        mov 30h,#20
On:     setb P1.0     ;вмикання світлодіода
```

```
        mov a,#03h
        mov 31h,a   ;запуск затримки 1 на 3 с
zdoff:  mov a,31h   ;очікування затримки на вимикання
        jnz zdoff
Of:     clr P1.0
        mov a,#03h
        mov 32h,a   ; запуск затримки 1 на 3 с
zdon:   mov a,32h   ; очікування затримки на вмикання
        jnz zdon
        ljmp On
end
```

Пояснення до програми. Дана програма використовує таймер 0 для створення апаратного відмітчика часу з періодом 50 мс. Вектор переривання від таймера 0 перебуває за адресою 0bh. В оброблювачі переривань здійснюється перезавантаження таймера, яке забезпечує час до наступного переривання, що дорівнює 50 мс. В осередку 30h створений програмний постдільник на 20. Таким чином за допомогою постдільника одержуємо період відмітника 1 с. Така змінна дуже зручна, якщо необхідно робити періодичні дії з періодом 1 с. В осередках 31h і 32h створюємо затримки від 1 до 255 с.

В основній програмі (мітка м0) робимо ініціалізацію таймера і переривання від нього, після чого запалюємо світлодіод, підключений до біта 0 порту P1, і за допомогою затримки в 1 с залишаємо увімкненим його на 3 с. Після цього гасимо його і залишаємо вимкненим на 3 с за допомогою затримки 2. Командою LJMP On зациклюємо програму на миготіння.

Установивши точки зупинки на рядках з мітками On і Off, у моменти зупинки програми у вікні Таймінга можна бачити, що зміна стану світлодіода відбувається кожні 3000000 мкс.

Методика виконання

Заносимо програму у вікно її редактора.

Контроль за роботою програми здійснюємо за допомогою редактора оточення. Для цього підключаємо на вихід P1.0 світлодіод.

Установлюємо точки зупинки відповідно до пояснень до програми.

Створюємо вікно для спостереження за змінними в процесі налагодження (рис. 10.19). Для цього в меню вибираємо пункти Вид – Перегляд значень. Ініціюємо правою кнопкою миші і додаємо у вікно регістри, що цікавлять нас, і осередки пам'яті (th0, tl0, 30h, 31h, 32h) за допомогою вікна створення перегляду і конструктора виразів (рис. 10.20).

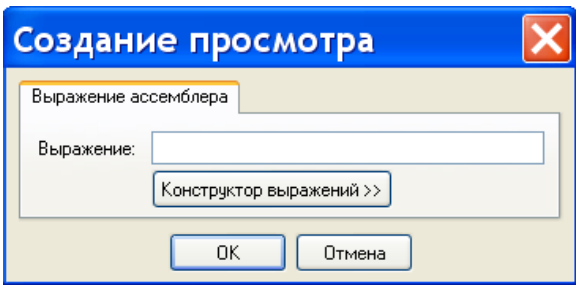


Рисунок 10.19 – Вікно для заповнення вікна налагодження

У результаті одержуємо вікно перегляду значень зі змінними, які нам необхідні для налагодження (рис. 10.21).

Для більш повного розуміння процесів, пов'язаних з перериванням програми таймером, та виконання програми до першого переривання робимо симуляцією в покроковому режимі. Для цього перший цикл переривання при ініціалізації виконаний скороченним.

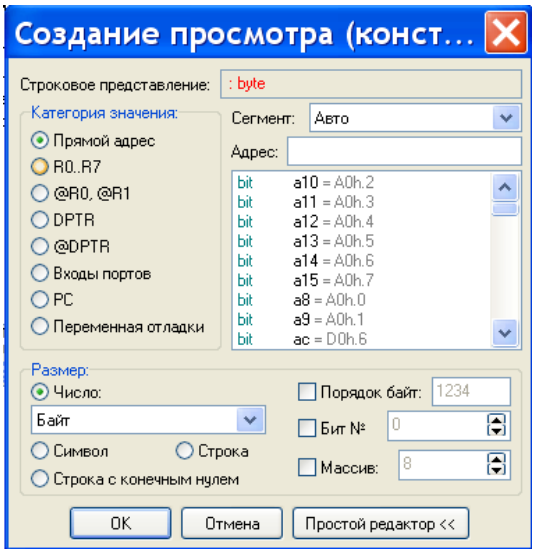


Рисунок 10.20 – Вікно конструктора виразів

Рисунок 10.21 – Вікно перегляду значень

Запускаємо симуляцію в автоматичному режимі і контролюємо час виконання до зупинки.

Звіт має містити тексти програми та самого завдання**.**

Варіанти завдань

Варіант 1. Реалізувати затримку в 2 с і вивести на порти P1.0-P1.3 у вигляді хвилі, що біжить, зліва направо**.**

Варіант 2. Реалізувати затримку в 1 с і вивести на порти P1.0-P1.3 у вигляді тіні, що біжить, зліва направо.

Варіант 3. Реалізувати період миготіння 2 с результату комбінації, що спрацювала, для лабораторної роботи 4.

Варіант 4. Реалізувати розрахунок і індикацію на семисегментний індикатор змінної від 0 до 8 з періодом 2 с.

Варіант 5. Реалізувати затримку в 2 с і вивести на порти P1.0-P1.3 у вигляді хвилі, що біжить, справа наліво.

Варіант 6. Реалізувати затримку в 1 с і вивести на порти P1.0-P1.3 у вигляді тіні, що біжить, справа наліво.

Варіант 7. Реалізувати розрахунок і індикацію на трьох світлодіодах двійкового коду від 0 до 7 з періодом 2 с.

Варіант 8. Реалізувати програму, яка включає світлодіод, через 5 с після натиснення кнопки, а виключає, коли її відпускають.

Варіант 9. Реалізувати програму, яка включає світлодіод, після натиснення натиснення кнопки, а виключає через 5 с після її відпускання.

Варіант 10. Реалізувати програму, яка блимає світлодіодом з періодом 3 с, доки натиснута кнопка, а виключає, коли її відпускають.

10.8. Лабораторна робота 7

Дослідження послідовного інтерфейсу в мікропроцесорних системах

Мета роботи: навчитися писати програми обміну інформацією в мікропроцесорних системах з послідовним інтерфейсом.

Теоретичні відомості. Послідовний інтерфейс у мікроконтролері серії MCS-51 призначений для обміну інформацією з іншими обчислювальними системами на відстані від декількох метрів до декількох кілометрів.

Послідовний інтерфейс має власну назву, оскільки він перетворює дані, що надходять до нього в паралельному форматі по восьмирозрядній шині, в послідовний код, тобто в послідовний у часі потік бітових даних на виході передавача. Крім цього, він виконує зворотне перетворення. Дані в послідовному форматі, що надходять на вхід приймача, перетворюються в паралельні і можуть бути зчитані в акумулятор.

Перетворення виконується за допомогою зсувного регістра. Швидкість передачі задається за допомогою таймера 1, що працює в другому режимі без переривань.

Швидкість передачі визначається за формулою

$$C = \left(2^{SMOD} / 32\right)\left(f_{osc} / 12\right) / \left(256 - \left(TH1\right)\right),$$

де f_{osc} – частота кварцового резонатора мікроконтролера.

Текст програми записаний мовою «Асемблер», що реалізує передачу масиву даних, які розташовуються у внутрішній пам'яті даних за адресою 30h довжиною 10 байтів. Після передачі здійснюється прийом відповідних даних у масив, що розташовується у внутрішній пам’яті даних, за адресою 50h довжиною 10 байтів.

```
Mass equ 30h
Recivmas equ 50h; початок масиву, що приймається
Flag equ 0h; кінець обміну даними
Razmer equ 12h; R2 2 Банк
Ukaz equ 10h; R0 2 Банк
        ljmp main; перехід на основну програму
        org 23h; вектор переривання від послідовного інтерфейсу
ljmp transmit
;ОСНОВНА ПРОГРАМА
main: mov TMOD,#20h; режим 2 таймеру 1
        mov PCON,#80h;SMOD=1
        mov TL1,#0fdh
        mov TH1,#0fdh; частота обміну 19.2 kgc
        setb TCON.6; режим роботи USART
        setb SCON.6
        setb SCON.7
        setb SCON.3
        mov Ukaz,#mass+1
        mov Razmer,#10; довжина масиву
        setb Flag
;ПЕРЕДАЧА
        mov SBUF, Mass; перший байт пішов
        setb IE.7; дозвіл переривань
        setb IE.4
m1:   jb Flag,m1; контроль закінчення передачі
;ПРИЙМАННЯ
        mov Ukaz,#recivmas; початок масиву приймання
        mov Razmer,#10; довжина масиву приймання
        setb SCON.4
setb Flag
m1r:  jb Flag,m1r
me:   ljmp me
;ПІДПРОГРАМА ОБРОБКИ ПЕРЕРИВАНЬ ВІД USART
transmit:
        push Acc
        push PSW
        setb PSW.4; активний банк 2
        clr PSW.3
        jb SCON.0,reciw; виявлення від кого запит
        djnz r2,m2t; контроль закінчення передачі
```

```
clr Flag; передачу закінчено
        ljmp m2
m2t:    mov SBUF,@R0; передача елемента масиву в передавач
        inc r0
m2:     clr TI
m2r:    pop PSW
        pop Acc
        reti
reciw:          mov @R0,SBUF; приймання
        inc R0
        clr RI
        djnz r2,m2r; контроль кінця приймання
        clr SCON.0
        clr Flag
        ljmp m2r
        end
```

Опис програми. В основній програмі ініціалізується таймер і послідовний інтерфейс. А так само проводиться запуск передачі за допомогою запису першого байта в SBUFF. Контроль над передачею здійснюється апаратно через виклик переривання від біта TI. Оскільки вектор переривання від приймача й передавача один – 23h, то при вході в оброблювач визначається від кого запит. Передача інших байтів масиву в SBUFF відбувається в оброблювачі переривання. Зчитування прийнятих байтів із приймача виконується так само, як і в оброблювачі переривань.

Методика виконання

1. Заносимо програму у вікно її редактора.

2. Програму виконуємо в режимі симуляції з використанням підсистеми послідовного порту (натиснути кнопку ) і вікна «Статистика UART».

3. Симулятор послідовного порту дозволяє спостерігати процес передачі і прийому даних із прив'язкою до часу тактування мікроконтролера. Вікно симулятора УСАПП зображено на рис. 10.22. Крім цього, у вікні можна сформувати повідомлення для приймача та імітувати в ручному або автоматичному режимі подачу його на вхід приймача.

4. Емуляцію появи інформації на вході приймача виконуємо в ручному режимі. Формувати список байтів для приймання можна, натиснувши кнопку «+» у полі керування буфером приймання. Для того, щоб черговий байт із буфера потрапив на вхід приймача, потрібно натиснути кнопку «Надіслати».

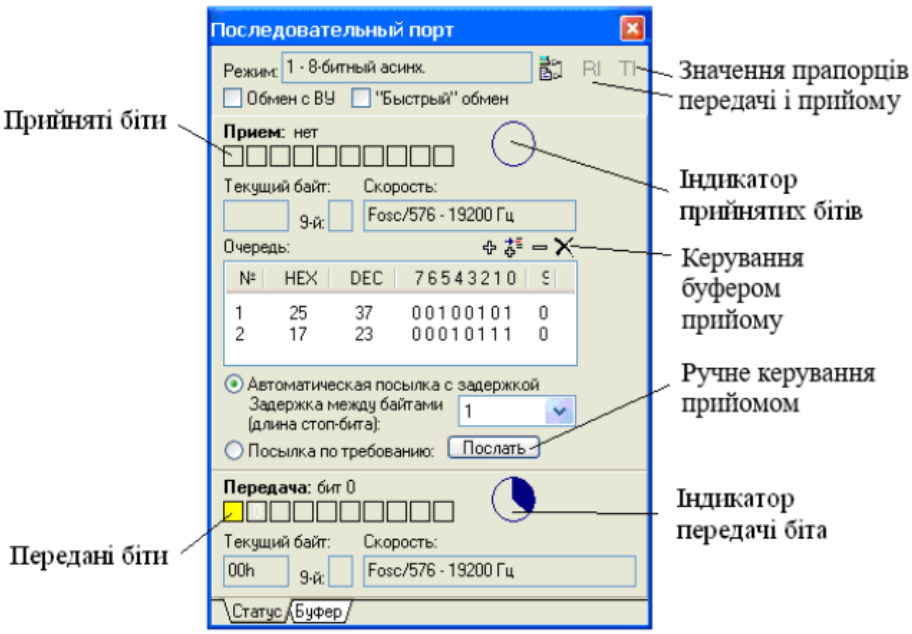


Рисунок 10.22 – Вікно симулятора послідовного обміну

Звіт повинен містити текст програми, вихідні дані і блок-схему алгоритму.

Завдання. Організувати обмін інформацією мікроконтролера з іншим мікроконтролером. Переданий масив розташувати в пам'яті програм за адресою і довжиною, що наведені в таблиці, згідно з варіантом. Прийняті дані розмістити у внутрішній пам'яті за адресом і довжиною, що наведені в таблиці, згідно варіанта. В оброблювачі переривань треба використовувати банк регістрів загального призначення згідно з варіантом. Швидкість обміну даними вибирається також згідно з варіантом (табл. 10.4).

Таблиця 10.4 – Індивідуальні завдання на лабораторну роботу

Номер варіанта	Початкова адреса переданого масиву	Довжина переданого масиву	Початкова адреса прийнятого масиву	Довжина прийнятого масиву	Номер банку оброблювача переривання	Швидкість обміну, біт/с
1	100h	8	20h	9	1	1200
2	150h	6	50h	11	2	2400
3	220h	4	40h	13	3	4800
4	50h	10	10h	7	2	9600
5	60h	5	60h	12	1	19200
6	80h	7	20h	10	3	1200
7	120h	9	50h	8	1	2400
8	170h	11	40h	6	2	4800
9	30h	13	10h	4	3	9600
10	10h	4	60h	13	1	19200

СПИСОК ЛІТЕРАТУРИ

1. Бродин В.Б. Системы на микропроцессорах и БИС программируемой логике / В.Б. Бродин, А.В. Калинин. – Москва: ЭКОМ, 2002. – 400 с.

2. Костинюк Л.Д. Мікропроцесорні засоби та системи / Л.Д. Костинюк, Я.С. Парганчук. – Львів.: Львівська політехніка, 2001. – 200 с.

3. Сташин В.В. Проектирование цифровых устройств на однокристальных микроконтроллерах / В.В. Сташин, А.В. Урусов, О.Ф. Мологонцева. – Москва: Энергоатомиздат, 1990. – 224 с.

4. Інтернет ресурс https://www.nuvoton.com/export/resource-files/DS_N76E003_EN_Rev1.09.pdf.

5. Інтернет ресурс https://www.nuvoton.com/export/resource-files/DS_ML51_ML54_ML56_Series_V2.00.pdf.

Навчальне видання

Ткачов Віктор Васильович
Проценко Станіслав Миколайович
Козарь Микола Володимирович
Шевченко Владислав Іванович

МІКРОПРОЦЕСОРНА ТЕХНІКА

Підручник

Друге видання, доповнене
і перероблене

Редактор Л.О. Чуїщева

Технічний редактор Н.М. Безгінова

Підписано до друку 31.01.2022. Формат 30х42/4.
Папір офсет. Ризографія. Ум. друк. арк. 13,6.
Обл.-вид. арк. 17,6. Тираж 16 пр. Зам. №

Підготовлено до друку та видрукувано
в Національному технічному університеті «Дніпровська політехніка»
Свідоцтво про внесення до Державного реєстру ДК № 1842 від 11.06.2004.
49005, м Дніпро, просп. Д. Яворницького,19