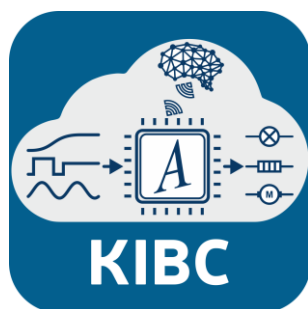


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Кафедра Кіберфізичних та інформаційно-вимірювальних систем



В.В. Ткачов, С.М. Проценко, М.В. Козарь, В.І. Шевченко

МІКРОПРОЦЕСОРНА ТЕХНІКА

Підручник



**ДНІПРОВСЬКА
ПОЛІТЕХНІКА
1899**

Дніпро
НТУ ДП
2021

УДК 004.31 (075.8)
ББК 32.973.26-04я73
Т-59

Рекомендовано Вченою радою НТУ «Дніпровська політехніка» як підручник для студентів бакалаврів вищих навчальних закладів спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології» і 152 «Метрологія та інформаційно-вимірвальна техніка» (№ від 2021 р.).

Рецензенти:

С.А. Положаєнко, д-р техн. наук, професор, завідувач кафедри «Комп'ютеризовані системи керування» Одеського Національного політехнічного університету.

А.І. Купін, д-р техн. наук, професор, завідувач кафедри «Комп'ютерних систем та мереж» ДВНЗ «Криворізький Національний університет».

А.Й. Наконечний, д-р техн. наук, професор, завідувач кафедри «Комп'ютеризовані системи управління» Національного технічного університету «Львівська політехніка»

Ткачов В.

Т-59 Мікропроцесорна техніка [Текст]: підручник / В.В. Ткачов, С.М. Проценко, М.В. Козарь, В.І. Шевченко. М-во освіти і науки України, НТУ «Дніпровська політехніка». – Дніпро: НТУ «ДП». – 2021. – 235 с.

ISBN

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

Посібник укладено відповідно до програми дисципліни «Мікропроцесорна техніка» для студентів, що навчаються за спеціальностями 151 «Автоматизація та комп'ютерно-інтегровані технології», 152 «Метрологія та інформаційно-вимірвальна техніка», а також для студентів інших спеціальностей.

УДК 004.31 (075.8)
ББК 32.973.26-04я73

ISBN

© В.В. Ткачов, С.М. Проценко, М.В. Козарь, В.І. Шевченко, 2021.
© НТУ «Дніпровська політехніка», 2021.

ЗМІСТ

	Стор.
ПЕРЕДМОВА.....	5
1. ЗАГАЛЬНІ ПОНЯТТЯ Й СТРУКТУРА СИСТЕМ КЕРУВАННЯ.....	6
1.1 Дискретні або цифрові СК.....	8
1.2 Основи обчислювальної техніки.....	9
2. ЗАГАЛЬНІ ВІДОМОСТІ ПРО МІКРОКОНТРОЛЕРИ.....	50
2.1 Структура й функціональні можливості базової моделі MCS-51 (МК51).....	51
2.2 Програмно доступні ресурси МК51.....	57
3. СИСТЕМА КОМАНД МІКРОКОНТРОЛЕРА СЕРІЇ MCS-51.....	65
3.1 Загальні положення.....	65
3.2 Способи адресації.....	68
3.3 Команди мікроконтролера.....	75
4. ПОРТИ ВВЕДЕННЯ/ВИВЕДЕННЯ ІНФОРМАЦІЇ ВІД ЗОВНІШНІХ ПРИСТРОЇВ.....	110
5. ОРГАНІЗАЦІЯ ПЕРЕРИВАНЬ У МІКРОПРОЦЕСОРНИХ СИСТЕМАХ.....	122
6. ОРГАНІЗАЦІЯ ТИМЧАСОВИХ ЗАТРИМОК У МІКРОПРОЦЕСОРНИХ СИСТЕМАХ.....	132
7. ПРИКЛАДИ ВВЕДЕННЯ І ВИДЕННЯ ІНФОРМАЦІЇ В ДИСКРЕТНИХ СИСТЕМАХ.....	143
7.1 Особливості контролю дискретних датчиків на реальних об'єктах керування.....	143
7.2 Приклад виконання програмної частини курсового проекту.....	145
7.3 Система керування кроковим двигуном.....	152
8. ОРГАНІЗАЦІЯ ПЕРЕДАЧІ ІНФОРМАЦІЇ В ПОСЛІДОВНОМУ ФОРМАТІ В МІКРОПРОЦЕСОРНИХ СИСТЕМАХ (МПС).....	164
8.1 Принципи передачі інформації з послідовного каналу зв'язку...	164
8.2 Послідовний інтерфейс у мікроконтролері MCS51.....	166
8.3 Режим роботи 0.....	168
8.4 Режим роботи 1.....	169
8.5 Режим роботи 2.....	174
8.6 Режим роботи 3.....	174
9. ПОСЛІДОВНІ ШИННІ СИСТЕМИ (КАНАЛИ ЗВ'ЯЗКУ).....	175
9.1 Загальні поняття про електронні шинні системи.....	175

9.2	Різновиди послідовних шинних систем.....	176
9.3	Окремі приклади інтерфейсів та послідовних шин.....	179
10.	ПРАКТИЧНА ЧАСТИНА.....	205
10.1	Правила запису програми на мові асемблера.....	205
10.2	Лабораторна робота № 1.....	207
10.3	Лабораторна робота № 2.....	212
10.4	Лабораторна робота № 3.....	216
10.5	Лабораторна робота № 4.....	220
10.6	Лабораторна робота № 5.....	223
10.7	Лабораторна робота № 6.....	225
10.8	Лабораторна робота № 7.....	229

ПЕРЕДМОВА

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

Завдання предмету – формування здатності застосовувати знання фізики, електротехніки, електроніки і мікропроцесорної техніки, в обсязі, необхідному для розуміння процесів в системах автоматизації та комп'ютерно-інтегрованих технологіях і вміння обґрунтовувати вибір структури та розробляти прикладне програмне забезпечення для мікропроцесорних систем управління 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 авторів по створенню і впровадженню сучасних мікропроцесорних управління, удосконаленню лабораторної бази дисципліни, методичного та мультимедійного забезпечення, а також значне оновлення елементної бази мікропроцесорної техніки.

Матеріал підручника відповідає лекційному курсу дисципліни, який впродовж кількох років викладався авторами у Національному технічному університеті «Дніпровська політехніка». Книга може бути використана для студентів 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.

Враховуючи навчальне 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 в галузі мікропроцесорної техніки.

Підручник розроблено колективом викладачів Національного технічного університету «Дніпровська політехніка». Перший розділ написано В.В. Ткачовим і М.В. Козарем, третій – В.І. Шевченком. Другий, сьомий, восьмий і дев'ятий – В.В. Ткачовим спільно зі С.М. Проценком, четвертий, п'ятий і шостий – В.В. Ткачовим, десятий – С.М. Проценком. Загальна редакція – В.В. Ткачова.

Автори з вдячністю отримують зауваження та побажання, що сприятимуть подальшому покращенню змісту книги.

1. ЗАГАЛЬНІ ПОНЯТТЯ Й СТРУКТУРА СИСТЕМ КЕРУВАННЯ

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

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

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

Технологічний процес – це виконання послідовності 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 обраних правил на підставі 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 впливів здійснюються людиною, то керування зветься автоматизованим. За аналогією системи теж поділяються на автоматичного й автоматизованого керування. 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, температури, тиску, що розвиваються, й 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.

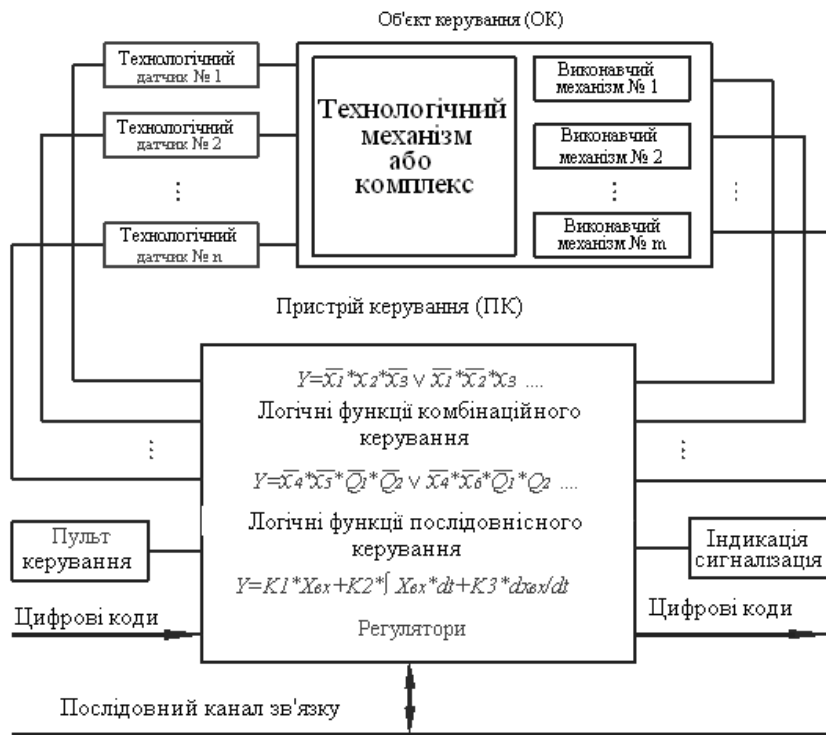


Рисунок 1.1 – Структурна схема системи керування

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 органи 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 вироблення керуючих впливів згідно з 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 встаткування. Це 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 виконавчих 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 виробляє 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 СК

Якщо припустити, що об'єкт керування відомий, тобто, визначені виконавчі механізми та засоби комутації енергії до них, обрані відповідні датчики, визначені зовнішні зв'язки, сформульовані необхідні вимоги до алгоритмів керування й параметрів регулювання, то залишається створити або розробити самий ПК. За своєю суттю ПК є пристроєм перетворення вхідної інформації у вихідну. Але оскільки вхідні і вихідні сигнали неоднакові, 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, заснованих 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 також, 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 в результаті цієї обробки. За зазначеним принципом будуються контури неперервного регулювання, де потрібно або підтримувати, або змінювати регульовану величину вихідного сигналу відповідно до обраного закону керування. Такі регулятори одержали назву цифрові, 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, \dots, x_n . У силу визначення логічні змінні можна називати також двійковими.

Логічна – це функція 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», 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 визначаються відношення еквівалентності, позначуване символом рівності « $=$ », і три операції: логічного додавання (диз'юнкції), логічного множення (кон'юнкції), логічного заперечення (інверсії), позначувані відповідно символами:

+ або \vee – операція диз'юнкції;

\cdot або \wedge або $\&$ – операція кон'юнкції;

$\bar{}$ – операція інверсії ($*$ – символ аргументу або функції).

З огляду на постулати при виконанні перерахованих операцій відносини еквівалентності мають вигляд:

а)	б)	в)	
$0 + 0 = 0$	$0 \cdot 0 = 0$	$\bar{0} = 1$	
$0 + 1 = 1$	$0 \cdot 1 = 0$	$\bar{1} = 0$	(1.1)
$1 + 0 = 1$	$0 \cdot 1 = 0$		
$1 + 1 = 1$	$1 \cdot 1 = 1$		

На підставі постулатів (1) можна вивести наступні співвідношення (закони) алгебри логіки:

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

а)	б)	в)	
$x + 1 = 1$	$x + 0 = x$	$x + x = x$	(1.2)
$x \cdot 1 = x$	$x \cdot 0 = 0$	$x \cdot x = x$	

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

а)	б)	в)	
$\overline{\overline{x}} = x$	$x + \bar{x} = 1$	$\overline{x_1 + x_2} = \bar{x}_1 \cdot \bar{x}_2$	(1.3)
	$\overline{x \cdot x} = 0$	$\overline{x_1 \cdot x_2} = \bar{x}_1 + \bar{x}_2$	

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

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

Закони подвійності (3, в), які також мають назву де-Моргана, були узагальнені К. Шенноном на випадок довільного (n) числа аргументів.

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, \dots, 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 випускаються 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:

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

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 більше ніж над однією змінною.

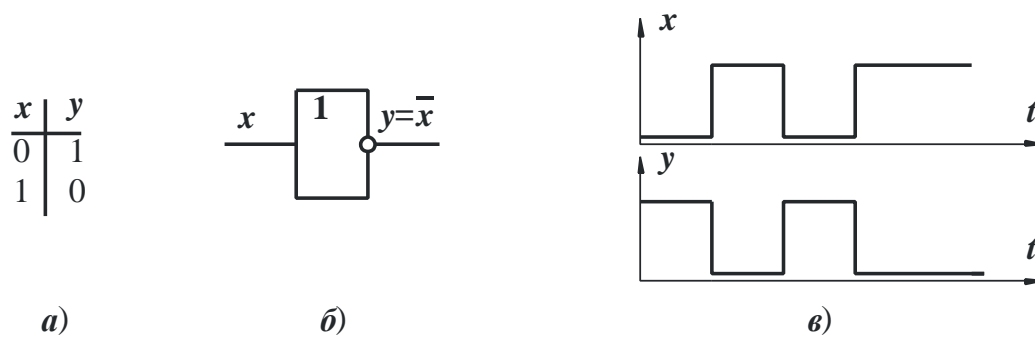


Рисунок 1.2 – Елемент «НІ»: а) – таблиця істинності; б) – умовне зображення; в) – тимчасові діаграми

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

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

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: а) – таблиця істинності; б) – умовне зображення; в) – тимчасові діаграми двовходового кон'юнктора.

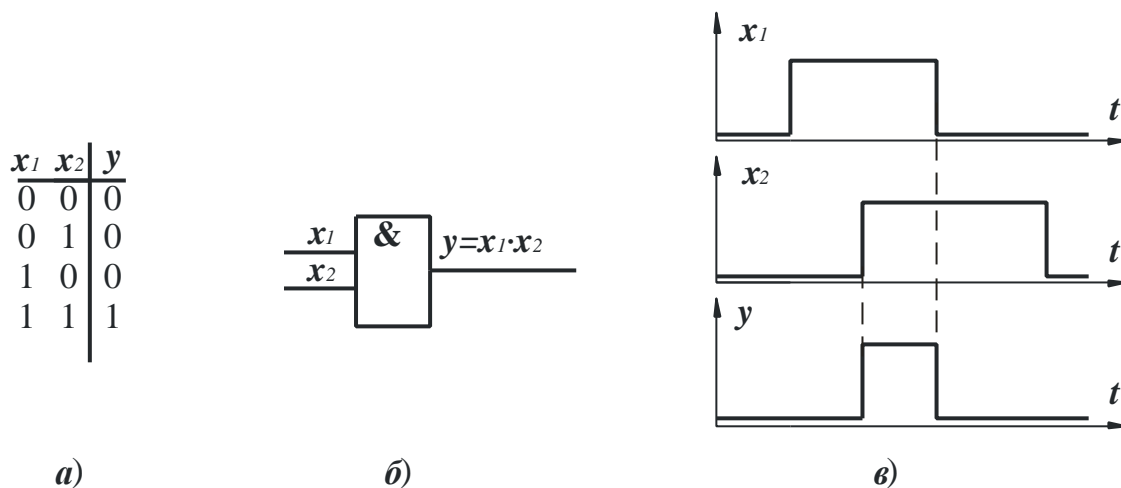


Рисунок 1.3 – Елемент «І»: а) – таблиця істинності, б) – умовне позначення, в) – тимчасові діаграми

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 вигляд:

$$y = \overline{x_1 \cdot x_2} \quad (1.7)$$

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

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

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

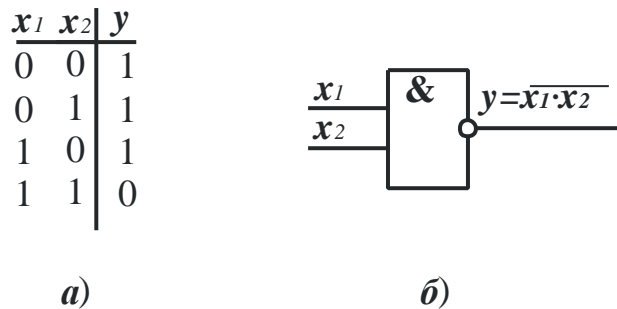


Рисунок 1.4 – Елемент «І – НІ»: а) – таблиця істинності, б) – умовне зображення

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: а) – таблиця істинності; б) – умовне зображення; в) – тимчасові діаграми.

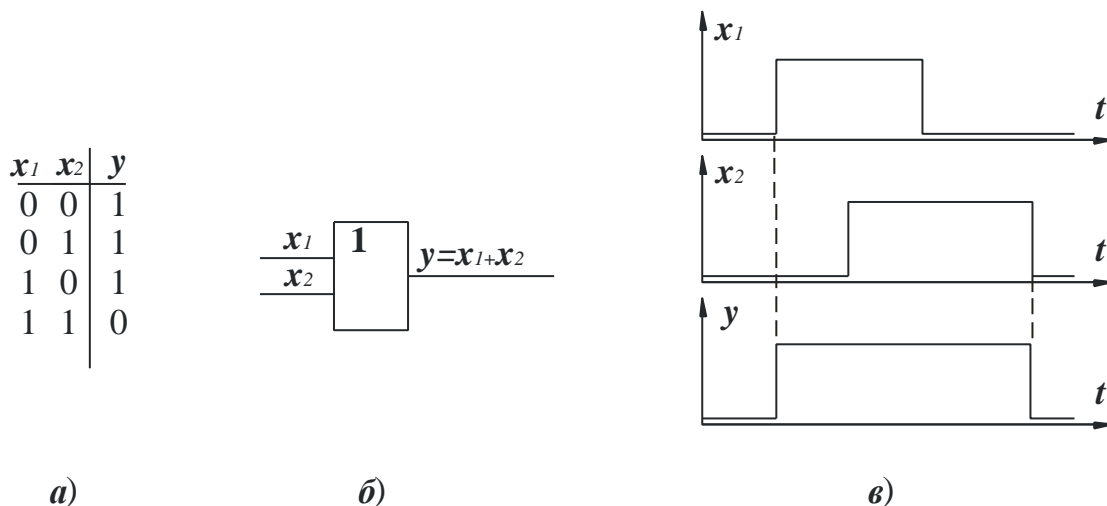


Рисунок 1.5 – Елемент «АБО»: а) – таблиця істинності, б) – умовне позначення, в) – тимчасові діаграми

Функція заперечення диз'юнкції («АБО – НІ») одержується шляхом інвертування самої функції диз'юнкції і її запис має вигляд:

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

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

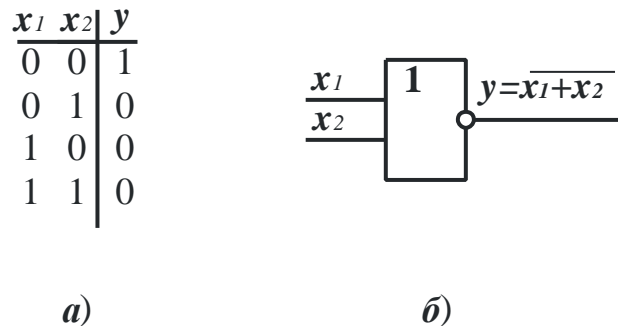


Рисунок 1.6 – Елемент «АБО – НІ»: а) – таблиця істинності; б) – умовне зображення

Функція «додавання за модулем два» (M2) найчастіше використовується як функція двох аргументів. У цьому випадку її називають також функцією «нерівнозначності». Аналітична форма запису такої функції має вигляд:

$$y = x_1 \oplus x_2. \quad (1.10)$$

Назва функції пов'язана з тим, що $x_1 \oplus x_2$ є арифметичною сумою двійкових чисел x_1 і x_2 в межах одного розряду: $0+0=0$; $0+1=1$; $1+0=1$; $1+1=10$. 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 M2 має деяку властивість, яку корисно знати.

$$\begin{array}{ccc} \text{а)} & \text{б)} & \text{в)} \\ \overline{x_1 \oplus x_2} & = x_1 \oplus \overline{x_2} & = \overline{x_1 \oplus x_2} \end{array} \quad (1.11)$$

При інвертуванні одного з аргументів уся функція інвертується. У цьому випадку реалізується логічна функція «рівнозначності» $x_1 \equiv x_2$. Вона дорівнює одиниці, якщо $x_1 = x_2$. Очевидні також і співвідношення (12).

$$x \oplus 0 = x; \quad x \oplus 1 = \overline{x} \quad x \oplus x = 0 \quad x \oplus \overline{x} = 1 \quad (1.12)$$

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 «неправда» – розімкнений (рис. 1.7). Часто «істинний» і «хибний» стани відповідають термінам «високий» і «низький» рівень, «так» – «ні», «увімкнено» – «вимкнено», «одиниця (1)» – «нуль (0)».

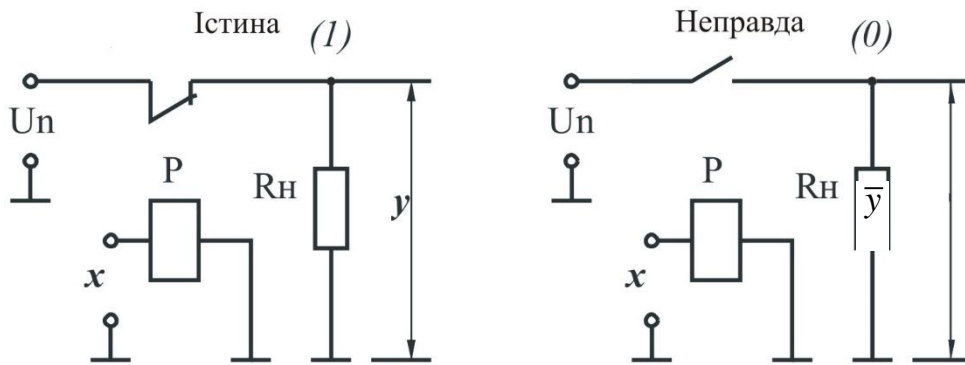


Рисунок 1.7 – Релейно-контактні логічні елементи

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_n проходить струм, під дією якого на R_n спадає напруга, що відповідає на виході логічній одиниці.

Функція «І» (рис. 1.9, б) реалізується з використанням додаткового джерела зсуву U_{zc} . За відсутності входніх сигналів x_1, x_2, \dots, x_n через резистор R_6 , діоди $VD1, \dots, VDn$ і входні опори джерел сигналу x проходить струм зсуву, і вся напруга виділяється на резисторі R_6 , а на виході – низький потенціал (0). Під час подачі на вхід сигналів x діоди $VD1, \dots, VDn$ замикаються, тому на виході буде високий потенціал (1). Якщо навіть на одному із входів відсутній входній сигнал, то на виході буде 0, що й відповідає функції «І».

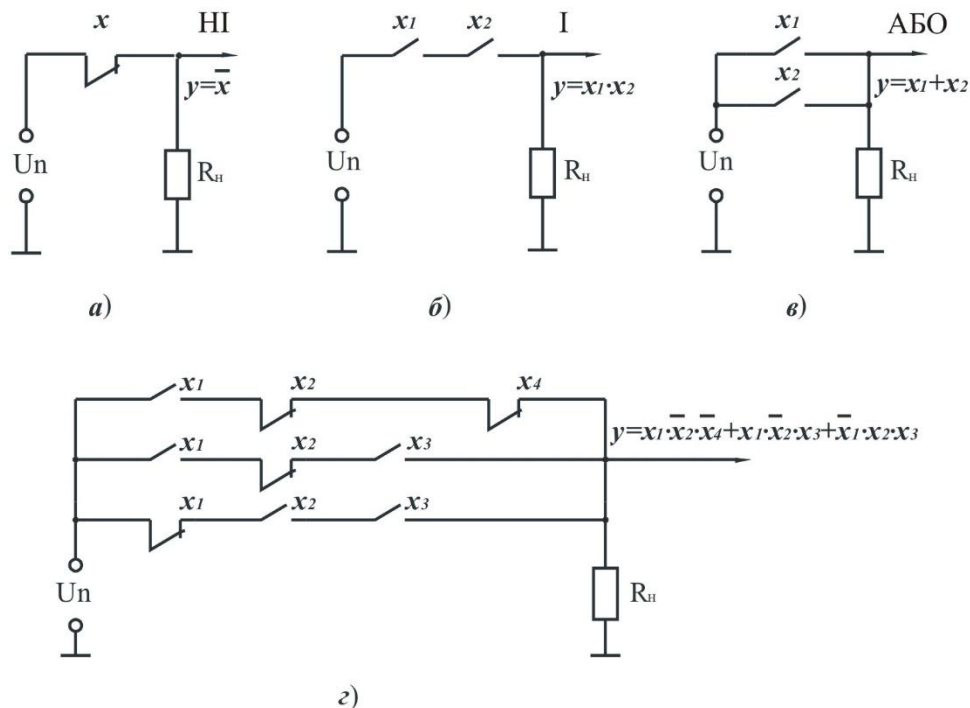


Рисунок 1.8 – Реалізація логічних функцій на РКЛ елементах

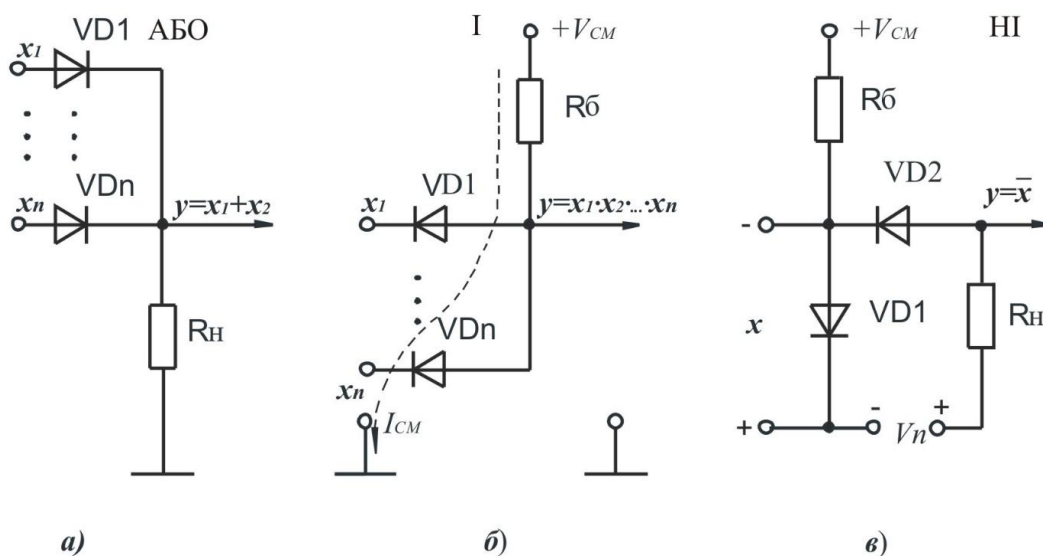


Рисунок 1.9 – Діодні логічні елементи

Функція «НІ» реалізується з використанням двох джерел живлення (рис. 1.9, в). Якщо відсутній вхідний сигнал x , то діод $VD1$ відкритий і на виході буде високий потенціал (1) від V_n . Якщо на вхід поданий негативний потенціал, то діод $VD1$ закривається, струм через R_n 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 здобули найбільшого поширення, оскільки мають кілька 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 відкрито 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 ланцюзі колектора (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 крапки» (тобто «0»).

Використовуючи діод-транзисторні ЛЕ, можна реалізувати логічну функцію у функціонально повному базисі «І – НІ» (рис. 1.10, з), а застосовуючи резисторно-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 кілька 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 ($x = 0$), 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 ($y = 1$).

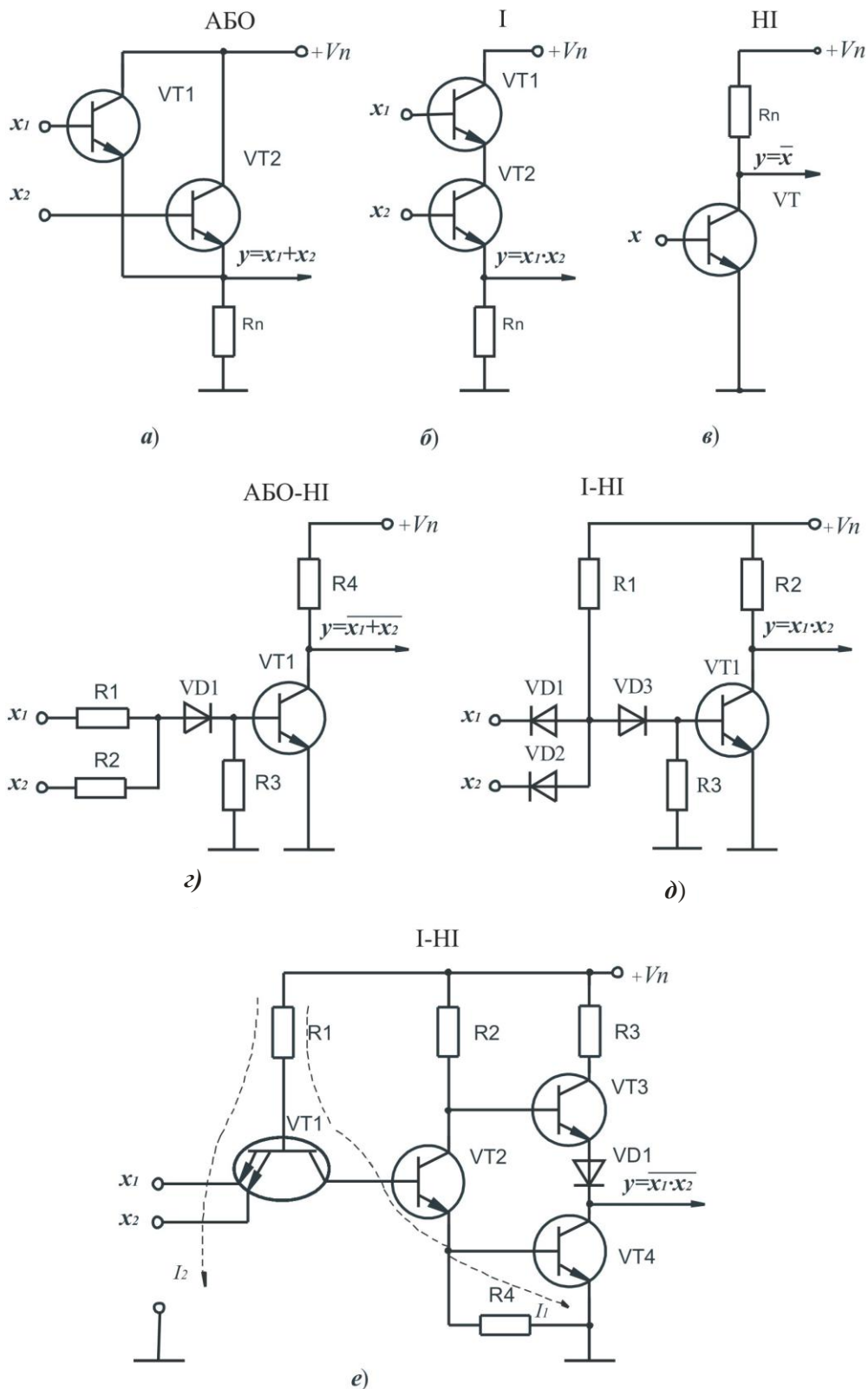


Рисунок 1.10 – Транзисторні логічні елементи

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 проектуваного пристрою – не головна вимога. Елемент, наведений на рис. 1.10, е, є базовим або типовим для ТТЛ і випускається 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 кілька входів і виходів, у яких певним комбінаціям вхідних сигналів відповідають цілком певні комбінації вихідних сигналів. Іноді такі пристрої називають декодерами. Різноманітність дешифраторів дуже широка, 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» у деякому розряді m -розрядного двійкового коду й $m-1$ нулів, або зворотним, якщо один «0» і $m-1$ одиниць. Наприклад, запис унітарного коду для $m=4$ буде мати вигляд:

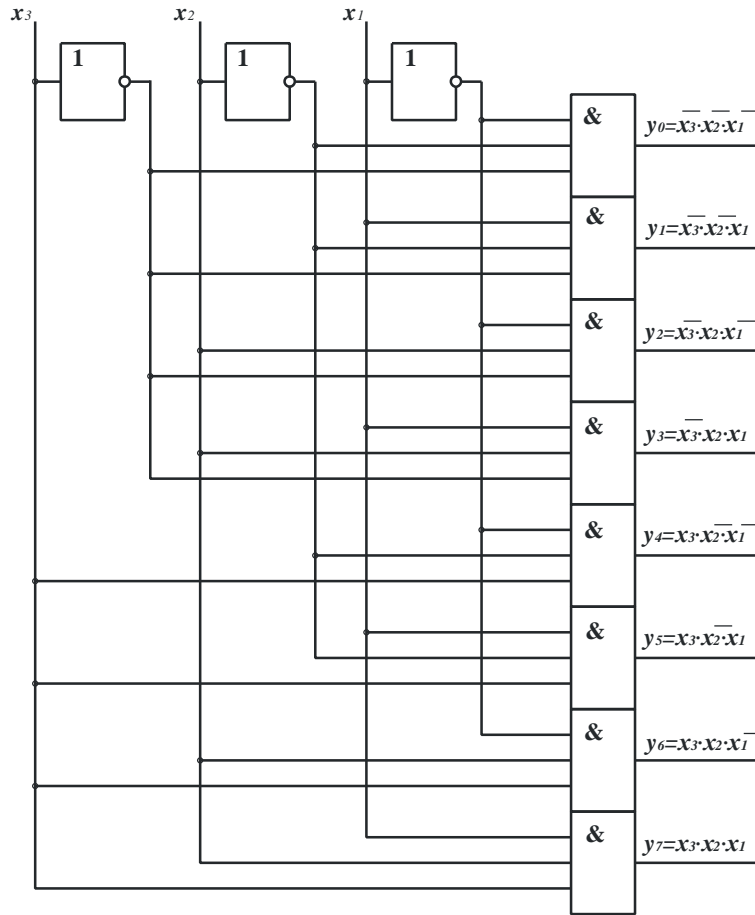
прямого – 1000, ..., 0010, 0001
зворотного – 0111, ..., 1101, 1110

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

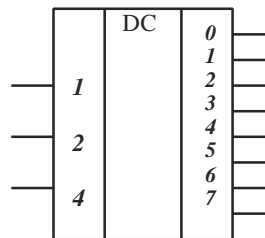
Функціональна схема дешифратора побудована відповідно до рівнянь $y_7 \dots y_0$ і являє собою 8 кон'юнкторів з 3-я входами, кожний з яких реалізує одну з вихідних функцій. Якщо виходи дешифратора проінвертувати, то вийде дешифратор зі зворотним унітарним кодом або, як його називають, дешифратор з інверсними виходами.

Таблиця 1.1 – Приклад дешифратора для $n = 3$ входів

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



a)



б)

Рисунок 1.11 – Дешифратор: а) – функціональна схема; б) – умовне позначення

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

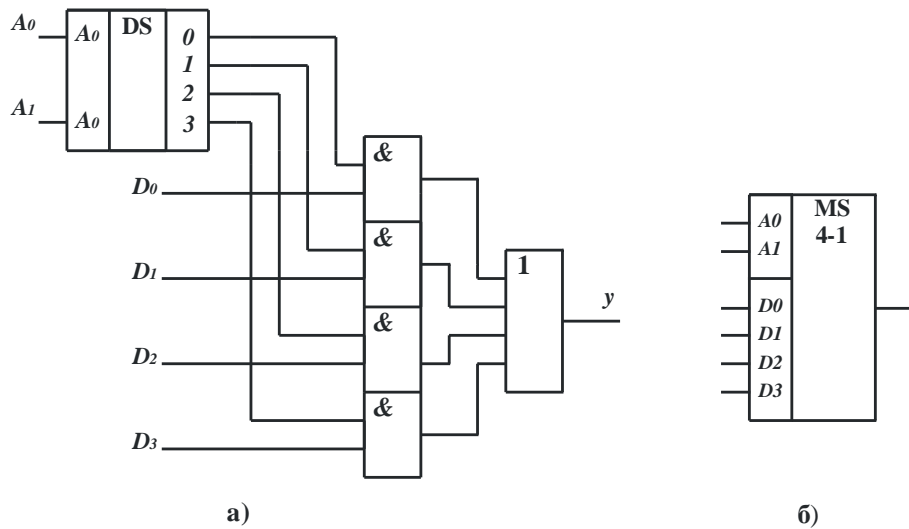


Рисунок 1.12 – Мультиплексор: а) – функціональна схема; б) – умовне зображення

Мультиплексор побудований як сукупність дешифратора двійкового адресного коду й чотирьох двохходових елементів «І», виходи яких об'єднані елементом «АБО». Очевидно, що стан виходу буде залежати від адресної комбінації A_0, A_1 й стану відповідної їй входу D_i .

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

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

Таблиця 1.2 – Приклад функціонування демультимплексора для $n = 4$ виходів

D	A_0, A_1	ϑ_0	ϑ_1	ϑ_2	ϑ_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 демультимплексора і його умовне зображення показані на рис. 1.13.

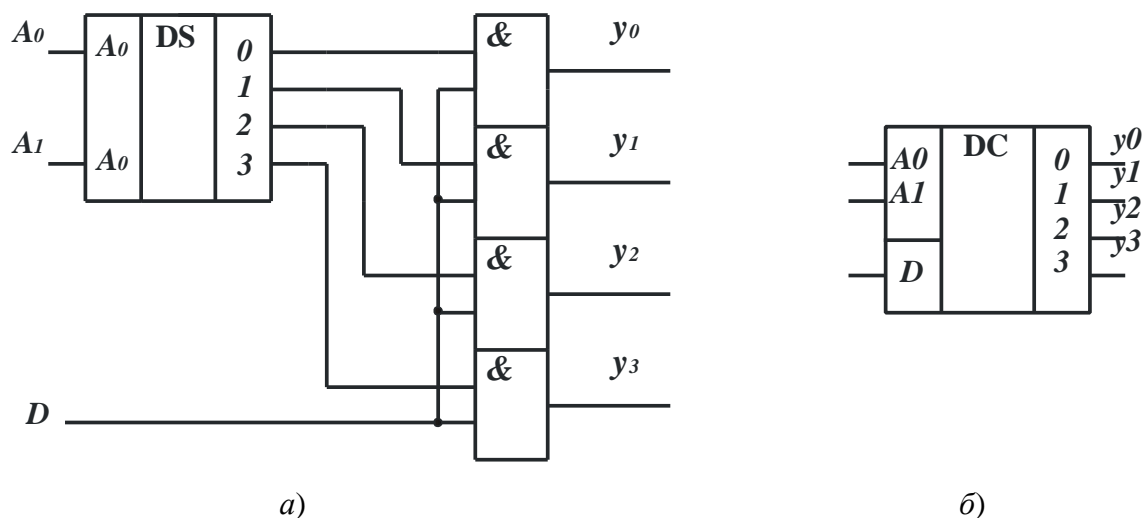


Рисунок 1.13 – Демультимплексор: а) – функціональна схема; б) – умовне зображення

Схему реалізовано з використанням тривходових елементів «І» і двох інверторів. Передача сигналу із входу (D) 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» (M2). 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 показані на рис. 1.14. Очевидно, що на виході напівсуматора буде 1 тільки при комбінаціях вхідних сигналів 01 і 10.

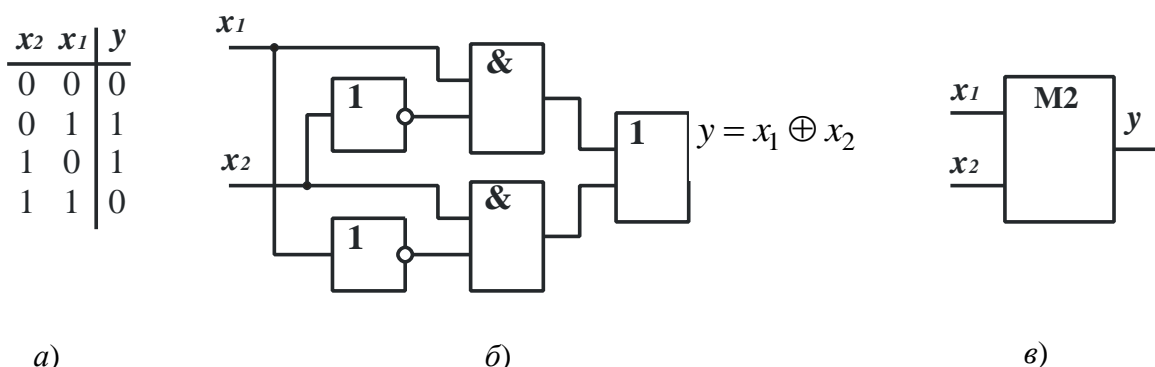


Рисунок 1.14 – Функція M2: а) – таблиця істинності; б) – функціональна схема; в) – умовне зображення

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 й зведені в табл. 1.3 і 1.4.

Таблиця 1.3 – Правила й приклад виконання арифметичного додавання

Двійкове додавання			
Доданки k -го розряду	Сума k -го розряду	Перенос в $k + 1$ -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 й приклад виконання арифметичного вирахування

Двійкове віднімання				
Зменшуване k -го розряду	Від’ємне k -го розряду	Різниця k -го розряду	Позика з $k + 1$ -го розряду	Приклад
0	– 0	= 0	0	010 – позика
0	– 1	= 1	1	1101 – зменшуване
1	– 0	= 1	0	– 1010 – від’ємний
1	– 1	= 0	0	0011 – різниця

Для виконання арифметичних операцій над двійковими числами зі знаком вводять додатковий (знаковий) розряд, який указує, чи є число позитивним або негативним. Якщо число позитивне, у знаковий розряд проставляється символ 0, якщо негативне, то символ 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 привласнюється код знака доданків, наприклад:

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

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 привласнити 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. Позитивні числа в прямому, зворотному й додатковому кодах мають той самий вид, а негативні – різний.

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

При записі негативного двійкового числа в додатковому кодi, 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 підсумовування багаторозрядних двійкових чисел – двійковому суматорі, необхідно передбачити ланцюг циклічного переносу. У випадку використання додаткового коду цей ланцюг відсутній.

З наведеного вище можна зробити наступний висновок. Застосування простого математичного «трюку» (подання двійкових чисел у зворотному або додатковому кодi) 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
$5 - 3 = 5 + (-3) = 2$	$x_1 = 0,101$ $x_2 = 1,100$ $\begin{array}{r} 0,101 \\ + \underline{1,100} \\ 10,001 \\ \quad \uparrow \\ 0,010 \end{array}$ Перенос у молодший розряд. Сума позитивна	$x_1 = 0,101$ $x_2 = 1,101$ $\begin{array}{r} 0,101 \\ + \underline{1,101} \\ 10,010 \\ \quad \uparrow \text{ x } \\ 0,010 \end{array}$ Одиниця переносу в молодший розряд ігнорується. Сума позитивна
$3 - 5 = 3 + (-5) = -2$	$x_1 = 0,011$ $x_2 = 1,010$ $\begin{array}{r} 0,011 \\ + \underline{1,010} \\ 1,101 \end{array}$ Перенос у молодший розряд відсутній. Сума негативна й зображена у зворотному коді	$x_1 = 0,011$ $x_2 = 1,011$ $\begin{array}{r} 0,011 \\ + \underline{1,011} \\ 1,110 \end{array}$ Сума негативна й зображена в додатковому коді

Однорозрядний двійковий суматор

Підсумовування багаторозрядних двійкових чисел $A = a_n a_{n-1} \dots a_0$ і $B = b_n b_{n-1} \dots b_0$ 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) й перенос у наступний старший розряд (p_{i+1}).

Принцип дії однорозрядного суматора наведена в табл. 1.6.

Для синтезу схеми однорозрядного суматора необхідно записати логічні рівняння для функцій виходів S_i і P_{i+1} й виконати перетворення відповідно до правил алгебри логіки. Для цього необхідно перегрупувати вихідне рівняння за прямим і інверсним значенням аргумента p_i , винести їх за дужки й перетворити у функції нерівнозначності (M2). Це перетворення наведено у формулах 1.13 і 1.14.

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

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 перетворюється й рівняння для функції P_{i+1}

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

Умовна позначка й функціональна схема однорозрядного суматора, що побудована відповідно з отриманими виразами, наведені на рис. 1.15.

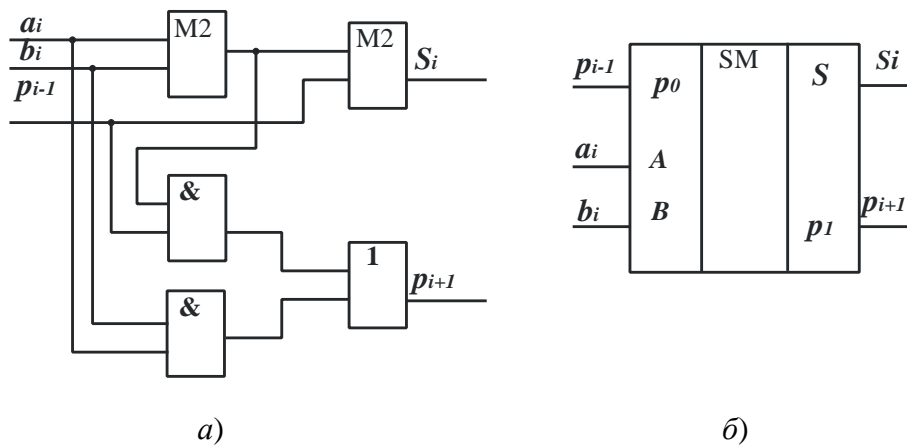


Рисунок 1.15 – Однорозрядний суматор: а) – функціональна схема; б) – умовна позначка

Багаторозрядний паралельний суматор може бути складений з однорозрядних суматорів, число яких дорівнює числу розрядів, що складаються шляхом з'єднання виходу, на якому 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 $11\dots11$ і $00\dots01$ буде мати місце «пробіг» одиниці переносу через увесь суматор від самого молодшого до самого старшого розряду. Даний тип суматора найбільш простий з погляду схеми ланцюгів поширення переносу.

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

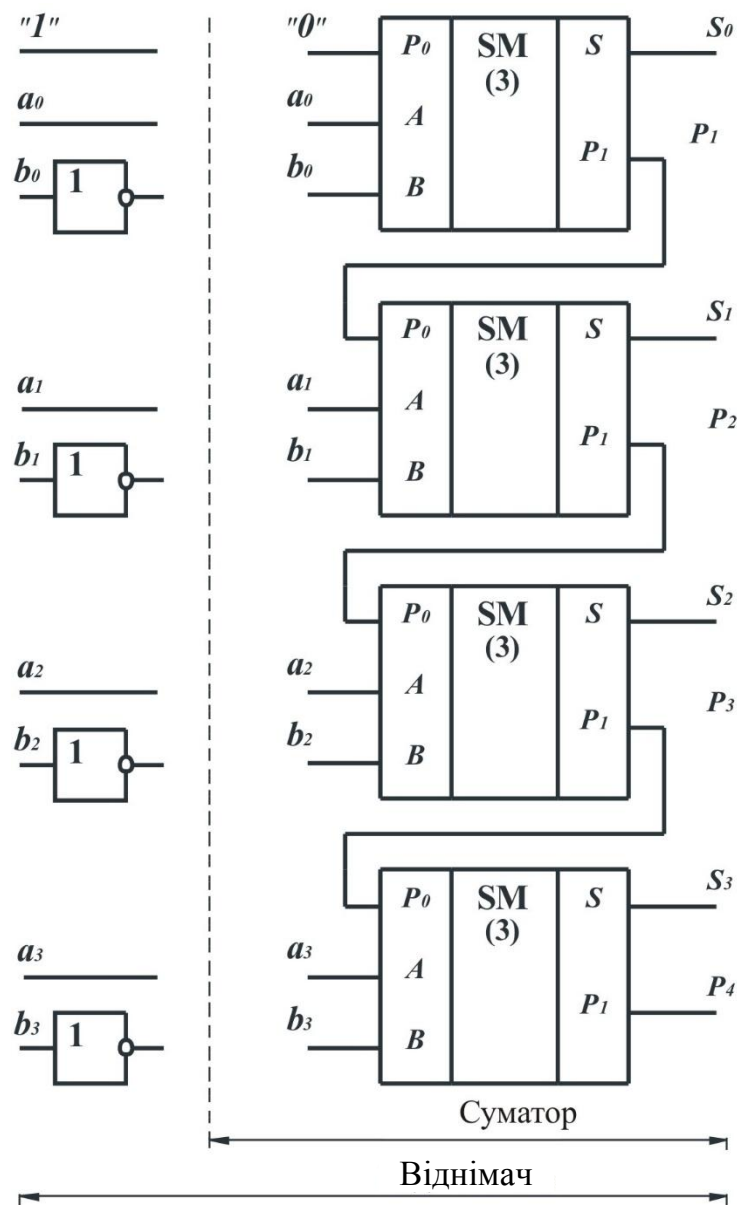


Рисунок 1.16 – Багаторозрядний суматор-віднімач

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, АЛП являють собою пристрої більш високого класу. В АЛП перетворення виконуються над багаторозрядними двійковими операндами.

До основних виконуваних операцій відносяться:

- логічні – порозрядні диз'юнкція або кон'юнкція;
- арифметичні – додавання, віднімання, додавання за модулем 2;
- спеціальні – інверсія, інкремент або декремент, зрушення 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 включення 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 проводиться 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.

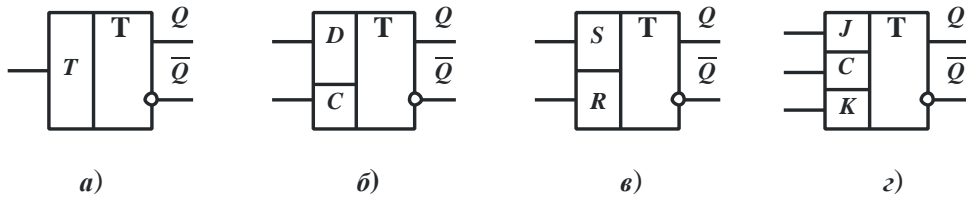


Рисунок 1.17 – Умовні зображення: а) – T -тригер; б) – D -тригер; в) – $R-S$ -тригер; г) – $J-K$ -тригер

Логіка функціонування тригерів залежить не тільки від значення вхідних змінних (керуючих сигналів), що надходять у 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 сполучають. У таких таблицях стану тригери на попередньому й наступних кроках записуються як Q^k і Q^{k+1} . Функціонування переходів і виходів для T -тригера й D -тригера наведені в табл. 1.7. У цих тригерах є по одному керуючому входу, які позначаються відповідно символами T і D .

Таблиця 1.7 – Функціонування переходів і виходів: а) – T -тригера; б) – D -тригера

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 символом C . У проміжку між синхросигналами стан тригера не змінюється. Функціонування D -тригера будуть розглянуті нижче.

Стан переходів і виходів для тригерів $R-S$ і $J-K$ наведена в табл. 1.8. У цих тригерах є окремі входи для установки виходів у стан «1» відповідно S і J і для установки 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 $J-K$ -тригерів

$R^k (K^k)$	$S^k (J^k)$	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 веде, і веденого. Ці тригери завжди такі, що синхронізуються. Перший ступінь перемикається за фронтом синхроімпульсу, а другий – за зрізом. Час перемикання веденого ступеню 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 розрізняють тригери асинхронні та синхронні або такі, що синхронізуються. В асинхронних тригерах перехід до нового стану залежить тільки від стану 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 позначки входів синхронізації показані на рис. 1.18.

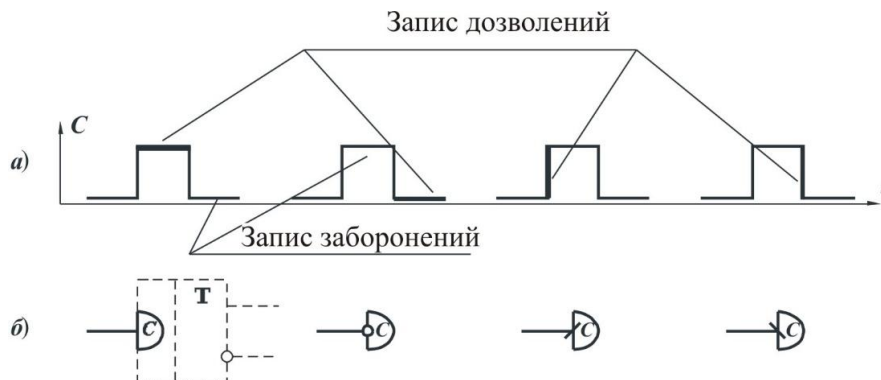


Рисунок 1.18 – Входи синхронізації: а) – тимчасові діаграми; б) – умовні позначки

Найпростішим із тригерів 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 показує, що функціонування тригера не залежить від типу елементів, на яких вони реалізовані. Відмінністю є те, що тригер, виконаний на елементах «АБО – НІ», керується прямими входними сигналами R й S (рівень логічної 1), а на елементах «І – НІ» – інверсними сигналами \bar{R} й \bar{S} (рівень логічного 0). Для варіанта а на рис. 1.19 наведено тимчасову діаграму. Тригер встановлюється при $S = 1$ й скидається при $R = 1$.

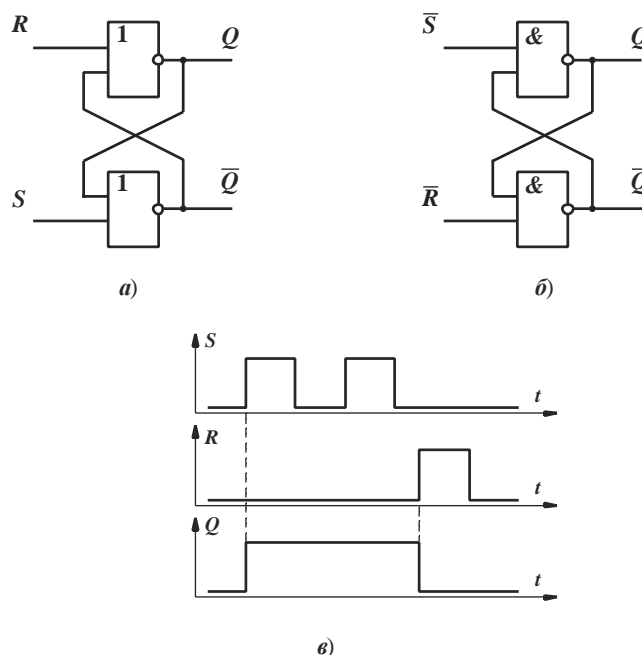


Рисунок 1.19 – Асинхронний $R-S$ -тригер: а) – у базисі «АБО – НІ»; б) – у базисі «І – НІ»; в) – тимчасові діаграми

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 й тимчасові діаграми її роботи наведено на рис. 1.20.

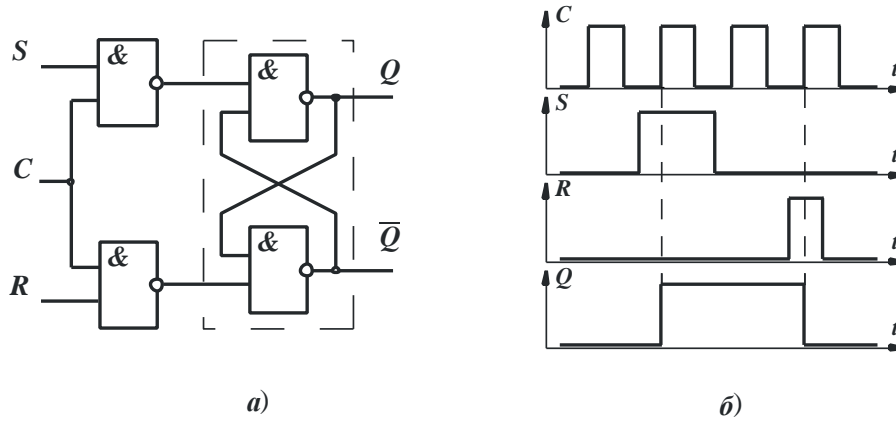


Рисунок 1.20 – Синхронний $R-S$ -тригер: *a)* – функціональна схема; *б)* – тимчасові діаграми

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 й тимчасові діаграми роботи D -тригера так само можна побачити на рис. 1.21.

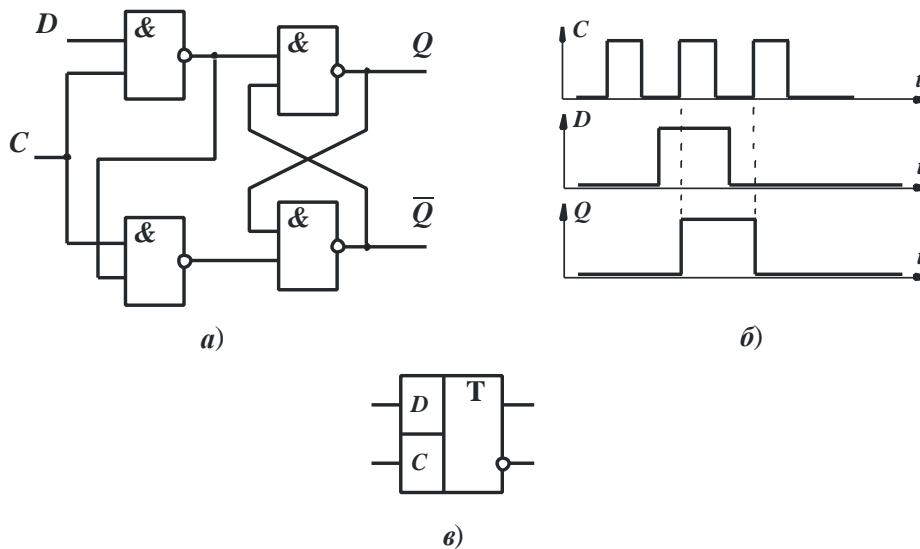


Рисунок 1.21 – D -тригер: *a)* – функціональна схема; *б)* – тимчасові діаграми; *в)* – умовна позначка

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

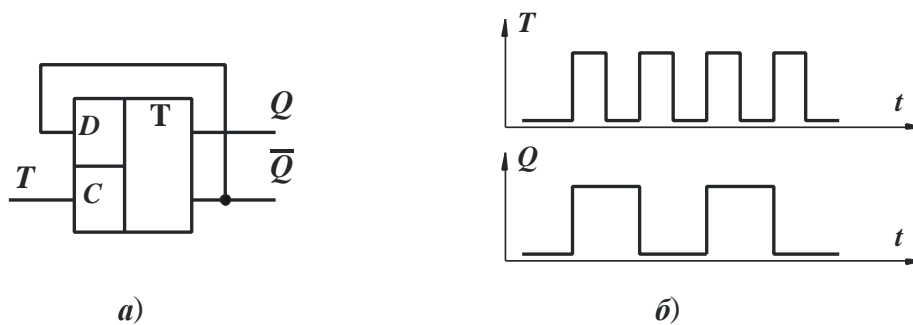


Рисунок 1.22 – T -тригер: а) – функціональна схема; б) – тимчасові діаграми

Регістри

Регістрами називаються цифрові пристрої, використовувані для запису й зберігання n -розрядного двійкового слова. Вони будуються на базі тригерів, число яких у схемі регістру звичайно відповідає числу розрядів двійкового слова. Особливістю роботи схеми регістру 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 комітками 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 зрушення на один або кілька розрядів убік молодшого або старшого розрядів; інкрементування або декрементування вмісту регістру. У структурі мікропроцесора це основний елемент.

Паралельний регістр

Паралельним називають регістр, у якому n -розрядне двійкове слово записується й зчитується одночасно по 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 схемної реалізації чотирирозрядного паралельного регістру і його умовну позначку наведено на рис. 1.23.

Загальними для розрядів регістру є ланцюги синхронізації (C) й скидання (R). Значення розрядів двійкового слова 0, 1, 2, 3 запишуться у відповідних тригерах тільки після подачі імпульсу синхронізації $C=1$. Після цього тригери

переходять у режим зберігання. Зчитування слова з регістру здійснюється із прямих Q -виходів тригерів. Інверсні виходи (\bar{Q}) використовуються рідше.

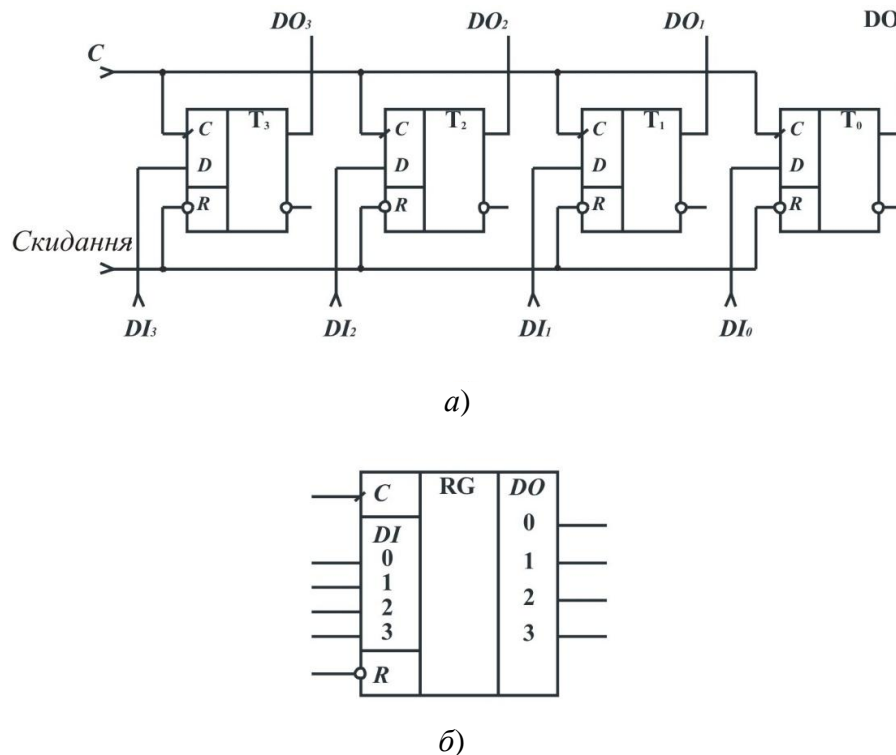


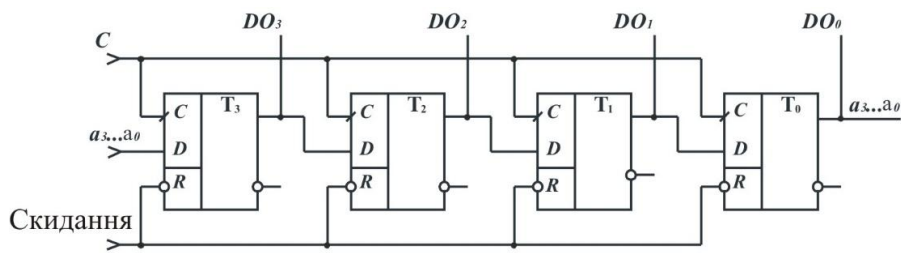
Рисунок 1.23 – Паралельний регістр: а) – функціональна схема; б) – умовна позначка

Слід нагадати, що після подачі живлення на схему регістру тригери встановлюються в довільні, заздалегідь непевні стани. Тому в наведеній схемі є окрема лінія установки регістру 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.

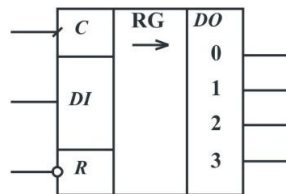
Послідовний регістр

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

На рис. 1.24 наведено приклад функціональної схеми й умовна позначка чотирирозрядного регістру, що виконує зрушення вправо.



а)



б)

Рисунок 1.24 – Регістр зрушення вправо: а) – функціональна схема; б) – умовна позначка

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 вступі імпульсу синхронізації на вхід C . За наступним синхроімпульсом значення з виходу Q_3 передається в тригер T_2 , а на вхід T_3 повинен подаватися наступний розряд слова, що вводиться. Таким чином, із надходженням кожного чергового синхроімпульсу проводиться зрушення інформації на один розряд вправо, і після четвертого синхроімпульсу регістр буде заповнений усіма розрядами слова, що вводиться.

Для реалізації процедури зрушення вліво використовуються двоступінчасті тригери, у яких за фронтом синхроімпульсу встановлюється перший щабель, а за зрізом – другий. На рис. 1.25 наведено приклад схеми такого регістру. Для передачі інформації з молодшого до старшого розряду вихід кожного тригера молодшого розряду з'єднано із входом тригера старшого розряду. До закінчення синхроімпульсу кожний тригер утримує інформацію, записану на попередньому кроці. Захоплення ж інформації відбувається за фронтом синхроімпульсу, що й забезпечує її зрушення вліво.

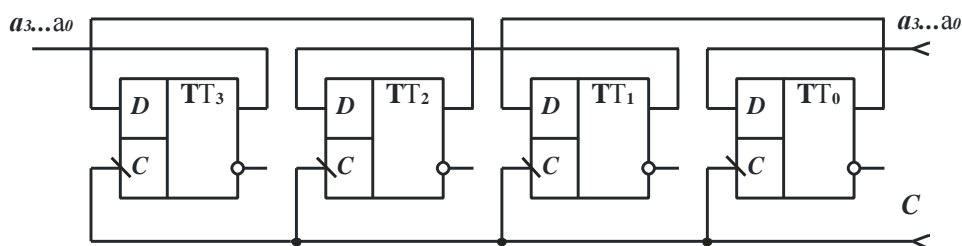


Рисунок 1.25 – Регістр зрушення вліво

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

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, яке може бути підраховане лічильником, звичайно 2^k . Де k дорівнює числу розрядів лічильника.

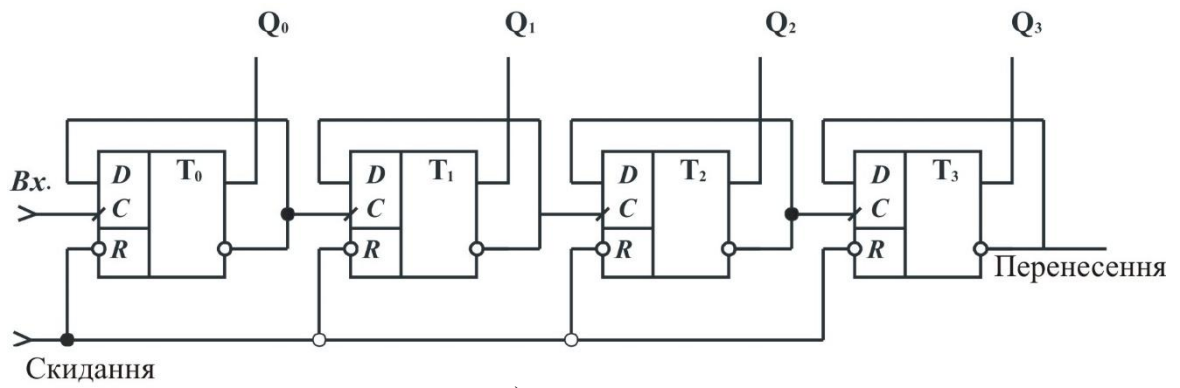
Двійкові лічильники можуть бути побудовані й на синхронних D -тригерах і $J - K$ -тригерах, перетворених у T -тригери.

Підсумовуючі двійкові лічильники

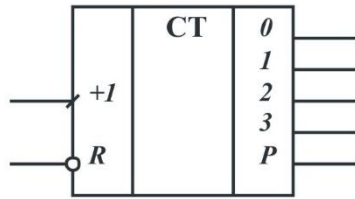
На рис. 1.26 наведено схему й тимчасові діаграми чотирирозрядного підсумовуючого двійкового лічильника з ланцюгами послідовного переносу. Інверсний вихід i -го розряду (тригера) з'єднано із входом $(i + 1)$ -го розряду. Лічильник побудовано на D -тригерах, які перетворені в асинхронні T -тригери, що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

Входом лічильника слугує 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_3 – старший). Ємність розглянутого лічильника $2^4 = 16$, тому максимальне його показання відповідне до подачі на вхід 15 рахункових імпульсів. 16-й рахунковий імпульс устанавлює всі тригери у вихідний (нульовий) стан, отже, шина «скидання» (установка «0») необхідна лише на початку роботи лічильника.

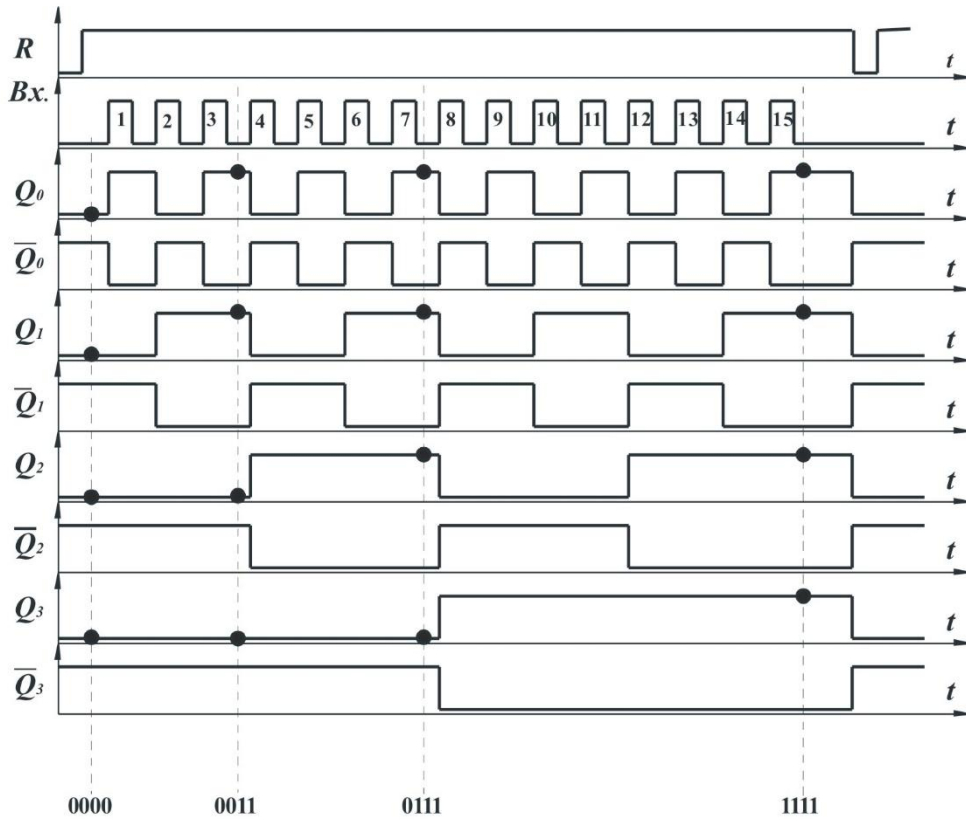
Після подачі кожного чергового вхідного імпульсу T -тригер переходить у протилежний стан. З тимчасової діаграми можна побачити, що період проходження імпульсів на виходах кожного розряду у два рази більший, ніж на його вході. У будь-який момент часу стан тригерів лічильника однозначно визначає число імпульсів, що надійшли на його вхід.



а)



в)



б)

Рисунок 1.26 – Підсумовуючий двійковий лічильник: а) – функціональна схема; б) – умовна позначка; в) – тимчасові діаграми

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 й зчитування змінюваних даних. У даному виді пам'яті зберігається інформація, що модифікується й використовується в процесі роботи. Це можуть бути різні змінні або результати проміжних і остаточних обчислень. Інформація при такому виді пам'яті губиться після вимикання живлення.

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

- 1К – це 1024 (2¹⁰) і читається як «кіло-» або «ка-». Обсяг пам'яті приблизно дорівнює однієї тисячі;
- 1М – це 1048576 (2²⁰) і читається як «мега-». Обсяг пам'яті приблизно дорівнює одному мільйону;
- 1Г – 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 2ⁿ байтів. У свою чергу кожний байт складається з 8 бітів.

Під коміркою 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 2Кx8 має 2048 восьмирозрядних комірок пам'яті типу ОЗП.

Оперативні запам'ятовувальні пристрої

Основою елементарної комірки ОЗП є тригер. На рис. 1.27 зображена її функціональна схема. Запам'ятовування й 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.

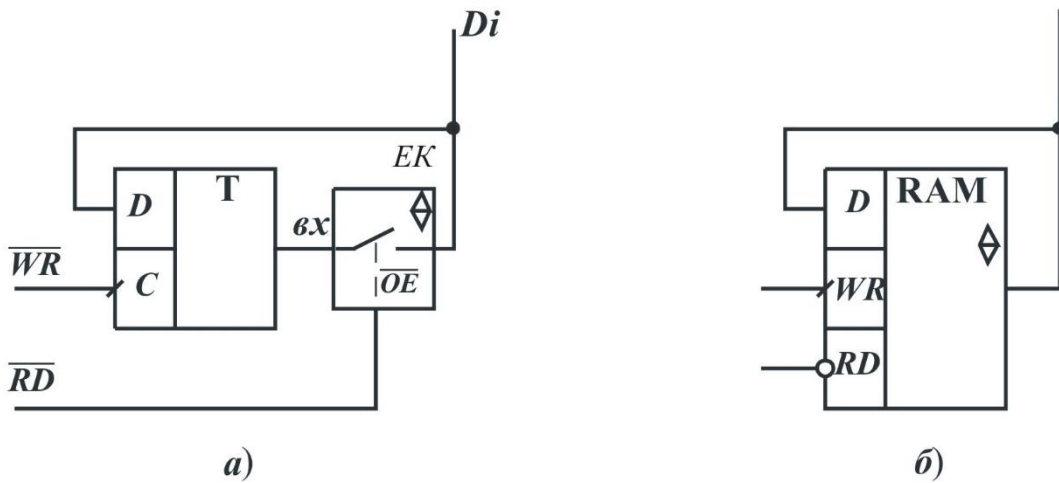


Рисунок 1.27 – Елементарна комірка ОЗП

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

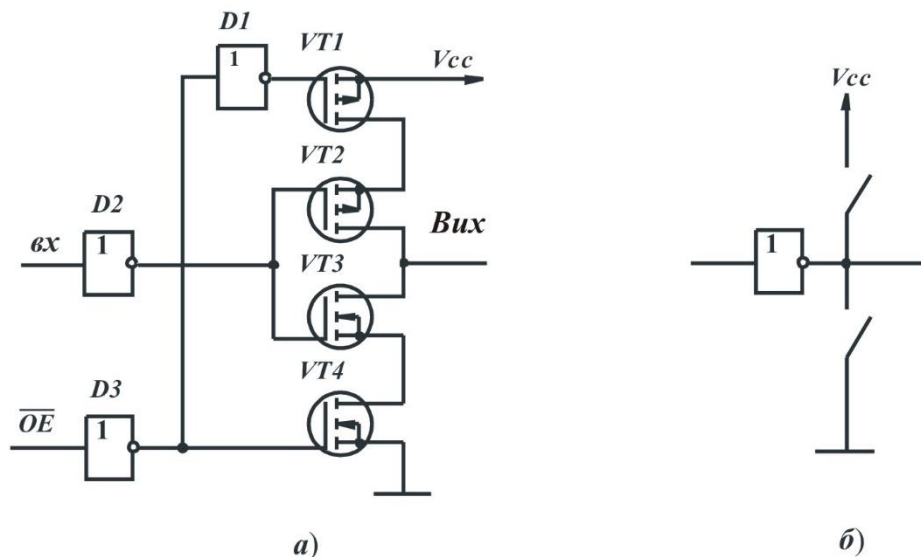


Рисунок 1.28 – Електронний ключ

Організація комірки пам'яті у такий спосіб дає можливість об'єднати її вхід і вихід на одній лінії даних *Di*. Для керування записом і читанням інформації є керуючі сигнали *WR* й *RD*. Сигнал *WR* подається на синхровхід тригера, а

сигнал RD – на вхід керування електронним ключем OE . Тимчасові діаграми запису й читання наведені на рис. 1.29.

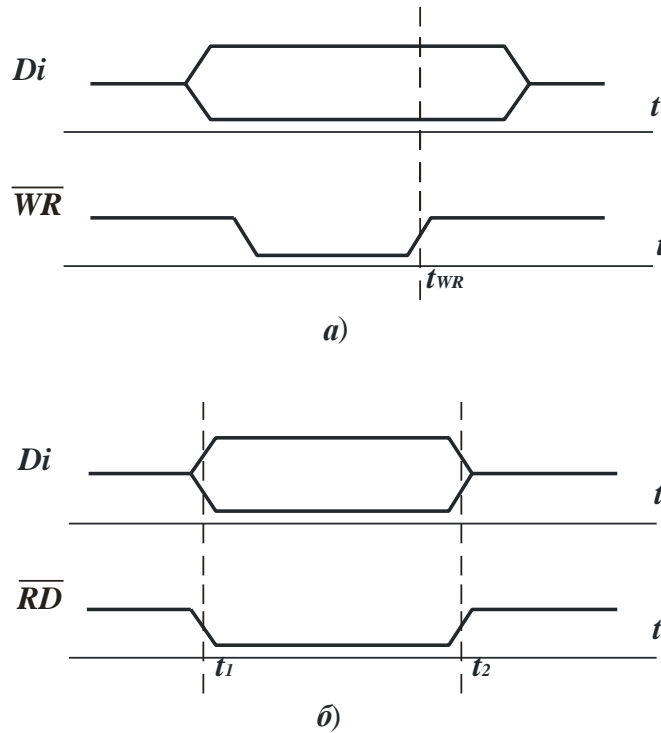


Рисунок 1.29 – Тимчасові діаграми: а) – запис ОЗП; б) – читання ОЗП

При записі інформації дані виставляються на лінію Di , після чого формується строб-імпульс на лінії WR . Установка 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 комірки 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 подається відповідна адресна комбінація $A_0...A_n$. Таким чином, кожна комірка пам'яті має адресу, за якою зберігається вміст (інформація).

В ОЗП є три керуючих входи. Для запису або читання – входи WR , RD , для дозволу роботи дешифратора – CS . Декодування адреси й установка 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 комірок в ОЗП. За сигналом WR проводиться 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.

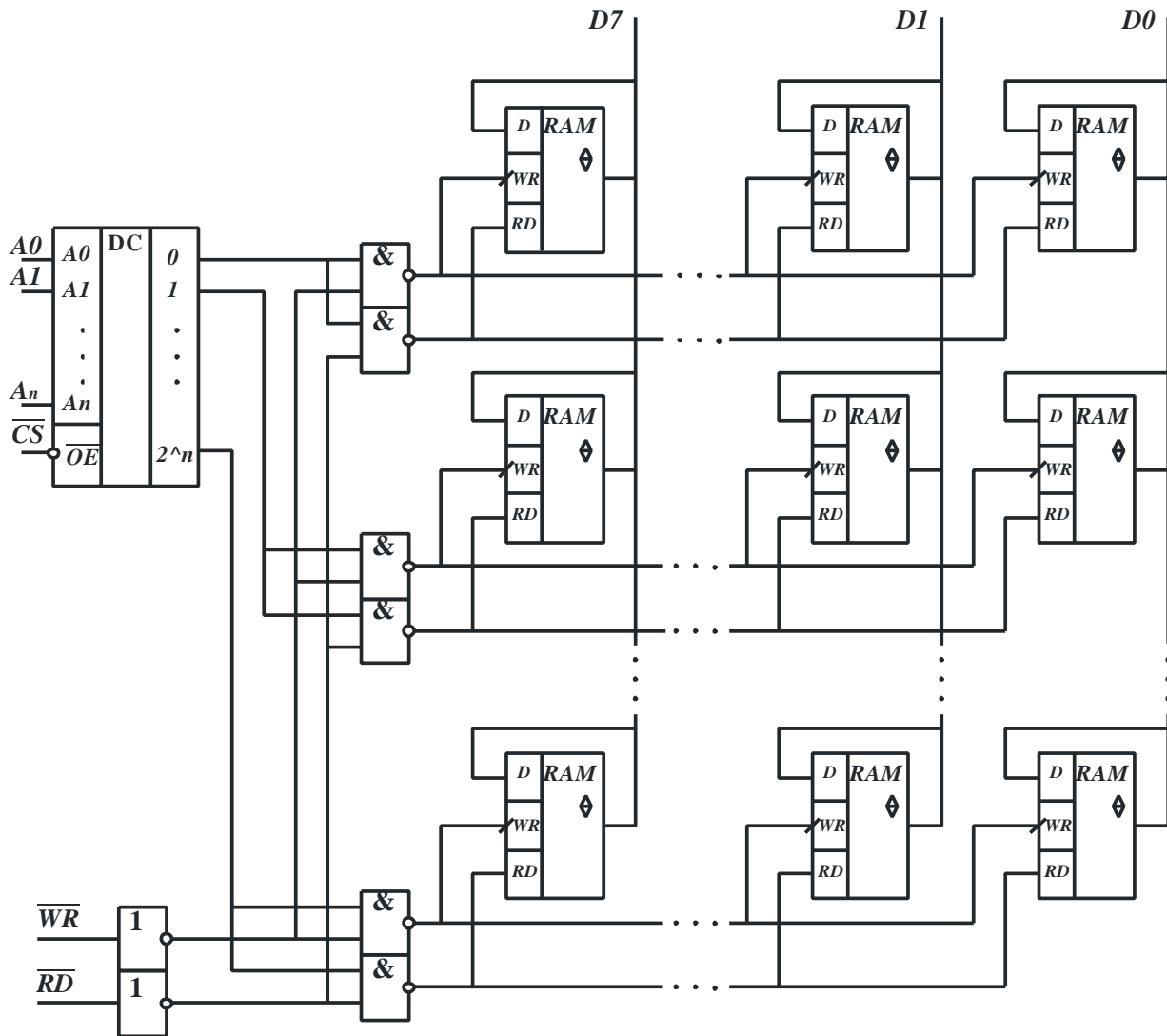


Рисунок 1.30 – Організація багаторозрядного ОЗП

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

Постійні запам'ятовувальні пристрої

Постійні запам'ятовувальні пристрої (ПЗП) різняться за принципом запису й 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.

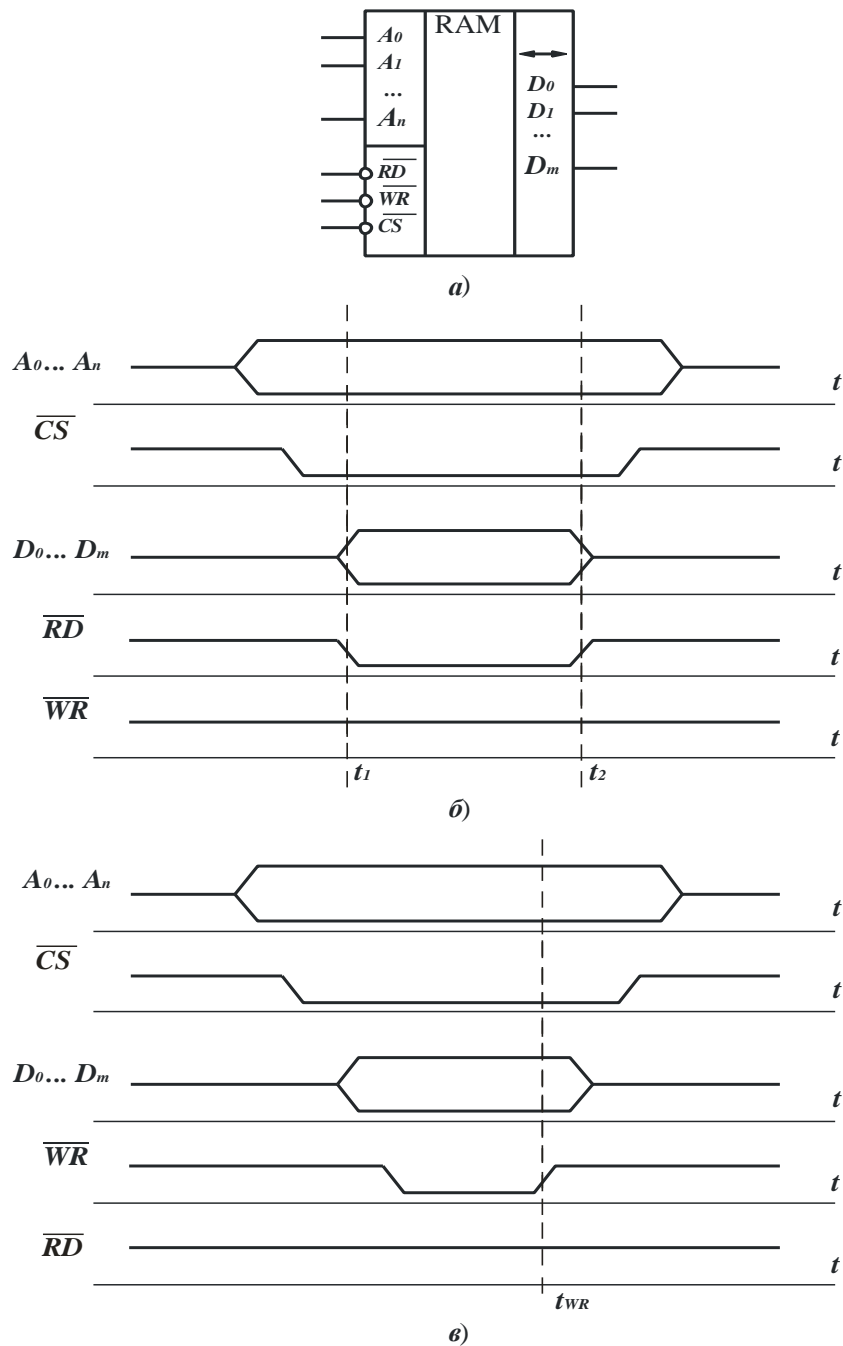


Рисунок 1.31 – Багаторозрядне ОЗП: а) – умовна позначка;
б), в) – тимчасові діаграми читання й запису

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. У якості елементарної комірки пам'яті використовується тригер, що виконано на базі польового транзистора з утримуючою ємністю. Функціональна схема такої комірки зображена на рис. 1.32. При вирядженій ємності $C1$ транзистор $VT1$ замкнено і на його виході втримується високий рівень сигналу (логічна 1). Заряд ємності проводиться при програмуванні ПЗП й у тих бітах, які повинні зберігати стан нуля. Для керування зарядом і розрядом ємності $C1$ використовуються електронні ключі $EK1$ і $EK3$.

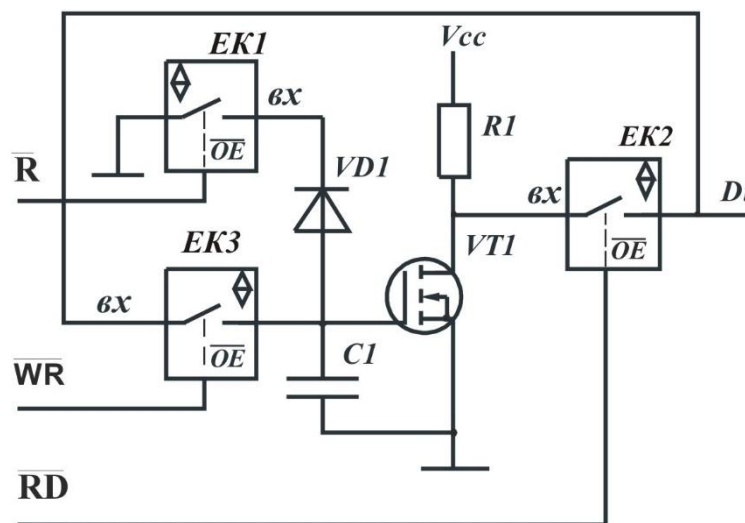


Рисунок 1.32 – Елементарна комірка ОЗП

При стиранні ПЗП керуючий сигнал подається на ключ $EK1$ і ємність розряджається через діод $VD1$. Після стирання всі комірки пам'яті встановлюються в одиничний стан. Керування записом інформації виконується за допомогою ключа $EK3$. Під час його відмикання на ємність $C1$ надходить сигнал з лінії даних (Di). Його високий рівень виробляє заряд ємності. Зчитування інформації із $VT1$ проводиться через ключ $EK2$, для чого повинен бути сформований керуючий сигнал на лінії RD . Оскільки електронні ключі забезпечують лінії із трьома станами, то вхід і вихід комірки пам'яті об'єднано.

Спосіб організації ПЗП є аналогічним організації багаторозрядного ОЗП. При звичайній роботі із ПЗП проводиться тільки зчитування інформації, тому на схемах умовних позначок лінії запису й стирання часто не зображуються. На рис. 1.33 наведено умовну позначку й тимчасову діаграму роботи ПЗП.

У ПЗП є два керуючих входи RD й CS , а також адресні входи $A_0...A_n$. Кількість адресних входів визначає ємність пам'яті. Так, при n рівному 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 повинно відбутися в проміжку між $t_1...t_2$.

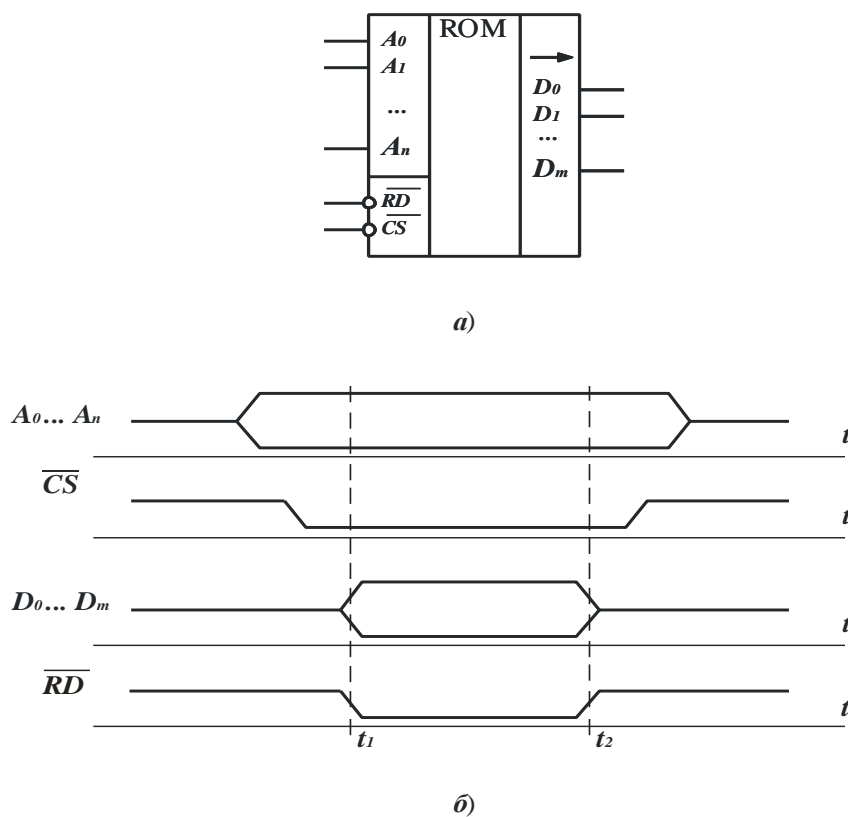


Рисунок 1.33 – ПЗП: а) – умовна позначка; б) – тимчасові діаграми

1.2.8. Методи й способи реалізації дискретних і цифрових СУ

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 ВІС ПЗП або ПЛМ, нарешті, з використанням програмних методів на базі мікропроцесорної техніки. Усе це є інструментом для розв'язку

поставленого завдання. Кожний зі способів має свої переваги й 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 й пристрої можуть бути реалізовані за допомогою дискретних вентильних схем або на ВІС ПЗП, ПЛМ. Крім того, існує чимало стандартних елементів, які випускаються промисловістю. Наприклад, перетворювачі двійково-десятькового 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 урахувувати, що якщо надалі виникне необхідність зміни алгоритму функціонування пристрою, то це приведе 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 й 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 алгоритма реалізації зазначених функцій зображено на рис. 1.34.

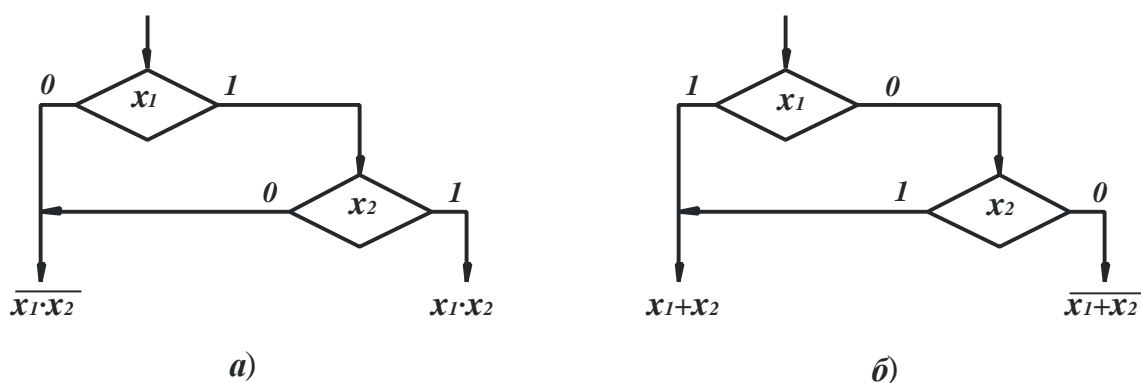


Рисунок 1.34 – Граф схема алгоритма реалізації логічних функцій «І» і «АБО»

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.

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 ПЗП або на ПЛМ являє собою програму, яку потрібно записати в 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 й АЛП являють собою процесор ЕОМ (рис. 1.35). Коли досягнення технології дозволили ПК й АЛП розмістити на одному кристалі, тоді й з'явився термін мікропроцесор (префікс мікро- більше стосується розмірів і вартості мікропроцесора).

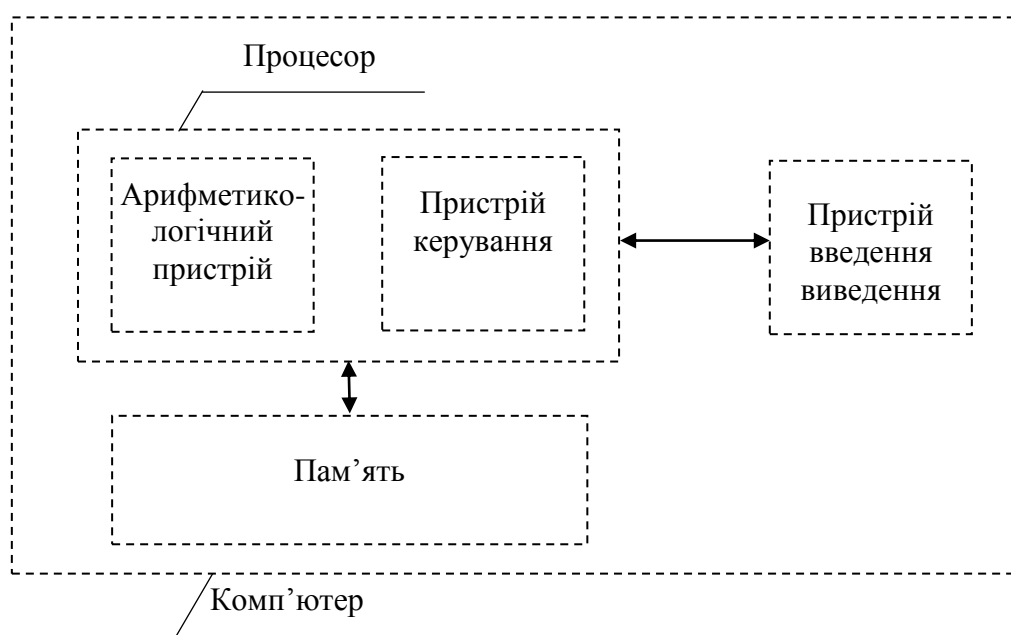


Рисунок 1.35 – Узагальнена структурна схема ЕОМ або комп'ютера

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

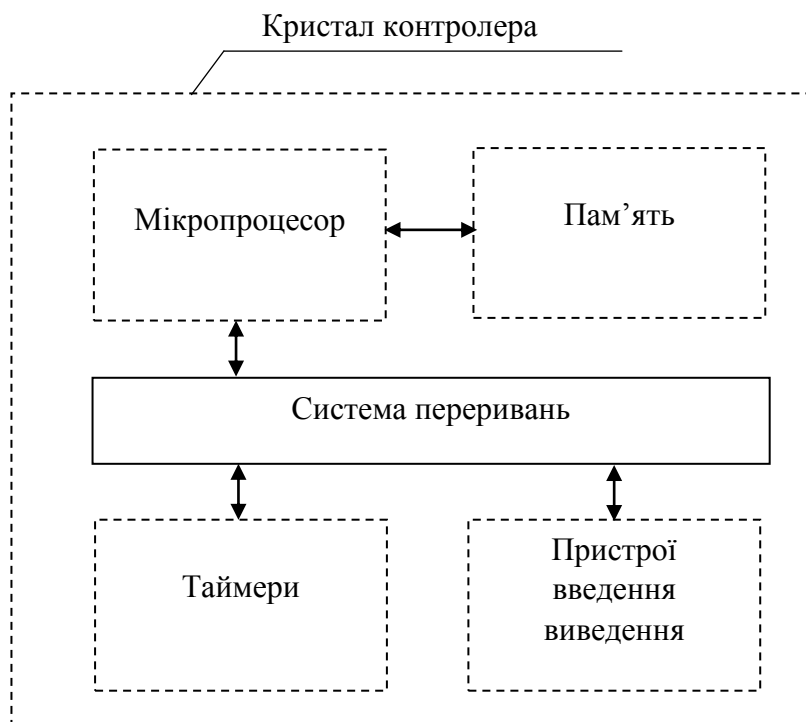


Рисунок 1.36 – Структурна схема мікроконтролера

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

Принцип роботи мікропроцесора розглянемо на прикладі його взаємодії з пам'яттю (рис. 1.36). Мікропроцесор містить шину адреса (16 двійкових розрядів), шину даних (як правило 8 двійкових розрядів), апаратні сигнали читання (RD) і запису інформації (WR), які визначають напрям пересилання інформації до мікропроцесора (RD) або від мікропроцесора (WR), Мікропроцесор на шину адреса завжди пересилає вміст лічильника команд (PC). При подачі напруги живлення на мікропроцесор формується сигнал скидання (RESET), при цьому, обнуляються всі вузли мікропроцесора і вміст лічильника команд складає 0000H, це і є адреса нульової комірки при цьому. При цьому формується сигнал читання RD і вміст нульової комірки пам'яті по шині даних пересилається до МП. Таким чином, мікропроцесор завжди починає свою роботу з читання нульової комірки пам'яті, вміст якої декодується і визначається яку операцію надалі буде виконувати мікропроцесор. Вміст

лічильника команд РС збільшується на 1 при цьому формується адреса наступної комірки пам'яті.

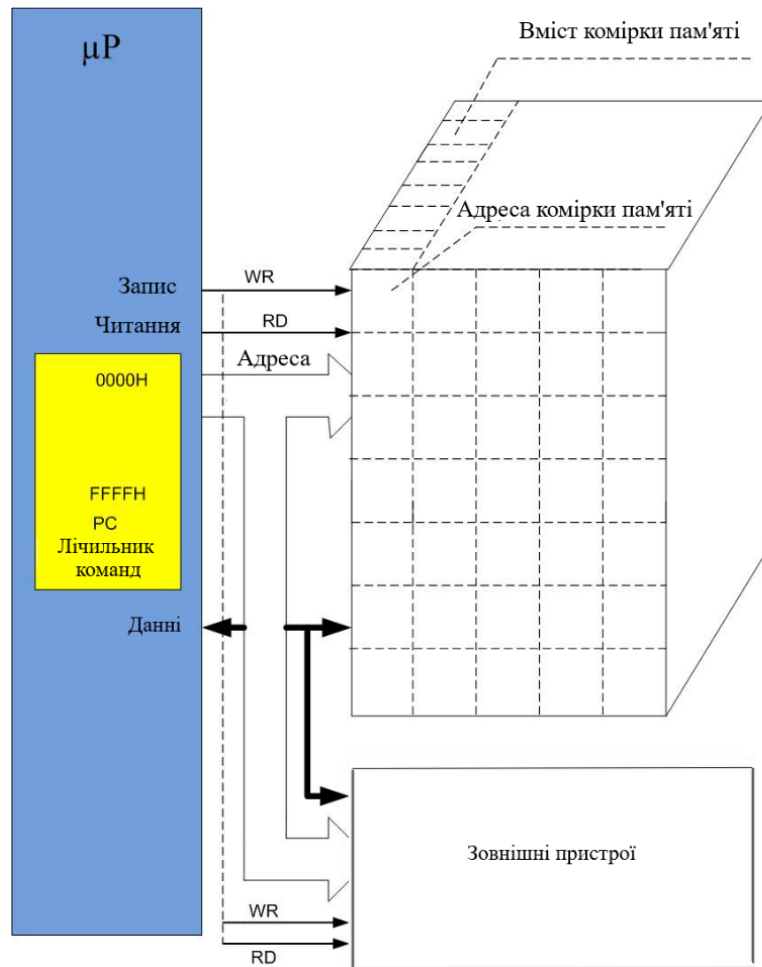


Рисунок 1.37 – Функціональна схема взаємодії мікропроцесора з вузлами комп'ютера

Контрольні запитання

1. Які види інформації можуть надходити на вхід системи керування?
2. Що таке об'єкт керування? Які бувають різновиди об'єктів керування?
3. Що таке логічна функція керування?
4. Що є аргументами функції керування?
5. Сформулювати основні закони булевої алгебри.
6. Сформулювати правила складання таблиць істинності.
7. Навести приклади базових логічних елементів.
8. Скласти таблицю істинності для повного лінійного дешифратора на 4 входи. Навести приклад функціональної схеми.
9. Навести приклади функціональних схем мультиплексора й демультиплексора на базі дешифратора на 3 входи.

10. Скласти таблицю істинності для суматора за модулем 2. Навести приклад функціональної схеми.

11. Навести приклади віднімання двійкових чисел для випадків, коли зменшуване більше від'ємника й коли зменшуване менше від'ємника.

12. Скласти таблицю істинності для однорозрядного двійкового суматора. Навести приклад функціональної схеми.

13. Що таке тригер? Які бувають різновиди тригерів?

14. Скласти таблицю переходів і виходів *D-Тригера*. Навести його функціональну схему й тимчасову діаграму.

15. Скласти таблицю переходів і виходів *T-Тригера*. Навести його функціональну схему, виконану на базі *D-Тригера*, й тимчасову діаграму.

16. Що таке паралельний регістр? Навести приклад його функціональної схеми.

17. Що таке послідовний регістр і які бувають його різновиди? Навести приклади їх функціональних схем.

18. Призначення й принцип роботи двійкового суматора.

19. Призначення й різновиди запам'ятовувальних пристроїв.

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

21. Розкрити принцип побудови елементарної комірки ОЗУ. Навести приклад функціональної схеми й тимчасової діаграми роботи.

22. Розкрити принцип побудови багаторозрядного ОЗУ. Навести приклад функціональної схеми.

23. Розкрити принцип побудови елементарної комірки ПЗУ. Навести приклад функціональної схеми й тимчасову діаграму роботи.

24. Які методи й 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 й комплект 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. 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 якої 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 одержала 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 випускають фірми Siemens, Philips, Atmel та ін.

Удосконалення технології й 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 й налагодження програмного забезпечення.

У 2020 році компанія Novaton виставила на ринок вдосконалений мікроконтролер 1Т8051, в якому повністю збережена система команд MCS-51 і 8-бітний високопродуктивний мікроконтролер на базі 1Т 8051 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 й 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 вводу/виводу, контролер переривання. Мікросхема живиться від одного джерела напругою +5В, потужністю до 1,5 Вт, допускає експлуатацію в діапазоні температур від -40 до +100°C. Загальними характеристиками для всієї сім'ї 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 (РС);
- 16-розрядний показчик даних ДРТР;
- синхронно-асинхронний приймач-передавач послідовного порту зі змінюваною швидкістю передачі;
- генератор тактових імпульсів (ГТІ);
- усі вхідні й вихідні сигнали узгоджуються ТТЛ рівнем.

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

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

Більшість команд МК51 виконуються з використанням акумулятора – регістру з бітовою адресацією, що забезпечує значне спрощення аналізу даних з бітовою структурою інформації. При використанні акумулятора передбачено два види адресації:

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

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

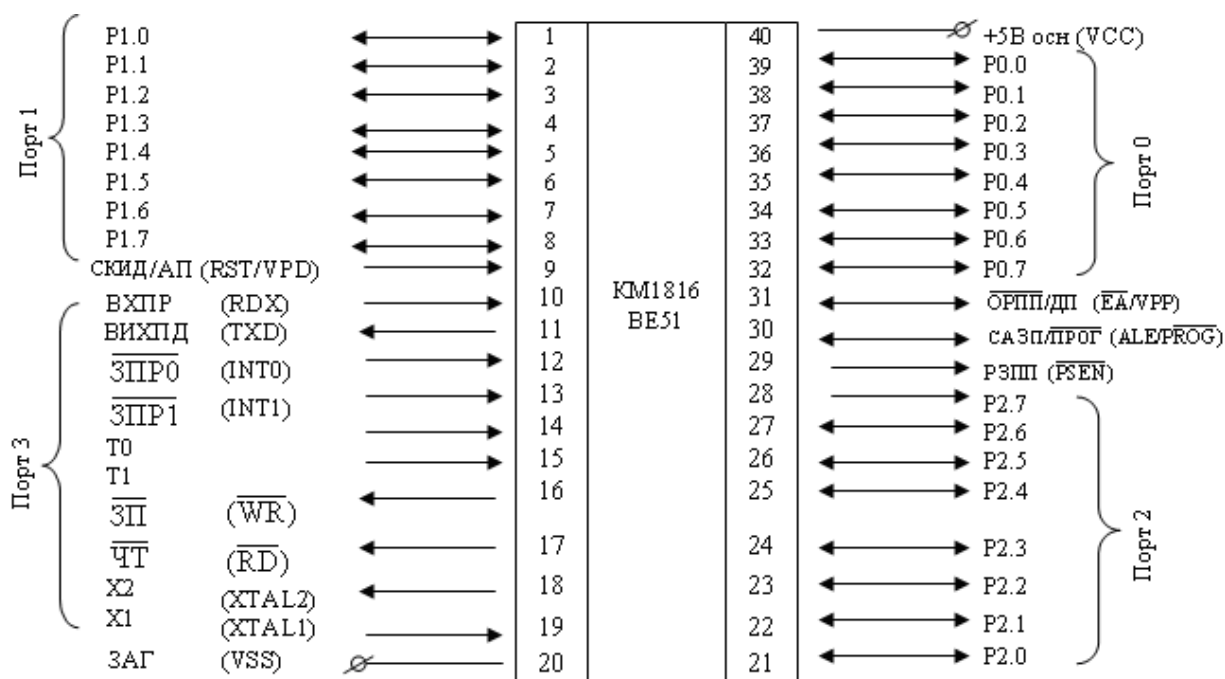


Рисунок 2.1 – Принципова схема мікроконтролера МК51

Таблиця 2.1 – Призначення виведення 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 й сигналів керування під час програмування й перевірки РПП
P3.0...P3.7	Вхід-Вихід	Порт 3 (P3) – восьмирозрядний 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	INT0 – вхід запиту від зовнішнього джерела переривання з умовним номером 0

P3.3	Вхід	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)	Строб адреси зовнішньої пам'яті. Використовується для керування режимом мультиплексування адреси й даних, які передаються через порт P0 при звертанні до ЗП. Якщо ALE=1, на контактах виведення порту P0 перебуває адреса. При програмуванні МК на вивід подається логічний «0»
#PSEN	Вихід	Дозвіл зовнішньої пам'яті програм. Виконує роль строба приймання 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, будуть виконуватися команди, розміщені в РПП, якщо (PC)=0000...0FFFH. Якщо подано #EA=0, будуть виконуватися команди, розміщені тільки у ЗПП (РПП повністю недоступна). Під час програмування МК на цей вивід подається імпульс напругою +21В
#EA/VPP	Вхід	Сигнал вимкнення резидентної пам'яті програм (РПП). Якщо подано #EA=1, будуть виконуватися команди, розміщені в РПП, при цьому (PC)=0000...0FFFH. Якщо подано #EA=0, будуть виконуватися команди, розміщені тільки у ЗПП (РПП повністю недоступна). Під час програмування МК на цей вивід подається імпульс напругою +21В
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 підсилювача-генератора синхросигналів. Підключається до зовнішнього джерела синхронізації (кварцового резонатора, увімкненого за схемою з «середньою крапкою»), рис. 2.5
XTAL2	Вихід	Вихід підсилювача-генератора синхросигналів. Підключення є аналогічним підключенню XTAL1
Vcc	-	Підключення до джерела живлення напругою Vcc=+5 В±10%
Vss	-	«Загальний» контакт

Регістр PSW є регістром ознак або регістром стану програми. У ньому формуються ознаки результатів, що виконуються в АЛП. Регістр PSW забезпечує керування вибором робочого банку регістрів, допускає побітову адресацію з використанням символічних імен PSW.3, PSW.4.

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

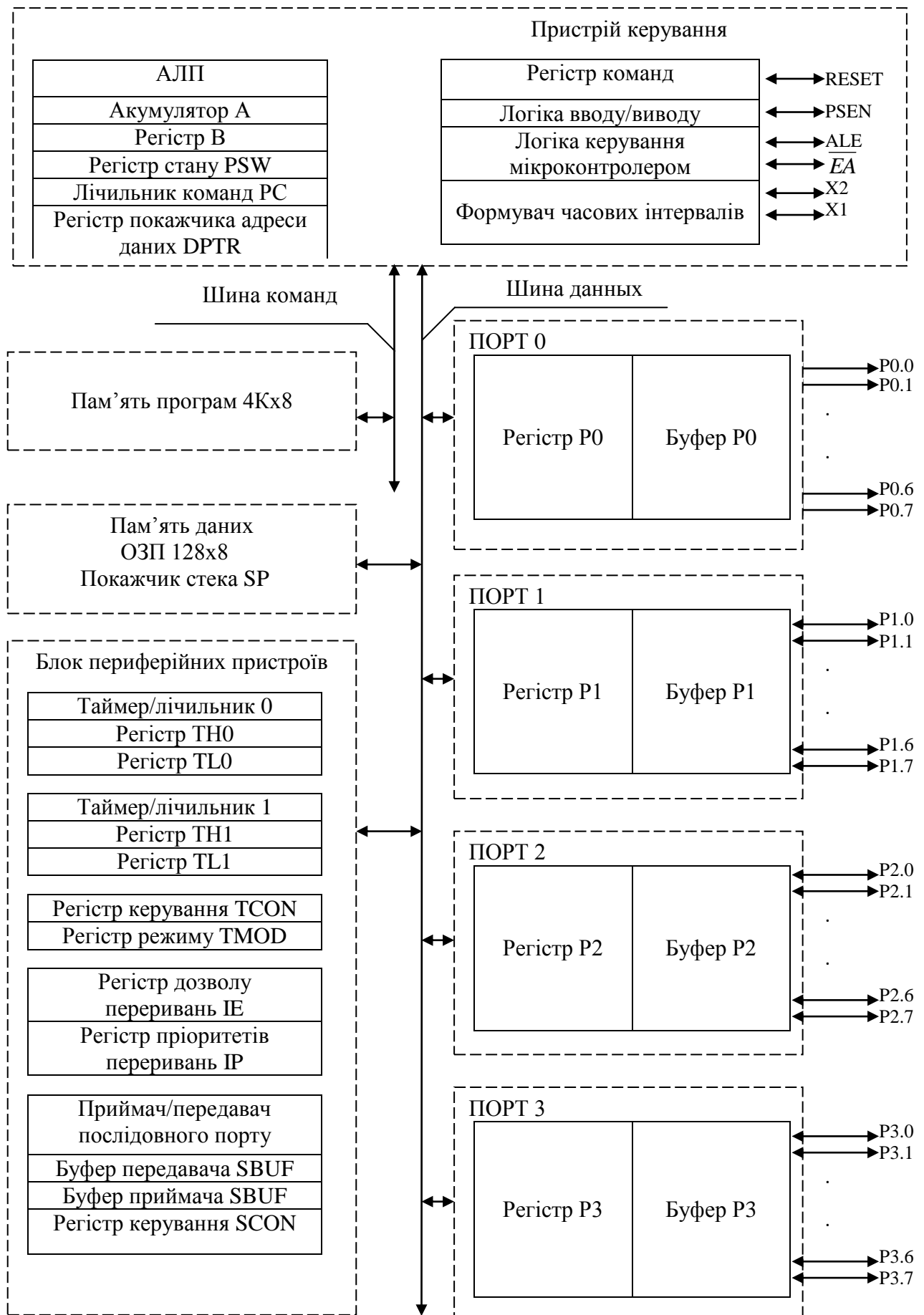


Рисунок 2.2 – Внутрішня структура МК51

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 INT0, INT1) і один сигнал від буфера послідовного обміну (RI ∨ TI).

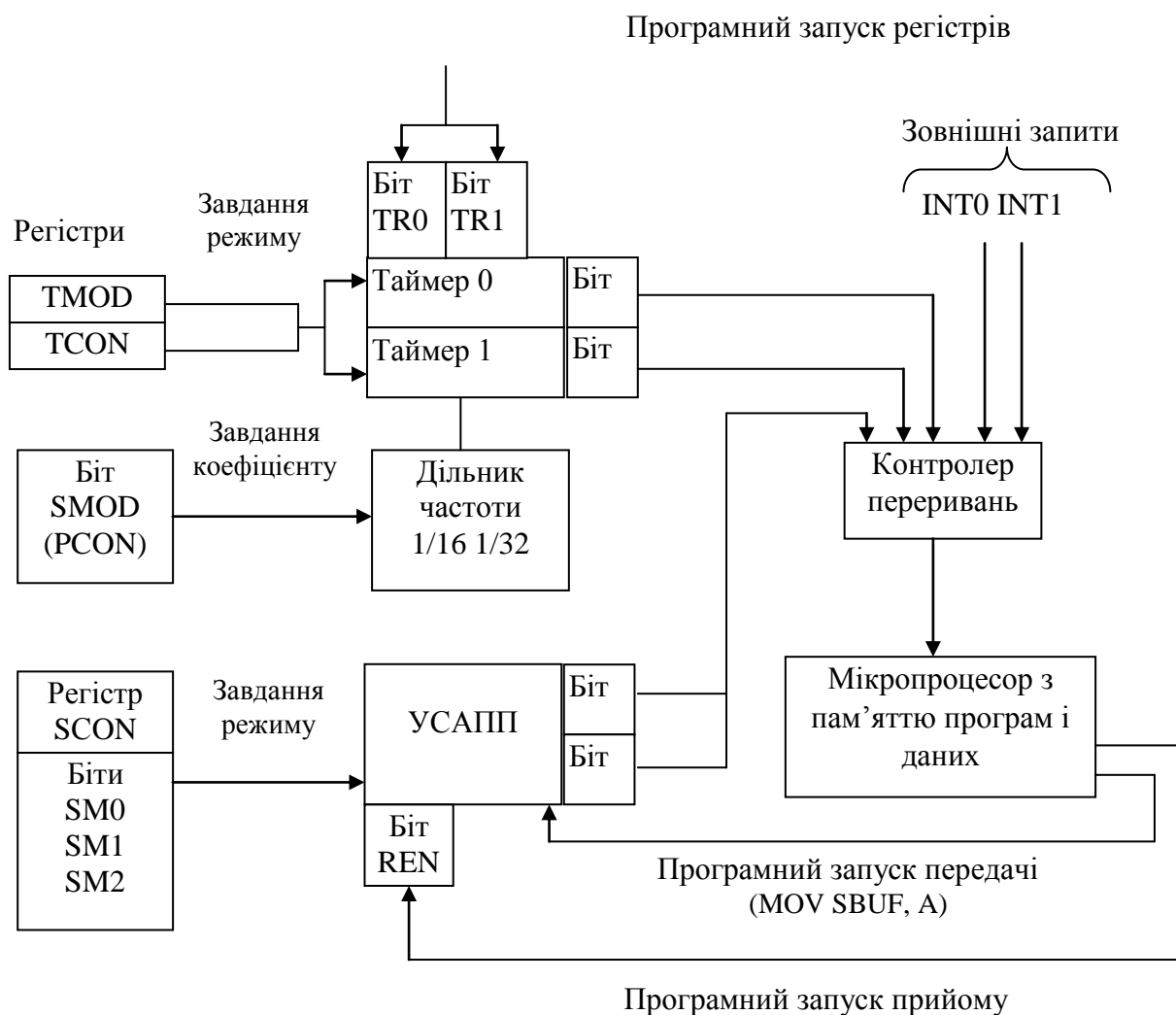


Рисунок 2.3 – Функціональна структура зв'язків внутрішніх вузлів МК51

Роботу мікроконтролера розділено 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 й 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).

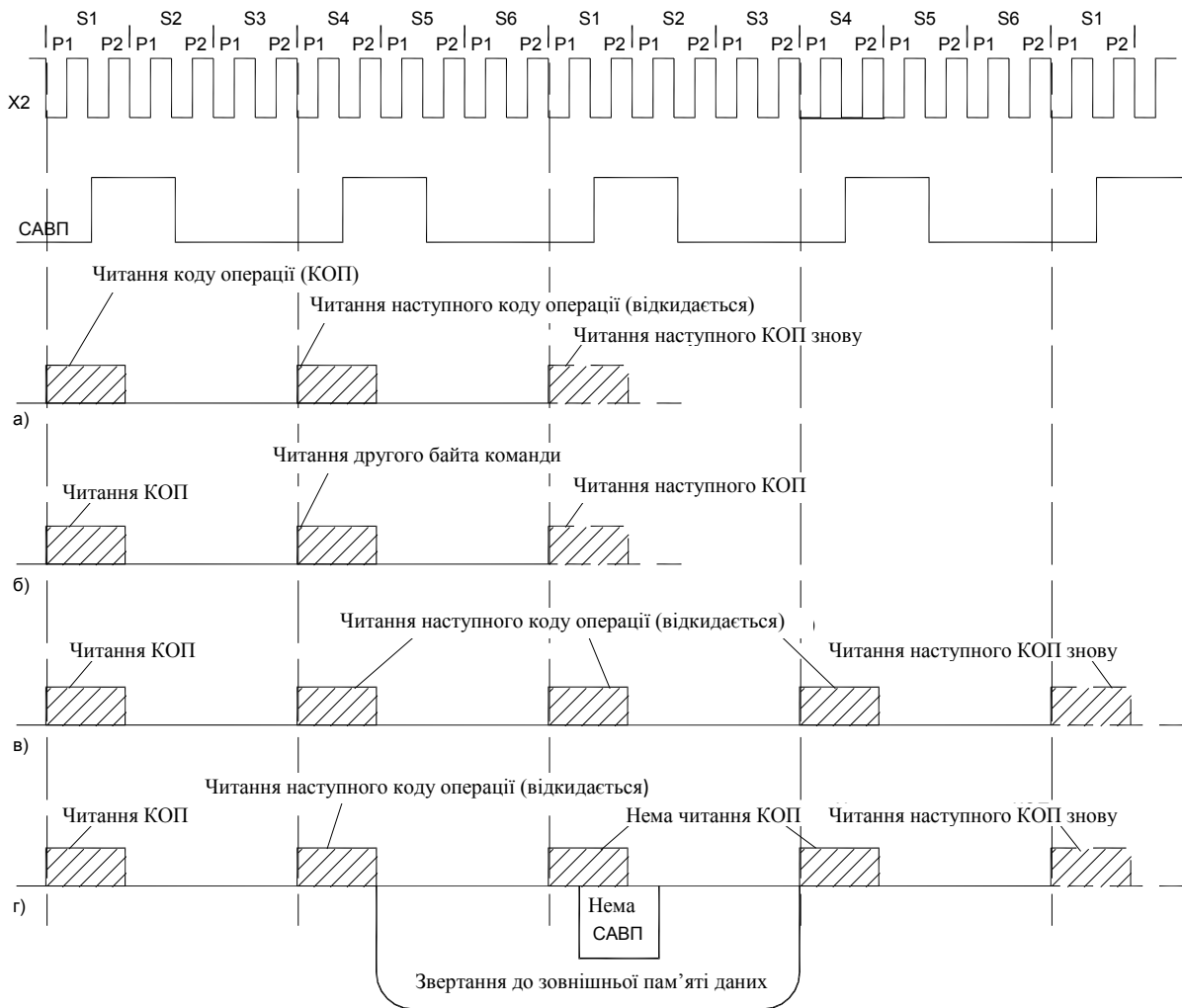


Рисунок 2.4 – Основні машинні цикли роботи МК51

Машинні цикли мають фіксовану тривалість, яка дорівнює 12 періодам коливань резонатора або шести станам первинного керуючого автомата (S1-S6). Кожний стан керуючого автомата містить дві фази (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, починаючи з фази S1P1 і закінчуючи фазою S6P2 (рис. 2.4). Усі заштриховані сигнали, наведені на рис. 2.4, є внутрішніми й недоступні користувачеві для спостереження.

Спостережуваними є сигнали резонатора й сигнал строба адреси зовнішньої пам'яті (САВП), який формується двічі за один машинний цикл (S1P2-S2P1 і S4P2-S5P1). Цей сигнал використовується для керування процесом пересилання інформації між МК51 і зовнішньою пам'яттю.

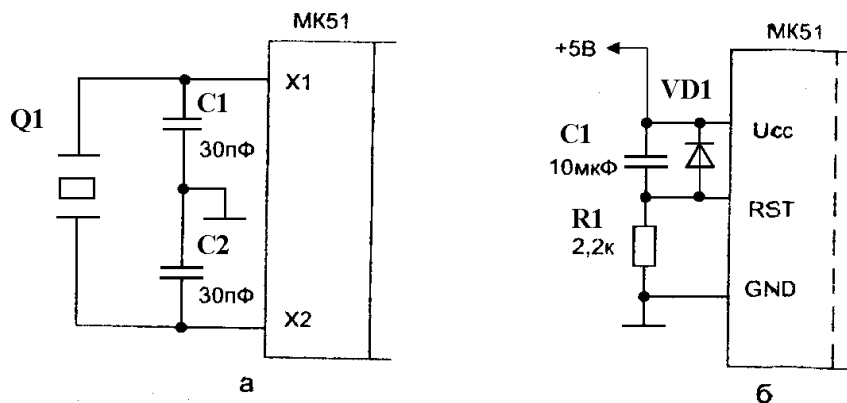


Рисунок 2.5 – Схеми підключення кварцових резонаторів до МК51

2.2. Програмно доступні ресурси МК51

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

Структуру пам'яті можна зобразити у вигляді (рис. 2.6). Пам'ять програм розділено на внутрішню (4 Кбайт) і зовнішню (60 Кбайт). При звертанні до молодших 4 К адрес сигнал $\overline{EA}=0$. При звертанні до старших 60 К 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 байт – 128 байт внутрішнього ОЗП й 128 адрес регістрів спеціальних функцій.

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

Область внутрішнього ОЗП визначена з адреси 00H до 7FH (табл. 2.2). 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 непрямо регістрової 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 з адреси 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.

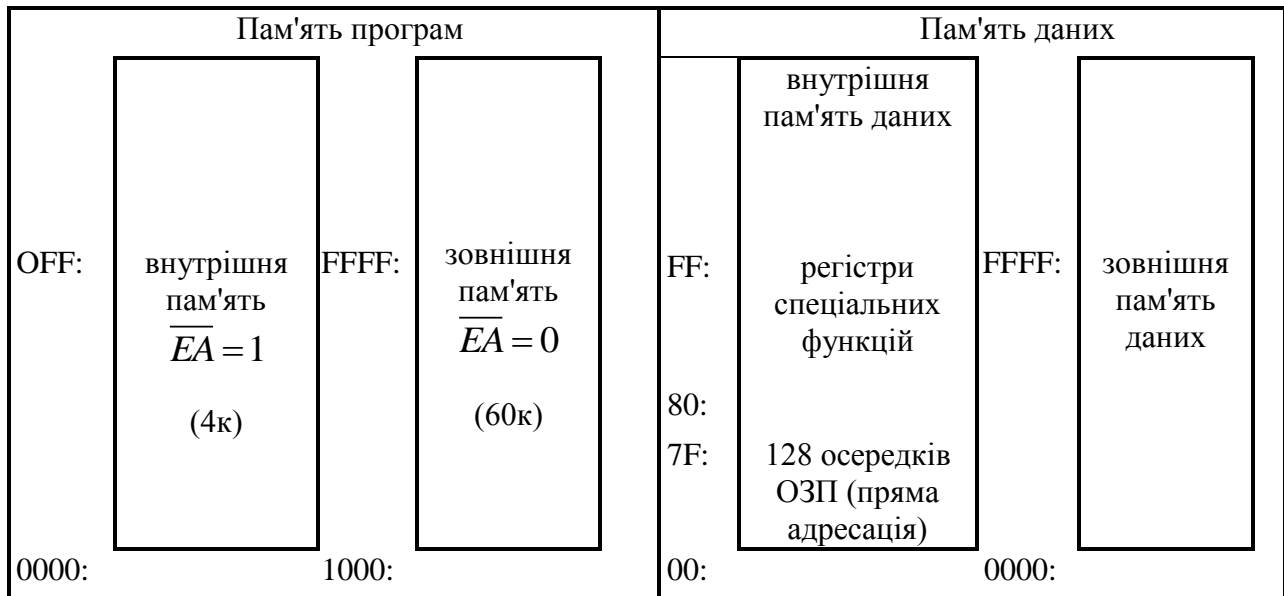


Рисунок 2.6 – Адресний простір мікроконтролера

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:	R7								31
18:	.								24
	R0								

17:	R7 . .				БАНК2				23
10:	R0								16
0F:	R7 . .				БАНК1				15
08:	R0								8
07:	R7 . .				БАНК0				7
00:	R0								0

Таблиця 2.3 – Адресний простір регістрів спеціальних функцій

Шістнадцяткова адреса					Десяткова адреса					Ім'я регістру
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 управління 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 управління 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 управління 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 управління 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 управління стековою пам'яттю										
81:										SP
Регістр показчик 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 являється регістр стану програми PSW.

Призначення окремих розрядів регістру PSW наведено в табл. 2.4.

Таблиця 2.4 – Призначення окремих розрядів регістру PSW

Назва біта	Позиція	Призначення																				
C	PSW.7	Прапор переносу. Встановлюється й скидається апаратно при виконанні арифметичних, логічних і бітових операцій, а також програмно																				
AC	PSW.6	Прапор допоміжного переносу. Встановлюється й скидається тільки апаратно при виконанні команд підсумовування й віднімання у випадку виникнення переносу або позики в біті 3 акумулятора																				
F0	PSW.5	Вільний прапор. Може бути змінено програмно й використовується за призначенням, встановленим програмістом																				
RS1 RS0	PSW.4 PSW.3	Вибір банку регістрів. Біти встановлюються й скидаються програмно для вибору активного (робочого) банку регістрів: <table border="1" data-bbox="587 1055 1439 1245"> <thead> <tr> <th>RS1</th> <th>RS0</th> <th>Активний банк</th> <th>Адреси РПД</th> </tr> </thead> <tbody> <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>08H-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-1FH</td> </tr> </tbody> </table>	RS1	RS0	Активний банк	Адреси РПД	0	0	0	00H-07H	0	1	1	08H-0FH	1	0	2	10H-17H	1	1	3	18H-1FH
RS1	RS0	Активний банк	Адреси РПД																			
0	0	0	00H-07H																			
0	1	1	08H-0FH																			
1	0	2	10H-17H																			
1	1	3	18H-1FH																			
OV	PSW.2	Прапор переповнення. Встановлюється й скидається апаратно при виконанні арифметичних операцій у випадку переповнення акумулятора. Дає можливість коректно виконувати дії над числами, зображеними в додатковому коді																				
-	PSW.1	Не використовується																				
P	PSW.0	Прапор паритету. Встановлюється й скидається апаратно в кожному циклі команди, фіксує факт непарної кількості «1» в акумуляторі																				

До РСФ можна звертатися як до звичайних осередків РПД. Більш детальне призначення й 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, званої APROM, в якій передбачено уміст користувача, рис. 2.4.

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

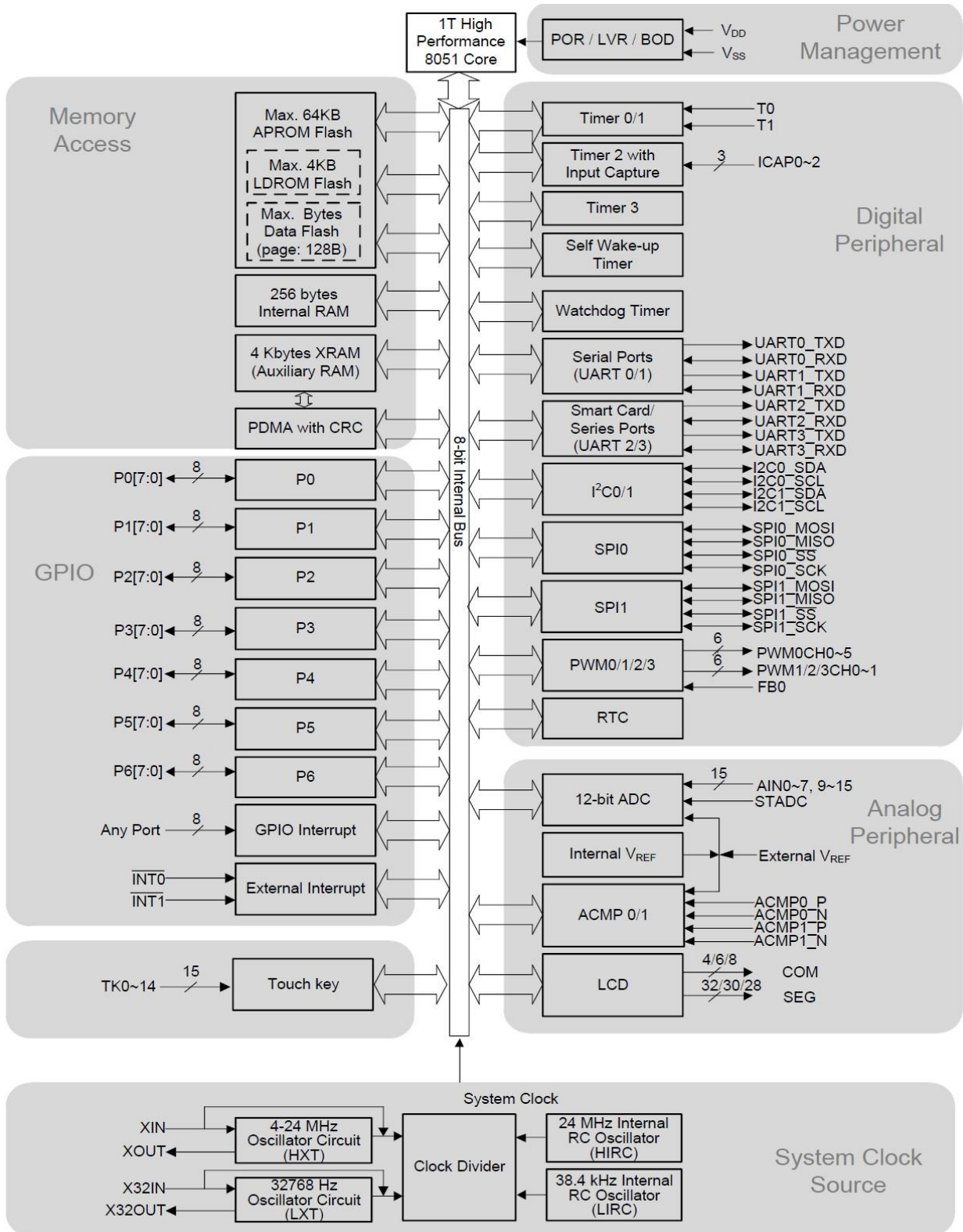


Рисунок 2.7 – Внутрішня структура мікроконтролера 1T8051

Існує додаткова флеш-пам'ять під назвою LDROM (табл. 2.5) для проведення внутрісистемного програмування (ISP). Розмір LDROM налаштовується до 4KBytes за допомогою конфігурування бітів регістра спеціальних функцій (рис. 2.5). Для полегшення програмування і перевірки Flash дозволяє програмувати і читати вміст комірок пам'яті. Удосконалений мікроконтролер забезпечує багату периферію, включаючи 256 байтів SRAM, 768 байтів допоміжної оперативної пам'яті XRAM (рис. 2.6) з адресацією командою робота з зовнішньою пам'яттю, до 18 входів/виходів загального призначення, два 16-розрядних таймера/лічильника 0/1, один 16-розрядний таймер 2 з трьома, модуль захоплення вхідного каналу, один сторожовий таймер (WDT), один таймер самовідтворення (WKT), чотири послідовних інтерфейса, два- UART, один SPI інтерфейс, один I2C, п'ять поліпшених вихідних каналів ШІМ, 12-розрядний АЦП. Периферійні пристрої 18 джерел з можливістю переривання з пріоритетом 4 рівня. 1Т8051 використовує три джерела імпульсів і підтримує перемикання на льоту за допомогою програмного забезпечення, три джерела синхронізації включають в себе зовнішній вхідний сигнал, внутрішній генератор 10 кГц і один внутрішній 16 МГц точний генератор, настрій на заводі до $\pm 1\%$ при кімнатній температурі забезпечує 4-рівнева виявлення відключення, яке можна скинути при включенні напруги джерела.

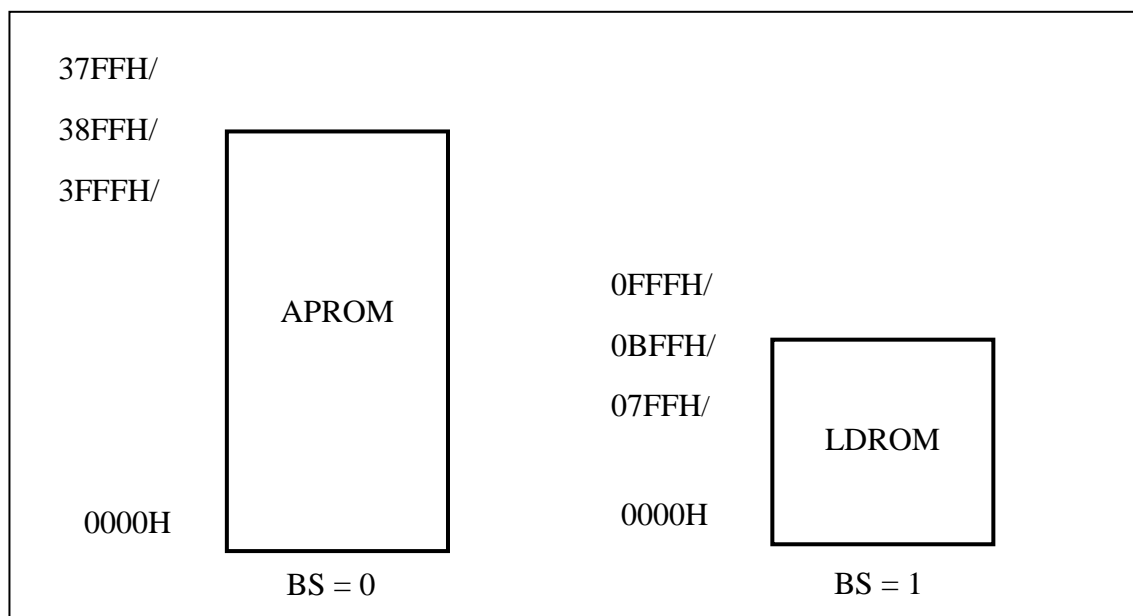


Рисунок 2.8 – Структура пам'яті програм мікроконтролера 1Т8051

Таблиця 2.5 – Налаштування розміру внутрішньої 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 18K Bytes 110 = LDROM is 1K Bytes. APROM is 17K Bytes 101 = LDROM is 2K Bytes. APROM is 16K Bytes 100 = LDROM is 3K Bytes. APROM is 15K Bytes 0xx = LDROM is 4K Bytes. APROM is 14K Bytes

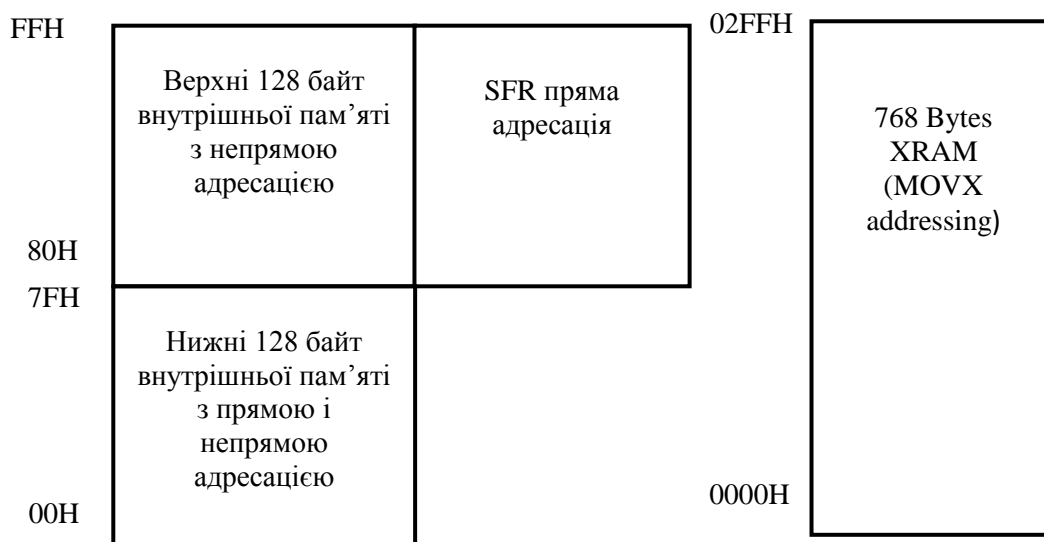


Рисунок 2.9 – Структура пам'яті даних мікроконтролера 1T8051

Таблиця 2.6 –

Незайняті адреси в просторі SFR, позначені «-» зарезервовані для подальшого використання. Доступ до цих областей матиме невизначений ефект і його слід уникати.

Контрольні запитання

1. Який максимальний об'єм зовнішньої пам'яті даних в MCS-51?
2. Скільки програмованих портів містить MCS-51?
3. Який мінімальний об'єм пам'яті програм в MCS-51?
4. Яку кількість розрядів містить лічильник команд MCS-51 ?

5. Скільки запитів на переривання опрацьовується в MCS-51 ?
6. Скільки двійкових розрядів містить універсальний програмований порт в MCS-51?
7. Яку кількість розрядів містить показчик 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 напрям 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 об'єм зовнішньої пам'яті програм в MCS-51?

3. СИСТЕМА КОМАНД МІКРОКОНТРОЛЕРА СЕРІЇ MCS-51

3.1. Загальні положення

Команда мікропроцесора представляється машинним кодом. Для виконання команди мікропроцесор завжди здійснює певну послідовність дій:

- витяг команди – читання з комірки 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 (в залежності 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 – *Assembler*. 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 асемблера в здійснений машинний код (обчислення виразів, розкриття макрокоманд, заміна мнемонік власне машинними кодами і символічних адрес на абсолютні або відносні адреси) проводиться асемблером – програмою-транслятором, яка і дала мові асемблера його назву.

У мові асемблера 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-го до 3-х).

Стосовно до 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 Assembler складається з мнемонічного позначення функції, слідом за яким йдуть операнди, що вказують методи адресації і типи даних. Різні типи даних або режими адресації визначаються зазначеними операндами, а не зміною мнемонічних позначень функції.

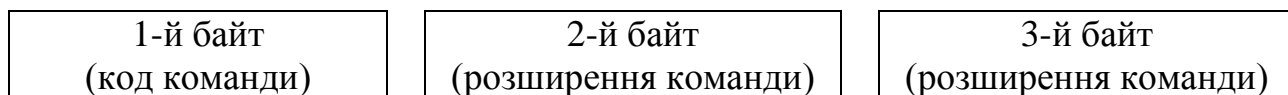
В загальному вигляді команда мікроконтролера серії MCS-51 в асемблерній формі має такий вигляд:

Команда Операнд1, Операнд2, Операнд3

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 випадку команда мікроконтролера серії MCS-51 в машинних кодах структурно має такий вигляд:



Як вже відзначалося, команда в машинних кодах може мати розмірність один, два або три байти. 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 (в залежності 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 наступні 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» – символічне позначення адреси комірки 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 байти;
direct	– пряма адреса комірки внутрішньої пам'яті даних. Значення direct може набувати значення від 00H до FFH. Слід зазначити, якщо значення direct більше ніж 7FH, то адресація здійснюється до регістрів спеціальних функцій;
label	– адреса комірки пам'яті, що визначається міткою;
R16	– один з 16-бітових регістрів. В якості 16-бітового регістра 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 у якості вказівника;
Rn	– один з регістрів загального призначення (R0, R1, ..., R7) в поточному банку регістрів загального призначення. Структура зв'язків програмної моделі MCS51 наведена рис. 3.1, з якої видно що більшість команд мікропроцесора використовують регістр – акумулятор.

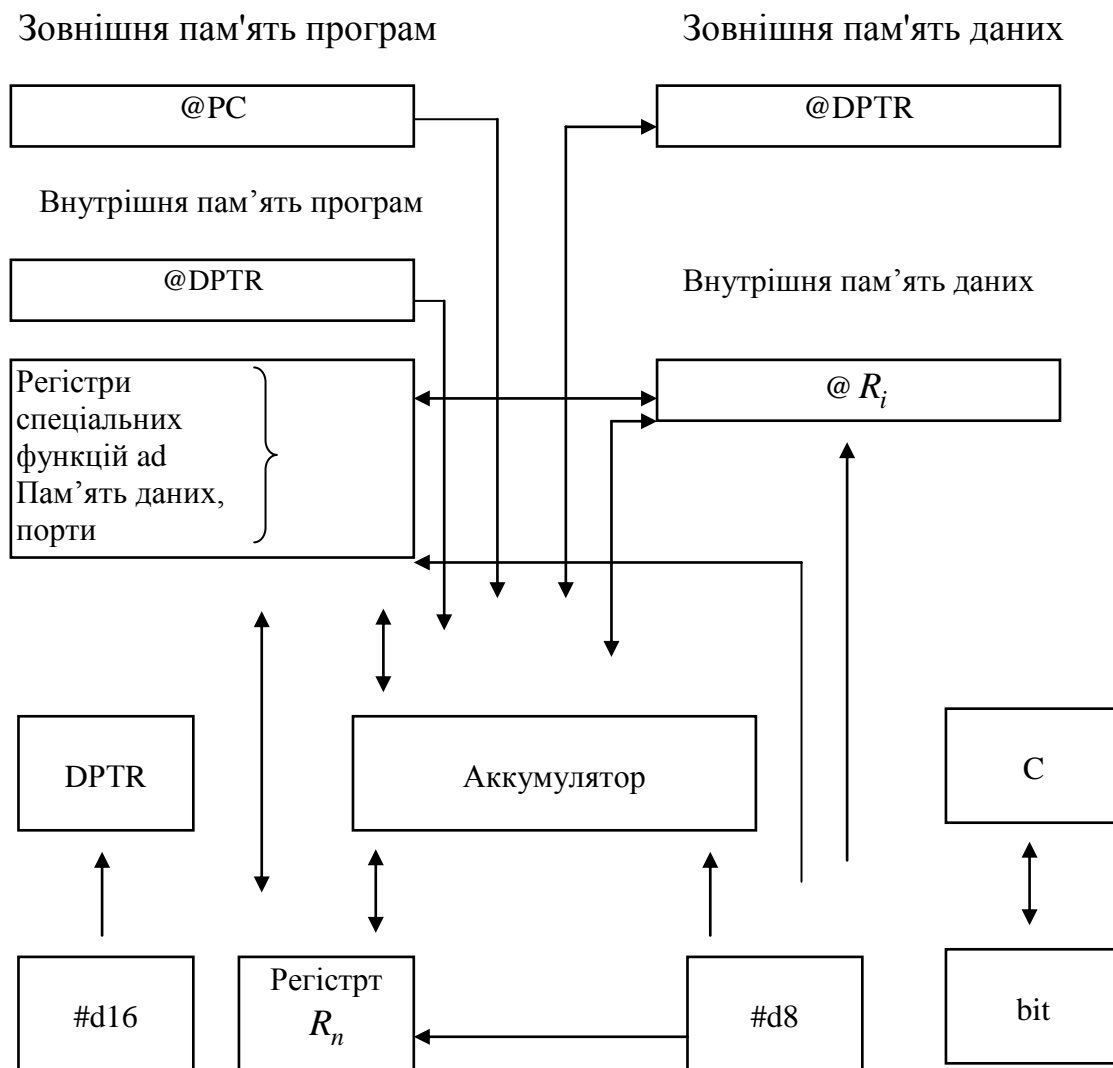


Рисунок 3.1 – Структура зв'язків програмної моделі MCS51

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

Способи адресації операндів-призначення:

- регістрова адресація;
- пряма адресація;

- непряма-регістрова адресація;

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 з використанням прямої адресації і непрямо-регістрової адресації (як до звичайних комірок внутрішньої пам'яті даних).

Регістрова адресація також використовується для звернення до регістрової пари АВ, регістрів А, В, DPTR і до прапора переносу С. 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-ми бітів в бітовій зоні внутрішньої пам'яті даних і до окремих бітів в регістрах спеціальних функцій. Старший біт прямої адреси визначає звернення до бітової зони внутрішньої пам'яті даних або до окремих бітів в регістрах спеціальних функцій.

Непрямо-регістрова адресація

Непрямо-регістрова адресація використовується для звернення до комірок внутрішньої пам'яті даних. В якості вказівника можуть використовуватися регістри R0, R1 поточного банку регістрів загального призначення.

При виконанні команд PUSH і POP (запис в стек і читання із стека) в якості вказівника використовується регістр SP – вказівник стека.

Непрямо-регістрова адресація також використовується для звернення до комірок зовнішньої пам'яті даних. Можливі два способи організації непрямо-регістрової 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 в якості вказівника може використовуватися регістр DPTR, який має розрядність 16 біт. Це дозволяє отримати доступ до будь-якої комірки зовнішньої пам'яті даних в 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 в якості вказівника 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.

Непрямо-регістрова адресація за сумою базового та індексного регістрів

Непрямо-регістрова адресація за сумою базового та індексного регістрів дозволяє здійснювати вибір даних (констант) з таблиць, розміщених в пам'яті програм.

В якості базового регістра може виступати регістр DPTR або лічильник команд PC, а в якості індексного регістра – акумулятор A.

Перш ніж розглядати команди мікроконтролера серії MCS-51 необхідно засвоїти способи адресації. Найбільш наочно можна це зробити на прикладі команд пересилань. В загальному вигляді формат команди пересилання має наступний 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 – команда, перекладена в машинні коди, третє поле – мнемокод команди, четверте поле – операнди. У першому і другому полі значення представлені в шістнадцятковій формі. На початку і наприкінці прикладів додані команди NOP – «порожня операція» для більшої наочності розміщення машинних кодів в 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 3ЕН пересилається в акумулятор А. В результаті виконання команди в акумуляторі буде значення 3ЕН. У машинному коді ця команда представлена двома байтами: перший – код команди (74Н), другий – безпосередньо значення (3ЕН), що пересилається в акумулятор. У пам'яті програм команда знаходиться в комірках з адресами 0001Н і 0002Н.

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

Цей приклад ілюструє: другий операнд вказано у вигляді безпосередньої адресації (тобто задано безпосереднє значення) і знаходиться 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 00NOP
```

У прикладі 2.1 вміст комірки внутрішньої пам'яті даних з адресою 49Н пересилається в акумулятор А. В результаті виконання команди в акумуляторі буде значення, яке міститься в комірці внутрішньої пам'яті даних, адреса якої 49Н. У машинному коді ця команда представлена двома байтами: перший – код команди (Е5Н), другий – адреса комірки внутрішньої пам'яті даних (49Н), з якої дані пересилаються в акумулятор. У пам'яті програм ця команда знаходиться в комірках з адресами 0001Н і 0002Н.

У прикладі 2.2 вміст комірки внутрішньої пам'яті даних з адресою 2ЕН пересилається в акумулятор А. В результаті виконання команди в акумуляторі буде значення, яке міститься в комірці внутрішньої пам'яті даних, адреса якої 2ЕН. У машинному коді ця команда представлена двома байтами: перший – код команди (Е5Н), другий – адреса комірки внутрішньої пам'яті даних (2ЕН), з якої дані пересилаються в акумулятор. У пам'яті програм ця команда знаходиться в комірках з адресами 0003Н і 0004Н.

Цей приклад ілюструє: другий операнд вказано у вигляді прямої адресації (тобто вказаний прямий адрес комірки внутрішньої пам'яті даних) і знаходиться 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 A. В результаті виконання команди в акумуляторі буде значення, що міститься в регістрі R0. У прикладі 3.2 вміст регістра R1 пересилається в акумулятор A. В результаті виконання команди в акумуляторі буде значення, що міститься в регістрі R1. У прикладі 3.3 вміст регістра R7 пересилається в акумулятор A. В результаті виконання команди в акумуляторі буде значення, що міститься в регістрі R7.

У всіх прикладах (приклади 3.1, 3.2, 3.3) в машинному коді дана команда представлена 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 вказано 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 реєстр R0 завантажено значення 33Н, в реєстр R1 – 40Н, в акумулятор – 11Н.

У прикладі 4.1 вміст акумулятора А пересилається в комірку внутрішньої пам'яті даних, адреса якої міститься в реєстрі R0. В результаті виконання команди вміст акумулятора (11Н) буде пересланий в комірку внутрішньої пам'яті даних з адресою 33Н (тому що в реєстрі R0 знаходиться значення 33Н).

У прикладі 4.2 команда виконується таким же чином як і в прикладі 4.1, але в реєстрі R0 знаходиться значення 34Н, тому що після виконання команди INC R0 вміст реєстра R0 збільшився на 1 (було 33Н, стало 34Н). В результаті виконання команди значення з акумулятора (11Н) буде переслане в комірку внутрішньої пам'яті даних з адресою 34Н.

У прикладі 4.3 команда виконується таким же чином як і в прикладі 4.2, але в реєстрі R0 знаходиться значення 35Н, тому що після виконання команди INC R0 вміст реєстра R0 збільшився на 1 (було 34Н, стало 35Н). В результаті виконання команди значення з акумулятора (11Н) буде переслане в комірку внутрішньої пам'яті даних з адресою 35Н.

У прикладі 4.4 вміст акумулятора А пересилається в комірку внутрішньої пам'яті даних, адреса якої міститься в реєстрі R1. В результаті виконання команди вміст акумулятора (11Н) буде пересланий в комірку внутрішньої пам'яті даних з адресою 40Н (тому що в реєстрі R1 знаходиться значення 40Н).

У всіх прикладах (приклади 4.1, 4.2, 4.3, 4.4) в машинному коді дана команда представлена 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 біт є кодом команди, а молодший біт – адресою (номером) реєстра, що використовується в якості вказівника на комірку внутрішньої пам'яті даних.

Цей приклад ілюструє: перший операнд вказано у вигляді непрямо-реєстрової адресації. Даний спосіб називається непрямо-реєстровий: непрямо – тому, що адреса одержувача інформації задана непрямо (адреса комірки внутрішньої пам'яті даних міститься в реєстрі) і реєстровий – тому, що

вказівник задано неявно (номер регістра, що виступає в ролі покажчика, знаходиться в пам'яті програм безпосередньо в кодї 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 комірки пам'яті програм, адреса якої визначається як сума значень, що знаходяться в регістрі DPTR і акумуляторі А, пересилається в акумулятор. В результаті виконання команди вміст комірки пам'яті програм з адресою 01A5H (тому що $01A0H + 05H = 01A5H$) буде переслане в акумулятор.

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

У прикладі 5.2 вміст комірки пам'яті програм, адреса якої визначається як сума значень, що знаходяться в лічильнику команд PC і акумуляторі А, пересилається в акумулятор. В результаті виконання команди вміст з комірки пам'яті програм з адресою 006DH (тому що $0009H + 64H = 006DH$) буде переслане в акумулятор.

Приклади 5.1 і 5.2 демонструють непрямо-регістрову адресацію за сумою базового та індексного регістрів. В якості базового регістра може виступати регістр DPTR або лічильник команд PC, а в якості індексного регістра – акумулятор А.

Непрямо-регістрова адресація за сумою базового та індексного регістрів дозволяє здійснювати вибірку даних (констант) з таблиць, розташованих в 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 5СН пересилається в регістр R0.

Приклад 6.2 – вміст комірки внутрішньої пам'яті даних з адресою 5СН пересилається в регістр R0.

Приклад 6.3 – вміст комірки внутрішньої пам'яті даних з адресою 5СН пересилається в комірку пам'яті даних, адреса якої міститься в регістрі R0.

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

3.3. Команди мікроконтролера

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

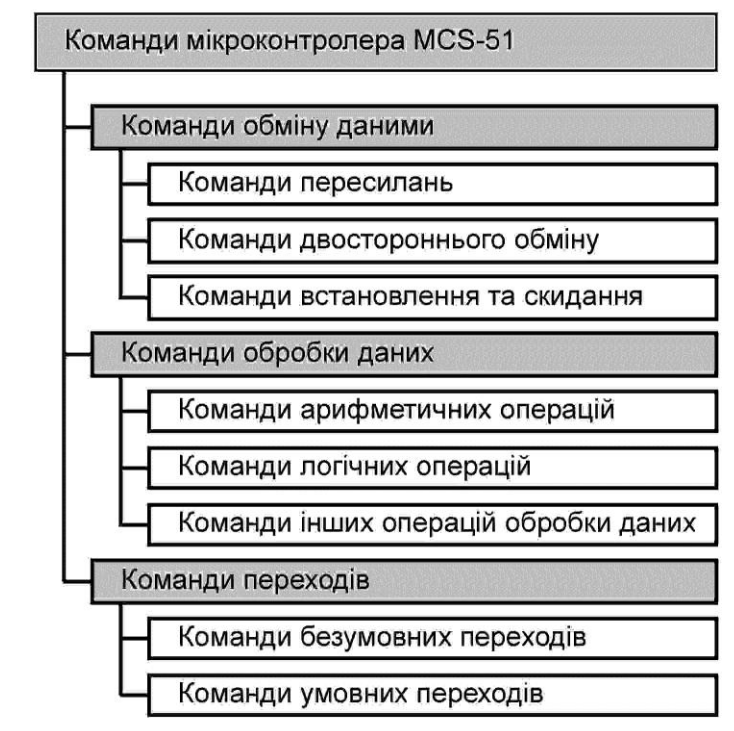


Рисунок 3.2 – Структура команд мікроконтролера серії MCS-51

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».

Команда NOP

Команда «немає операції» – команда, що не містить ніякої функції (операції). Не впливає на вміст регістрів, прапорів, комірок пам'яті даних.

NOP ;1байт, 1мц

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

3.3.1. Команди обміну даними

В літературі команди обміну даними часто називають командами передачі даних, командами пересилань, командами читання та запису, командами загрузки і деякими іншими термінами. В залежності від процесора серед команд обміну даними можна виділити за функціональними ознаками ті чи інші групи команд. В системі команд мікроконтролера серії MCS51 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, а також зовнішніми складовими елементами контролера керування. Архітектура мікроконтролерів серії MCS-51 обумовлює обмін даними як між внутрішніми складовими частками, так і з зовнішніми так само як і з елементами (комірками) пам'яті. Тому фактично весь обмін даними будується на базі команд пересилань з пам'яттю. Група команд пересилань представлена командами:

- MOV – для роботи з внутрішньою пам'яттю даних;
- MOVX – для роботи з 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 наступний 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 данні, а другий – звідкіля. Вміст байта-джерела не змінюється. Ця команда на прапори та інші регістри не впливає. Операнди джерела та призначення допускають п'ятнадцять комбінацій адресації:

MOV A,Rn ;1байт, 1мц

Вміст регістра Rn пересилається в регістр A.

MOV Rn,A ;1байт, 1мц

Вміст регістра A пересилається в регістр Rn.

MOV A,direct ;2байти, 1мц

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

MOV direct,A ;2байти, 1мц

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

MOV A,@Ri ;1байт, 1мц

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

MOV @Ri,A ;1байт, 1мц

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

MOV Rn,direct ;2байти, 2мц

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

MOV direct,Rn ;2байти, 2мц

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

MOV direct,@Ri ;2байти, 2мц

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

MOV @Ri,direct ;2байти, 2мц

Вміст комірки внутрішньої пам'яті даних з адресою, що вказана в якості другого операнда пересилається в комірку внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri.

MOV direct,direct ;3байти, 2мц

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

MOV A,#data ;2байти, 1мц

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

MOV Rn,#data ;2байти, 1мц

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

MOV direct,#data ;3байти, 2мц

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

MOV @Ri,#data ;2байти, 1мц

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

Команда MOV <біт-призн.>,<біт-джер.>

Команда «переслати змінну-біт» пересилає значення розмірністю один біт з біта-джерела в біт-призначення. Тобто перший операнд визначає куди пересилаються данні, а другий – звідкіля. Вміст біта-джерела не змінюється. Одним із операндів повинен бути прапор 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 ;2байти, 2мц

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

MOV bit,C ;2байти, 2мц

Вміст прапора 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 командою, яка одночасно завантажує дані розмірністю 16 біт.

MOV DPTR,#data16 ;3байти, 2мц

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

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

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

MOVX A,@DPTR ;1байт, 2мц

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

MOVX A,@Ri ;1байт, 2мц

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

MOVX @DPTR,A ;1байт, 2мц

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

MOVX @Ri,A ;1байт, 2мц

Вміст регістра A пересилається в комірку 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 комірки пам'яті програм. Адреса комірки пам'яті програм визначається (обчислюється) як сума 8-бітового вмісту акумулятора без знаку і вмісту 16-бітового регістра, який вказано в другому операнді команди. В якості 16-бітового регістра 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 складається з вмістом акумулятора. Додавання виконується так, що перенесення з молодших восьми біт може поширюватися на старші біти.

MOVC A,@DPTR+A ;1байт, 2мц

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

MOVC A,@PC+A ;1байт, 2мц

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

Команда PUSH <байт-джер. >.

Команда «запис в 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 комірку стека

(внутрішньої пам'яті даних), адреса якої міститься в регістрі вказівнику стека SP. Вміст байта-джерела залишається незмінним. Застосовується ця команда для тимчасового збереження вмісту регістрів спеціальних функцій при вході в підпрограму, а також для передачі параметрів підпрограмам через стек.

PUSH direct ;2байти, 2мц

Вміст регістра вказівника стека SP збільшується на 1, після чого вміст комірки пам'яті даних, адреса якої вказана в якості операнда у вигляді прямої адреси, пересилається в комірку стека (внутрішньої пам'яті даних), адреса якої міститься в регістрі вказівнику стека SP.

Примітка. Слід згадати, що пряма адресація до комірок внутрішньої пам'яті даних з 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 комірки і пересилання в байт-призначення. Команда виконується в два етапи: перший – вміст комірки стека (внутрішньої пам'яті даних), адреса якої міститься в регістрі вказівнику стека SP пересилається в байт-призначення, другий – вміст вказівника стека SP зменшується на 1. Вміст комірки 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 в деяких випадках для передачі параметрів з підпрограм через стек.

POP direct ;2байти, 2мц

Вміст комірки стека (внутрішньої пам'яті даних), адреса якої міститься в регістрі вказівнику стека SP пересилається в комірку пам'яті даних, адреса якої вказана в якості операнда у вигляді прямої адреси. Після чого вміст вказівника стека SP зменшується на 1.

Примітка. Слід згадати, що пряма адресація до комірок внутрішньої пам'яті даних з адресою від 80H до FFH здійснює звернення до регістрів спеціальних функцій.

3.3.3. Команди двостороннього обміну

Команди двостороннього обміну в мікроконтролерах серії MCS51 дозволяють здійснювати одночасний обмін вмісту регістра A з вмістом

внутрішнього ресурсу контролера (регістр загального призначення, регістр спеціальних функцій, комірка внутрішньої пам'яті даних). Розмірність даних обміну (в залежності від команди) становить один байт або одна тетрада. Група команд двостороннього обміну представлена командами:

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

Команди двостороннього обміну в загальному форматі мають наступний вигляд:

XCH A,<байт-джер.>

XCHD A,<байт-джер.>

Команда XCH A,<байт-джер.>

Команда «обмін вмісту акумулятора зі змінною-байтом» здійснює обмін вмісту акумулятора з вмістом байта-джерела. Операнди джерела допускають три комбінації адресації:

XCH A,Rn ;1байт, 1мц

Вміст регістра A обмінюється з вмістом регістра Rn, що вказаний в другому операнді. Тобто після виконання команди регістр A містить вміст регістра Rn до виконання команди, а регістр Rn містить вміст регістра A до виконання команди

XCH A,direct ;2байти, 1мц

Вміст регістра A обмінюється з вмістом комірки пам'яті даних, пряма адреса якої вказана в другому операнді. Тобто після виконання команди регістр A містить вміст комірки пам'яті даних до виконання команди, а комірка пам'яті даних містить вміст регістра A до виконання команди

XCH A,@Ri ;1байт, 1мц

Вміст регістра A обмінюється з вмістом комірки пам'яті даних, адреса якої міститься в регістрі Ri. Тобто після виконання команди регістр A містить вміст комірки пам'яті даних до виконання команди, а комірка пам'яті даних містить вміст регістра A 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) комірки

внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri. На старші біти (біти 7 – 4) ця команда не впливає

XCHD A,@Ri ;1байт, 1мц

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

3.3.4. Команди встановлення та скидання

Команди встановлення та скидання – це команди, 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 наступний вигляд:

SETB <біт-призн.>

CLR <біт-призн.>

CLR <байт-призн.>

Команда SETB C

Команда «встановити біт прапора перенесу» встановлює біт прапора перенесу C в 1.

SETB C ;1байт, 1мц

Вміст біта прапора перенесу C приймає значення 1.

Команда SETB <біт-призн.>

Команда «встановити біт» встановлює біт-призначення в 1.

SETB bit ;2байти, 1мц

Вміст біта, пряма адреса якого визначається операндом, приймає значення 1.

Команда CLR C

Команда «скинути біт прапора переносу» скидає біт прапора переносу C в 0.

CLR C ;1байт, 1мц

Вміст біта прапора переносу C приймає значення 0.

Команда CLR <біт-призн.>

Команда «скинути біт» скидає біт-призначення в 0.

CLR bit ;2байти, 1мц

Вміст біта, пряма адреса якого визначається операндом, приймає значення 0.

Команда CLR A

Команда «очистити акумулятор» обнуляє вміст акумулятора.

CLR A ;1байт, 1мц

Вміст акумулятора приймає значення 0.

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 наступний 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 виникненні перенесень з розрядів 7-го і 3-го, встановлюються відповідно прапори переносу

C і додаткового переносу AC, в іншому випадку ці прапори скидаються. При додаванні цілих чисел без знака прапор переносу C вказує на появу переповнення. Прапор 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 прапор OV скидається. При додаванні цілих чисел зі знаком прапор OV вказує на негативну величину, отриману при підсумовуванні двох позитивних операндів або на позитивну суму для двох негативних операндів. Операнди джерела допускають чотири комбінації адресації:

ADD A,Rn ;1байт, 1мц

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

ADD A,direct ;2байти, 1мц

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

ADD A,@Ri ;1байт, 1мц

Вміст комірки внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, додається до вмісту регістра A, сума пересилається в регістр A. Прапори C, AC, OV приймають значення відповідно до результату складання.

ADD A,#data ;2байти, 1мц

Значення, що вказане в другому операнді додається до вмісту регістра A, сума пересилається в регістр A. Прапори C, AC, OV приймають значення відповідно до результату складання.

Команда ADDC A,<байт-джер.>

Команда «складання з переносом» складає одночасно вміст акумулятора A, значення прапора переносу C 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 прапори переносу C і додаткового переносу AC, в іншому випадку ці прапори скидаються. При додаванні цілих чисел без знака прапор переносу C вказує на появу переповнення. Прапор 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 прапор OV скидається. При додаванні цілих чисел зі знаком прапор OV вказує на негативну величину, отриману при підсумовуванні двох позитивних операндів або на позитивну суму для двох негативних операндів.

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

ADDC A,Rn ;1байт, 1мц

Одночасно складається вміст регістра Rn, регістра A та прапора переносу C, сума пересилається в регістр A. Прапори C, AC, OV приймають значення відповідно до результату складання.

ADDC A,direct ;2байти, 1мц

Одночасно складається вміст комірки внутрішньої пам'яті даних, адреса якої вказана в другому операнді, регістра A та прапора переносу C, сума пересилається в регістр A. Прапори C, AC, OV приймають значення відповідно до результату складання.

ADDC A,@Ri ;1байт, 1мц

Одночасно складається вміст комірки внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, регістра A та прапора переносу C, сума пересилається в регістр A. Прапори C, AC, OV приймають значення відповідно до результату складання.

ADDC A,#data ;2байти, 1мц

Одночасно складається значення, що вказане в другому операнді, вміст регістра A та прапора переносу C, сума пересилається в регістр A. Прапори C, AC, OV приймають значення відповідно до результату складання.

Команда SUBB A,<байт-джер.>

Команда «віднімання з позичкою» віднімає вміст байта-джерела разом з прапором переносу C від вмісту акумулятора A, результат розташовується в регістрі A. В залежності від результату віднімання модифікуються прапор переносу C, прапор додаткового переносу AC та прапор переповнення OV.

Прапор переносу (позичання) C встановлюється, якщо при відніманні для біта 7 необхідна позика, в іншому випадку – скидається. Прапор додаткового переносу AC встановлюється, якщо позика необхідна для біта 3 і скидається в іншому випадку. Прапор 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 прапор *OV* вказує на негативне число, яке утворюється при відніманні негативної величини від позитивної, або позитивне значення, яке утворюється при відніманні позитивного числа від негативного. Операнд джерела допускає чотири режими адресації:

SUBB A,Rn ;1байт, 1мц

Вміст регістра *Rn* разом з прапором переносу *C* віднімається від вмісту регістра *A*, різниця пересилається в регістр *A*. Прапори *C*, *AC*, *OV* приймають значення відповідно до результату віднімання.

SUBB A,direct ;2байти, 1мц

Вміст комірки внутрішньої пам'яті даних, адреса якої вказана в другому операнді, разом з прапором переносу *C* віднімається від вмісту регістра *A*, різниця пересилається в регістр *A*. Прапори *C*, *AC*, *OV* приймають значення відповідно до результату віднімання.

SUBB A,@Ri ;1байт, 1мц

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

SUBB A,#data ;2байти, 1мц

Значення, що вказане в другому операнді, разом з прапором переносу *C* віднімається від вмісту регістра *A*, різниця пересилається в регістр *A*. Прапори *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 в наступним чином: старший байт в регістрі *B*, а молодший байт в регістрі *A*. Якщо добуток більше, ніж 255, то встановлюється прапор переповнення *OV*, в іншому випадку він скидається. Прапор переносу *C* завжди скидається. Ця команда має лише один різновид і слід звернути увагу на її синтаксис – написання регістрів «*AB*» здійснюється без коми та пробілу (як один цільний операнд).

MUL AB ;1байт, 4мц

Здійснює множення вмісту регістрів А та В. Добуток розподіляється в регістрах А (молодший байт) та В (старший байт). Прапор OV приймає значення відповідно до результату множення, а прапор C скидається.

Команда DIV AB

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

DIV AB ;1байт, 4мц

Здійснює ділення вмісту регістра А на вміст регістра В. Результат ділення: в регістрі А – ціла частина частки, а в регістрі В – залишок від ділення. Прапор AC приймає значення відповідно до результату ділення, а прапори C та OV скидаються.

Примітка. Ділення на нуль неможливе, тому якщо регістр В містить нуль, то після команди DIV вміст регістрів А і В буде невизначеним, також це призведе до скидання прапора переносу C та встановлення прапора переповнення OV.

Команда DEC <байт-призн.>

Команда «декремент» зменшує на 1 вміст байта-призначення. У разі, коли вміст має початкове значення, що дорівнює 00H, виконання декременту призведе до значення FFH. Операнди байта-призначення допускають чотири комбінації адресації:

DEC A ;1байт, 1мц

Зменшується на 1 вміст регістра А.

DEC Rn ;1байт, 1мц

Зменшується на 1 вміст регістра загального призначення Rn, який вказаний в якості операнда.

DEC direct ;2байти, 1мц

Зменшується на 1 вміст комірки внутрішньої пам'яті даних, пряма адреса якої вказана в якості операнда.

DEC @Ri ;1байт, 1мц

Зменшується на 1 вміст комірки внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, що вказаний в якості операнда.

Команда INC <байт-призн.>

Команда «інкремент» збільшує на 1 вміст байта-призначення. У разі, коли вміст має початкове значення, що дорівнює FFH, виконання інкременту призведе до значення 00H. Операнди байта-призначення допускають чотири комбінації адресації:

INC A ;1байта, 1мц

Збільшується на 1 вміст регістра A.

INC Rn ;1байт, 1мц

Збільшується на 1 вміст регістра загального призначення Rn, який вказаний в якості операнда.

INC direct ;2байти, 1мц

Збільшується на 1 вміст комірки внутрішньої пам'яті даних, пряма адреса якої вказана в якості операнда.

INC @Ri ;1байт, 1мц

Збільшується на 1 вміст комірки внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, що вказаний в якості операнда.

Команда INC DPTR

Команда «інкремент вказівника DPTR» збільшує на 1 вміст 16-розрядного вказівника DPTR. У разі коли вміст має початкове значення, що дорівнює FFFFH, виконання інкременту призведе до значення 0000H. Команда має одну комбінацію адресації:

INC DPTR ;1байт, 2мц

Збільшується на 1 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 наступний 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-джерело і поміщає результат в байт-призначення. Ця операція не впливає на стан прапорів. Операнди джерела та призначення допускають шість комбінацій адресації:

ANL A, Rn ; 1байт, 1мц

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

ANL A, direct ; 2байти, 1мц

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

ANL A, @Ri ; 1байт, 1мц

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

ANL A, #data ; 2байти, 1мц

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

ANL direct,A ;2байти, 1мц

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

ANL direct,#data ;3байти, 2мц

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

Примітка. Якщо команда ANL застосовується для зміни вмісту порту, то значення, яке використовується в якості даних порту, буде зчитуватися з «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 прапора переносу С та вмістом біта-джерело і поміщає результат в прапор переносу С. Існує один різновид цієї команди:

ANL C,bit ;2байти, 2мц

Виконується функція «логічне І» над вмістом прапора переносу С та вмістом біта, адреса якого визначається другим операндом. Результат пересилається в прапор переносу С.

Команда ANL C,/<біт-джер.>

Команда «логічне І для змінних-бітів» виконує операцію логічного «І» над вмістом прапора переносу С та інверсним значенням вмісту біта-джерело і поміщає результат в прапор переносу С. Існує один різновид цієї команди:

ANL /C,bit ;2байти, 2мц

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

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

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

ORL A,Rn ;1байт, 1мц

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

ORL A,direct ;2байти, 1мц

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

ORL A,@Ri ;1байт, 1мц

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

ORL A,#data ;2байти, 1мц

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

ORL direct,A ;2байти, 1мц

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

ORL direct,#data ;3байти, 2мц

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

Примітка. Якщо команда ORL застосовується для зміни вмісту порту, то значення, яке використовується в якості даних порту, буде зчитуватися з «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 прапора переносу C та вмістом біта-джерело і поміщає результат в прапор переносу C. Існує один різновид цієї команди:

ORL C,bit ;2байти, 2мц

Виконується функція «логічне АБО» над вмістом прапора переносу C та вмістом біта, адреса якого визначається другим операндом. Результат пересилається в прапор переносу C.

Команда ORL C,/<біт-джер.>

Команда «логічне АБО для змінних-бітів» виконує операцію логічного «АБО» над вмістом прапора переносу C та інверсним значенням вмісту біта-джерело і поміщає результат в прапор переносу C. Існує один різновид цієї команди:

ORL /C,bit ;2байти, 2мц

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

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

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

XRL A,Rn ;1байт, 1мц

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

XRL A,direct ;2байти, 1мц

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

XRL A,Ri ;1байт, 1мц

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

XRL A,#data ;2байти, 1мц

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

XRL direct,A ;2байти, 1мц

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

XRL direct,#data ;3байти, 2мц

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

Примітка. Якщо команда XRL застосовується для зміни вмісту порту, то значення, яке використовується в якості даних порту, буде зчитуватися з «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 НЕ» над вмістом регістра А. Регістр А одночасно є байтом-джерелом та байтом-призначення.

CPL A ;1байт, 1мц

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

Команда CPL C

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

CPL C ;1байт, 1мц

Виконується функція «логічне НЕ» над вмістом прапора переносу C, результат розташовується в прапорі переносу C.

Команда CPL <біт-джер.>

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

CPL bit ;2байти, 1мц

Виконується функція «логічне НЕ» 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 з «засувки» порту, а не з виводів мікроконтролера.

3.3.5.3. Команди інших операцій обробки даних

В окрему групу команд обробки інформації можна виділити команди, що здійснюють операції зсуву та операції обміну. В мікроконтролерах серії MCS51 здійснення цих операцій можливо тільки над вмістом регістра A. Склад вище означених 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 0-й біт (розряд) акумулятора.

RL A ;1байт, 1мц

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 прапор переносу», здійснює зсув вмісту восьми біт акумулятора на один розряд вліво, при цьому вміст 7-го біта (розряду) акумулятора зсувається в прапор переносу C, а вміст прапора переносу C зсувається в 0-й біт (розряд) акумулятора.

RLC A ;1байт, 1мц

Здійснюється зсув вмісту акумулятора на один розряд вліво, при цьому вміст 7-го розряду акумулятора зсувається в прапор переносу C, а вміст прапора переносу C 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 0-го біта (розряду) акумулятора зсувається в 7-й біт (розряд) акумулятора.

RR A ;1байт, 1мц

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 прапор перенесення», здійснює зсув вмісту восьми біт акумулятора на один розряд вправо, при цьому вміст 0-го біта (розряду) акумулятора зсувається в прапор переносу C, а вміст прапора переносу C зсувається в 7-й біт (розряд) акумулятора.

RRC A ;1байт, 1мц

Здійснюється зсув вмісту акумулятора на один розряд вправо, при цьому вміст 0-го розряду акумулятора зсувається в прапор переносу C, а вміст прапора 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 команда чотирьох циклічних зсувів (якого можна досягти послідовним виконанням чотирьох команд RL A або чотирьох команд RR A).

SWAP A ;1байт, 1мц

Здійснюється взаємний обмін вмісту старшої та молодшої тетради акумулятора. Тобто обмінюється вміст 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 команди, що дозволяють здійснювати переходи в програмі. Розробка програмного забезпечення неможлива без застосування команд умовного і безумовного переходу.

Команди безумовного переходу дозволяють здійснити перехід в 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, в разі якщо умова не виконується, то здійснюється виконання наступної команди. Якщо точку переходу, в команді умовного переходу, помістити вище за алгоритмом, то за допомогою даної команди можна організувати цикл в 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 представляє собою регістр, що містить адресу комірки пам'яті програм, в якій знаходиться поточна для виконання команда.

Таким чином, в результаті виконання переходу значення лічильника команд змінюється таким чином, що воно відповідає адресі комірки пам'яті програм в якій знаходиться команда, на яку здійснюється перехід. У початковому тексті програми для позначення точки переходу використовують мітку. Мітка це довільне символічне ім'я, яке в процесі компіляції програми перетворюється в значення яке модифікує вміст лічильника команд.

3.3.6.1. Команди безумовних переходів

В системі команд мікроконтролерів серії MCS-51 є декілька різновидів безумовного переходу, які можна розділити 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 таким набором:

AJMP <мітка>

LJMP <мітка>

SJMP <мітка>

Команди переходу, що обчислюється представлені командою:

JMP @A+DPTR

Команди переходу на підпрограму представлені командами:

ACALL <мітка>

LCALL <мітка>

Команди переходу (повернення) з підпрограми представлені командами:

RET

RETI

Команда LJMP <мітка>

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

LJMP label ;3байти, 2мц

Значення адреси, що визначається міткою, яка вказана в якості операнда, завантажуються (пересилається) в лічильник команд мікроконтролера РС

Команда SJMP <мітка>

Команда «короткий перехід» дозволяє здійснити перехід в 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 в якості операнда. Причому в машинному коді операнд буде представлено у вигляді 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 ;2байти, 2мц

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

Примітка. Використання команди «короткий перехід» (в 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 в будь-яку точку програми в межах однієї сторінки об'ємом 2 Кбайт пам'яті програм. Точка переходу визначається міткою, що вказана в команді в якості операнда. Причому в машинному коді операнд буде представлено у вигляді 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 одинадцятью молодшими розрядами і утворюють значення адреси переходу, яке пересилається до лічильника команд.

AJMP label ;2байти, 2мц

Одинадцять розрядів, що визначаються міткою, яка вказана в якості операнда, модифікують одинадцять молодших розрядів лічильника команд РС (після завершення зчитування команди).

Примітка. Практично ця команда не має використання, основна її мета – забезпечення сумісності і полегшення переходу від мікроконтролерів серії MCS48 до мікроконтролерів серії MCS51.

Команди переходу, що обчислюється:

Команда JMP @A+DPTR

Команда «непрямий перехід», дозволяє здійснити перехід в будь-яку точку програми в межах всього адресного простору пам'яті програм мікроконтролера (64Кбайт). 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 ;1байт, 2мц

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 в якості операнда. Причому в машинному коді операнд буде представлено у вигляді абсолютного значення адреси переходу, тобто по суті мітка являє собою значення адреси комірки пам'яті програм, в якій знаходиться перша команда підпрограми.

LCALL label ;3байти, 2мц

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

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

Завантаження в лічильник команд мікроконтролера адреси комірки 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 в якості операнда. Причому в машинному коді операнд буде представлено у вигляді 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 вище п'ять старших розрядів суміщаються з одинадцятю молодшими розрядами і утворюють значення адреси переходу, яке пересилається до лічильника команд.

ACALL label ;2байти, 2мц

Механізм виконання даної команди описується таким же 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 має використання, основна її мета – забезпечення сумісності і полегшення переходу від мікроконтролерів серії MCS48 до мікроконтролерів серії MCS51.

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 комірки пам'яті програм, яка є наступною за коміркою 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 ;1байт, 2мц

Механізм виконання цієї команди відбувається наступним чином. Вміст із стека (з комірки пам'яті даних, адреса якої знаходиться в регістрі SP) пересилається в молодший байт лічильника команд і вміст вказівника стека SP зменшується на 1. Потім вміст із стека (з комірки пам'яті даних, адреса якої знаходиться в регістрі SP) пересилається в старший байт лічильника команд і вміст вказівника стека SP зменшується на 1.

Команда RETI

Команда «повернення з підпрограми переривання» здійснює повернення з підпрограми переривання в точку, що розташована безпосередньо за командою на якій був здійснено 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, що дорівнює пріоритету тільки що обробленого переривання.

RETI ;1байт, 2мц

Механізм виконання цієї команди відбувається наступним чином. Вміст із стека (з комірки пам'яті даних, адреса якої знаходиться в регістрі SP) пересилається в молодший байт лічильника команд і вміст вказівника стека SP зменшується на 1. Потім вміст із стека (з комірки пам'яті даних, адреса якої знаходиться в регістрі SP) пересилається в старший байт лічильника команд і вміст вказівника стека SP зменшується на 1. 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) в 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:

JV <біт-джер.>,<мітка>

JNV <біт-джер.>,<мітка>

JVC <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 в якості умови 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-джерело встановлено (дорівнює 1), в іншому випадку – виконується наступна команда.

JB bit,label ;3байти, 2мц

Якщо вміст біта, що зазначений прямою адресою в першому операнді дорівнює 1 – здійснюється перехід за міткою, що визначається другим операндом, в іншому випадку – виконується наступна команда.

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

Команда JNB <біт-джер.>,<мітка>

Команда «перехід, якщо біт не встановлений» здійснює перехід за міткою, якщо біт-джерело не встановлено (дорівнює 0), в іншому випадку – виконується наступна команда.

JNB bit,label ;3байти, 2мц

Якщо вміст біта, що зазначений прямою адресою в першому операнді дорівнює 0 – здійснюється перехід за міткою, що визначається другим операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною командою 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-джерело встановлено (дорівнює 1), в іншому випадку – виконується наступна команда. В будь-якому випадку біт-джерело скидається (тобто приймає значення 0).

JBC bit,label ;3байти, 2мц

Якщо вміст біта, що зазначений прямою адресою в першому операнді дорівнює 1 – здійснюється перехід за міткою, що визначається другим операндом, в іншому випадку – виконується наступна команда. В будь-якому випадку вказаний біт скидається – приймає значення 0.

Команда не має взаємопротилежної команди.

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

Команда JC <мітка>

Команда «перехід, якщо біт переносу встановлений» здійснює перехід за міткою, якщо прапор переносу C встановлено (дорівнює 1), в іншому випадку – виконується наступна команда.

JC label ;2байти, 2мц

Якщо вміст прапора переносу C дорівнює 1 – здійснюється перехід за міткою, що визначається операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною командою є команда JNC <мітка>.

Команда JNC <мітка>

Команда «перехід, якщо біт переносу не встановлений» здійснює перехід за міткою, якщо прапор переносу C не встановлено (дорівнює 0), в іншому випадку – виконується наступна команда.

JNC label ;2байти, 2мц

Якщо вміст прапора переносу C дорівнює 0 – здійснюється перехід за міткою, що визначається операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною командою 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 наступна команда.

JZ label ;2байти, 2мц

Якщо вміст регістра A дорівнює 0 – здійснюється перехід за міткою, що визначається операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною командою 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 наступна команда.

JNZ label ;2байти, 2мц

Якщо вміст регістра A не дорівнює 0 – здійснюється перехід за міткою, що визначається операндом, в іншому випадку – виконується наступна команда.

Взаємопротилежною командою 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 не рівні, здійснює перехід за міткою, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним вміст «байта-призначення» та «байта-джерела» і супроводжується модифікацією прапора переносу C.

Прапор переносу C встановлюється в 1, якщо значення цілого без знака «байта-призначення» менше, ніж значення цілого без знака «байта-джерело», в іншому випадку прапор переносу C скидається в 0.

Операнди джерела та призначення допускають чотири комбінації адресації:

CJNE A,direct,label ;3байти, 2мц

Порівнюється вміст регістра A та вміст комірки внутрішньої пам'яті даних, пряма адреса якої визначається другим операндом. В разі якщо їх значення не рівні – здійснює перехід за міткою, що визначається третім операндом, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним значення, що порівнюються і супроводжується модифікацією прапора переносу C відповідно до результату порівняння.

CJNE A,#data,label ;3байти, 2мц

Порівнюється вміст регістра A та безпосереднє значення, що визначається другим операндом. В разі якщо їх значення не рівні – здійснює перехід за міткою, що визначається третім операндом, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним значення, що

порівнюються і супроводжується модифікацією прапора переносу С відповідно до результату порівняння.

`CJNE Rn,#data,label` ;3байти, 2мц

Порівнюється вміст регістра загального призначення Rn, який вказаний в якості першого операнда, та безпосереднє значення, що визначається другим операндом. В разі якщо їх значення не рівні – здійснює перехід за міткою, що визначається третім операндом, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним значення, що порівнюються і супроводжується модифікацією прапора переносу С відповідно до результату порівняння.

`CJNE @Ri,#data,label` ;3байти, 2мц

Порівнюється вміст комірки внутрішньої пам'яті даних, адреса якої міститься в регістрі Ri, який вказаний в якості першого операнда, та безпосереднє значення, що визначається другим операндом. В разі якщо їх значення не рівні – здійснює перехід за міткою, що визначається третім операндом, в іншому випадку – виконується наступна команда. Виконання команди залишає незмінним значення, що порівнюються і супроводжується модифікацією прапора переносу С відповідно до результату порівняння.

Команда не має взаємопротилежної команди.

За допомогою команд 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», а також 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 1 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` ;2байти, 2мц

Зменшується на 1 вміст регістра загального призначення Rn, який вказаний в якості першого операнда. Якщо вміст цього регістра не дорівнює 0, то здійснюється перехід за міткою, що вказана в якості другого операнда, в іншому випадку – виконується наступна команда.

DJNZ direct,label ;3байти, 2мц

Зменшується на 1 вміст комірки внутрішньої пам'яті даних, пряма адреса якої визначається першим операндом. Якщо вміст цієї комірки не дорівнює 0, то здійснюється перехід за міткою, що вказана в якості другого операнда, 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.

Мнемонічні позначення команд мікроконтролерів серії MCS-51 і 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 прапора переносу C
CLR	<біт-приз.>	Скидання біта
CPL	A	Інверсія акумулятора
CPL	C	Інверсія прапора 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 прапор переносу C встановлений
JMP	@A+DPTR	Непрямий перехід
JNB	<біт-приз.>,<мітка>	Перехід, якщо біт не встановлений
JNC	<мітка>	Перехід, якщо прапор переносу

Команда	Операнди	Функція команди
		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 в стек
RET		Повернення з підпрограми
RETI		Повернення з підпрограми переривання
RL	A	Зсув вмісту акумулятора вліво
RLC	A	Зсув вмісту акумулятора вліво через прапор переносу
RR	A	Зсув вмісту акумулятора вправо
RRC	A	Зсув вмісту акумулятора вправо через прапор 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 запитання

1. Скільки банків даних містить мікроконтролер MCS-51?
2. Де розташована стекова пам'ять в MCS-51?
3. Яку кількість розрядів містить покажчик стекової пам'яті в MCS-51?
4. Яку кількість регістрів містить банк 0 в MCS-51?

5. Яка початкова адреса стекової пам'яті в MCS-51?
6. Який мінімальний об'єм 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 банку даних до акумулятора?
10. Яка команда пересилає константу до регістру, що знаходиться в банку даних?
11. Яка команда пересилає інформацію з комірки у комірку, що знаходяться в пам'яті даних?
12. Яка команда пересилає інформацію з 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 вводу/виводу інформації рис. 4.1. Вони призначені для двонаправленого обміну інформацією із зовнішніми пристроями, такими як кнопки, датчики, виконавчі пристрої, індикатори і т.п., а також із зовнішньою пам'яттю програм і даних. Крім цього, порти використовуються під час програмування внутрішньої пам'яті програм контролера.

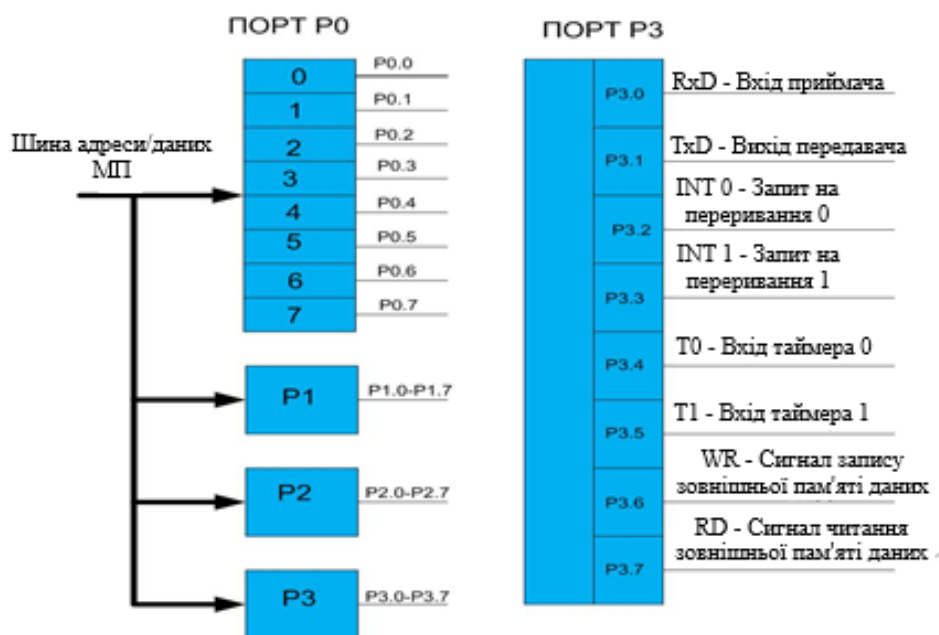


Рисунок 4.1 – Структурна схема портів MCS-51

Функціональна схема ліній портів показана на рис. 4.2 і 4.3. Базовою є модель порту 1. Вона містить керований тригер-засувку, виконаний у вигляді D-Тригера, вхідний буфер В1, буфер В2 читання тригера-засувки й вихідний драйвер, виконаний на польовому транзисторі з ізольованим затвором і навантажений на резистор. Схему порту 0 доповнено мультиплексором MS (на рисунку показаний 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 2 доповнено мультиплексором MS, який дозволяє підключати до виходів порту старший байт адресної шини процесора при роботі із зовнішньою пам'яттю. Схему порту 3 доповнено буферами підключення периферійних пристроїв. Адресація до портів проводиться так само, як до комірок пам'яті. Фізичні адреси портів введення/виведення:

P0-80H, при бітовій адресації 80H-87H;
P1-90H, при бітовій адресації 90H-97H;
P 2-A0H, при бітовій адресації A0 H-A7H;
P 3-B0H, при бітовій адресації B0 H-B7H.

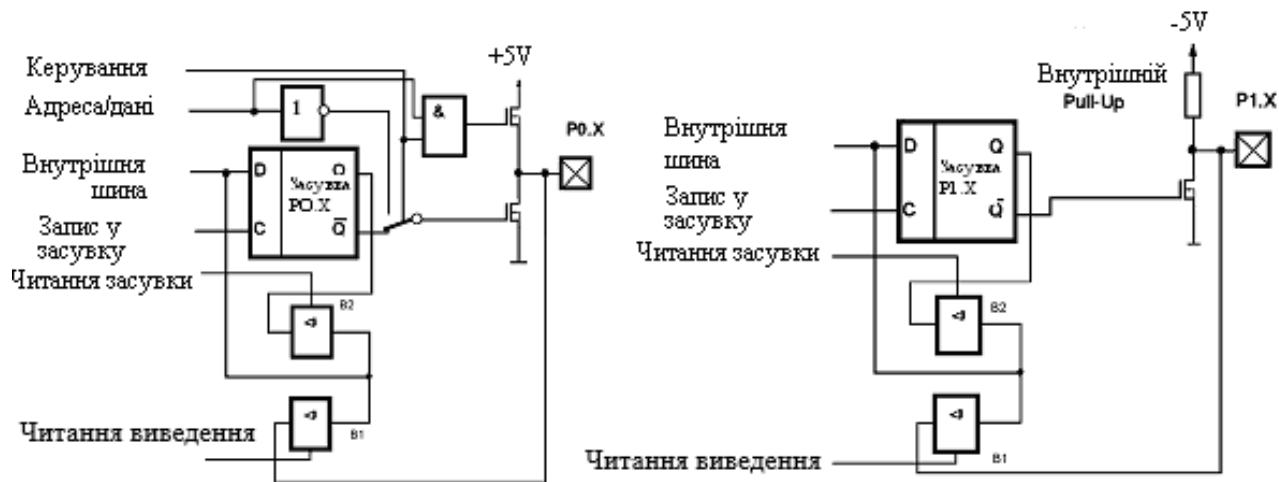


Рисунок 4.2 – Функціональні схеми каналів портів 0 і 1



Рисунок 4.3 – Функціональні схеми каналів портів 2 і 3

Порт 0 є двонаправленим, а порти 1, 2, 3 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 1. При цьому вихідний МОП -транзистор закривається і на виведенні мікросхеми встановлюється високий рівень напруги (тобто 1). При включенні 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 P0 проводиться видача молодшого байта адреси й приймання або видача даних, а на P2 видається старший байт адреси. У тригері-засувці порту 0 при обміні даними записуються всі одиниці.

У режимі програмування порт 0 використовується в якості шини даних, через порт 1 подається молодший байт адреси, через порт 2 – старші розряди адреси й керуючі сигнали. Розряди порту 3, крім функцій вводу/виводу, мають альтернативні функції, наведені в табл. 3.1.

Навантажувальна здатність порту P0 – два TTL входу, портів P1, P2, P3 – один TTL вхід. Для приєднання потужнішого навантаження необхідно використовувати або буфери (рис. 1.32, розділ 1), або транзисторні підсилювачі (рис. 1.33, розділ 1).

Операцію приймання сигналів від зовнішнього пристрою контролером прийнято називати введенням. Для настроювання порту або окремих бітів порту на ввід у тригери-засувки відповідних розрядів необхідно занести 1.

Прочитати стан портів можна командою

MOV A, P0 ; занесення в акумулятор значення порту 0.

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 квазі-двонаправлена структура. Кожен порт має свій регістр управління портами (реєстр P_x). Запис і читання регістра управління портом здійснюються по різному.

Запис в регістр управління портом встановлює логічне значення регістра 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: квазі-двонаправлений (структура порту MCS 8051), двотактний, тільки вхідний і відкритий стік.

Режими роботи портів встановлюються за допомогою регістрів спеціальних функцій P_xM1 і P_xM2 (P_x – лінія порту). Стан бітів портів P_xM1.n P_xM2.n для різних режимів введення/виведення: 00 – квазі-двонаправлений, 01 – двотактний, 10 – тільки вхід (високоімпедансний), 11 – відкритий стік.

Всі лінії портів введення/виведення можуть бути запрограмовані на використання сигналів рівня TTL або входів тригерів Шмітта, вибором

відповідних бітів в регістрі PxS. Вхід з тригером Шмітта забезпечує краще придушення перешкод.

Всі лінії введення/виведення також мають можливість вибору швидкості наростання фронтів з допомогою бітів. регістрів управління PxSR. У вихідних режимах, швидкість наростання повільна.

У квазі-двонаправленій структурі введення/виведення, є три польових транзистора (рис. 4.4), які забезпечують підтягування виходу порту до потенціалу джерела живлення.

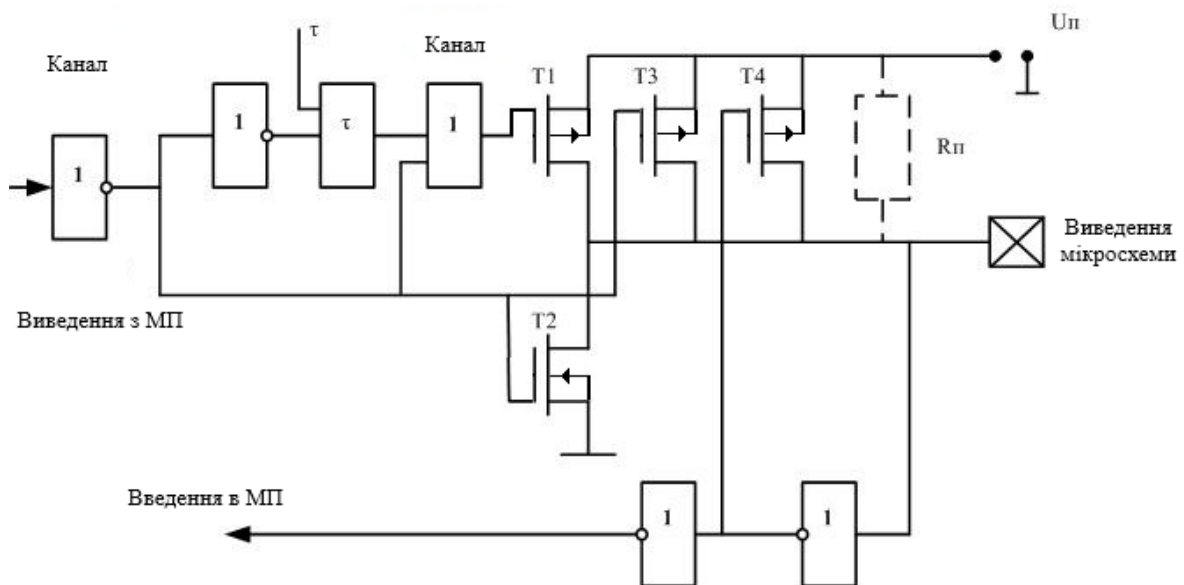


Рисунок 4.4 – Функціональна схема квазідвонаправленого режиму введення/виведення лінії порту мікроконтролера 1T8051

Коли мікропроцесор на вивід 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 виведенням типу «сухий контакт». Сенсори з безконтактними виходами бувають двох видів. З вихідним транзистором NPN типу для підключення до систем з загальним плюсом (рис. 4.5, а), при цьому транзистор комутує мінусовий потенціал, і транзистором PNP типу для підключення до систем з загальним мінусом (рис. 4.5, б), при цьому транзистор комутує

плюсовий потенціал. На підключення сенсорів типу «сухий контакт» не впливає який потенціал в системі є загальним

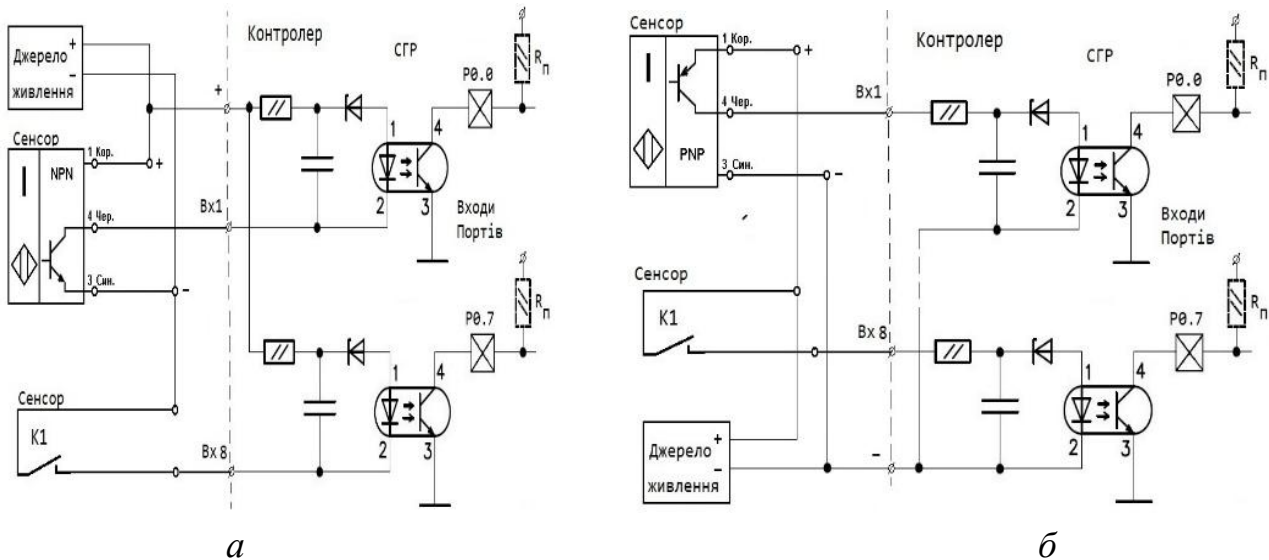


Рисунок 4.5 – Функціональна схема введення інформації від сенсорів

Якщо під час введення інформації від зовнішніх пристроїв на виведенні порту виведено логічна одиниця 1, а зовнішній пристрій тягне вихідний потенціал порту до 0 (рис. 4.5), (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 більше, ніж один вхід TTL (декілька світлодіодів оптронів).

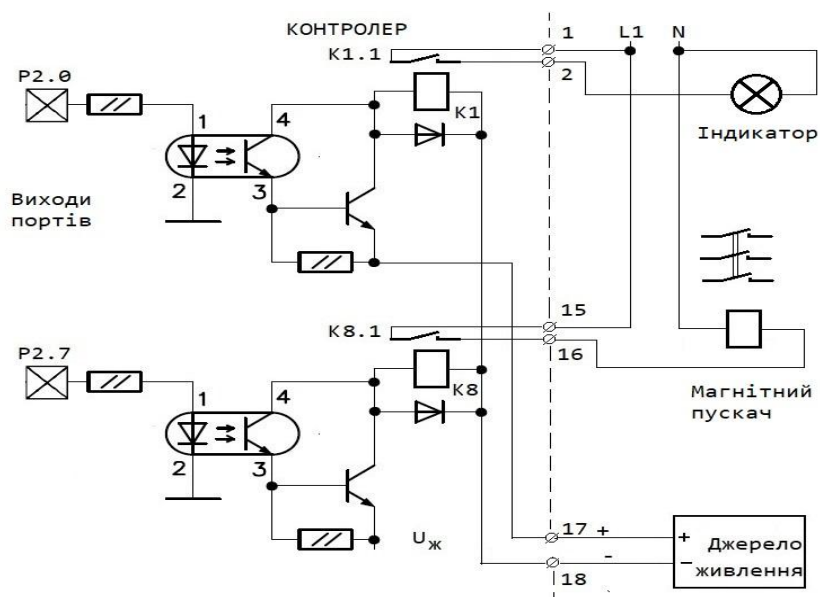


Рисунок 4.6 – Функціональна схема виведення інформації з порту

У режимі квазі-двонаправленого виведення порту, при зміні стану засувки порту з логічного 0 на логічний 1 для забезпечення високої крутизни фронтів при перемиканні включається режим «сильного підтягування» (рис. 4.6), при цьому на час двох тактів мікропроцесора з використанням сигналу t , включається польовий транзистор T2 (рис. 4.3), щоб швидко змінити вихід порту з низького потенціалу на високий. «Слабкий» і «дуже слабкий» режими підтягування продовжують залишатися на рівні потенціалу виведення порту.

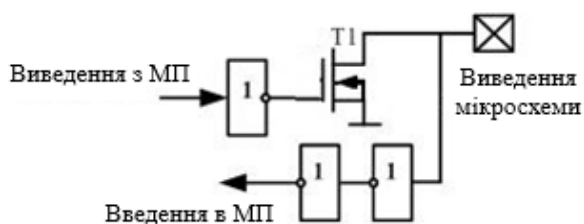


Рисунок 4.7 – Функціональна схема двотактного режиму

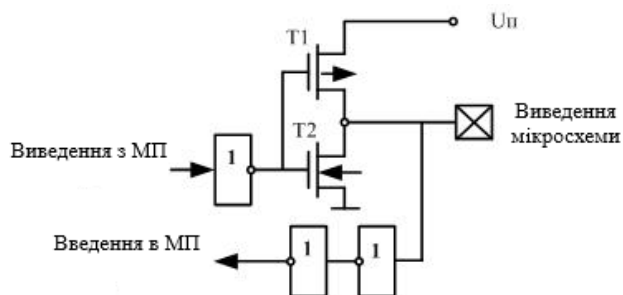


Рисунок 4.8 – Функціональна схема режиму «тільки введення»

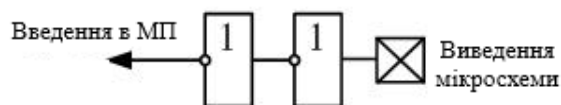


Рисунок 4.9 – Функціональна схема режиму «відкритий стік»

Двотактний режим (Push-Pull, рис. 4.7) має ту ж структуру з низькою напругою, що і квазі-двонаправлений режим, але забезпечує безперервний двотактний режим, коли засувка порту записується логікою 1. Зазвичай двотактний режим використовується в якості вихідного контакту, коли вихідна лінія порту потребує великий струм навантаження

В режимі «тільки вхід» (рис. 4.8) лінія порту підключається до джерела напруги через резистор.

Режим з відкритим стоком (рис. 4.9) відключає всі транзистори високої напруги (рис. 4.3) і управляє низьким рівнем напруги на виході порту тільки тоді, коли засувка порту задається логікою 0. Якщо вихід засувки порту встановлений в 1, вона веде себе так, як ніби вона працює тільки в режимі введення.

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

JВ P2.0, M1 ; перехід на мітку M1, якщо розряд P2.0 дорівнює 1

JNB P0.0, M2 ; перехід на мітку M2, якщо розряд P0.0 дорівнює 0.

Таблиця 4.1 – Альтернативні функції порту 3

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 0. Сприймається сигнал низького рівня або зріз.
TxD	P3.1	Вихід передавача послідовного порту в режимі УАПІ. Вихід синхронізації в режимі регістру, що зсуває.
RxD	P3.0	Вхід приймача послідовного порту в режимі УАПІ. Ввід-вивід даних у режимі регістру, що зсуває.

Операцію передачі сигналів від мікроконтролера до зовнішнього пристрою називають виводом. У цьому випадку на розрядах портів мікроконтролер установлює потрібні логічні рівні сигналів і в такий спосіб проводить керування зовнішніми пристроями. При виконанні операції виводу в порт нове значення записується в засувку у фазі S6P2 останнього машинного циклу команди. Однак новий зміст засувки виводиться безпосередньо на вихідний контакт тільки у фазі S1P1 наступного машинного циклу. Вивід 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, процесор автоматично реалізує спеціальний режим, який називається «читання–модифікація–запис». Цей режим обігу припускає ввід сигналів не із зовнішніх виводів порту, а з його регістру-засувки, що дозволяє виключити неправильне зчитування раніше виведеної інформації, оскільки передбачається модифікування вмісту тригера-засувки, а не вхідної інформації. Цей режим звертання до портів реалізовано у наступних командах:

ANL – логічне І, наприклад ANL P1, A;

ORL – логічне АБО, наприклад ORL P2, A;

XRL – виключає 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 й 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 – установка біта Y порту X;

CLR PX.Y – скидання біта Y порту X.

Приклади роботи з портами.

Порти в мікроконтролері призначені для вводу вхідних сигналів і реалізації функцій дискретного керування. Фактично, завдання дискретного керування об'єктами з використанням програмувальних контролерів полягає у вводі вхідних сигналів, обчисленні логічної функції, що зв'язує вхідні й вихідні сигнали, й виводу результату обчислення на виходи контролера. У першій главі розглянуто завдання дискретного керування, яке можна записати у вигляді комбінаційних і послідовнісних автоматів. Задання умови для комбінаційного автомата можна зобразити у вигляді таблиці істинності. Розглянемо завдання 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.

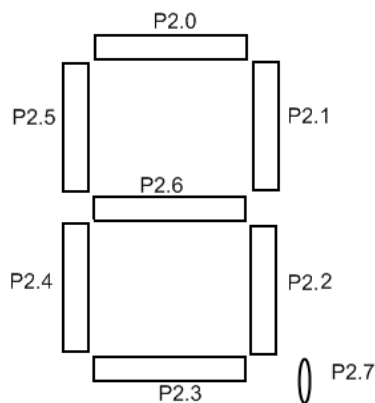


Рисунок 4.10 – Підключення семисегментного індикатора до мікроконтролера

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

Сформуємо в пам'яті програм контролера масив значень байта виводу згідно до 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

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 наступні 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, заснований на програмній реалізації логічних кон'юнктивно диз'юнктивних рівнянь.

Як і в попередньому випадку, функціонування автомата задається у вигляді таблиці істинності, наприклад:

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 наступна:

JV P1.2,M1

JNB P1.4,M1

JNB P2.1,M1

JNB P2.3,M1

SETB P1.7

LJMP EXIT

M1: ; перевірка наступного умови.

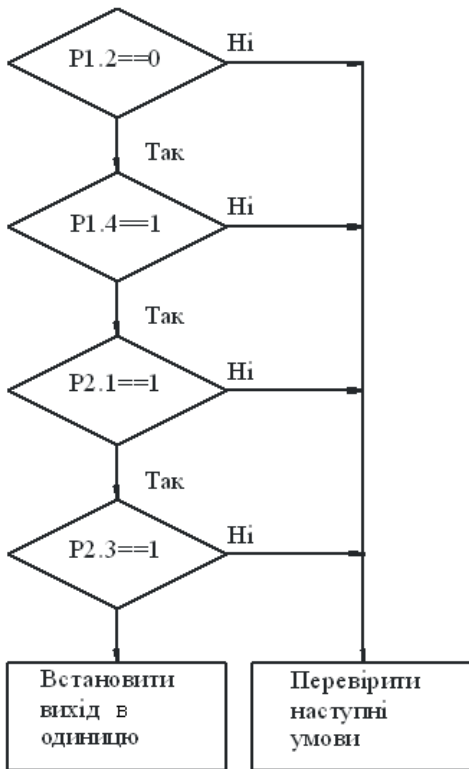


Рисунок 4.11 – Схема алгоритму програмної реалізації логічної функції «І»

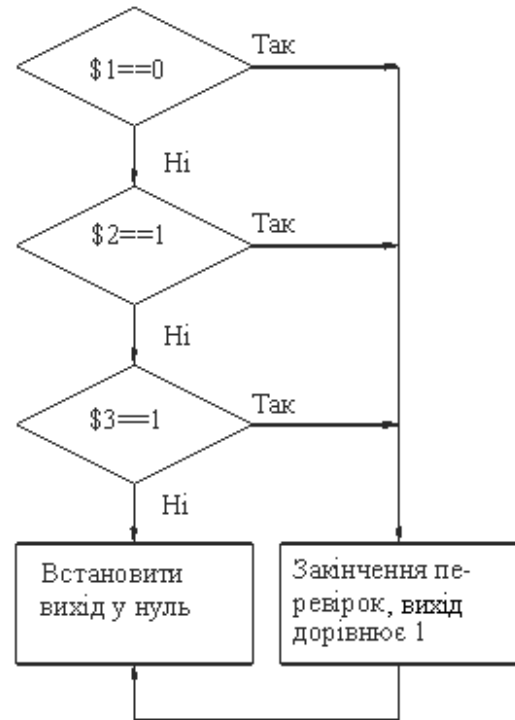


Рисунок 4.12 – Схема алгоритму програмної реалізації логічної функції «АБО»

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 перевіряють інверсну умову. Тоді послідовне виконання таких команд і буде відповідати виконанню вихідної умови логічної функції.

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

```

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 ; i 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 жодна з умов

EXIT:

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

Контрольні запитання

1. Який порт MCS-51, крім порту 0, використовується при зверненні 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 налагоджується на ввід інформації?
5. Яка навантажувальна здатність порту 0?
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. Професор 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. Професор запам'ятовує, у якому місці перервана лекція (мікропроцесор запам'ятовує у стековій пам'яті адресу, на якій перервана програма), і після цього відповідає на запитання студента (мікропроцесор виконує підпрограму обслуговування цього запиту). Після відповідей на запитання студента (після виконання підпрограми мікропроцесором) професор 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 – збільшується на 1, якщо виконується однобайтова команда, збільшується на 2, якщо виконується двобайтова команда й збільшується на 3, 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 2 ($PC+3$), при виконанні команди ACALL вміст лічильника команд збільшується на 1 ($PC+2$) і запам'ятовується в стеку, а потім уже в лічильник команд завантажується адреса підпрограми, яка перебуває в другому й третьому байті команд LCALL або ACALL.

Таким чином, у стеку запам'ятовується адреса команди, при якій 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 з адресою підпрограми, відповідної до даного запиту. Завдяки цій команді МП поточне значення лічильника команд запам'ятовує в стеку, а в лічильник команд заноситься другий і третій байт 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 MCS51.

Функціональну схему взаємодії МП із периферійними пристроями в MCS51 наведено на рис. 5.1.

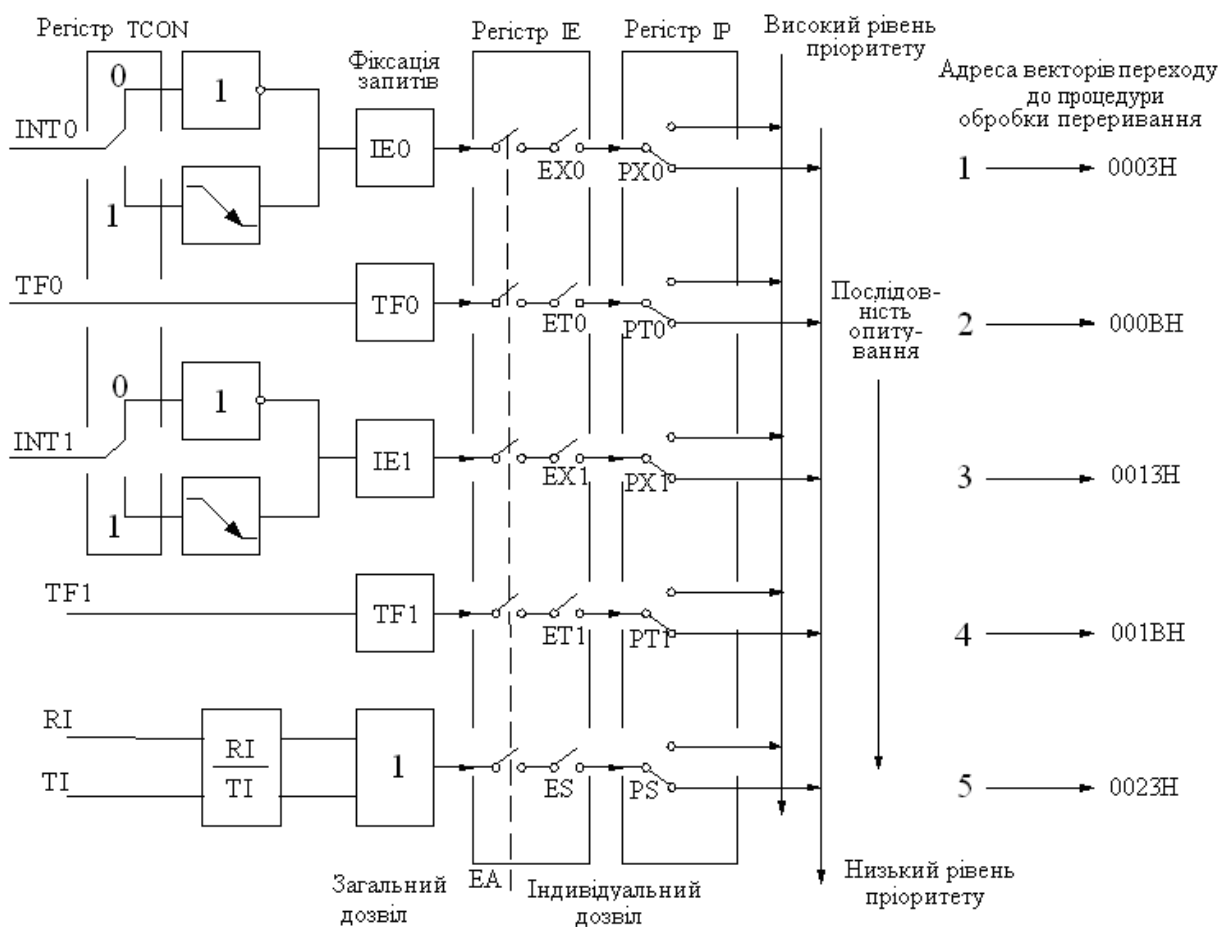


Рисунок 5.1 – Логіка функціонування системи переривань МК51

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.п.). 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 прапорами переповнення TF0 і TF1 у регістрі керування TCON (TCON.5, TCON.7) і прапорами TI,RI у регістрі керування послідовним портом SCON (SCON1, SCON0).

Спрощено логіка функціонування системи переривань зображена на рис. 5.1, де показано п'ять сигналів запитів переривань. *INT0*, *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 регістру керування TCON (біти TCON.0 і TCON.3) табл. 6.2.

Якщо зазначені біти встановлені в 1, то активним є перехід з 1 в 0, 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, відповідні до кожного запиту, фіксовані для кожного джерела й формуються в контролері переривання у відповідності до табл. 4.1.

Таблиця 5.1 – Адреси підпрограм відповідних запитів переривання

Запит переривання	Адреса підпрограми
Зовнішній запит INT0	0003H
Запит від таймера T/30	000BH
Зовнішній запит INT1	0013H
Таймер лічильник T/31	001BH
Послідовний порт	0023H

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

Типову структуру підпрограми обслуговування переривання можна уявити в такий спосіб (рис. 5.2). На початку будь-якої підпрограми зберігається в стеку проміжна інформація основної програми (уміст акумулятора й регістрів загального призначення та ін.). Здійснюється це командами PUSH.

PUSH A ; завантаження в стек умісту акумулятора
 PUSH PSW ; завантаження в стек умісту регістру стану програми
 Тіло підпрограми
 POP PSW ; вивантаження з вершини стеку в регістр стану

POP A ; вивантаження з вершини стеку в акумулятор
RETl ; повернення на адресу перерваної програми

?Рисунок 5.2 – Типова структура підпрограми обслуговування

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 стека буде перебувати адреса перерваної програми. Кількість команд PUSH і POP повинно бути однаковим і визначається самим розробником підпрограми, оскільки від цього залежить час обслуговування переривання. Будь-яка підпрограма обслуговування переривання завершується командою RETl. Завдяки цій команді з вершини стека пересилається в лічильник команд (PC) адреса перерваної програми, крім того, вона скидається прапор запиту переривання, відповідний до цієї підпрограми.

Аналіз структури підпрограми показує, що розмістити підпрограму обслуговування переривання у восьми комірках пам'яті програм далеко не завжди можливо. Тому, як правило, за адресою підпрограм розміщується 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 регістру керування перериваннями ІЕ (табл. 5.2).

Установка відповідного біта регістру ІЕ (SETB ІЕ.х) в одиницю (еквівалентно замиканню перемикача на рис. 4.1) дозволяє обслуговування цього запиту, скидання в 0 (еквівалентно розмиканню перемикача) забороняє обслуговування. Крім того, скидання біта ІЕ.7 (CLR ІЕ.7) забороняє обслуговування всіх запитів переривання.

З одночасною появою декількох запитів переривання в МК51 передбачено двоступінчастий механізм пріоритетів переривань. Перший щабель механізму пріоритетів забезпечується станами бітів регістру пріоритетів ІР (табл. 5.3).

Встановлення відповідного біта регістру ІР в 1 (еквівалентно верхньому положенню перемикача РХ) привласнює «високий» рівень, а скидання в 0 (еквівалентно нижньому положенню перемикача РХ) «низький».

Другий щабель реалізовано у послідовності опитування запитів переривання, яка встановлює фіксовану черговість обслуговування. При цьому запиті INT0 привласнюється найвищий пріоритет, а послідовному порту (запити RI, TI) найнижчий (рис. 5.1).

Таблиця 5.2 – Організація регістру дозволу переривань ІЕ

Символічне позначення	Позиційне позначення	Назва й призначення
ЕА	ІЕ.7	Біт дозволу переривань. Програмно скидається для заборони всіх переривань і встановлюється для дозволу переривань залежно від стану бітів ІЕ.4- ІЕ.0
	ІЕ.6	Не використовується
	ІЕ.5	Не використовується
ЕS	ІЕ.4	Біт керування перериваннями послідовного порту. Програмно встановлюється (скидається) для дозволу (заборони) переривань за прапорами ТІ і RЛ
ЕТ1	ІЕ.3	Біт дозволу переривання від таймера 1. Програмно встановлюється й скидається для дозволу (заборони) переривань від таймера (лічильника) 1
ЕХ1	ІЕ.2	Біт дозволу зовнішнього переривання 1. Програмно встановлюється (скидається) для дозволу (заборони) апаратного переривання ІNT1
ЕТ0	ІЕ.1	Біт дозволу переривання від таймера 0. Програмно встановлюється (скидається) для дозволу (заборони) переривань від таймера (лічильника) 0
ЕХ0	ІЕ.0	Біт дозволу зовнішнього переривань 0. Програмно встановлюється (скидається) для дозволу (заборони) апаратного переривання ІNT0

Таблиця 5.3 – Організація регістру пріоритетів переривань ІР

Символічне позначення	Позиційне позначення	Назва й призначення
	ІР.7	Не використовується
	ІР.6	Не використовується
	ІР.5	Не використовується
PS	ІР.4	Біт керування пріоритетом переривань послідовного порту. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету за допомогою переривання послідовного порту
PT1	ІР.3	Біт керування пріоритетом переривання від таймера 1. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету за допомогою переривання від таймера (лічильника) 1
PX1	ІР.2	Біт керування пріоритетом зовнішнього переривання 1. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету апаратному перериванню 1
PT0	ІР.1	Біт керування пріоритетом переривання від таймера 0. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету за допомогою переривання від таймера (лічильника) 0
PX0	ІР.0	Біт керування пріоритетом зовнішнього переривання 0. Програмно встановлюється (скидається) для присвоєння високого (низького) пріоритету апаратному перериванню ІNT0

Алгоритм функціонування системи обслуговування запитів переривання наведено на рис. 5.3. 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. 5.4). З рисунку можна побачити, що основну програму на адресі 0007H перериває підпрограма, що відповідає запиту T1.

Потім виконання цієї підпрограми на адресі 1002H перериває підпрограма з більш високим пріоритетом, що відповідає запиту INT1, яку на адресі 1303H перериває підпрограма з найвищим пріоритетом, відповідна до запиту INT0.

Як можна побачити з рисунку, спочатку завершується виконання підпрограми, відповідної до запиту 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. 5.4 також зображено уміст стеку на момент виконання підпрограми, відповідної до запиту 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 з системою переривання MCS 8051, але має розширене число запитів переривань від периферійних пристроїв. В залежності від моделі 1T8051 має до 18 джерел переривань з чотирма рівнями пріоритетів. В табл. 5.4 наведено перелік запитів на переривання від периферійних пристроїв для моделей 1T8051 (ML51, ML54, ML56).

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

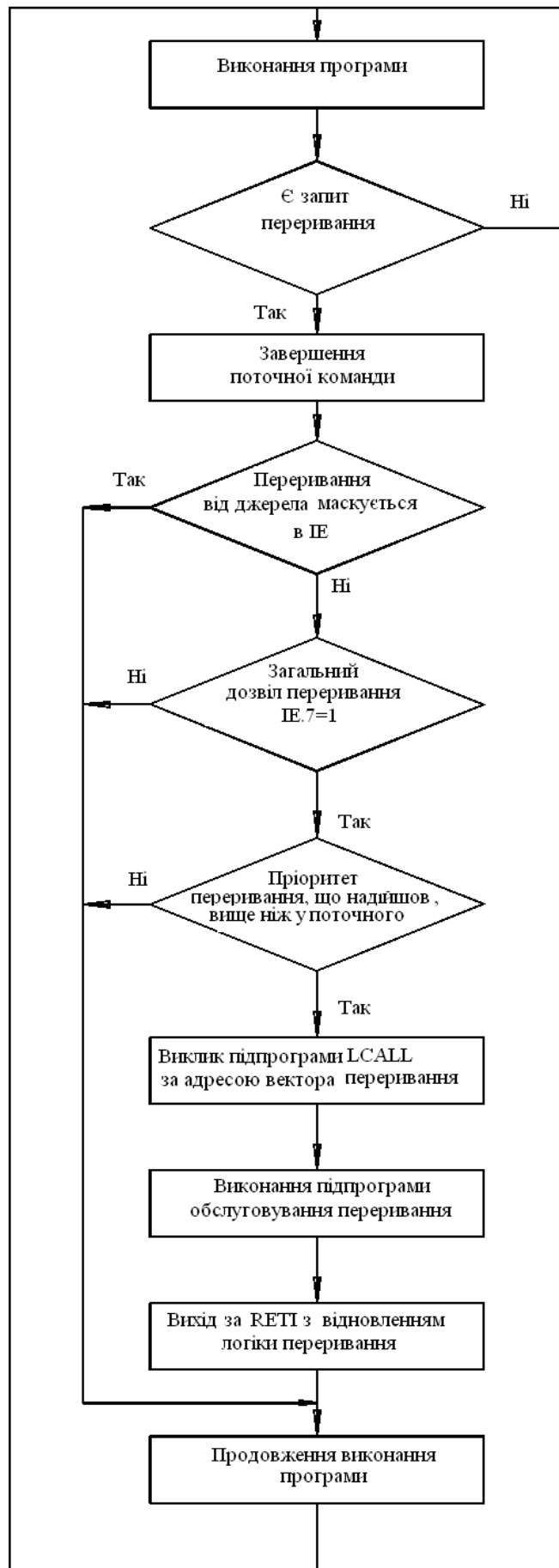


Рисунок 5.3 – Алгоритм обробки переривань у MCS 51

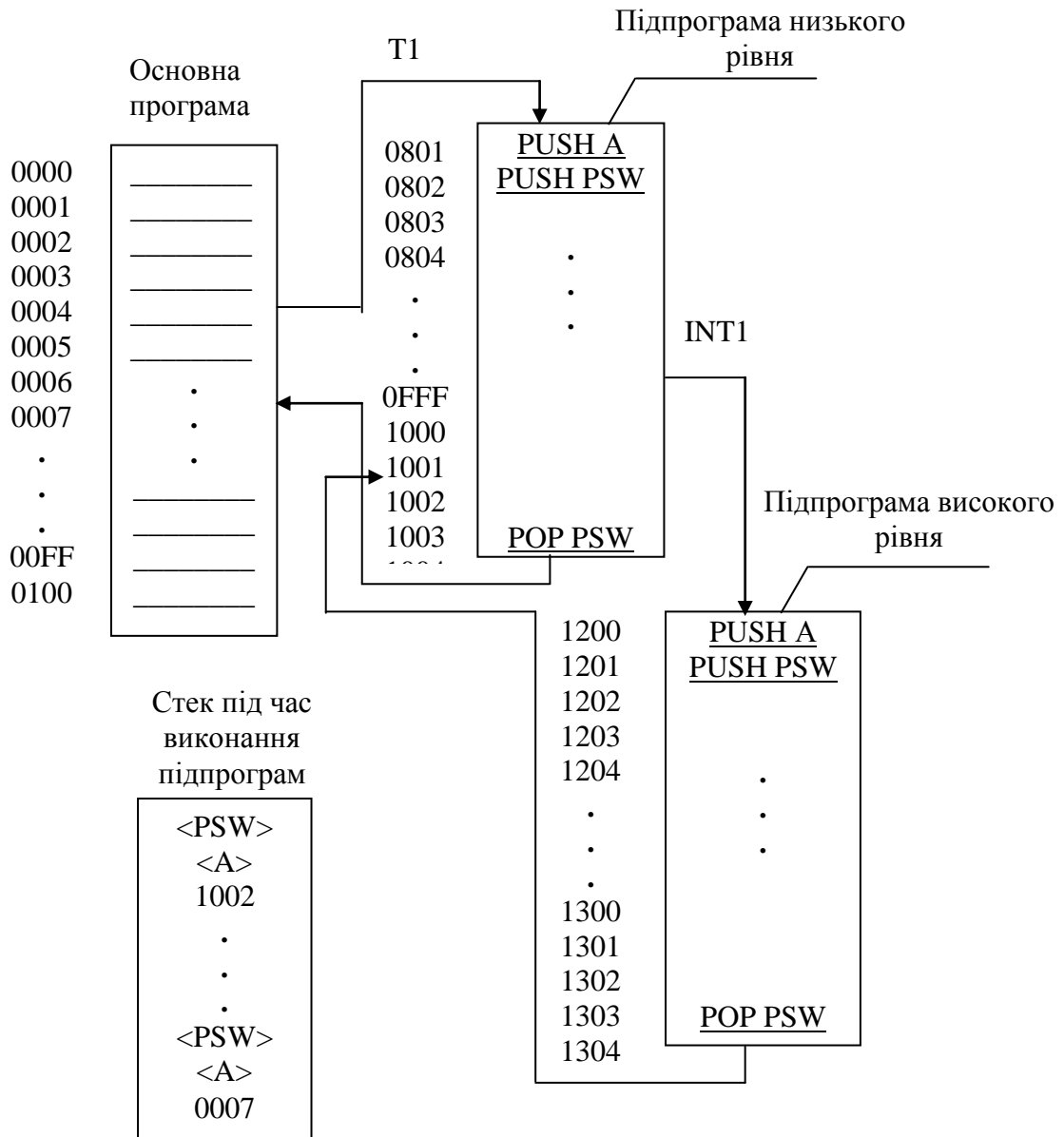


Рисунок 5.4. – Режим вкладених переривань

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

Source Запит переривання	Vektor Address Адреса вектора	Vektor Number Номер вектора	Source Запит переривання	Vektor Address Адреса вектора	Vektor Number Номер вектора
Reset Скидання	0000H	-	Serial port 1 interrupt Переривання послідовного порта 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	0013H	2	CPU Hard Fault interrupt	0093H	18

Зовнішній запит переривання 1			Вихід за кордони пам'яті		
Timer 1 overflow Переривання таймера 1	001BH	3	SMC 0 interrupt Переривання послідовного порта 2	009BH	19
Serial port 0 interrupt Переривання послідовного порта 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 по зниженню 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 порта 3	00DBH	27
Input capture interrupt Переривання по 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 гальма ШІМ	0073H	14	RTC interrupt Годинник реального часу	00F3H	30

Контрольні запитання

1. Скільки сигналів переривань обробляється в MCS51 і скільки в 1T8051?
2. Скільки сигналів переривань обробляється від зовнішніх пристроїв в МК51?
3. Який регістр MCS51 призначено для керування режимом переривань?
4. Який регістр MCS51 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 усіяких сигналів, наприклад, увімкнення реле, тиристора), так і програмно, при цьому сигнали керування виробляються 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 повинно 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 й програмного впливів на об'єкт здійснюється паралельно, як правило, цей час називають системним. Він організований 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_n), закінчення (t_o) інтервалу й, власне, величина тимчасового інтервалу (t_s) (рис. 6.1).

У базовій моделі MCS51 для формування тимчасових інтервалів є два таймери, які являють собою регістри, що працюють у рахункових режимах.

Ідею 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 до верху відбувається відкривання дна, рідина виливається, посудина стає порожньою, дно закривається й система приходить у вихідний стан (роботу механізму відкривання й закривання дна посудини для спрощення не розглядаємо).

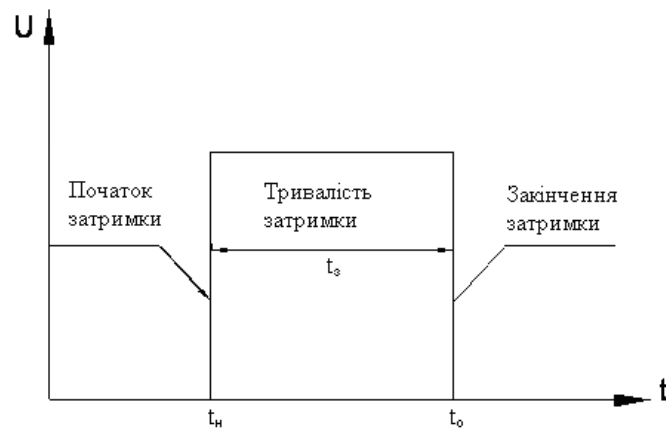


Рисунок 6.1 – Графічне зображення тимчасового інтервалу

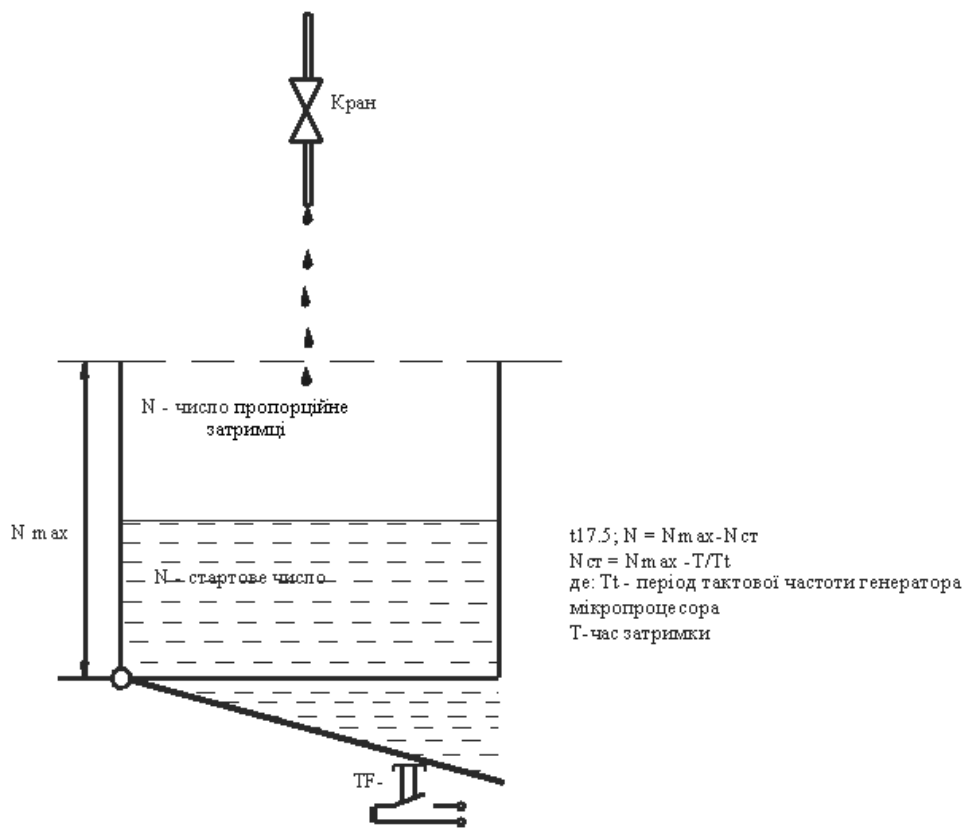


Рисунок 6.2 – Принцип роботи таймера

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 до верху так само буде менше.

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

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

У базовій моделі MCS51 використовуються два 16-розрядних лічильники, Таймер 0 і Таймер 1, кожний з яких працює незалежно в чотирьох режимах 0, 1, 2, 3.

Логіку функціонування таймера зображено на рис. 6.3, а графіки тимчасового аналізу роботи на рис. 6.4.

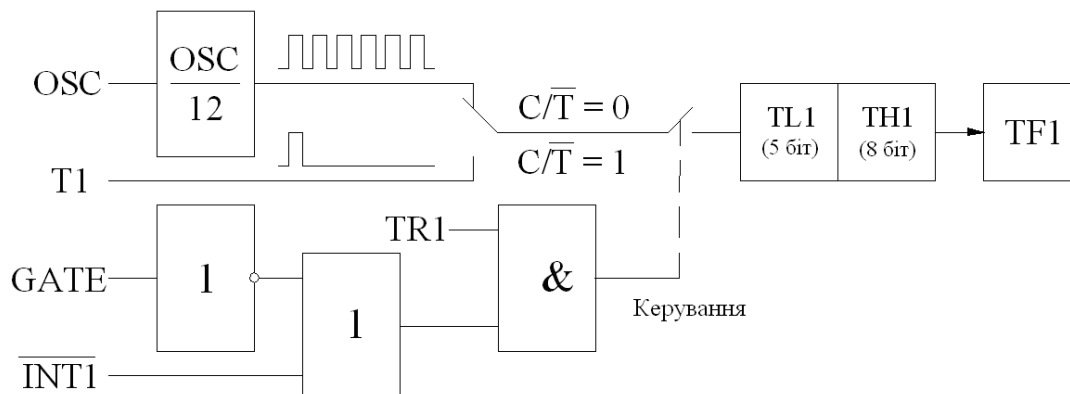


Рисунок 6.3 – Функціональна схема таймера в режимі 0

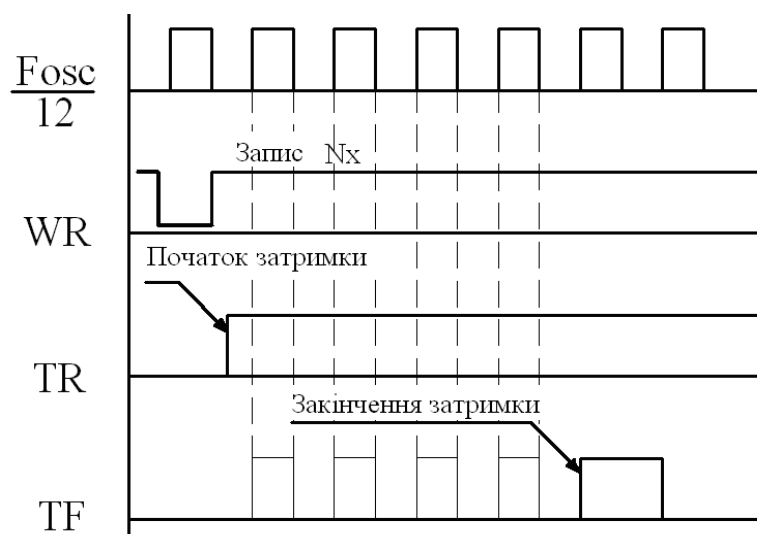


Рисунок 6.4 – Графік тимчасового аналізу роботи таймера в режимі 0

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

Для керування організацією взаємодії таймерів із системою переривання й установа 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 регістру лічильника збільшується на 1 (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 необхідна умова TR1=1 і INT1#=1.
3/T1#	TMOD.6	Біт вибору типу подій для Таймера 1. При C/T1# = 1 він працює як лічильник, при C/T1# = 0 як таймер.
M1.1	TMOD.5	Біт 1 визначення режиму роботи Таймера 1.
M1.0	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 INT0#=1.
3/TO#	TMOD.2	Біт вибору типу подій при таймері 0. При C/TO# = 1 він працює як лічильник, при C/TO# = 0 як таймер.
		Біт 1 визначення режиму роботи Таймера 0.
		Біт 0 визначення режиму роботи Таймера 0.
M1.0	TMOD.1	
M0.0	TMOD.0	

У режимі лічильника інкрементування регістру лічильника (TH, TL) здійснюється при переході вхідного сигналу (T0 або T1) з «1» в «0».

Входи аналізуються у фазі S5P2 кожного машинного циклу. Інкрементування вмісту регістру 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 машинні цикли роботи мікропроцесора, частота рахунку в цьому випадку складе $F_{osc} / 24$ (частота рахунку 500 КГц).

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

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

Таблиця 6.2 – Структура керуючого слова регістру TMOD

TMOD.7	TMOD.66.	TMOD.55	TMOD.44	TMOD.33	TMOD.22	TMOD.11	TMOD.00
GATE1	C/T	M1	M0	GATE0	C/T	M1	M0
Таймер 1				Таймер 0			

Біти TMOD.0 і TMOD.1 регістру режиму 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 включені 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 повинен 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) ухвалюємо такими, що дорівнюють 0.

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 рахунку)

здійснюється бітами запуску таймера TR0 або TR1 (біти SETB TCON.6 або SETB TCON.4) регістру керування TCON (табл. 6.4.).

Таблиця 6.4 – Призначення бітів регістру TCON

Ім'я біта	Номер біта	Функція
TF1	TCON.7	Прапор переповнення таймера 1. Установлюється при переході рахункового регістру таймера зі стану FFH у стан 00H. Очищається при передачі керування на процедуру обробки переривання.
TR1	TCON.6	Біт запуску Таймера 1. При TR1=1 рахунок дозволено. Установлюється й скидається програмно.
TF0	TCON.5	Прапор переповнення таймера 0. Установлюється при переході рахункового регістру таймера зі стану FFH у стан 00H. Очищається при передачі керування на процедуру обробки переривання.
TR0	TCON.4	Біт запуску Таймера 0. При TR0=1 рахунок дозволено.
IE1	TCON.3	Прапор запиту зовнішнього переривання за входом INT1. Установлюється апаратно при виявленні запиту за входом INT1. Скидання залежить від стану біта IT1.
IT1	TCON.2	Біт керування типом сигналу переривання INT1. При IT1=1 – визначається за зрізом (перехід 1-0). Скидання IE1 здійснюється апаратно (автоматично). При IT1=0 – визначається за низьким рівнем сигналу на вході INT1. Скидання IE1 здійснюється програмно.
IE0	TCON.1	Прапор запиту переривання по вхід INT0 (аналогічно IE1). Біт керування типом сигналу переривання INT0 (аналогічно IT1).
IT0	TCON.0	

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

Таким чином, уміст лічильника прямо пропорційний тривалості імпульсу сигналу INT0 або INT1, тобто в цьому режимі за допомогою таймера можна вимірювати тривалість імпульсів, які подаються на входи INT0 або INT1.

Після того, як лічильник повністю заповнено (посудина наповнилася до верху) стан «усі одиниці» (FFFFH) переходить у стан «усі нулі» (0000H) (відкрилося дно посудини), формується сигнал закінчення рахунку 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, яке розміщується в посудині), яке може порахувати рахунковий регістр складає FFFFH (65536 десятковий еквівалент), то максимальний часовий інтервал на одному таймері (на початку рахунку посудина пуста) складе 65,536 мс.

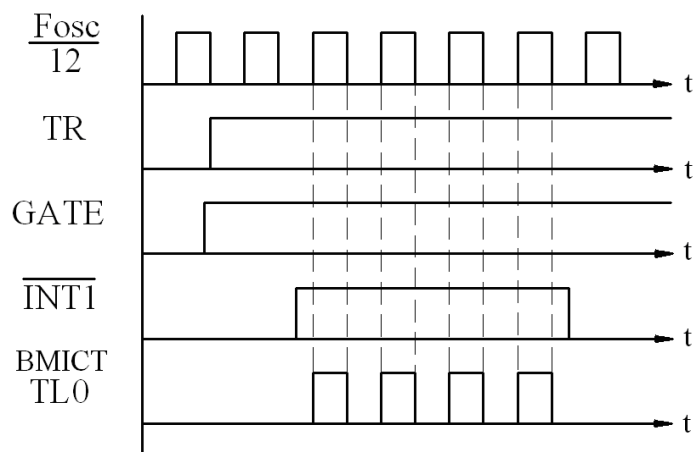


Рисунок 6.5 – Графік тимчасового аналізу роботи таймера в режимі 0 і 1

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

$$N_3 = N_{\max} - N_{CT},$$

а $N_3 = \frac{T_3}{T_t}$, де N_3 – число, яке пропорційне величині часової затримки, тоді

$$N_{CT} = N_{\max} - \frac{T_3}{T_t},$$

$$\frac{T_3}{T_t} = N_{\max} - N_{CT},$$

$$T_3 = (N_{\max} - N_{CT})T_t.$$

Наприклад, потрібно сформувати часову затримку тривалістю $T_3 = 1$ мс, тоді

$$N_{CT} = N_{\max} - \frac{T_3}{T_t} = 65536 - \frac{1 \cdot 10^{-3}}{1 \cdot 10^{-6}} = 65536 - 1000 = 64536.$$

Це число попередньо необхідно записати в рахунковий регістр Таймера, а потім встановити біт запуску Таймера (TR0 або TR1).

У режимі 0 рахунковий регістр містить 13 розрядів, у якому регістр ТН працює як 8-розрядний лічильник, а регістр TL як 5-бітовий переддільник.

У режимі 1 рахунковий регістр містить 16 розрядів, тобто ТН має 8 розрядів і TL – 8 розрядів.

Режим 2 можна назвати режимом програмувального генератора, рис. 6.6.

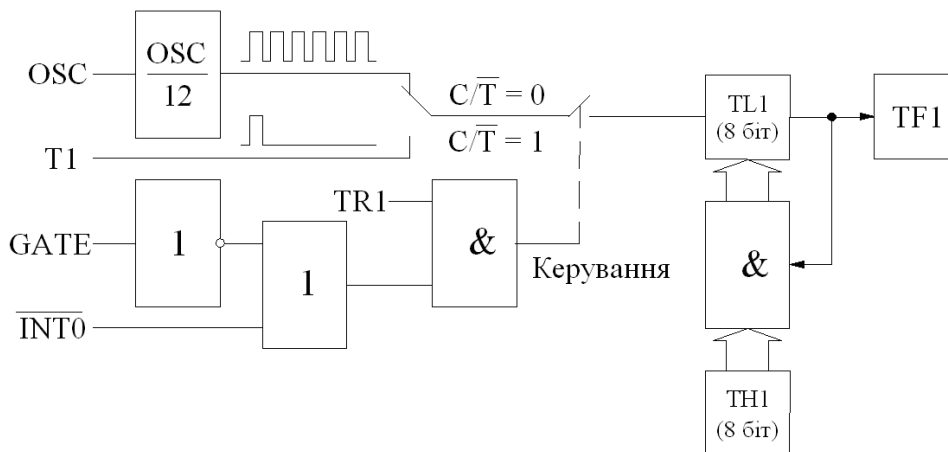


Рисунок 6.6 – Функціональна схема таймера в режимі 2

При цьому регістр TL працює як 8-розрядний рахунковий регістр, а в ТН завантажується число, яке визначає частоту імпульсів на виході таймера.

Після переповнення регістру TL автоматично вміст регістру ТН перезавантажується в регістр TL, при цьому вміст регістру ТН зберігається. На графіку тимчасового аналізу (рис. 6.7) для прикладу показано, що в регістр ТН попередньо занесене число ЕАН (250 у десятковому еквіваленті, тобто для заповнення посудини до верху необхідно всього 5 крапель) і через кожні 5 імпульсів на виході таймера буде формуватися імпульс, частота повторення якого в розглянутому прикладі складе

$$F = 1\text{МГц} / 5 = 200\text{КГц}.$$

Таким чином, змінюючи вміст регістру ТН (MOV Thx, #nx), можна змінювати частоту імпульсів на виході рахункового регістру.

У режимі 3 Таймер 1 заблоковано еквівалентно (TR1=0), а Таймер 0 працює як два 8-розрядні рахункові регістри, причому регістр TL0 керується бітами Таймера 0, а ТН0 – бітами Таймера 1 (рис. 6.8).

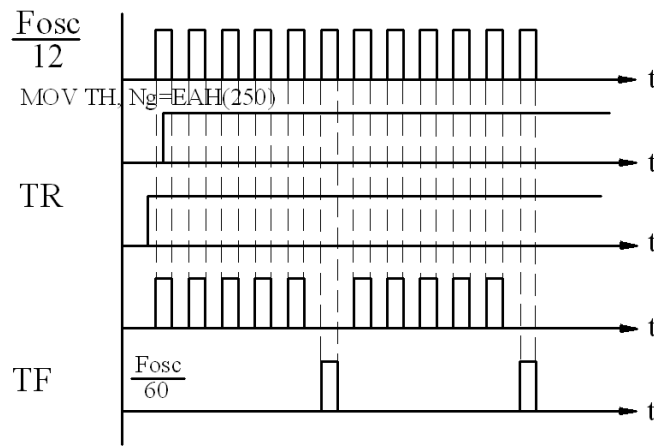


Рисунок 6.7 – Графік тимчасового аналізу роботи таймера в режимі 2

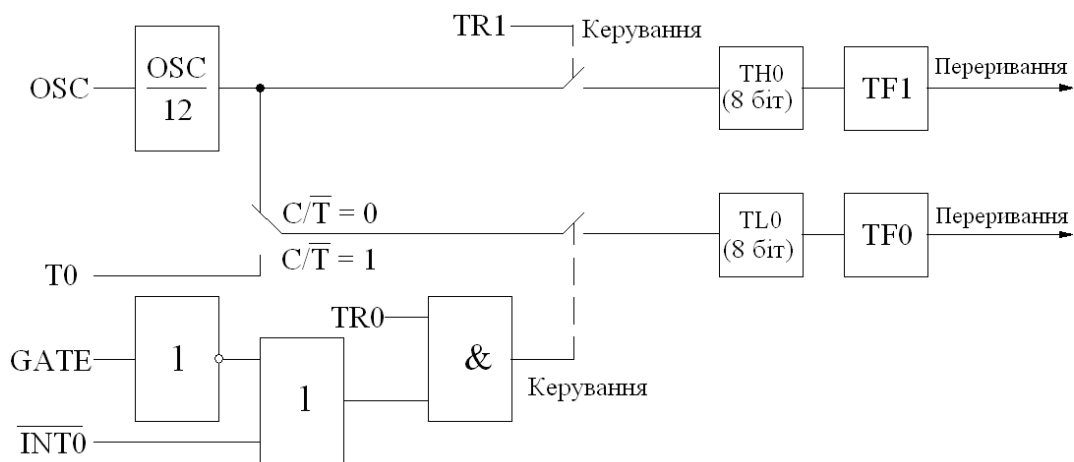


Рисунок 6.8 – Функціональна схема таймера в режимі 3

Взаємодія таймера із системою переривання визначає регістр TCON, у якому установлена бітова адресація. Призначення бітів цього регістру наведено в табл. 5.4. Біти TCON.7 і TCON.5 фіксують переповнення таймера й 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 INT0 і INT1. При одиничному стані TCON.2 аналіз сигналу INT1 виконується за зрізом, а скидання автоматично (апаратно). При нульовому стані біта TCON.2 аналіз запиту сигналу INT1 здійснюється за низьким рівнем сигналу на вході, а скидання сигналу INT1 виконується програмно після закінчення підпрограми обслуговування переривання.

У режимі 0 рахунковий регістр містить 13 розрядів, у якому регістр TH працює як 8-розрядний лічильник, а регістр TL як 5-бітовий переддільник.

У режимі 1 рахунковий реєстр містить 16 розрядів, тобто 8 розрядів ТН і вісім ТЛ.

Режим 2 можна назвати режимом програмного генератора. При цьому реєстр ТЛ працює як 8-розрядний рахунковий реєстр, а в ТН завантажується число пропорційне до частоти імпульсів на виході таймера.

Після переповнення реєстру ТЛ автоматично вміст реєстру ТН перезавантажується в реєстр ТЛ, при цьому вміст реєстру ТН зберігається.

Таким чином, змінюючи вміст реєстру ТН (MOV Thx, #nx), можна змінювати частоту імпульсів на виході рахункового реєстру.

У режимі 3 (рис. 5.8) Таймер 1 заблоковано еквівалентно (TR1=0), а Таймер 0 працює як два 8-розрядні рахункові реєстри, причому реєстр TL0 керується бітами Таймера 0, а TH0 – бітами Таймера 1.

Розглянемо приклад організації часової затримки в 1 секунду на одному таймері. Оскільки максимальна величина затримки в одному таймері складає 65мс, то спочатку організуємо затримку 50 мс, а потім створим програмний лічильник з коефіцієнтом 20 (50x20=1000мс). Розрахуємо число, яке необхідно завантажити в таймер T0 для отримання затримки 50 мс. Згідно наведеним вище виразам $N_{ст} = 65536 - 50000 = 15536$, переведемо у шістнадцяткову систему

$N_{ст} = 3cb0h$, наступним етапом визначимо число, яке необхідно занести в реєстр TMOD для забезпечення необхідного режиму. Аналіз показує, 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 мілісекунд,
setb tcon.4 ; запуск таймера (установка біта TR0 у реєстрі TCON)
setb ie.7 ; дозвіл переривань
setb ie.1 ; дозвіл переривань від Таймера 0
mov prescal,#20 ; коефіцієнт множення затримки 50 мілісекунд,
m0 pushA (підпрограма переривань
mov th0,#3ch ; перезавантаження таймера (50 мілісекунд)
mov tl0,#0b0h ;
djnz prescal,m0; перевірка затримки 1 секунда,
pop acc ; відновлюємо вміст акумулятора,
reti; вихід з підпрограми переривання.
end.
```

Контрольні запитання

1. Скільки таймерів містить MCS51?
2. Скільки режимів роботи має таймер MCS51?
3. Який сигнал MCS51 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. Вміст таймера в процесі роботи?
8. Який регістр забезпечує установку 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.п. Схеми підключення датчиків до МК51 наведено на рис 7.1, а. Ввід інформації із цих датчиків має свої особливості.

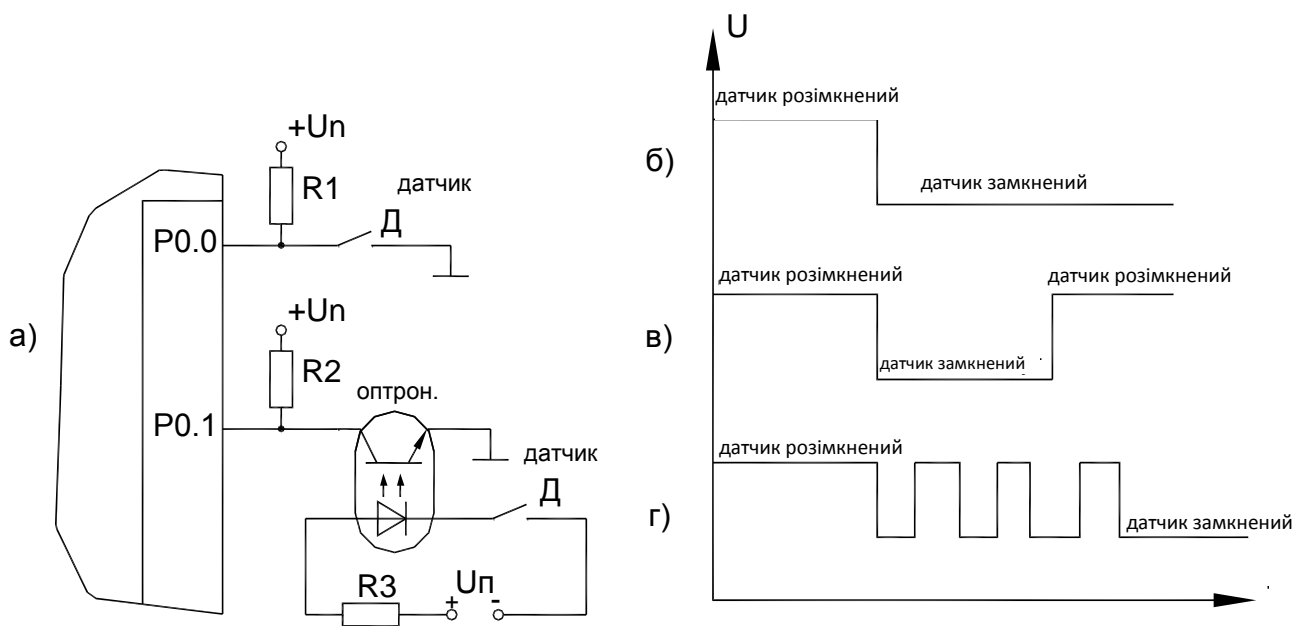


Рисунок 7.1 – Схеми підключення дискретних датчиків

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).

Причому 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 здійснює команди операції, які відповідають розімкненому стану контакту. Схему алгоритму опитування стану датчика наведено на рис. 7.2, а.

Якщо спостерігається деренчання контакту, то усунути його можна шляхом вводу в схему алгоритму лічильника заданого числа станів, що збігаються (рис. 7.2, б), або шляхом вводу тимчасової затримки (рис. 7.2, в).

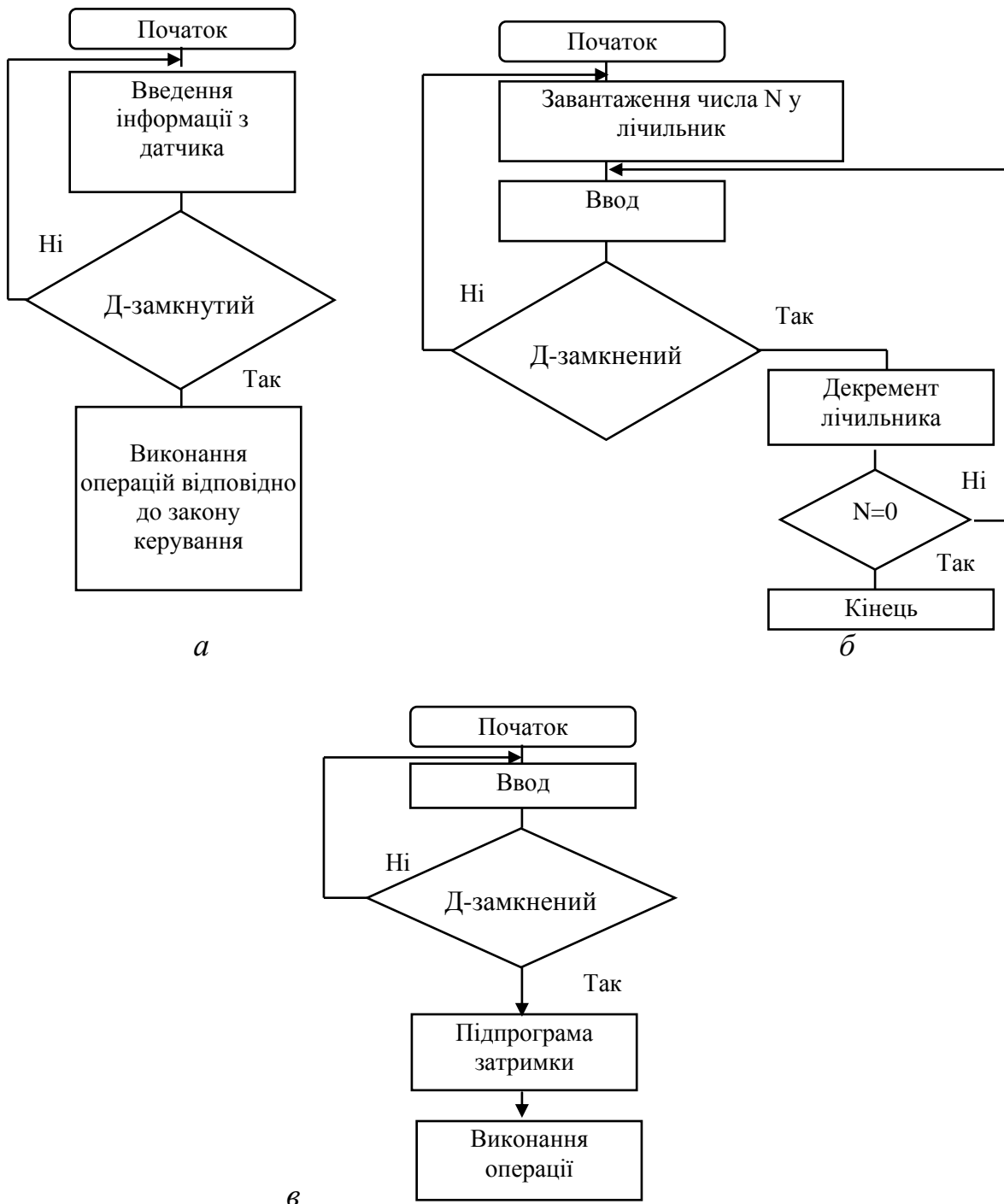


Рисунок 7.2 – Схема алгоритму аналізу інформації з дискретного датчика:
 а – опитування датчика; б – з використанням лічильника;
 в – з використанням часової затримки

7.2. Приклад виконання програмної частини курсового проєкта

Як приклад об'єкта управління буде розглянута технологічна установка для отримання рідких сумішей шляхом змішування вихідних компонентів в необхідних пропорціях. Установки такого типу застосовуються для виготовлення будівельних сумішей, лакофарбових покриттів та ін. Технологічна схема для суміші яка складається з двох компонентів показана на рис. 7.3.

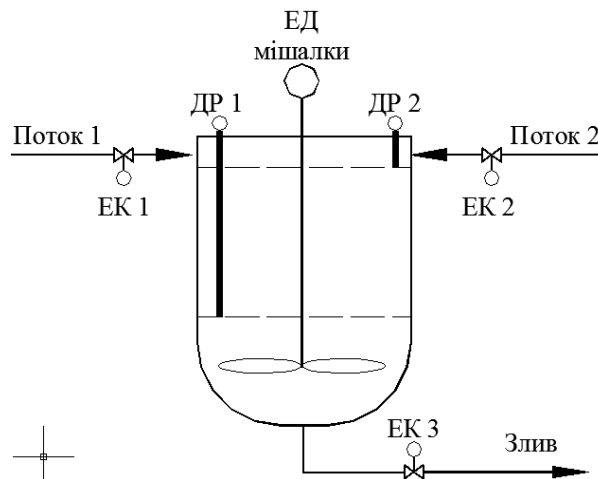


Рисунок 7.3 – Технологічна схема приготування рідких сумішей

У міксерну ємність вихідні компоненти подаються по черзі через електроклапан ЕК1 для першого потоку і ЕК2 – для другого. Дозування компонентів виконується по датчикам 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 тех. процесу зводиться до наступного: по сигналу від кнопки «пуск» система вмикає електроклапан ЕК1 та заливає компонент потоку 1 до моменту спрацьовування датчика ДР1, після чого електроклапан ЕК1 вимикається; далі вмикається електроклапан ЕК2 та заливається компонент потоку 2 до моменту спрацьовування датчика ДР2 після чого електроклапан ЕК2 вимикається; вмикається ЕД мішалки на 4 хвилини; після вимикання ЕД мішалки включається зливний клапан ЕК3 на 1,5 хвилини.

Якщо в момент пуску ДР1 показує, що компонент потоку 1 є а іншого компонента немає, то необхідно виконати заливку потоку 2 і далі виконати перемішування і злив готового продукту. У разі початку роботи при повній ємності необхідно виконати процеси перемішування та зливу, відповідно до алгоритму приведену в попередньому абзаці.

Приведений матеріал являє собою словесний опис об'єкта управління. Така робота завжди виконується на початку проектування.

При виконанні курсового проекту словесний опис об'єкта управління видається в завданні.

На другому етапі проектування виконується розробка алгоритму керування. Задачу логічного управління, найкраще описувати алгоритмами роботи автоматів з пам'яттю. Такі алгоритми зручно представляти у вигляді спрямованого графа-станів. В цьому випадку вершини графа позначають фізичні стани, в яких може перебувати об'єкт управління або система, а на дугах вказуються логічні умови переходів між цими станами. При використанні моделі автомата Міллі на дугах також вказуються і вихідні сигнали.

При програмних методах реалізація алгоритму представленого у вигляді графа можливості використання вхідних і вихідних сигналів значно розширюються порівняно з апаратною реалізацією. В якості вхідних змінних 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. Над вхідними змінними можуть виконуватися арифметичні або логічні операції, а також операції порівняння (більше, менше, дорівнює). Над таймерами – визначення кінця затримки. Вихідними можуть бітові та числові змінні, лічильники та таймери. Над ними виконуються операції установки, скидання, присвоєння значення, інкремент, декремент або скидання в нуль. Над таймерами – запуск та зупинку.

Приклад графа для представленого словесного опису наведено на рис. 7.4. Ідентифікація вхідних та вихідних змінних, функцій управління часовим витримкам наведені в табл. 7.1. У таблиці також показані адреси портів для підключення датчиків, кнопок і виконавчих механізмів.

Таблиця 7.1 – Ідентифікація вхідних і вихідних змінних

Вхідні параметри об'єкта управління	Ім'я змінної	Ідентифікатор змінної	Порт МК (адреса)
електро двигун мішалки, ЕД	electricstirremotor	ESM	P2.0 (0A0h)
електро клапани: ЕК1 (поток 1) ЕК2 (поток 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)
датчикирівня: ДР1	levelsensor 1	LS1	P1.1 (91h)

ДР2	levelsensor2	LS2	P1.2 (92h)
Витримки часу – змішування – 4 хв. – злив – 1.5 хвил	timedelay1	Tm1	240 с 90 с

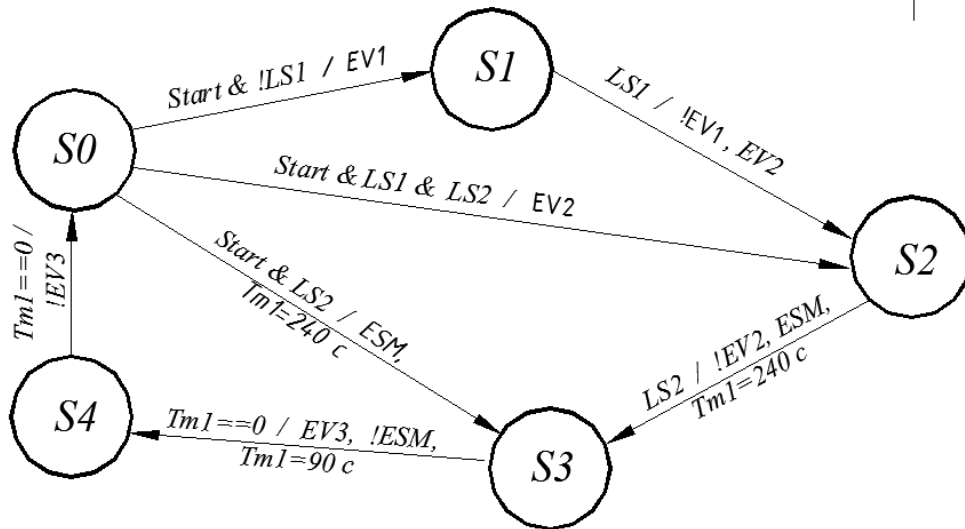


Рисунок 7.4 – Граф управління мішалкою

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 – включення/виключення клапанів компонентів 1 і 2 та ESM – управління двигуном мішалки.

S1 – стан заповнення міксера компонентом 1.

S1 – S2 після спрацьовування датчика рівня ДР1 (LS1) відключається ЕК1 (!EV1) і включається 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 включаються перша витримка часу (Tm1=240).

S3 – стан змішування.

S3 – S4 після закінчення витримки 240 с. ($Tm1=0$) виконується перехід до зливу готової суміші. На цій дузі вимикається двигун мішалки (!ESM), вмикається ЕКЗ (EV3), і включається друга витримка часу ($Tm1=90$).

S4 – стан зливу готової суміші.

S4 – S0 після закінчення витримки 90 с. ($Tm1=0$) вимикається ЕКЗ (!EV3), і здійснюється перехід у початковий стан.

Програму, що реалізує алгоритм управління, записаний у вигляді графа, слід оформляти як підпрограму. Такий підхід дозволяє легко організувати багатозадачність роботи мікроконтролера, де кожен об'єкт управляється від окремої підпрограми. Виклик таких підпрограм проводиться або циклічно, або по лічильник 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 наступне. 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 повинен 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 включає в себе частину ініціалізації периферійних пристроїв і підпрограму обслуговування переривання від таймера.

```

;*****ТЕКСТ ПРОГРАММИ*****
;*****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 комірки для зберігання коду стану
;*****ДОДАТКОВІ ЗМІННІ
Tm1        equ  41h  ; адреса комірки для затримки часу
Divider     equ  43h  ; адреса комірки 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    EV3 ; P2.3 – ЕК 3
;*****початкова установка реєстрів управління МК і таймера 0
mov  sp,#70h ; установка початкової адреси стекової пам'яті
mov  tmod,#01h ; установка режиму 1 для таймера 0
mov  th0,#3ch } ; занесення в таймер 0 числа 3cb0h, яке визначає
mov  tl0,#0b0h } ; період відмітчика часу 50 мілісекунд
setb tcon.4   ; запуск таймера (установка біта TR у реєстрі 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 сек.
mov  stateG1, #00;привласнення 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 викону-
    jbLS1,S0_S2 ; }ється то треба перейти на аналіз другої дуги S0–S2
;***** формування сигналів керування
    setb EV1 ; включення клапанаEK1
    mov stateG1,#01 ; зміннійstateG1=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 ; включення клапана EK2
    mov stateG1,#01 ; зміннійstateG1=01 (стан S1)
    sjmp ExitG1 ; вихід із підпрограми
;***** аналіз дуги S0–S3 (Start & LS1 @ LS2 / ESM, Tm1=240 c)
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 ; включити ЕД мішалки
    mov Tm1,#0F0h ; 240 секунд
    mov stateG1,#01 ; зміннійstateG1=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 ; виключення EK1
;***** формування вихідних сигналів керування та змінних
    setb EV2 ; включення EK2
    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 c)
    jnb LS2, ExitG1; перехід на вихід із підпрограми
    clr EV2 ; виключення EK1
    setb ESM ; включення ESM
;***** формування вихідних сигналів керування та змінних
    mov Tm1,#240 ; запуск витримки 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 c)
    mov a,Tm1      ; }перевірка закінчення витримки 240 сек.
    cjne a,#0, ExitG1; }
;***** формування вихідних сигналів керування та змінних
    clr  ESM       ; виключення ЕД мішалки
    setb EV3       ; включення клапану зливуЕК3
    mov  Tm1,#90   ; запуск витримки 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 витримки 90 сек.
    cjne a,#0, ExitG1; }
;***** формування вихідних сигналів керування та змінних
    clr  EV3       ; виключення клапану зливуЕК3
    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 секунда
    mov Divider,#20 ; }перезавантаження перед дільника
; ***** декриментування затримок
    mov a, Tm1    ; пересилання змінної Tm1 в акумулятор,
    jz, m1        ; перевірка закінчення затримки, якщо ні,
    decTm1        ; декремент змінноїTm1,
m1:  pop PSW      ; відновлення вміст PSW та акумулятора,
    roracc
    reti          ; вихід з підпрограми переривання.

    End           ; кінець програми

```

Курсовий проект виконується в середовищі розробки MCStudio. Для відлагодження програми та демонстрації її роботи викладачу використовувати «Середовище оточення» пакету, в якому присутня картинка технологічного

процесу. Значення стану датчиків та кнопок керування вводиться за допомогою елемента «кнопка», а виконавчі пристрої - елементом «індикатор».

Схему електричну принципову контролера розробляють на основі схем введення інформації з сенсорів та виводу інформації з порту приведених 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 призначено у першу чергу для точного позиціонування вала без застосування систем зворотного зв'язку. Обмотки КД є частиною статора. На роторі розташовано постійний магніт або у випадках зі змінним магнітним опором зубчастий блок з магнітом'якого матеріалу. Усі комутації проводяться зовнішніми схемами. Відносно до контролера відмінність між ними відсутня. На двигунах з постійними магнітами зазвичай є дві незалежні обмотки.

Крокові двигуни мають широкий діапазон кутових роздільних здатностей. Більш грубі мотори, звичайно с постійними магнітами, обертаються на 90° за крок, у той час як прецизійні двигуни можуть мати роздільність $1,8$ або $0,72^\circ$ на крок.

Для правильного керування біполярним кроковим двигуном необхідна електрична схема, яка повинна виконувати функції старту, зупинки, реверсу й зміни швидкості. Кроковий двигун транслює послідовність цифрових перемикачів у рух. «Обертове» магнітне поле забезпечується відповідними перемикачними напруг на обмотках. Слідом за цим полем буде обертатися ротор, з'єднаний за допомогою редуктора з вихідним валом двигуна.

Потужність крокових двигунів знаходиться у діапазоні від одиниць ватів до одного кіловата. Кроковий двигун має не менш двох положень стійкої рівноваги ротора в межах одного оберту. Напруга живлення обмоток керування кроковим двигуном являє собою послідовність однополярних або двополярних прямокутних імпульсів, що надходять від електронного комутатора (К). Результуючий кут відповідає числу перемикачів комутатора, а частота обертання двигуна – частоті перемикачів електронного комутатора.

Комутатор повинен забезпечувати зміну полярності струму в обмотках. Він являє собою два транзисторні мости, в діагональ яких включені 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 обертися вектор магнітного поля, а за ним і ротор, у прямій або зворотній послідовності. При такому керуванні двигун має 4 стійких стану на одному оберті ротора.

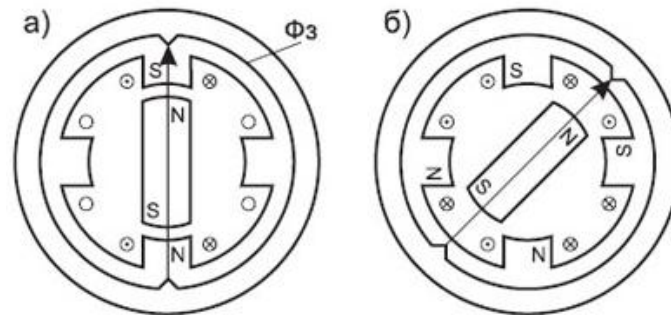


Рисунок 7.5 – Положення ротора при кроковому – а і напівкроковому керуванні – б

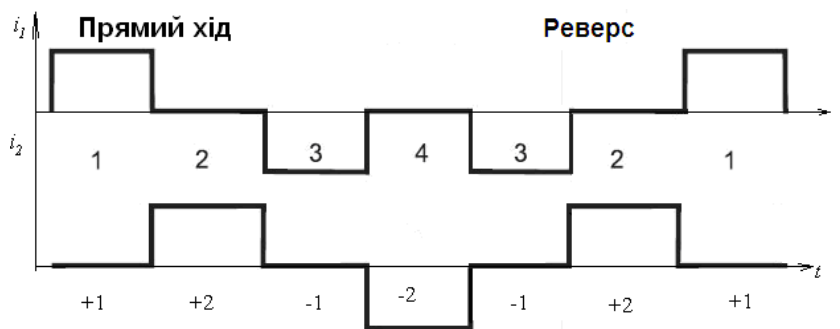


Рисунок 7.6 – Прямий і зворотний хід у кроковому двигуні

Одночасне включення двох обмоток приводить до орієнтування вектора магнітного поля із кроком 45° щодо вертикальної осі. Таке керування називається напівкроковим. Напівкроковий режим дозволяє вдвічі підвищити точність позиціонування крокових двигунів. Послідовність комутації обмоток двигуна в напівкроковому режимі наведена на рис. 7.7. При комутації по черзі включаються одна обмотка, а за нею дві разом і т.п. При такому керуванні двигун має 8 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 вигляд:

$$M - M_C = J \frac{\omega - \omega_0}{\Delta t}, \quad (7.1)$$

де M – обертаючий момент електродвигуна, Дж; M_C – статичний момент робочого механізму, наведений до вала двигуна, Дж; J – момент інерції системи, наведений до вала двигуна, Дж·с²; α – кут повороту, рад.; t – час, с.

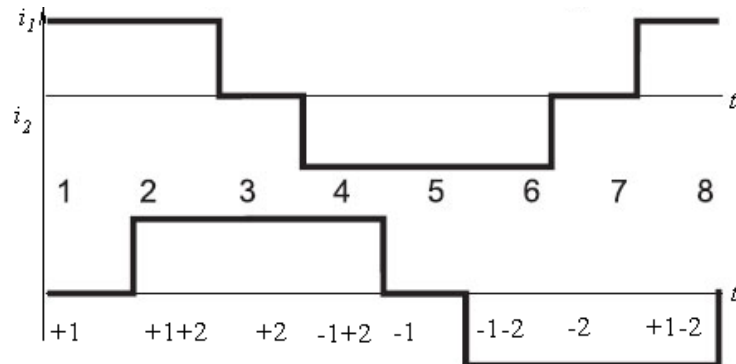


Рисунок 7.7 – Напівкрокове керування кроковим двигуном

$$\frac{\omega - \omega_0}{\Delta t} - \text{кутове прискорення, } 1/\text{с}^2; \quad (7.2)$$

$$\omega = \frac{\alpha - \alpha_0}{\Delta t} - \text{кутова швидкість, } 1/\text{с}^2. \quad (7.3)$$

Наявність моменту інерції приводить до того, що для розгону або гальмування двигуна необхідний час, інакше вектор положення ротора почне відставати або випереджати вектор поля й двигун втратить синхронізм, що призведе або до пропуску кроків, або зупинки двигуна.

Наведемо приклад 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 проводиться в напівкроковому режимі. Необхідно визначити час виходу двигуна на задану швидкість, а також кількість кроків для розгону й тривалість кожного кроку. Розрахунки проводяться без обліку моменту опору обертанню.

Розв'язок: оскільки початкова швидкість дорівнює нулю, то $\omega' = 0$. Тоді (1) можна переписати

$$M = J \frac{\omega}{\Delta t}, \quad (7.4)$$

$$\Delta t = \omega \frac{J}{M}, \quad (7.5)$$

$$\omega = \frac{60 \cdot 2\pi}{60} = 6,28 \frac{\text{рад}}{\text{с}}. \quad (7.6)$$

Звідси

$$\Delta t = 6,28 \frac{500}{1500} = 2,1 \text{ с.}$$

Визначимо, скільки кроків повинен виконати двигун до повного розгону.

Перед початком розгону $\alpha = 0$

$$M = J \frac{2\alpha}{\Delta t^2}, \quad (7.7)$$

звідси

$$\alpha = \frac{\Delta t^2}{2} \cdot \frac{M}{J} = \frac{2,1^2}{2} \cdot \frac{1500}{500} = 6,62 \text{ рад.}$$

У напівкроковому режимі кроковий двигун робить 8 кроків за один оберт.

Тоді один крок – це поворот на кут

$$S = \frac{2\pi}{8} = \frac{\pi}{4} \text{ рад.}$$

Виходить, до кінця розгону двигун зробить: $\frac{6,62 \cdot 4}{\pi} = 8,4$ кроків.

Тривалість першого кроку буде дорівнювати, виходячи з (7)

$$t_1 = \sqrt{2 \cdot \alpha_1 \cdot \frac{J}{M}} = \sqrt{2 \cdot \frac{\pi}{4} \cdot \frac{500}{1500}} = 0,72 \text{ с.}$$

Тривалість другого кроку визначимо як різницю часу повороту на два кроки мінус поворот на один крок. Тривалість третього – як різницю часу повороту на три й два кроки і т.п. Розрахунки значень зручніше за все проводити в програмі EXCEL.

$$t_2 = \sqrt{2 \cdot 2 \cdot \alpha_1 \cdot \frac{J}{M}} - \sqrt{2 \cdot \alpha_1 \cdot \frac{J}{M}} = 0,3 \text{ с;}$$

$$t_3 = \sqrt{3 \cdot 2 \cdot \alpha_1 \cdot \frac{J}{M}} - \sqrt{2 \cdot 2 \cdot \alpha_1 \cdot \frac{J}{M}} = 0,23 \text{ с;}$$

...

$$t_9 = \sqrt{9 \cdot 2 \cdot \alpha_1 \cdot \frac{J}{M}} - \sqrt{8 \cdot 2 \cdot \alpha_1 \cdot \frac{J}{M}} = 0,124 \text{ с.}$$

Тривалість кроку для швидкості обертання 60 об/хв. обчислюємо в такий спосіб:

$$t_s = \frac{S}{\omega} = \frac{\pi}{4 \cdot 6,28} = 0,125 \text{ с.}$$

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 керування складається із частин. Перша оформляється у вигляді підпрограми з передачею параметрів. У якості параметрів задається напрямок руху, кількість кроків. Підпрограма розраховує кількість кроків розгону й гальмування двигуна, а також кількість кроків з номінальною швидкістю. Алгоритм підпрограми, яка ініціалізує керування двигуном із прикладу розрахунків, наведений на рис. 7.8.

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 й гальмування програма використовує таблицю, 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 показано на рис.7.10.

Схему підключення крокового двигуна до мікроконтролера зображено на рис. 7.11. Вона складається із двох транзисторних мостів, у діагональ яких включені статорні обмотки крокового двигуна ОВ1 і ОВ2. Керування мостами проводиться чотирма молодшими бітами порту P0. Коли подається одиниця на P0.0, 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.

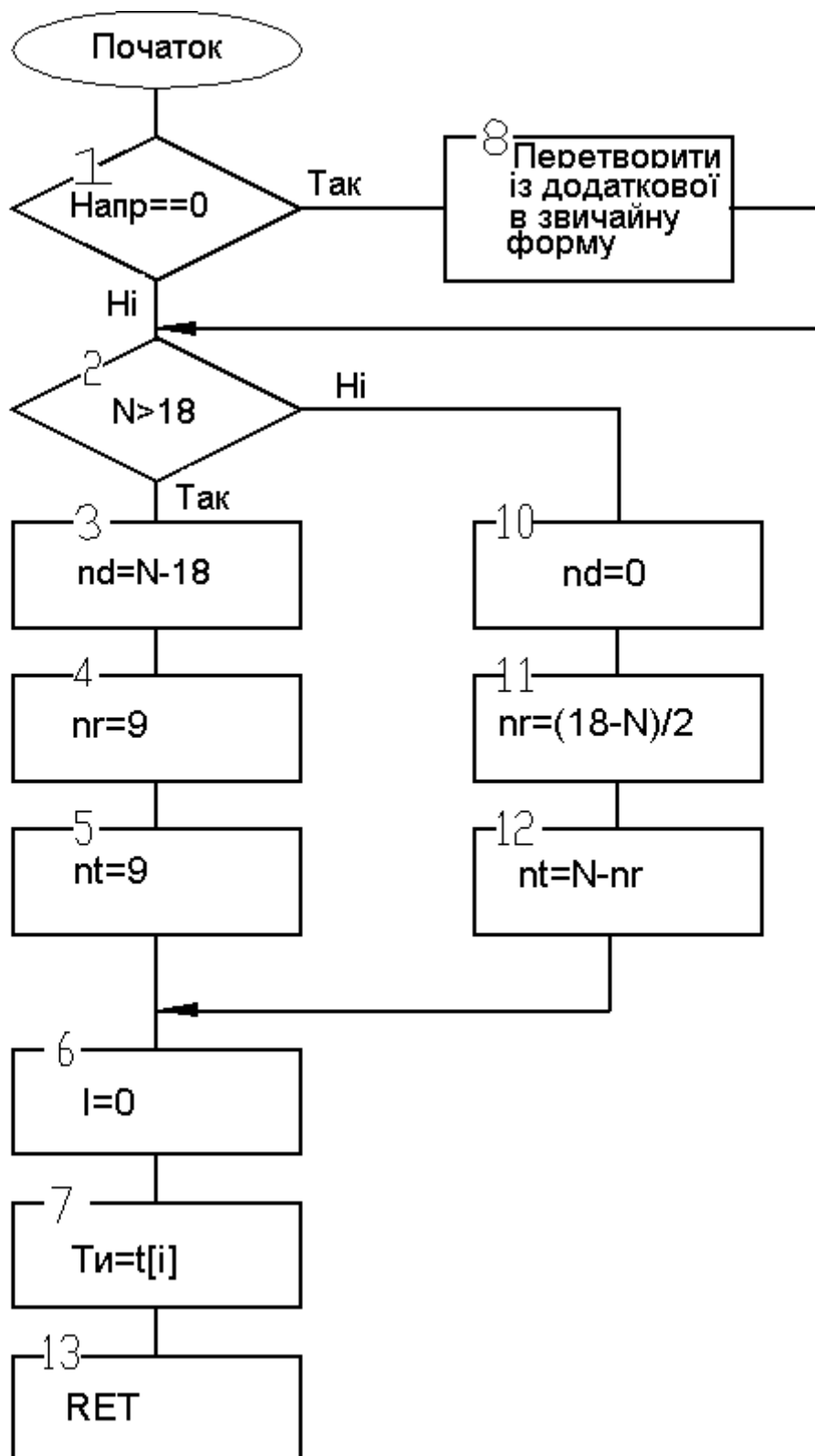


Рисунок 7.8 – Підпрограма ініціалізації руху

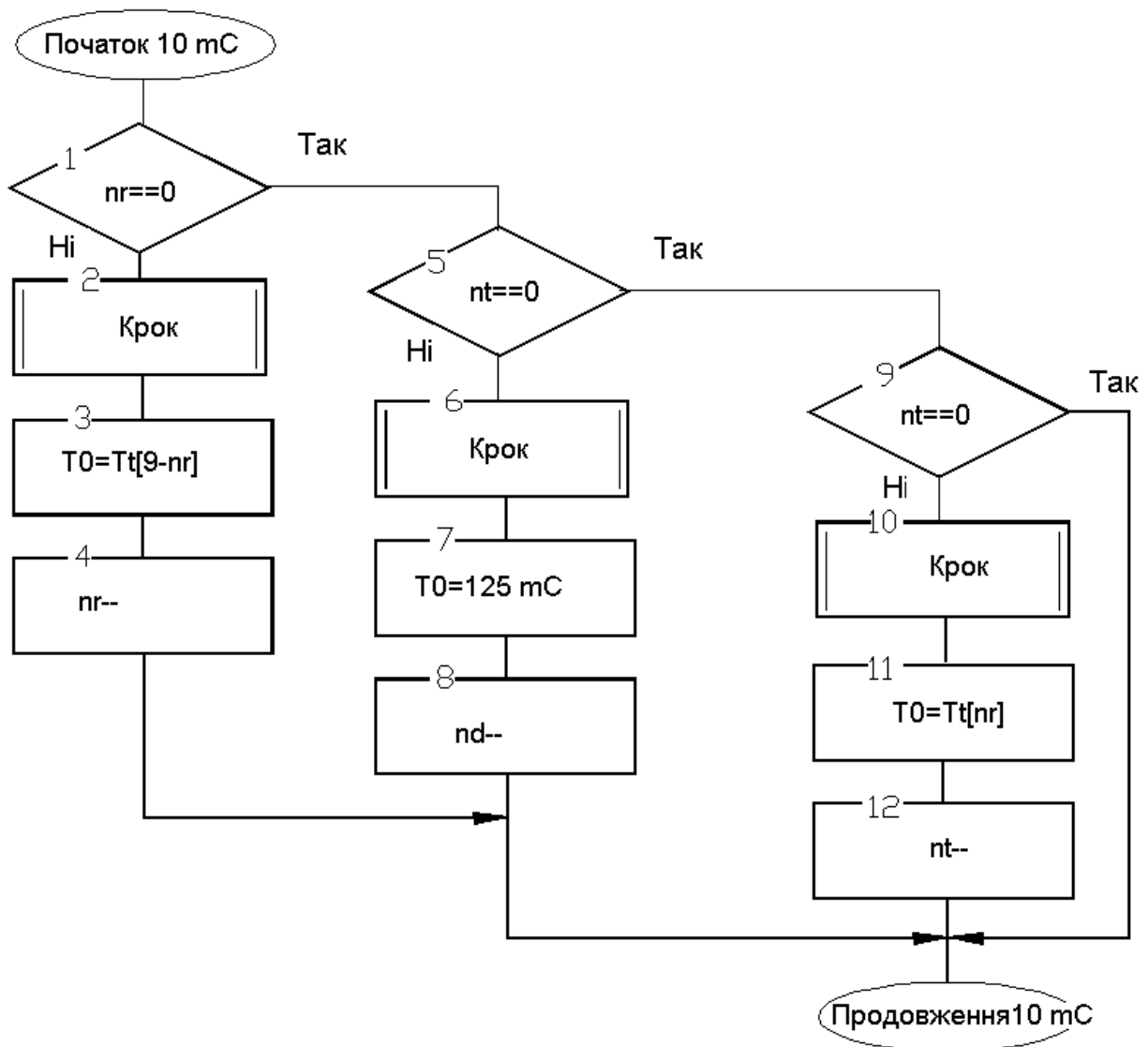


Рисунок 7.9 – Алгоритм відпрацювання руху крокового привода

Програма керування кроковим двигуном наведена нижче. Підпрограма Step_ini формує вихідні дані для прямого й реверсивного керування кроковим 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

fr EQU 08h; прапор непарної кількості кроків розгону

гальмування

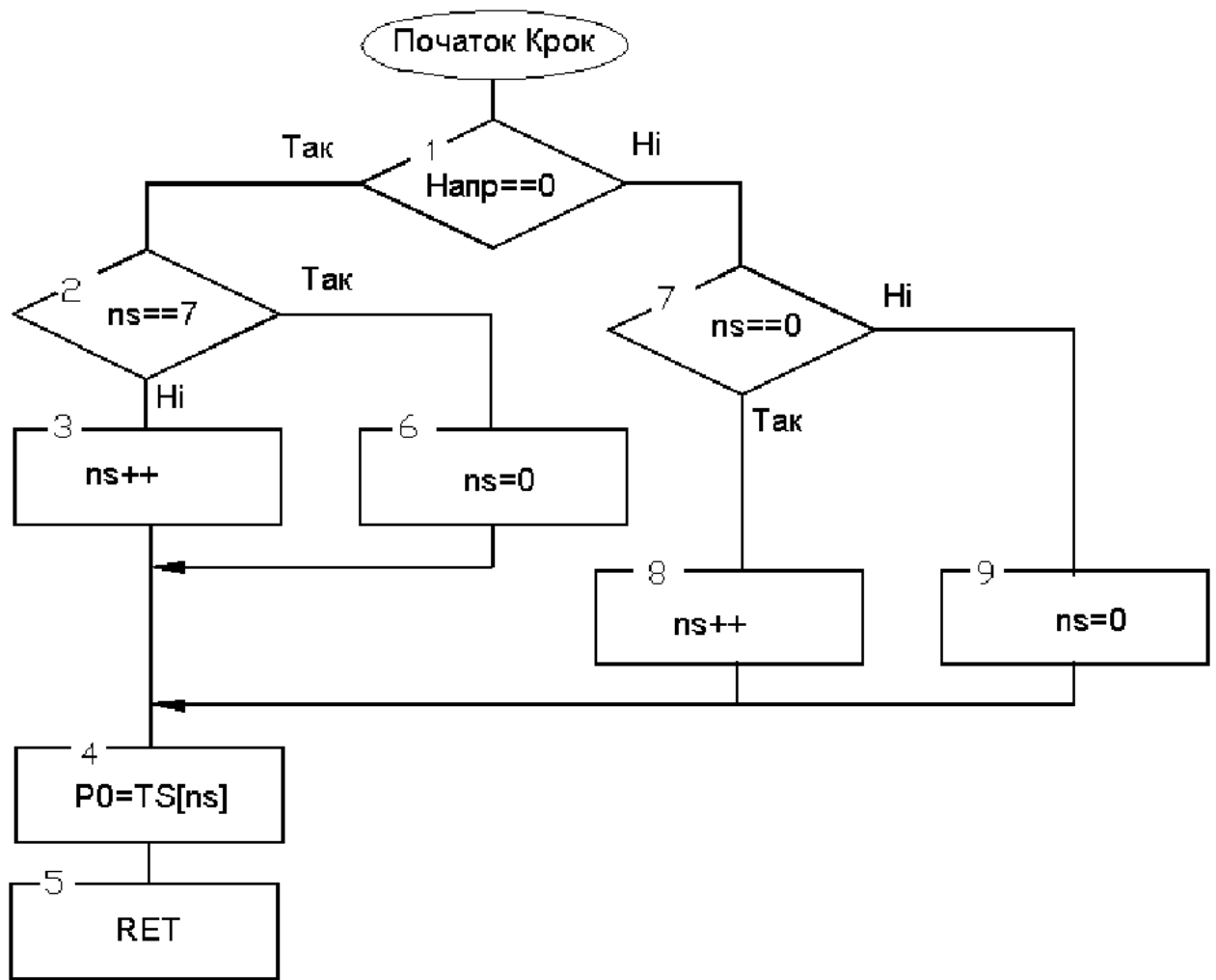


Рисунок 7.10 – Підпрограма формування сигналів керування мостами

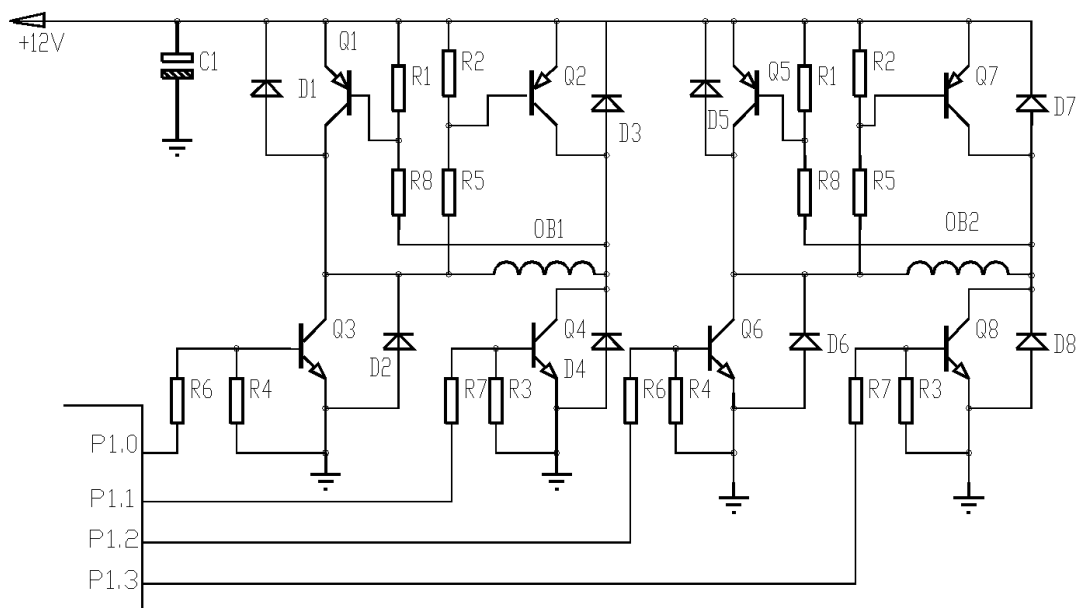


Рисунок 7.11 – Підключення крокового двигуна до мікроконтролера

Наведений фрагмент є описом призначення символічних 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; настроювання 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 настроюється на режим 1 з періодом переповнення 10 мС, в 0 встановлюються порти виводу і початкові значення змінних.

```

    . . .
        mov N,#8h
        lcall Step_ini
    . . .

```

Вихідні дані, напрямки руху й кількість кроків передаються в підпрограму через комірку 20 h, символічне 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; збереження прапора 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; устанавлюємо ознаку, 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 ; устанавлює 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
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
pop PSW
pop Acc
reti

```

m13:
ex:

В оброблювач переривання програма заходить кожні 10 мС, змінна tim є програмним таймером тривалості імпульсу кроку. При розгоні й 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 ps зберігає положення ротора. Оскільки крокові двигуни використовуються без датчиків зворотного зв'язку за положенням, алгоритм задання початкової системи координат виглядає в такий спосіб. У системі, як правило, є дискретний кінцевий датчик початку координат. При увімкненні живлення, якщо датчик не спрацював, кроковий двигун включається на рух до початку координат на мінімальній швидкості для перших 8 кроків. Таким чином, тривалість імпульсу дорівнює першому значенню з таблиці *gazgon*. Це дозволяє засинхронізувати положення вектора поля й ротора. При спрацьовуванні кінцевого вимикача процес зупиняється й фіксується початок координат. Мікропроцесорна система керування дозволяє максимально використовувати можливості крокового приводу. Вона забезпечує гнучке керування, особливо в перехідних режимах, зберігаючи точність позиціонування й виконуючи будь-які програми відпрацьовування напрямку й швидкості ходу.

Контрольні запитання?

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 провідникові передається окремий біт, у такий спосіб 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 невеликій 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, проводиться за дво- або трипровідниковою схемою. При цьому використовується так званий спосіб тимчасового ущільнення каналу. Його суть показано на рис. 8.1.

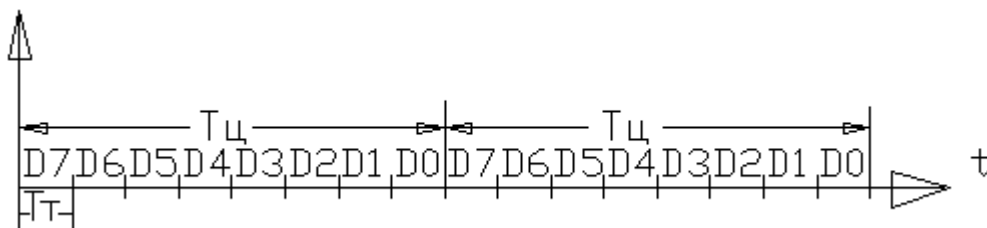


Рисунок 8.1 – Часове ущільнення каналу зв'язку

У цьому випадку інформація передається по двох проводах – 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 T_t . Таким чином, відбувається перетворення даних з паралельного формату в послідовний, яке здійснюється за

допомогою зсувного регістру й тактового генератора. Структурна схема цього пристрою показана на рис. 8.2.

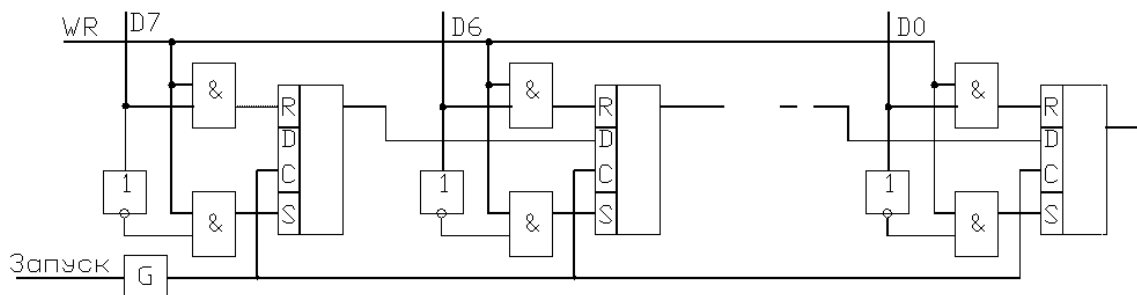


Рисунок 8.2 – Схема зсувного регістру з паралельним занесенням даних

Він складається зі зсувного регістру, виконаного на D-тригерах, схем занесення інформації в тригери (D0-D7) (паралельний формат) й генератора синхроімпульсів G.

Після занесення інформації в тригери запускається генератор. На останньому тригері послідовно в часі формується значення D0, за другим імпульсом генератора – D1 і т.п.

Час утримання значень бітів на виході визначається періодом генератора.

Під час такого способу передачі виникають наступні 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.

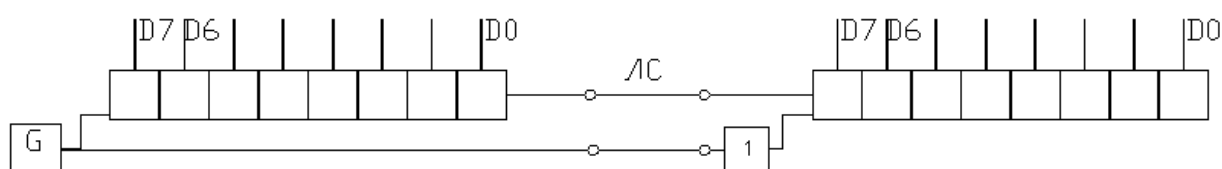


Рисунок 8.3 – Передача інформації з додатковим сигналом синхронізації

Такий спосіб дозволяє передавати інформацію з максимальними швидкостями передачі. 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.п.

Графік часового аналізу цього способу передачі інформації зображений на рис. 8.4.

Інвертування тактового сигналу приймача приводить до того, що підтвердження значення в ЛС проводиться у середині бітового інтервалу, при значенні сигналу, що вже встановилось.

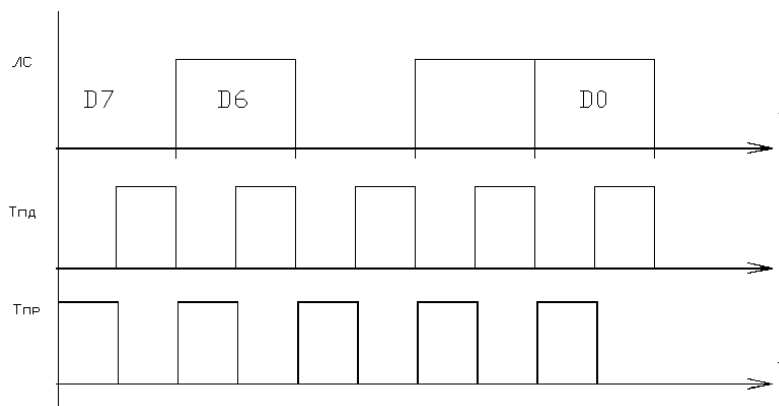


Рисунок 8.4 – Графік часового аналізу обміну інформацією при використанні додаткового тактового сигналу

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.

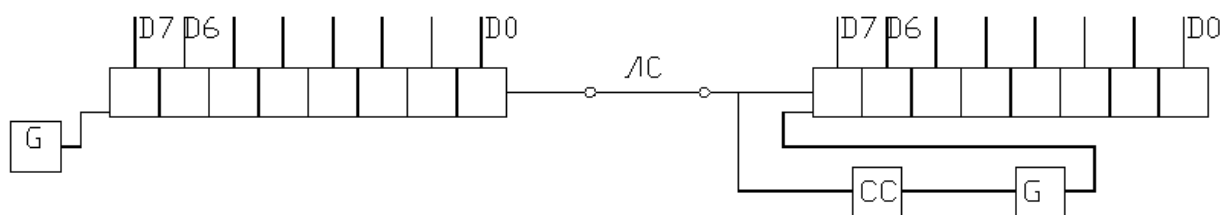


Рисунок 8.5 – Обмін інформацією з незалежним тактуванням

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 регістри, що приймають й передають, буфери приймача й передавача, а також схеми тактування передавача й синхронізації

приймача. Наявність додаткових буферів дозволяє суміщати операції зчитування вже прийнятого байта з прийманням чергового.

Послідовний інтерфейс може працювати за чотирма режимами і настраюється на відповідний за допомогою запису керуючого слова в регістрі SCON. Призначення бітів регістру керування наведено в табл. 3.2.

Для обміну інформацією із зовнішніми пристроями УСАПП використовують сигнали Rxd і Txd – альтернативне призначення бітів порту 3 розряди 0 і 1 відповідно. Залежно від режиму роботи УСАПП призначення цих розрядів різне.

Послідовний інтерфейс має символічне ім'я SBUF. Для передачі байта інформації необхідно передати цей байт в буфер прийомо-передавача послідовного інтерфейсу командою MOV SBUF, A; (Передача вмісту акумулятора в буфер послідовного обміну.). Після цього автоматично байт пересилається в зсувний регістр передавача і відразу починає побітно виштовхувати в лінію зв'язку. Після передачі останнього біта встановлюється в одиницю тригер TI, «запит на переривання передавача», який підключений до системи переривання мікроконтролера та інформує систему про те, що буфер передавача порожній.

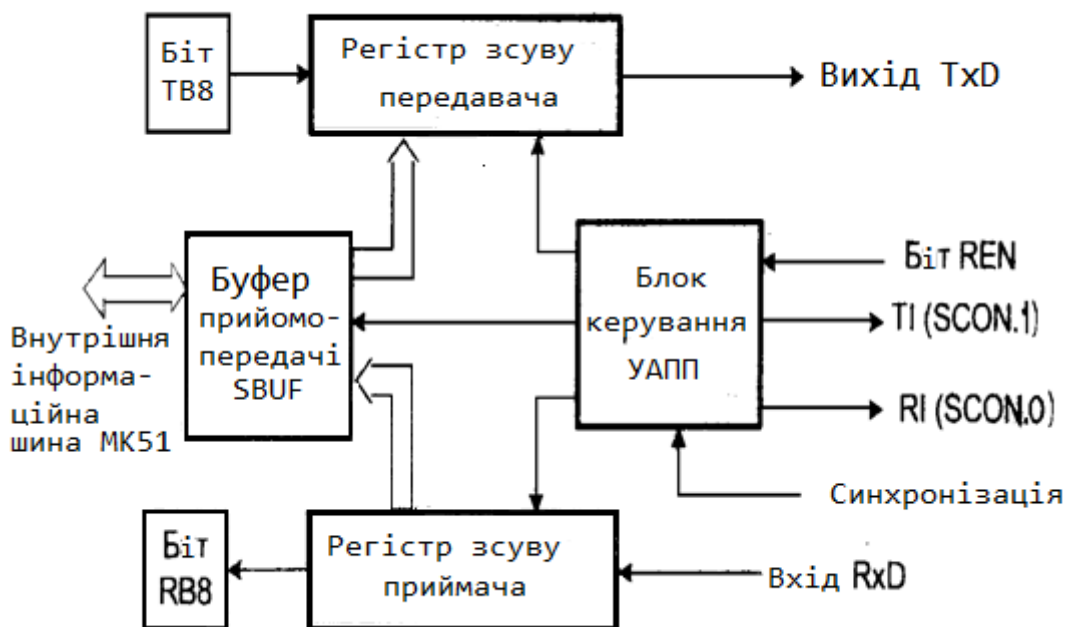


Рисунок 8.6 – Структурна схема УАПП

Приєм інформації з лінії зв'язку здійснюється побітно через вхід RxD. Після прийому останнього біта послілки встановлюється в одиницю тригер «готовність приймача» RI, інформує систему про те, що буфер приймача

повний і інформацію з буфера необхідно забрати. Якщо байт з буфера не буде прочитаний він буде втрачений.

Організація регістру керування SCON

Символічне позначення	Позиційне позначення	Назва й призначення бітів
SM0	SCON.7	Біти керування режимом послідовного порту
SM1	SCON.6	Програмно встановлюються й 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 8-м бітом
REN	SCON.4	Біт керування дозволом приймання. Програмно встановлюється й скидається для дозволу (заборони) приймання послідовних даних
TB8	SCON.3	8-й біт при передачі даних. Програмно встановлюється й скидається для задання 8-го біта в режимі 9-бітного послідовного порту
RB8	SCON.2	8-й біт при прийманні даних. Апаратно встановлюється й скидається залежно від значення 8-го біта й режимі 9-бітного послідовного порту
TI	SCON.1	Прапор переривання при закінченні передачі. Установлюється апаратно після передачі байта, скидається програмно після обслуговування переривання
R1	SCON.0	Прапор переривання при закінченні приймання. Установлюється апаратно після приймання байта, скидається програмно після обслуговування переривання

8.3. Режим роботи 0

Режим 0 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 й приймання 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 потрібно, передає запит на приймання 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 й дорівнює $1/12 F_{osc}$. Для ініціювання передачі необхідно занести до регістра SCON у розряди MS0, MS1 нулі й записати в SBUF перший байт повідомлення. Наступний байт може бути записано в SBUF тільки після передачі байта. Прапор 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 прапор необхідно скинути програмно. Допускається опитування прапора без використання переривання. Після встановлення прапора TI в SBUF можна записати наступний біт повідомлення. Якщо було передано останній байт повідомлення, досить скинути прапор TI і більше він формуватися не буде. Для поновлення передачі необхідно в SBUF записати перший байт повідомлення й процес передачі повторюється.

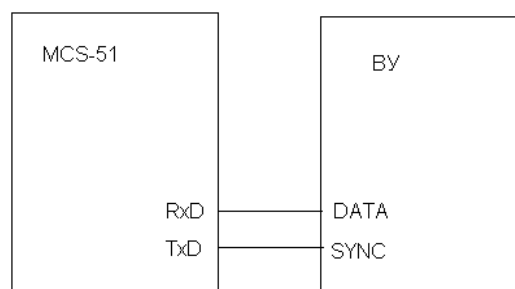


Рисунок 8.7 – Схема підключення периферійних пристроїв до контролера в режимі 0

Для переведення УСАПП у режим приймання потрібно встановити в SCON.4 одиницю.

Режим 0 призначено для обміну інформацією із пристроями, що розташовані на платі контролера, при дотриманні правил перешкодозахисту й мінімальній довжині провідників. Він забезпечує максимальну швидкість обміну по послідовному каналу.

8.4. Режим роботи 1

Режим асинхронного обміну застосовується частіше 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 1 зображено на рис. 8.8. У посилці передаються 8-бітові дані, які доповнюються 2 службовими бітами: старт-біт (передається на початку) і стоп-біт (передається наприкінці повідомлення).

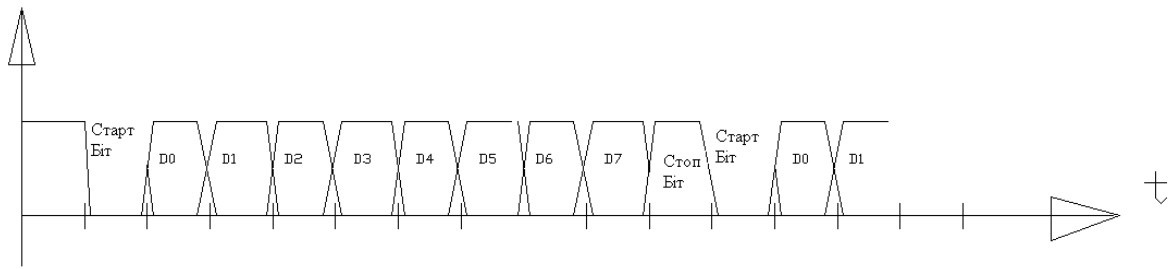


Рисунок 8.8 – Структура байтового повідомлення в режимі 1

Передача даних проводиться через вихід TxD. Прийом даних – через вхід RxD.

Якщо передача в режимі 1 відсутня, то вихід TxD перебуває в стані 1. Після занесення даних в SBUF передавача, до лінії зв'язку передається старт-біт, за ним 8 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, у повідомленні можна виділити початок і кінець передачі байта.

Структурну схему УСАПП у режимі 1 зображено на рис. 8.9.

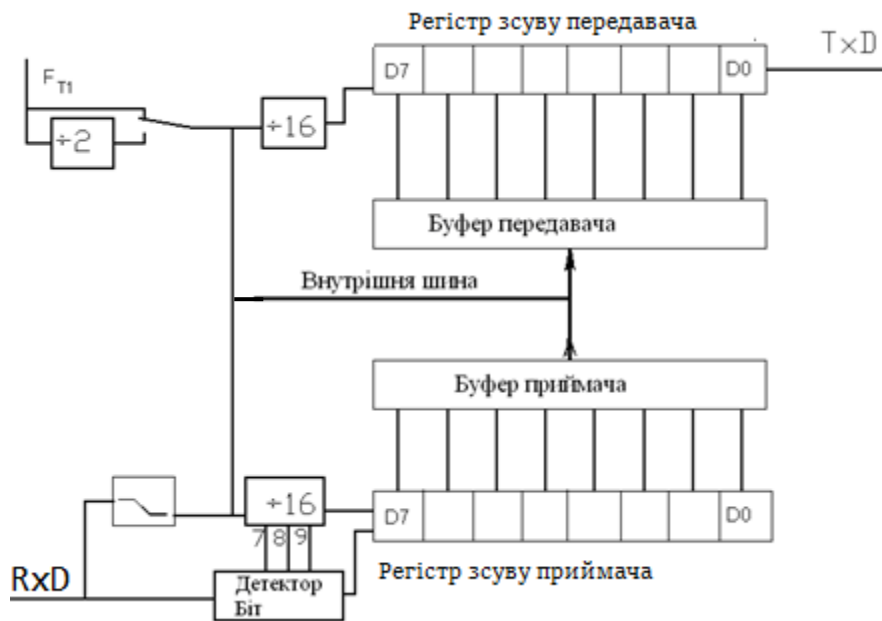


Рисунок 8.9 – Структурна схема УСАПП у режимі 1

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 ТН1.

На тактування зсувного регістра передавача частота з виходу таймера надходить через дільник на 16.

У режимі 1 відсутній сигнал зовнішнього тактування приймача. Імпульси синхронізації приймача формуються з тактової частоти таймера 1 спільно зі схемами детектора спаду, детектора бітів і керованого дільника на 16.

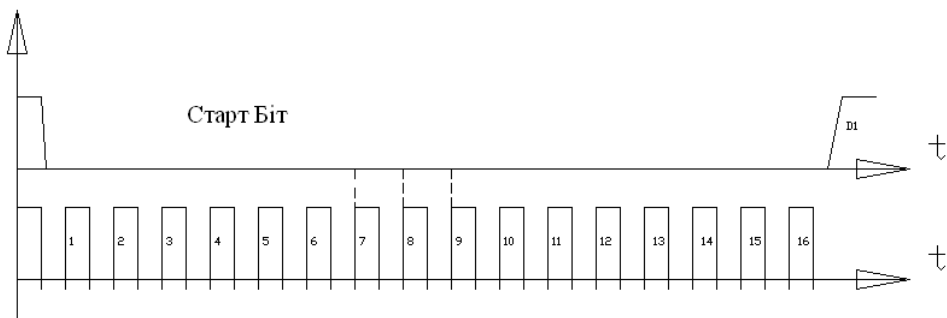


Рисунок 8.10 – Перевірка старт-біта в середині інтервалу для захисту від імпульсної перешкоди

Приймання починається під час виявлення детектором зміни сигналу на вході приймача з 1 в 0. При цьому скидається в 0 дільник на 16 приймача. На 7-му, 8-му і 9-му такті дільника опитується значення входу приймача RxD (рис. 8.10) і, якщо за трьома вимірами більше одиниць, старт-біт не зачитується, йде його подальший пошук. Таким чином, відбувається підстроювання фази тактування приймача за сигналами з лінії зв'язку. Вона забезпечує підтвердження значення прийнятих даних у середині тактового інтервалу. А мажоритарне опитування (за трьома значеннями на інтервалі) – захист від імпульсних перешкод.

При прийманні, крім контролю старт-біта, здійснюється контроль стоп-біта. Якщо він не дорівнює 1 – байт не зачитується й дані губляться.

Як правило, обмін даними між контролерами проводиться декількома байтами. Цю послідовність байтів називають повідомленням. Для визначення початку повідомлення використовують або не застосовуване в посилці оригінальне значення першого байта, або так званий тайм-аут, коли протягом певного часу передача даних перед передачею чергового повідомлення не відбувається.

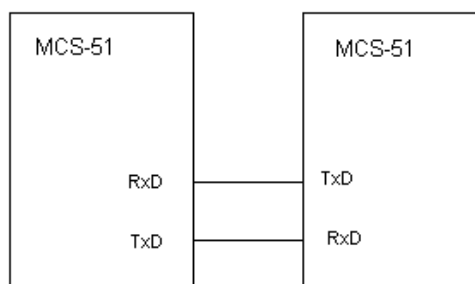


Рисунок 8.11 – Схема підключення двох контролерів у режимі 1

Як правило, обмін між пристроями в послідовному форматі відбувається за алгоритмом запит – відповідь, або передача – підтвердження. При цьому ініціатором обміну є провідний контролер або Master. Другий контролер називають веденим або Slave. Завдання на програму обміну інформацією між контролерами можна сформулювати в такий спосіб: необхідно передати через

послідовний інтерфейс масив даних, розміщених з адреси Adr1, довжиною в L байт; прийняти у відповідь масив довжиною Lotv і записати його з адреси Adr2.

На рис. 8.11 наведено схему алгоритму й 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 приймання
      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 приймання
m2r:  ljmp m2r
      END
```

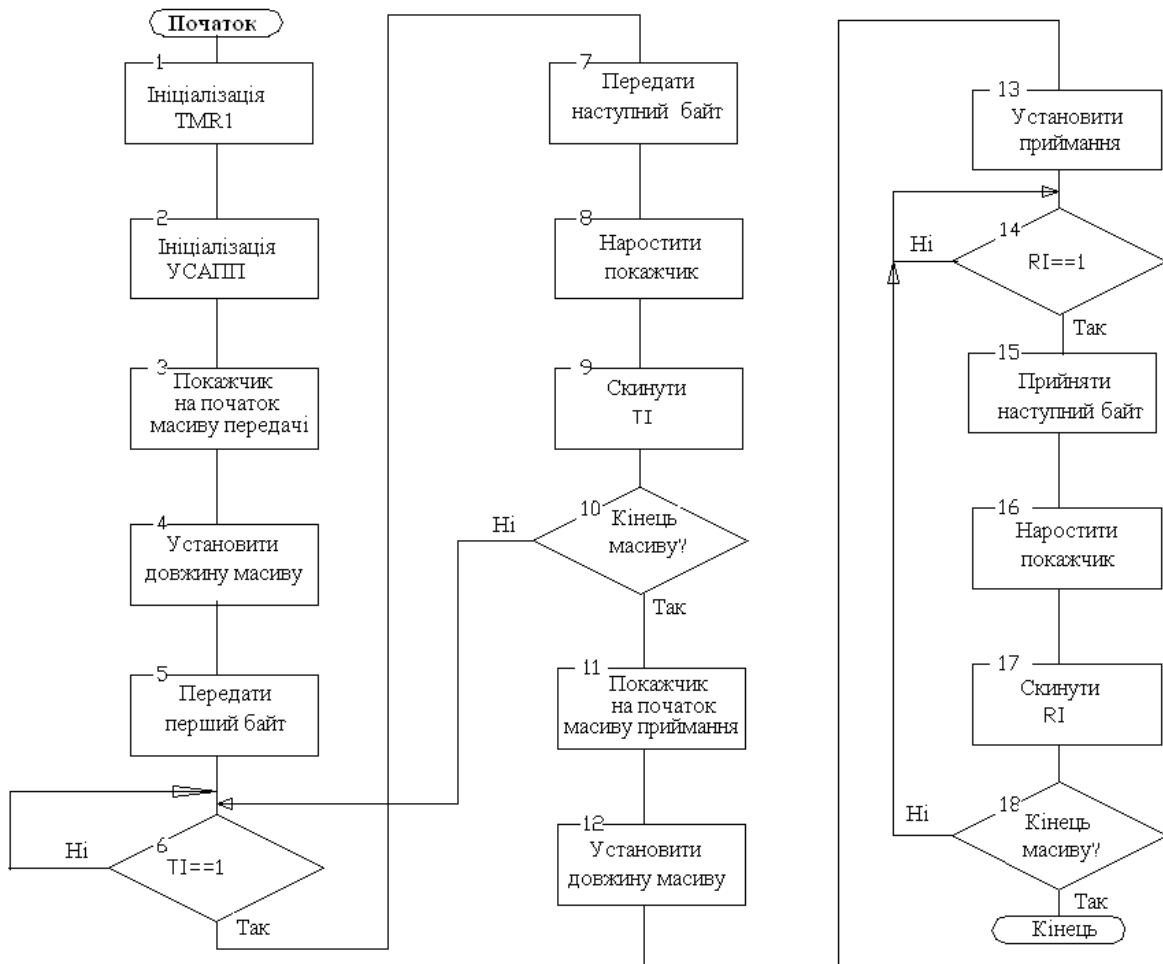


Рисунок 8.12 – Блок-схема алгоритму обміну повідомленнями між контролерами в режимі 1

На початку програми задається другий режим таймера 1, а також заноситься константа в ТН1, що забезпечує швидкість обміну 19,2 кбіт/с.

Потім настроюється УСАПП на перший режим роботи, а також заноситься перший байт масиву в SBUFF. Контроль передачі байта здійснюється шляхом програмного опитування біта ТІ в SCON. Після закінчення передачі чергового байта цей біт установлюється в одиницю. В SBUFF заноситься значення наступного елемента масиву, нарощується показчик масиву, скидається в 0 біт ТІ, декрементується й перевіряється на нуль лічильник елементів масиву. Якщо останній елемент масиву передано, програма переходить на приймання відповіді на запит, якщо він був не останнім, то вертається на контроль біта ТІ. Після закінчення передачі масиву УСАПП переводиться в режим приймання, для цього біт REN (SCON.4) установлюється в 1. Потім формується значення показчика масиву для прийому, установлюється його довжина й відбувається опитування й контроль біта RI (SCON.0). Одиниця в цьому біті означає, що приймач закінчив приймання чергового байта, тепер необхідно вчитати буфер, щоб звільнити його для

приймання наступного байта. Після зчитування з SBUFF значення заноситься в масив прийому, наращується покажчик елементів масиву, скидається біт RI, декрементується й перевіряється на нуль лічильник елементів масиву. Якщо останній байт масиву прийнято, біт REN скидається в 0, приймання припиняється й програма переходить на продовження виконання завдання керування.

8.5. Режим роботи 2

У цьому режимі, крім старт- і стоп-біта та 8 бітів даних, додається дев'ятий біт, значення якого можна задавати програмно. Він використовується для підвищення вірогідності приймання інформації (для передачі значення біта контролю парності), або для передачі ознаки початку повідомлення в багатоконтролерних мережах, де першим байтом передається номер адресата. Частота приймання/передачі може дорівнювати 1/32 або 1/64 частоти резонатора залежно від значення SMOD (PCON.7).

8.6. Режим роботи 3

Режим 3 збігається з режимом 2 за форматом повідомлення але швидкість обміну може задаватися таймером 1, як і в режимі 1.

Контрольні запитання

1. Яку кількість бітів містить посилка УАПП МК51 в режимі 1?
2. Яку кількість бітів містить посилка УАПП МК51 в режимі 2?
3. Який біт регістра SCON УАПП МК51 дозволяє роботу контролера в мережевому режимі?
4. Який сигнал УАПП МК51 використовується в системі переривань при передачі інформації?
5. Який сигнал УАПП МК51 використовується в системі переривань при прийманні інформації?
6. Скільки режимів роботи має послідовний порт МК51?
7. Яку кількість бітів містить посилка УАПП МК51 в режимі 0?
8. Який перехід в лінії зв'язку характеризує початок посилки?
9. Який перехід в лінії зв'язку характеризує завершення посилки?
10. Який біт регістра SCON УАПП МК51 дозволяє приймання інформації?

9. ПОСЛІДОВНІ ШИННІ СИСТЕМИ (КАНАЛИ ЗВ'ЯЗКУ)

9.1. Загальні поняття про електронні шинні системи

В 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.

В 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 безпроводне з'єднання.

В цьому розділі мова буде йти про неоптичні лінії передачі даних.

Таким чином, згідно з визначенням, до однієї шини можуть бути приєднані більш ніж два об'єкти, тобто існує декілька джерел передачі даних. Звідси одразу стає зрозуміло, що однією з основних особливостей шини має бути обробка конфлікту на лінії зв'язку або ж необхідні стратегії, які зможуть з самого початку ліквідувати подібні конфлікти під час передачі інформації.

Електронні шинні системи поділяються на паралельні та послідовні.

В паралельних шинних з'єднаннях перенесення даних відбувається паралельно, тобто одночасно по декількома паралельних ліній зв'язку. Паралельні шинні з'єднання часто розглядають як шини задньої стінки в комп'ютерних чи мікропроцесорних системах для з'єднання декількох електронних блоків, які знаходяться на різних платах.

В 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;
- коротка довжина шин;
- 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 в формі один за одним переданих бітів через один канал зв'язку.

До характерних особливостей послідовних шин відносяться:

- більш низька у порівнянні з паралельними шинами такої самої тактової частоти швидкість обміну даними;
- більш низькі 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», властивий послідовний обмін більш високої надійності, особливо це стосується електромагнітної сумісності для жорстких умов – використання в промисловості або в транспортних засобах. Перш за все, за допомогою польових шин було створено велику кількість досить різноманітних систем, що поширені також дуже по-різному. В якості приклада можна привести одну з найпоширеніших систем CAN (Controller Area Network). Це система, що 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 в сфері офісної техніки, не дуже велика. До них відносять USB (Universal Serial Bus), Ethernet , а також IEEE 1394 «Firewire».

На відміну від офісних, існує велика кількість різноманітних систем в сфері промисловості. В цьому випадку потрібно розрізняти так звані патентовані та «відкриті» шини. Патентовані системи мають різноманітні

характеристики, в залежності 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 звані 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 йшла мова 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 7 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 4, 5 і 6 в шинних системах, що розглядаються, виражені слабо і через свою простоту до уваги не приймаються. Рівень 7 (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 в транспортних засобах вже було показано, що в шинних системах різні рівні можуть бути стандартизованими чи 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 приймати в ній активну участь. Функціональна сумісність потребує загальних специфікацій значення, тобто інтерпретації переданих даних. В 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 розроблених 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 в сфері промисловості, перерахуємо 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.

Наведемо декілька прикладів промислових систем Ethernet, тобто систем, що 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 є найбільш відповідною до тих чи інших вимог, та системи, які мають розширення. У [GRU2001] детально розібрано засіб для порівняння, оцінки та вибору послідовної шини.

В наступних часткових розділах коротко наведені деякі приклади різноманітних систем зв'язку, які, без сумніву, мають відношення до мікрокомп'ютерів чи мікроконтролерів.

В 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 зв'язкового контролера та трансивера. Шинні контролери генерують та обробляють логічні фронти імпульсу та послідовність бітів, в той час як трансивери в якості безпосереднього інтерфейсу до комунікаційних ліній надають та сприймають сигнальні фронти імпульсу та рівні біта безпосередньо як сигнали на лінії.

З одного боку зв'язкові контролери можуть приєднуватись 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 доступні багато модифікацій родини мікропроцесорів 8051 з вже вбудованими комунікаційними контролерами, що відіграють особливу роль в цій книзі.

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 двокрапкове 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 демонструється 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 діапазоні +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.

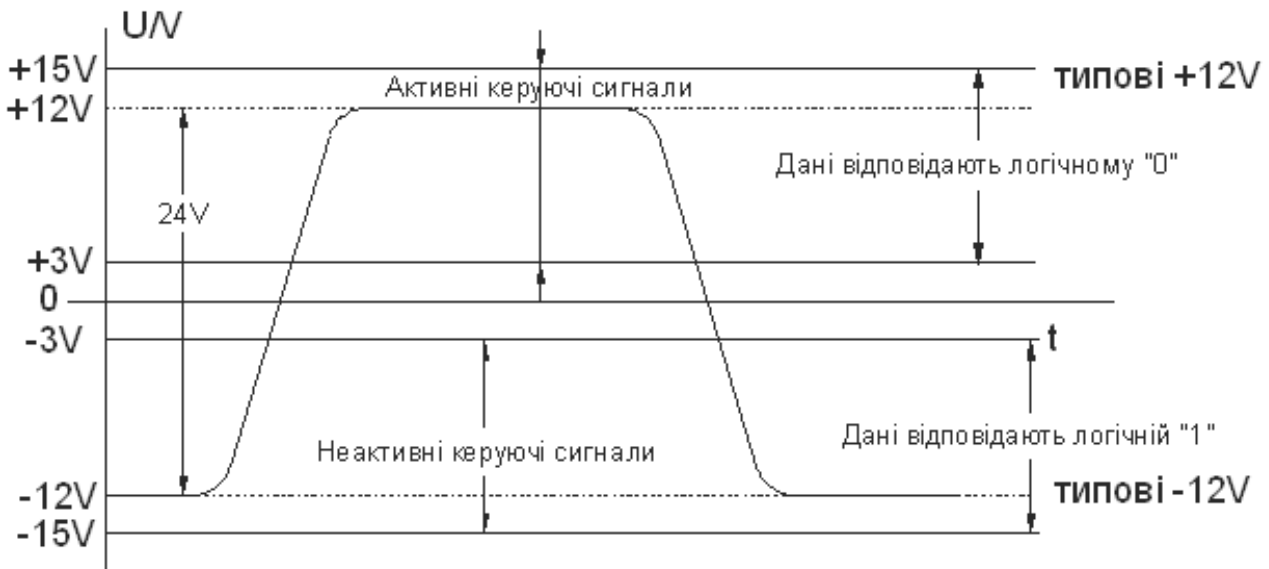


Рисунок 9.1 – Визначення логічного рівня RS 232 для ліній передачі даних та керуючих сигналів

9.3.1.2. Розведення контактів та сигналів

В якості роз'ємного з'єднання RS 232 використовують, наприклад, штекерний підроз'єм D. Його вигляд згори зображено на рис. 9.2.

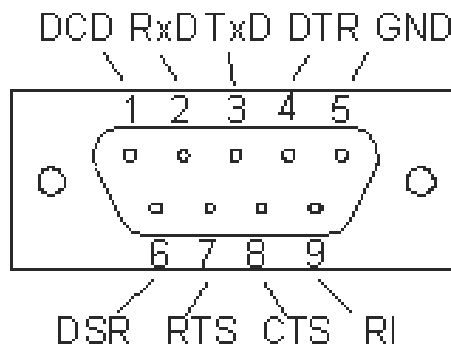


Рисунок 9.2 – Дев'ятиконтактний штекерний підроз'єм D інтерфейсу з призначенням виводів для інтерфейсу RS 232

Магістралі, що використовуються для передачі даних, можуть бути змонтовані у відповідності до розпайки виводів, що демонструється вище (рис. 9.2).

Функціональні характеристики інтерфейсу RS 232 можна виділити з наведених нижче ліній передач (рис. 9.3). Тут демонструються найбільш важливі лінії передач.

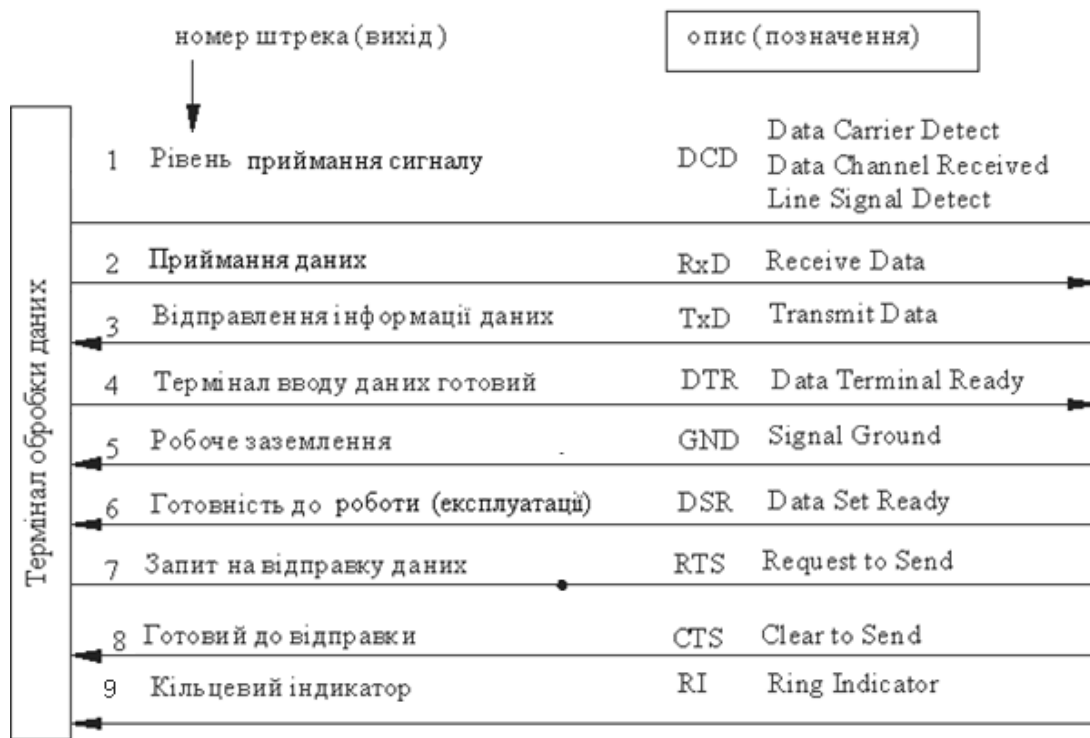


Рисунок 9.3 – Призначення сигналів інтерфейсу RS 232

На рис. 9.4 зображено просте двопровідне з'єднання між двома терміналами на основі RS 232.

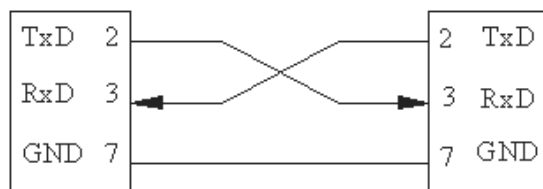


Рисунок 9.4 – Мінімальна конфігурація з'єднання RS232 між двома терміналами

9.3.1.3. Довжина 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 кабель RS232 шляхом зниження спотворення форми імпульсу та значного зниження паразитних зв'язків.

9.3.2. З'єднання RS485

З'єднання за допомогою EIA RS485 дає можливість підключення до шини, 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 двопровідникової лінії передач, клеми А і В, це означає, що вони ввімкнені паралельно (рис. 9.5).

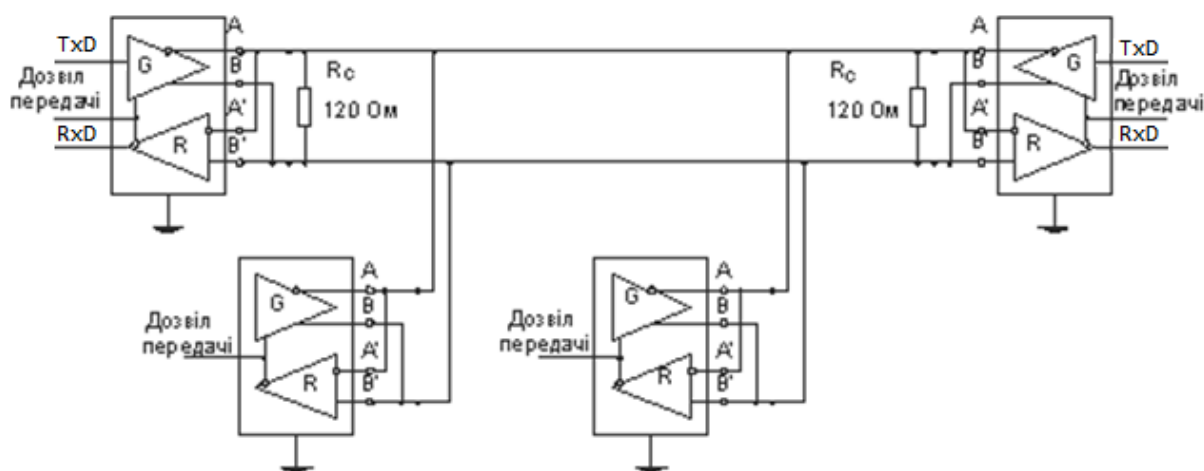


Рисунок 9.5 – Шинна система з інтерфейсом RS 485

В цьому інтерфейсі можливо підключення 32 користувачів (як передавачів, так і приймачів даних) до одного двопровідникового шинного кабелю з симетричним перенесенням даних. Для забезпечення симетрії провідники закручуються з фіксованим кроком. Такий кабель ще називають «вита пара». В такому кабелю електромагнітна перешкода буде наводити сигнал перешкоди однаковий для провідника А і провідника В. При цьому напруга на провіднику А буде рівна $U_a' = U_a + U_{\Pi}$, на провіднику В $U_b' = U_b + U_{\Pi}$. Оскільки, на вході приймача завжди діє різниця напруг U_a і U_b , корисний сигнал на вході приймача визначається, як $U_{ex} = U_a' - U_b' = U_a + U_{\Pi} - U_b - U_{\Pi} = U_a - U_b$. Завдяки симетричній конструкції інтерфейсу досягається високий рівень захищеності від перешкод.

Часто для з'єднання RS 485 використовується 9-контактний штекерний роз'єм. На рис. 9.6 зображено розташування виводів та призначення сигналів такого інтерфейсу.

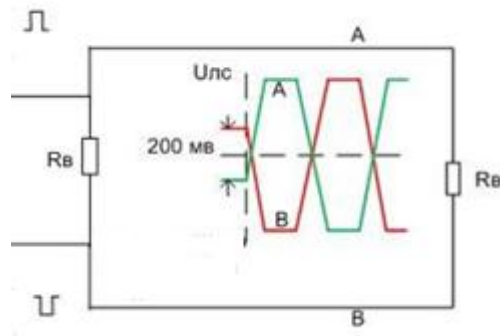
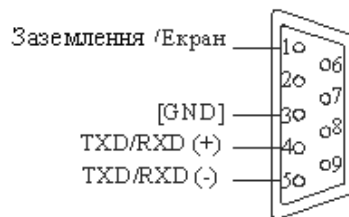


Рисунок 9.6 – Графік зміни сигналів при передачі 0 та 1



Вивід	Функція
1	Schutzerde/Schirm => Заземлення Екран
2	-
3	GND/Signalmasse => 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. Основи

Відтоді, як з 1980 року 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. В 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 використовуватись майже всіма заводами-виробниками транспортних засобів. В автоматичі, 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 7 з частково власними групами користувачів в автоматичі, до того ж в 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 засновані 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 мільярда CAN-вузлів.

9.3.3.2. Фізичні характеристики

CAN має лінійну архітектуру. Як середовище передачі даних застосовується вита пара з електричними рівнями, що відповідають ISO 11898. Згідно з цим,

максимальна протяжність складає 40 метрів при максимальній швидкості передачі в 1 Мбіт/с. Однак при знижені швидкості передачі даних до 50 Кбіт/с мережа може мати довжину до 1000 метрів. Принцип підключення користувачів до шинної лінії за допомогою різниці напруг є аналогічним вже розглянутим вище принципам функціонування інтерфейсу RS485, які зображені на рис. 9.6.

9.3.3.3. Характеристики протоколу рівня передачі бітів

На рівні передачі бітів CAN використовується метод CSMA/CA, основи якого вже було розглянуто на прикладі шини I²C (розділ 9.3.3.3 «Арбітраж доступу»).

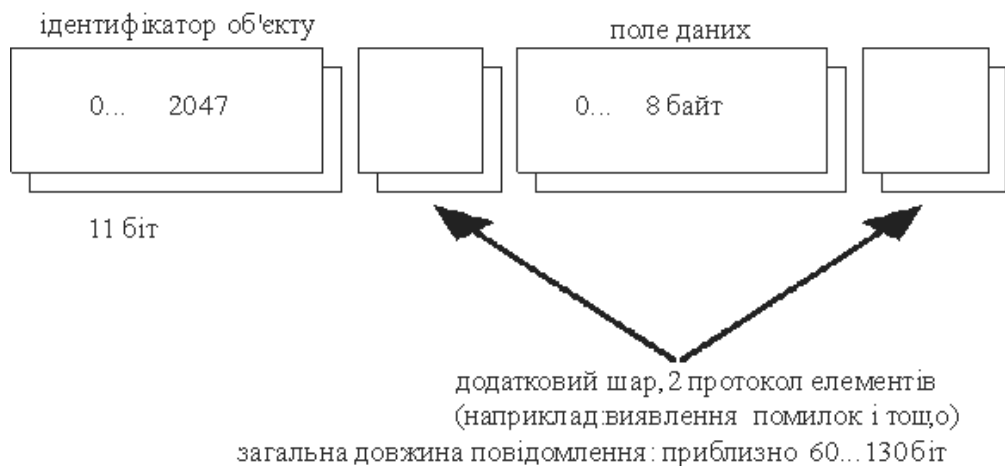


Рисунок 9.8 – Структура телеграми стандарту CAN

Поле даних може мати різну довжину від 0 до 64 бітів і мати інформацію, призначену для передачі.

Поле «Ідентифікатор об'єкта» (11 біт) має декілька функцій. Воно служить як мітка для телеграми. Через це воно і використовується для регулювання доступу до шини, тобто для ліквідації колізій. Дане поле надає сигналу певний пріоритет. Останній, в свою чергу, тим вище, чим менше сигналів логічної 1 має поле. Таким чином, найвищий пріоритет у сигналі з ідентифікатором об'єкта 0. CAN – це мережа з кількома ведучими пристроями, в якій сигнали різного пріоритету передаються широкомовлено. Її ключовий концепт – це спосіб розподілу доступу до шини, заснований на попередженні конфліктів (CSMA/CA). Далі ми розглянемо принцип функціонування розподілу доступу до шини (рис. 9.9).

Якщо шина вільна, будь-яка станція може розпочинати передачу сигналу, в якому вона надсилає заголовок телеграми. Він містить ідентифікатор, спеціально призначений для даного сигналу. Процес арбітражу тепер залежить

від цього ідентифікатора. Під час арбітражу доступу рецесивні біти ідентифікації («1») переписуються від доміантних бітів ідентифікації («0») іншого вузла, який в той самий час намагається отримати доступ до шини. Кожен передавач даних одночасно чекає сигналу від шини і одразу ж перемикається з функції передачі на отримання даних, як тільки зчитує доміантний біт, в той час як він надсилає рецесивний біт. Вузол відправлення інформації з найбільш високим ідентифікатором пріоритету виграє арбітраж і таким чином отримує доступ до шини. Його сигнал відправляється без втрати часу. Таким чином, арбітраж доступу не є деструктивним. Після програного арбітражу автоматично відбувається нова спроба доступу до шини.

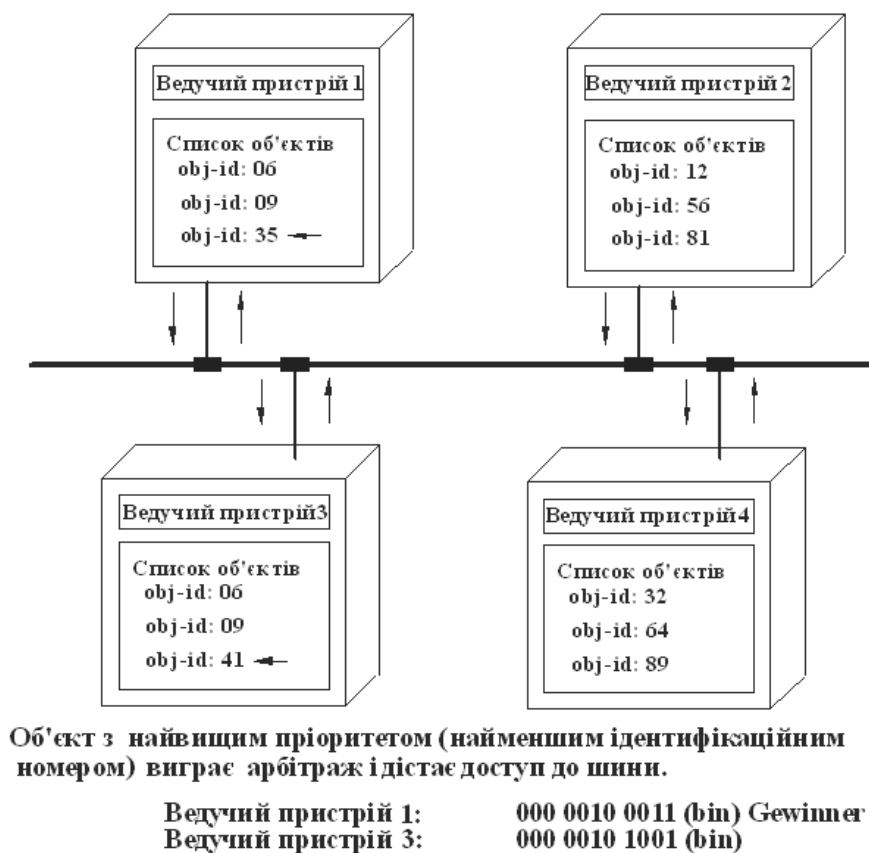


Рисунок 9.9 – Процес доступу до загальної шини CAN і недеструктивний арбітраж

Це означає, що сигнал з найвищим пріоритетом затримується тільки доти, доки сигнал, що відправляється в даний момент, не буде доставлено. Таким чином, для цього сигналу може бути задано максимальний час пошуку. Однак, доступ до шини сигналів з більш низьким пріоритетом при дуже високому її завантаженні може бути на деякий час ускладнено. Таким чином, CAN розподіляє доступну потужність передачі сигналів відповідно до їх пріоритету, а саме незалежно від завантаженості шини.

Принцип арбітражу доступу CAN передбачає, що максимальний час проходження сигналу в мережі значно менший, ніж час біта, тому що за час передачі біта кожен вузол має встигнути передати свою реакцію. Тому максимальна протяжність мережі тут залежить в більшій мірі від обраної швидкості передачі даних, ніж в інших шинних системах, в яких пригнічення сигналу частіше всього є обмежуючим фактором.

CAN має високу надійність передачі інформації. Арбітраж, фільтрація сигналів, виявлення та попередження помилок та підтвердження отримання сигналу реалізовані в апаратній частині. Для виявлення помилок CAN використовує комбінацію з п'яти різних механізмів: контроль розрядного рівня, контроль циклічного надлишкового коду (ЦНК), контроль підтвердження, заповнення бітів та контроль в межах сигналу.

Вузли мають систему самоконтролю та самостійно перемикаються при певній частоті помилок з *error active* на *error passive* (дані режими відрізняються засобом обробки помилок) і, нарешті, на *bus-off* режим.

Згідно (ISO) нормам 11898 мережа CAN специфікована тільки до ISO/OSI-Рівня 2. На цьому інтерфейсі контролери можуть обмінюватись сигналами, що містять до восьми бітів корисної інформації, та ідентифікаторами.

9.3.3.4. 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 в стандарті ISO11898 – це диференційно-керована двопровідна шина із загальним зворотним проводом. Два провідника, згідно полярності щодо заземлюючого проводу, називаються CANH і CANL. На вході приймального пристрою діє диференціальне 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 передачі), для збільшення довжини шини або для збільшення числа підключених вузлів використовують повторювачі (репітори).

Шина CAN і пристрої, підключені до неї, функціонують як схема «монтажне І», рис. 9.15.

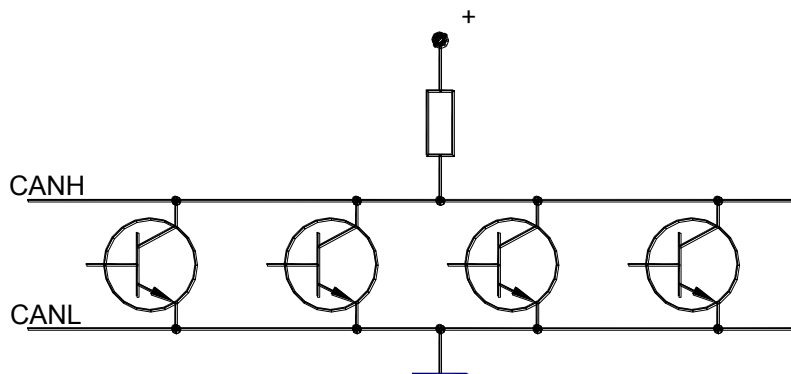


Рисунок 9.10 – Загальний вигляд схеми і принцип роботи «монтажне І»

Кожен з транзисторів, зображених на рис. 9.10, являє підключення вузла на шину CAN. Кожен вузол може передавати логічну одиницю (еквівалентно закриттю транзистора), при цьому на дроті шини CANH з'являються «+» напруги джерела живлення або логічна одиниця, відкриття транзистора призводить до заземлення проводу CANH, що 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. 9.11).

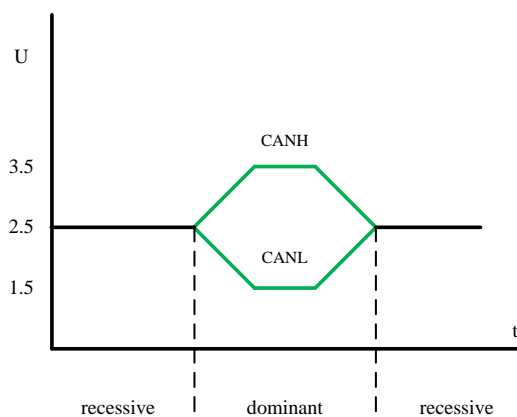


Рисунок 9.11 – Форма сигналів на CAN-шині

Передача даних по CAN шині здійснюється за допомогою посилок (фреймів), встановлених стандартом CAN 2.0. В (рис. 9.12).

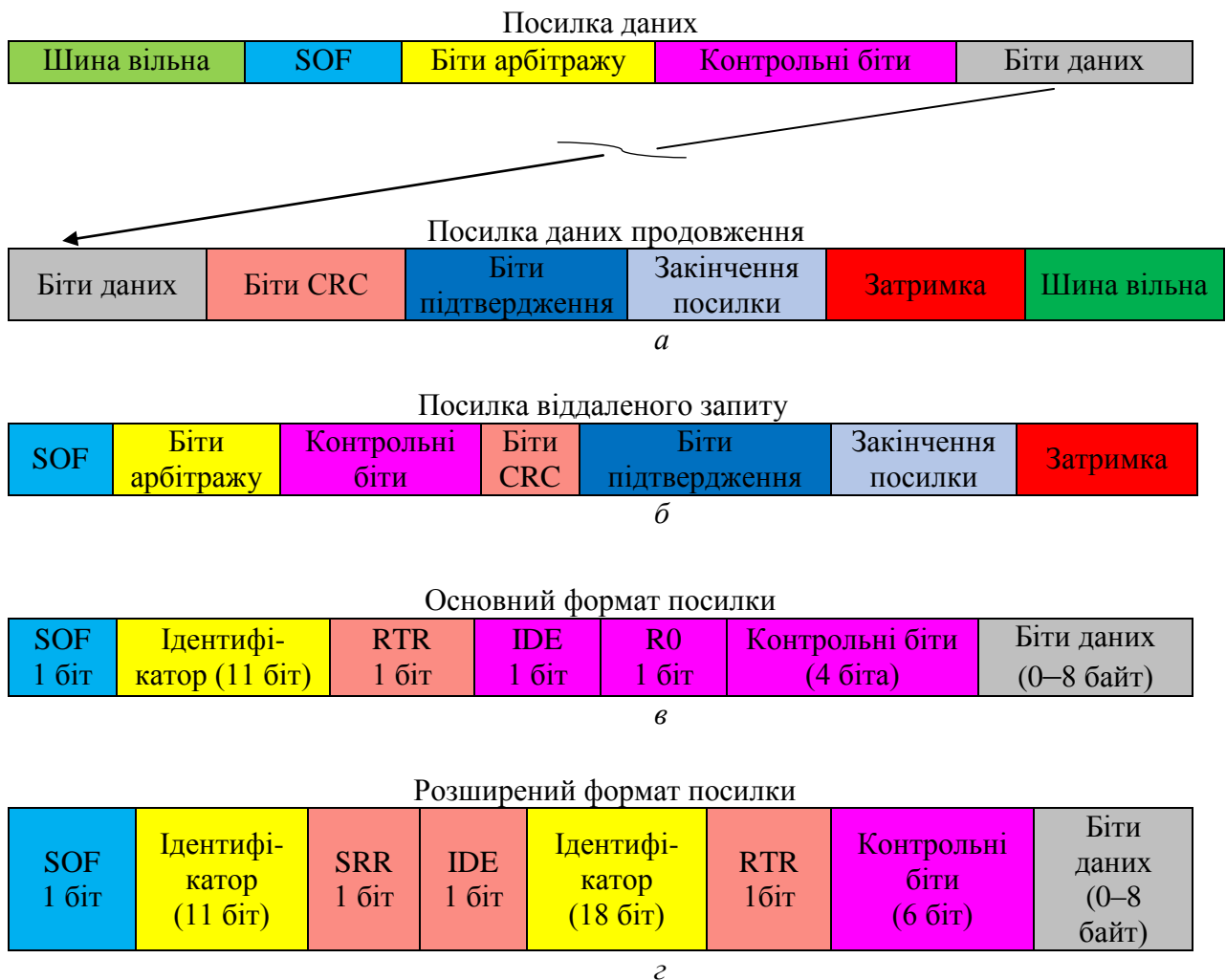


Рисунок 9.12 – Структура і формат посилок в CAN шині

Якщо шина вільна в лінії зв'язку встановлений одиничний сигнал. Посилка даних розділяється на поля (рис. 9.12, а). Поле початку послілки 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 біт і один біт RTR, який визначає це повідомлення як послілку даних (RTR = 0) або кадр запиту даних (RTR = 1).

Контрольні 6 біт (Control Field), 4 з яких визначають кількість байт даних, що передаються в послілці (до 8 байт даних), один біт розширеного кадру (IDE) використовується для вказівки розширеного ідентифікатора, один біт R0 резервний, Поле контрольної суми CRC Field містить 16 біт, 15 біт з яких складають контрольну суму, а один біт використовується як розділовий CRC Del. Поле підтвердження складається з 2 біт. Перший біт ACKSlot, який під час

передачі формується як recessive, а пристрій, який приймає кадр без помилок, підтверджує правильність прийому установкою цього біта в стан dominant.

Другий розділовий біт ACKDel завжди зберігає значення recessive. Закінчується кадр полем закінчення кадру End of Frame (7 біт). Всі біти мають стан recessive. Пауза між кадрами складається мінімум з трьох одиничних біт.

Таким чином, основний формат посилки (рис. 9.12, в.) Містить ідентифікатор з 11 біт, розширений формат (рис. 9.12, з) з 29 біт. Біт RTR, що визначає тип посилки (дані або запит даних) і в стандартному і в розширеному форматі розташовується відразу за ідентифікатором. Розширений формат посилки 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 же ідентифікатор тільки біт RTR в цьому повідомленні встановлено в 1. Оскільки даних в цьому повідомленні немає, то довжина такого повідомлення дорівнює 44 біта. За даними в посилці у всіх повідомлень розташовуються однакові блоки бітів. Контрольна сума, біти підтвердження правильного прийому посилки, біти закінчення посилки.

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 по їх порівнянні 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 управління доступом відповідальний за визначення, яким з вузлів в мережі надана шина для передачі повідомлення.

Домінуючий біт на шині, відповідно до логіки роботи «монтажного І» завжди переважає будь-якого числа рецесивних бітів, переданих іншими вузлами, отже, вузол, який робить спробу передачі, збереже контроль щодо шини тільки в тому випадку, якщо повідомлення, яке він передає, має наймолодший ідентифікатор серед усіх повідомлень, які передаються одночасно в шину. Цей механізм і є ідеєю управління доступом в CAN.

Розглянемо приклад, коли три вузла одночасно намагаються передати свої повідомлення.

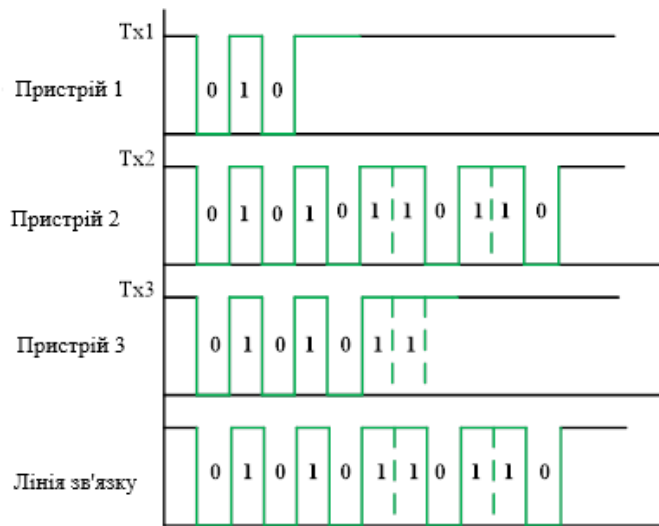


Рисунок 9.13 – Графік часового аналізу боротьби за доступ до CAN шини

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. В цьому випадку пристрій 1 апаратно відключається від шини. Приклад схемного рішення боротьби за канал наведено на рис. 6.21.

Пристрої 2 і 3 продовжують передачу повідомлень до восьмого біта, коли пристрій 3 передає одиничний сигнал, а пристрій 2 нульовою

В цьому випадку пристрій 3 відключається від шини, а пристрій 2 продовжує передавати своє повідомлення. Аналіз показує, що повідомлення, які передаються вузлом 1 та 3, мають більш низький пріоритет (більш високе значення ідентифікатора), таким чином, вузол 2 виграє боротьбу за канал. 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 же ідентифікатором, то доступ до каналу зв'язку отримає кадр даних, оскільки біт RTR (Remote Transmission Request) в кадрі запиту має значення (recessive), а в кадрі даних (dominant).

Функціональна схема, апаратно реалізує функцію боротьби за канал зв'язку приведена на рис. 9.14. При передачі CAN пристроями в лінію зв'язку різних сигналів RS тригер скидається і блокує передачу даних у того пристрою, який передає одиничний сигнал.

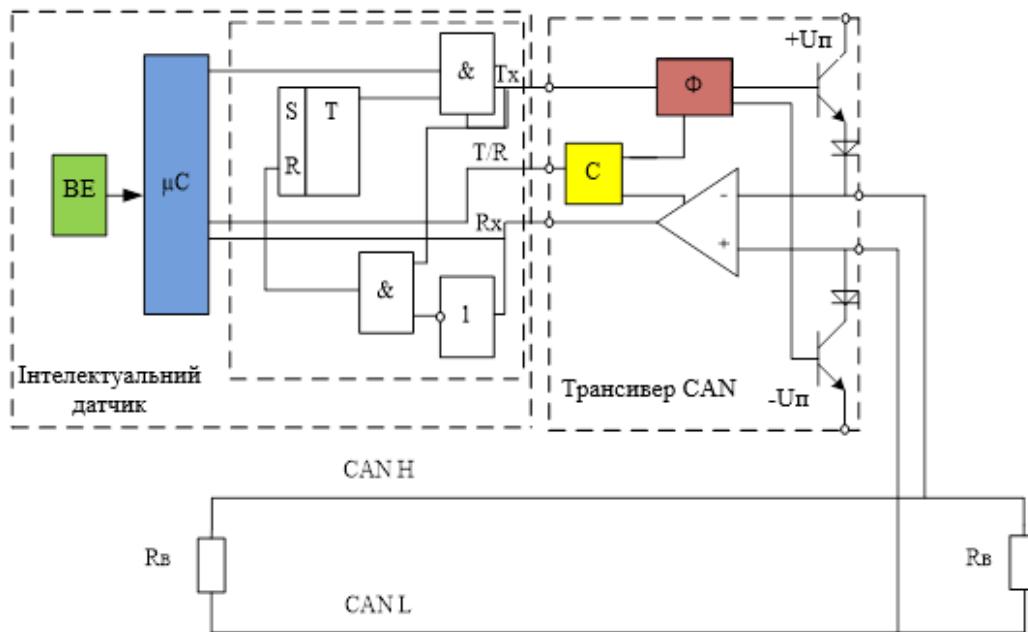


Рисунок 9.14 – Функціональна схема підключення інтелектуального датчика до CAN-шини

У CAN-шині передане повідомлення може бути прийнято декількома пристроями. При цьому пристрої можуть додаватися або виключатися з мережі без зміни програмного і апаратного забезпечення вузлів системи.

Алгоритм роботи CAN-шини сертифікований у вигляді стандарту CAN 2.0 В, що забезпечило виготовлення CAN-контролерів у вигляді окремих мікросхем для реалізації всіх функцій CAN-інтерфейсу на апаратному рівні.

Наприклад, широке поширення набули мікроконтролери, які мають на кристалі декілька послідовних інтерфейсів, таких як для міжплатним комунікацій (I2C, SPI), так і таких як RS485 і CAN

Функціонально мікроконтролер складається з мікропроцесора і CAN пристрою. Мікропроцесор через порти введення виведення пов'язаний з датчиками, а через SPI інтерфейс здійснюється зв'язок з CAN-пристроєм.

Фільтрація всіх повідомлень в CAN-контролері здійснюється налаштуванням фільтрів і масок. CAN-шина містить кілька рівнів протоколу передачі даних:

- рівень прикладної програми – приймає відфільтровані повідомлення і запитує інформацію з необхідним ідентифікатором;
- рівень повідомлення – здійснює фільтрацію повідомлень і визначає їх статус;
- транспортний рівень – визначає спосіб передачі повідомлення для передачі по фізичному каналу. Цей рівень відповідає за бітову синхронізацію,

розподіл повідомлення по кадрам, доступ до каналу, підтвердження повідомлення і розпізнавання помилок;

– фізичний рівень – забезпечує спосіб передачі бітів.

Бітове представлення, яке використовується для передачі повідомлень в шині CAN-NRZ кодування (без повернення до нуля), гарантує максимальну ефективність в бітовому кодуванні. Тому якщо в переданому повідомленні присутній 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 Кбіт/с, 500 Кбіт/с, 250 Кбіт/с, 125 Кбіт/с, 50 Кбіт/с і 10 Кбіт/с. Рекомендовані довжини шини для кожної з цих швидкостей в бодах – 25 м, 50 м, 100 м, 250 м, 500 м, 1000 м, 2500 м і 5000 м відповідно рис. 9.19.

Рівень передачі даних (DLL) можна розділити на два підрівні: засіб управління доступом (Medium Access Control) і управління логічної зв'язком (Logical Link Control).

CAN пристрій, який виявляє помилку в передачі, здатний перервати передачу і послати прапор 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 оцінкою помилок, виявлених в межах кожного вузла, з метою визначення дефектів в конкретному пристрої. При накопиченні помилок будь-яким пристроєм воно відключається від мережі.

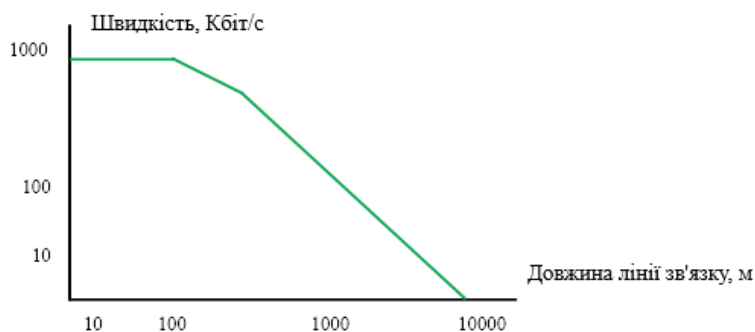


Рисунок 9.15 – Залежність швидкості передачі інформації по CAN-шині в залежності від довжини лінії зв'язку

Рівень управління 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 управління доступом (MAC) прозорі для верхніх рівнів. Практично, це означає, що управління логічним зв'язком (LLC) відповідально 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 виходить, як об'єднання прапорів 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 вузол перериває передачу поточного повідомлення і передає на шину повідомлення помилки, що складається з шести послідовних «dominant» бітів, таке повідомлення називають прапором помилки (рис. 9.16, а).

Всі пристрої, підключені до шини, розпізнають помилку і передають на шину свої прапори помилок. Таким чином, прапор помилки складатиметься від 6 до 12 біт, що генеруються одним або декількома вузлами.

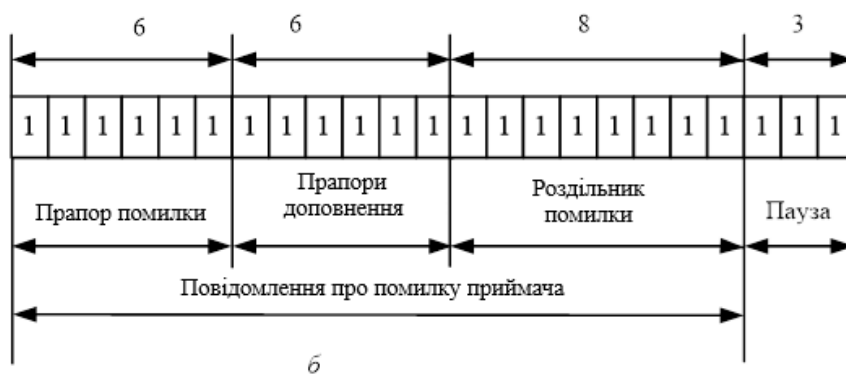


Рисунок 9.16 – Формати повідомлень про помилки

Повідомлення про помилку закінчується полем роздільник з 8 «recessive» біт, яке називають кадром помилки. Після завершення кадру помилок шина переходить в нормальний режим, і вузол повторює спробу знову передати перерване повідомлення.

Якщо приймаючий вузол визначає помилку на шині, то він формує пасивний прапор помилки з шести «recessive» біт, доповнює ще 6 біт заповнення і формує поле роздільник з 8 «recessive» біт (рис. 9.16, б).

Передача пасивного кадру помилок не порушує роботу інших вузлів, підключених до шини. Після передачі кадру помилок вузол повинен чекати шість послідовних «recessive» біт, щоб знову підключитися до шини.

Специфікації ISO/OSI рівня 2 не достатньо 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 автоматичі, 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 створено багато відкритих стандартів рівня застосування. Основні відкриті вищі протоколи в сфері автоматизації та інших споріднених сферах – CANopen та DeviceNet.

Протокол CANopen дуже поширений та застосовується в багатьох інших сферах окрім автоматизації.

На рис. 9.17 зображено базовий концепт комунікації пристрою з інтерфейсом CANopen.

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, встановлюється 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 розділяти на декілька сигналів. PDO використовують для швидкої передачі невеликих наборів даних з високим пріоритетом сигналу.

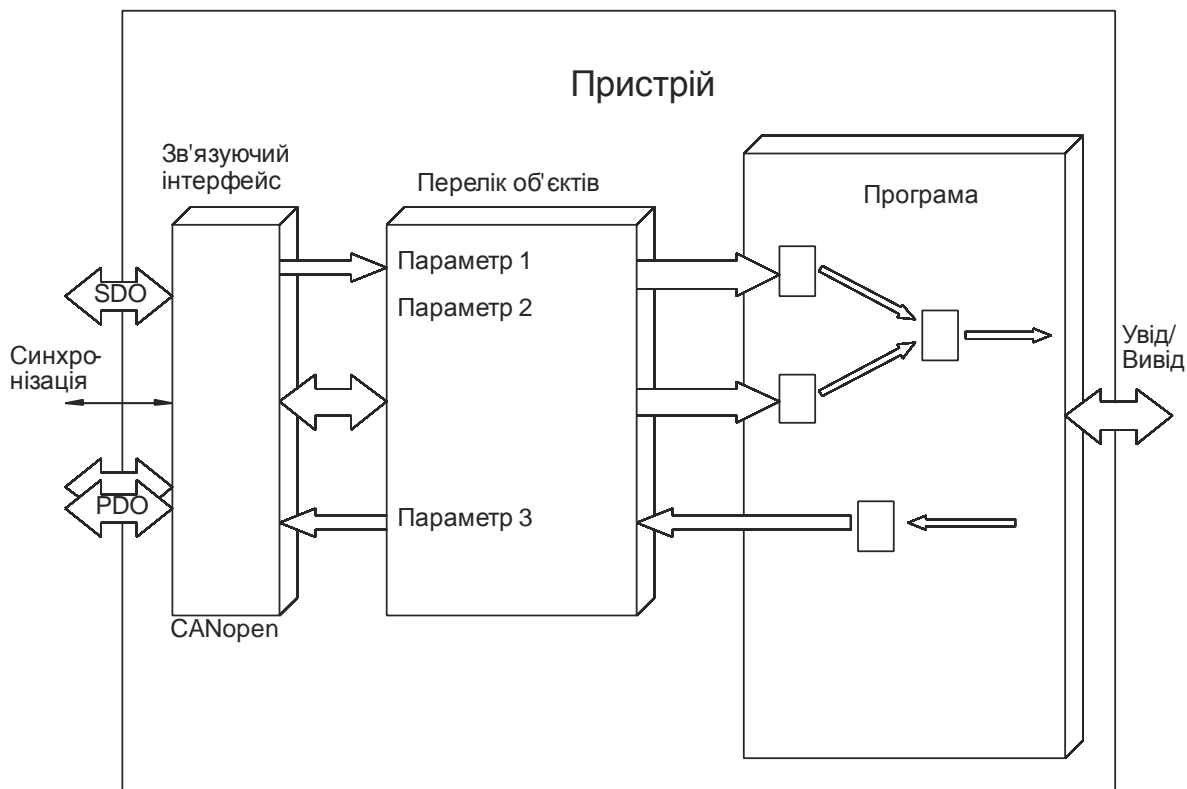


Рисунок 9.17 – Концепція комунікації пристрою CANopen

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 в записях 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 у відповідності 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.

9.3.3.5. Основні характеристики CAN

В табл. 9.1 ще раз зібрані основні характеристики мережі CAN

Таблиця 9.1 – Основні характеристики CAN

Походження	Розроблена компаніями Bosch и Intel для застосування в 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 1 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 – приблизно 130 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:6
Застосування	Автомобілі (легкові, вантажні, автобуси), промислова та будівельна автоматизація, ткацькі верстати, медичне устаткування, верстати та ін.

9.3.4. MODBUS

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 в такому випадку збільшується до макс. 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. В подібній техніці кабельного з’єднання повторювачі сигналів мають підключатись завжди між рівними користувачами (рис. 9.18).

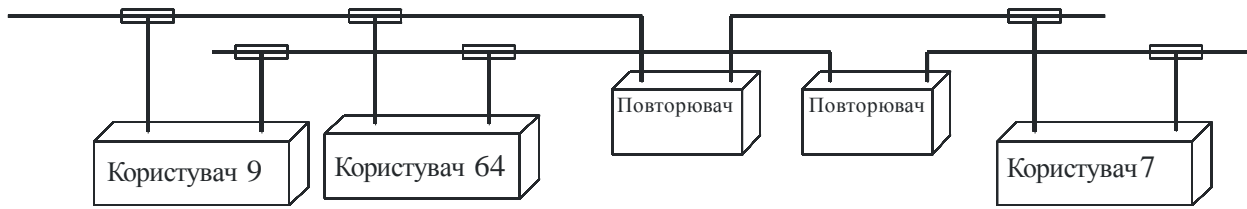


Рисунок 9.18 – Двокабельна мережа

За допомогою спеціальних повторювачів топологія може бути розширена, зіркоподібна або деревоподібна.

Відповідно до еталонної моделі ISO/OSI в Modbus Plus реалізовано разом з рівнями 1 і 2 також рівень 3. Мережі можуть з’єднуватись шляхом мостів. В 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 кілька мостів. Міст є активним користувачем шини з двома незалежними одна від одної адресами (рис. 9.19).

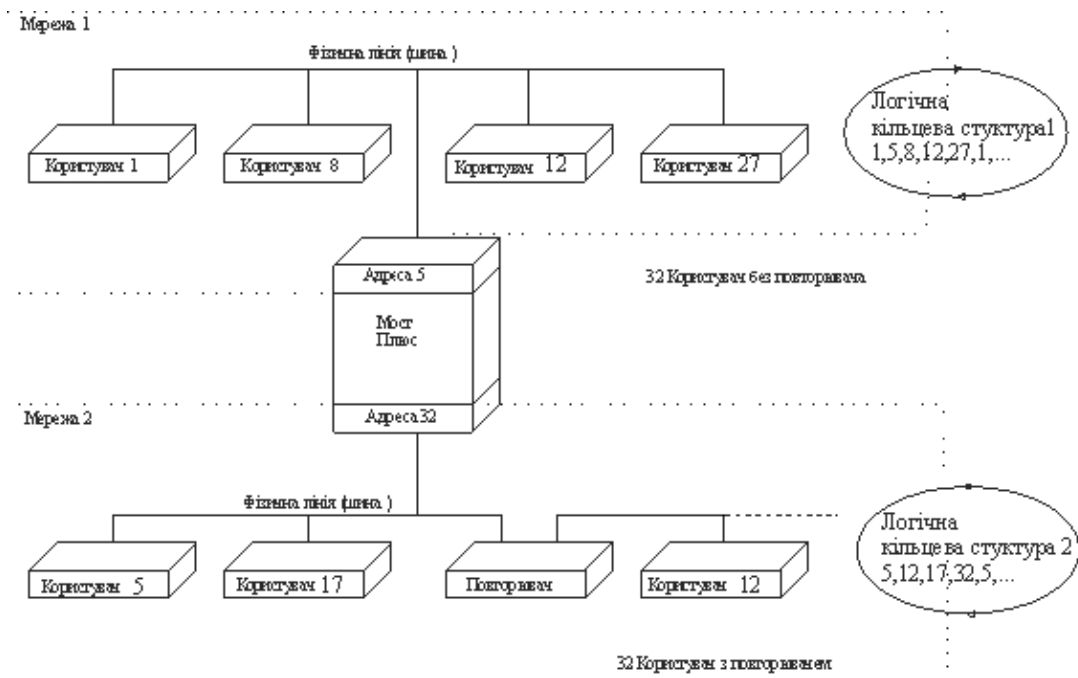


Рисунок 9.19 – Фізична та логічна структура Modbus

Якщо від одної мережі до іншої надсилається сигнал, то міст отримує відповідний сигнал, зберігає його та надсилає далі, як тільки сигнал отримав маркер іншої мережі. Таким чином можна посилати сигнали через 4 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 має бути двох однакових адрес.

Якщо два користувачі в одній кільцевій схемі мають однакові адреси, це передається прикладній програмі.

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 підключатись до шини. Він може надсилати сигнали довжиною максимум 512 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 прийнято сигнал підтвердження, той самий сигнал буде повторюватись. Якщо під час другої спроби сигнал підтвердження відправника не надходить, він надсилає повідомлення про помилку, яке може бути запитано додатком.

При передачі маркера одночасно може бути надіслано до 512 біт ширококомовлених 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 слідує 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. Поруч 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 в якості фізичного середовища 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 приводить до скорочення витрат на його вдосконалення. В 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 в цьому випадку не відправляється. Особливістю протоколу MODBUS являється те, що частина параметрів протоколу не змінюється. До них відносяться формат кадру посилки, 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 ASCII або RTU визначаються користувачем.

Кадр повідомлення (рис. 9.20) містить адресу одержувача, яку операцію повинен виконати одержувач, команду, дані для виконання команди і метод контролю достовірності передачі інформації.

Адреса одержувача 1 байт	Команда 1 байт	Дані 0–252 байта	CRC контрольна сума 2 байта
--------------------------	----------------	------------------	-----------------------------

Рисунок 9.20 – Кадр повідомлення протоколу MODBUS

Ведений пристрій отримує повідомлення, перевіряє його на правильність передачі, виконує команду і формує відповідь на повідомлення, яке було надіслане і має точно такий формат.

MODBUS передбачає два режими роботи. Режим ASCII, коли для кодування інформації використовується символ 0-9, A-F, і режим RTU (Remote Terminal Unit), коли інформація кодується в двійковій системі. Довжина

байті біт парності встановлюється в нуль, при цьому відбувається контроль непарності.

При відсутності контролю парності в посилці замість молодшого розряду встановлюється другий стоповий біт (рис. 9.21, б).

Повідомлення в MODBUS передаються посилками або фреймами. Послідовність передачі байтів у фреймі приведена на рис. 9.24.

Першим передається байт адреси, потім байт команди, в якому вказується, які функції має виконати ведене пристрій. За байтом команди йдуть байти даних, число яких може змінюватися від 0 до 252. Наприкінці фрейма слідує два байта контрольної суми, призначених для перевірки правильності прийнятого фрейму.

Таким чином, довжина фрейму є змінною величиною, але не більше 256 байт.

Передача фреймів здійснюється із забезпеченням тимчасової синхронізації. Графік часового аналізу передачі фреймів наведено на рис. 9.22. Фрейми передаються через інтервали часу $T_{\Pi} = 3,5 t_{\delta}$, де t_{δ} час передачі одного байта. При швидкості передачі 9600 біт / с тривалість одного біта $t_{\delta im} = 0,1$ мс, тоді $t_{\delta} = t_{\delta im} 11 = 1,14$ мс, а час паузи $T_{\Pi} = 3,5 t_{\delta} = 4$ мс. Початок передачі визначається пристроєм MODBUS, який отримав команду на передачу даних.

При цьому пристрій виявляє наявність паузи T_{Π} і починає передачу фрейму. Закінчення передачі фрейму контролюється виявленням затримки після передачі контрольної суми, яка: $t_z = 1,5 t_{\delta} = 1,7$ мс.

Якщо після закінчення часу і до закінчення паузи починається передача будь-яким пристроєм, то цей кадр вважається помилковим і ця посилка передається повторно і помилка фіксується у відповіді на запит.

Адреси пристроїв в MODBUS визначені від 1 до 247. Адреси 248-255 зарезервовані.

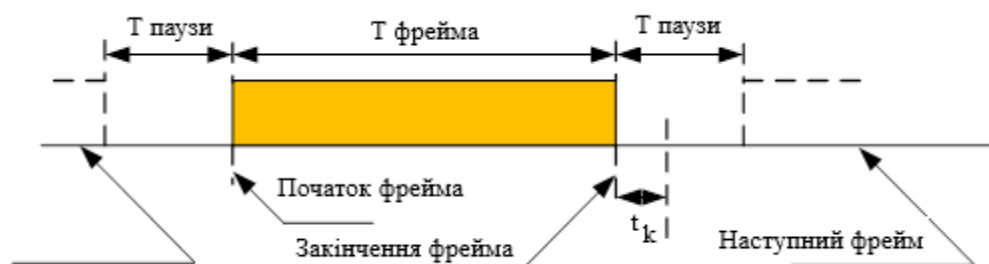


Рисунок 9.22 – Графік часового аналізу синхронізації передачі фреймів у протоколі MODBUS

Провідний пристрій не має адреси, оскільки ведений пристрій може передавати інформацію тільки ведучому.

При ширококомовному режимі встановлюється нульовий адрес 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, лежать 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 повинні бути вказані номер датчика, номер регістра або адреси осередків пам'яті, а при записі – вид 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 комірок пам'яті, в яких зберігаються значення вимірюваних параметрів.

Послідовність алгоритму формування контрольної суми CRC при передачі повідомлення наступна:

- у двобайтний регістр CRC завантажується число FFFFH (табл. 9.2);
- в пам'ять завантажується число, яке представляє поліном циклічного коду;
- перший байт повідомлення складається по модулю 2 (виключне АБО) з регістром CRC, результат якої залишається в регістрі CRC, зсувається в бік молодшого біта на один біт, при цьому старший біт регістра CRC заповнюється нулем.

Якщо при цьому молодший біт регістру 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 посилки. Сформована контрольна сума у вигляді двох байтів передається в кінці посилки. На приймальній стороні виконується зворотнє перетворення, і отримане число порівнюється з числом CRC в посилці.

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 запитання

1. Скільки об'єктів можна підключити до шини?
2. Перелічити особливості послідовних шин.
3. Які різновиди польових шин ви знаєте?
4. Інтерфейс RS 232. Назвати приклади використання та логічні рівні.
5. Скільки користувачів можна підключити до інтерфейсу RS 485?
6. Призначення шини I²C.
7. Скільки ліній необхідно для роботи шини I²C?
8. Як кодуються умови початку та кінця передачі по шині I²C?
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 асемблера

Оригінальний текст програми на мові асемблера 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 комірки 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 кінчається 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 (аббревіатурою) 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 управління операндом може бути число (LCALL 0135H), мітка (JMP LABEL), непряма адреса (JMP @A) або вираз (JMP \$ – 2, де \$ – поточний зміст лічильника команд).

Використовувані в якості операндів символічні імена і мітки повинні бути визначені, а числа представлені із зазначенням системи числення, для чого використовується суфікс (буква, що стоїть після числа): В – для двійковій, Q – для вісімковій, D – для десяткової і H – для шістнадцятиричної. Число без суфікса за замовчуванням вважається десятковим.

Асемблер MCS-51 допускає використання виразів в 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 розподіл, MOD – розподіл по модулю) і логічних (OR – АБО, AND – І, XOR – виключне АБО, NOT – заперечення) операцій у форматі 2-байтних слів. Наприклад, запис SUBB A, # ((NOT 27) +1) еквівалентна запису SUBB A, # 0E5H і забезпечує віднімання від вмісту акумулятора числа –27, представленим в додатковому коді. Широко використовуються також оператори LOW і HIGH, що дозволяють обчислити молодший і старший байти 2-байтного операнда.

Коментарі завжди починаються з 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 порта P0.0

SET призначає символічне ім'я числовим значенням або регістру. Ім'я може бути згодом змінено за допомогою директиви SET.

DB заносить в 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 й налагодження програмного забезпечення для контролерів серії MCS-51.

Мета роботи: вивчити технологію налагодження програмного забезпечення MCS-51, вивчити команди пересилань із прямою адресацією й арифметичних і логічних операцій із 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 наступному:

1. Складання тексту програми у вигляді мнемокоду процесора.
2. Трансляція в машинні інструкції процесора.
3. Налagodження програми на програмному симуляторі мікроконтролера.
4. Запис налагодженої програми в мікроконтролер.
5. Перевірка функціонування програми на реальному об'єкті.

Дані лабораторні роботи будуть виконуватися в інтегрованій системі MCStudio. Вона являє собою інтегровані в одну оболонку наступні 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, в процесі трансляції, налагодження і т.ін. система створює додаткові службові файли. Для керування цією сукупністю файлів використовується концепція проекту.

Детальний опис встановлення демо-версії системи на персональний комп'ютер та робота з нею приведений в [5].

Методика виконання

1. Проект створюється з головного меню MCStudio послідовним вибором пунктів Файл – Створити – Проект. Для вказівки місця розташування проекту, імені проекту й модулів, типу процесора заповнюється форма (рис. 10.1).

У результаті на екрані відкриваються вікна: «Проект», «Ресурси» і вікно редактори тексту програми із прописаним шляхом до вихідного модуля програми (рис. 10.2).

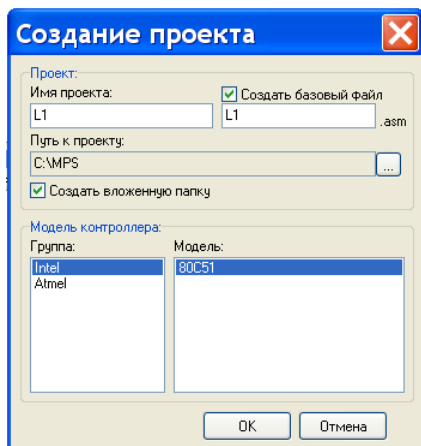


Рисунок 10.1 – Вікно створення проекту

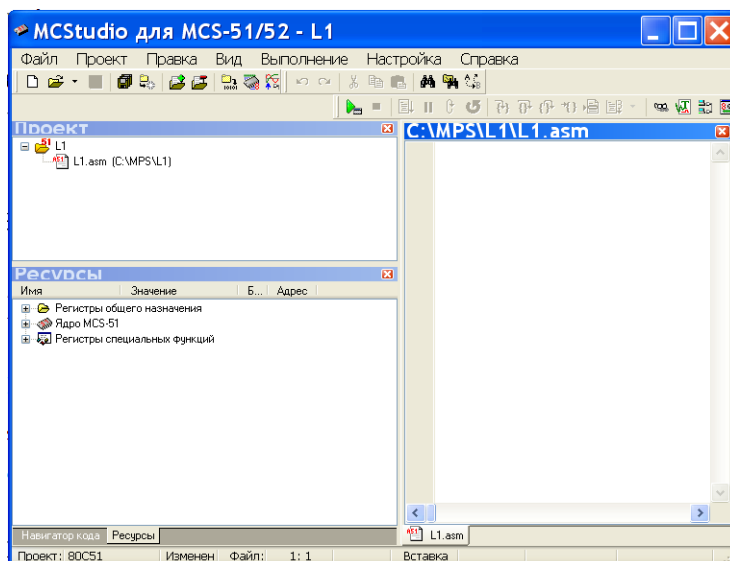



Рисунок 10.2 – Екран MCStudio після створення проекту

1. У текстовому редакторі набираємо текст програми, наведений у лабораторній роботі. При наборі тексту необхідно дотримуватись наступних 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 виявить і видасть повідомлення про характер помилки у вікні повідомлень(рис. 10.4). Якщо у вікні повідомлень по помилці клікнути лівою кнопкою миші, то курсор буде поміщено у рядок програми, де була виявлена помилка.

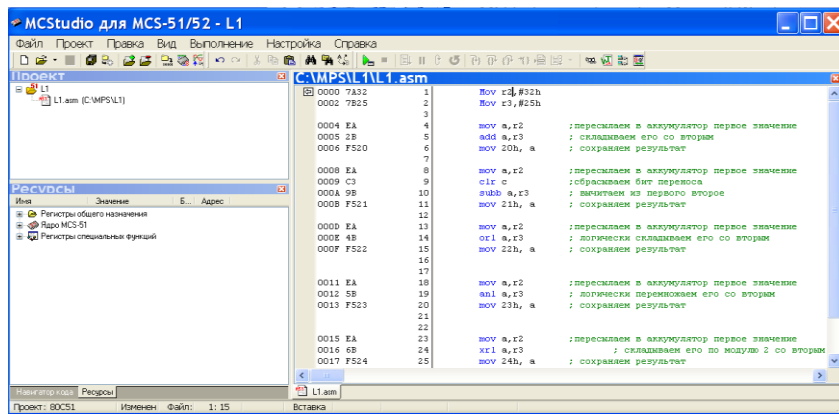


Рисунок 10.3 – Результат компіляції програми

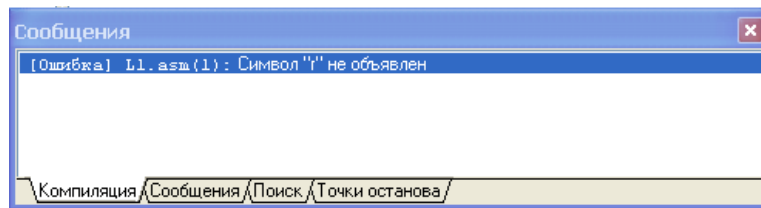



Рисунок 10.4 – Видяг вікна повідомлень про помилку

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, щоб переконатися, чи однаково розуміють виконання команди контролер і програміст.

5. Результати виконання команд переглядаємо у вікні відображення дерева ресурсів мікроконтролера (рис. 10.5).

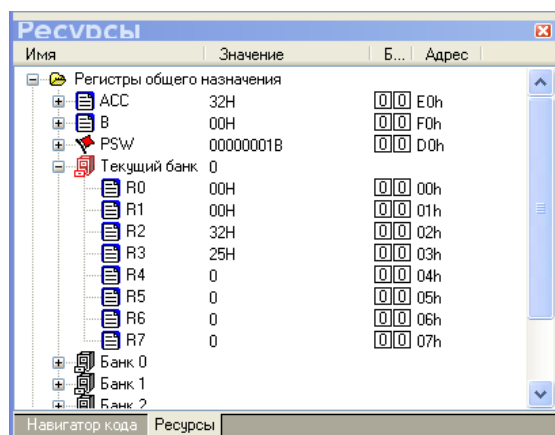


Рисунок 10.5 – Дерево ресурсів контролера на 6 кроці виконання програми

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 виконання команд пересилань із прямою адресацією.

Пересилання константи (рис. 10.6).

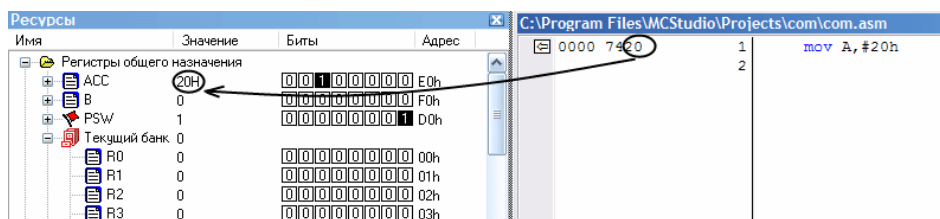


Рисунок 10.6 – Виконання команди mov A,# 20h

Команда MOV A,# d виконує пересилання константи, яке зберігається в другому байті команди, в акумулятор.

Пряма адресація (рис. 10.7).

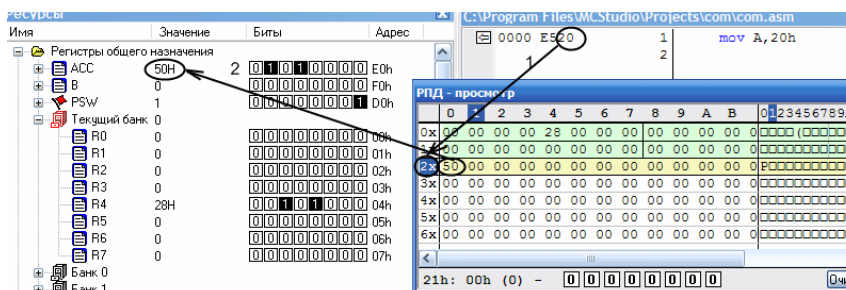


Рисунок 10.7 – Виконання команди mov A,20h

Команда MOV A,d виконує пересилання даних із внутрішньої пам'яті в акумулятор. Адреса комірки пам'яті перебуває в другому байті команди (рис. 10.8)

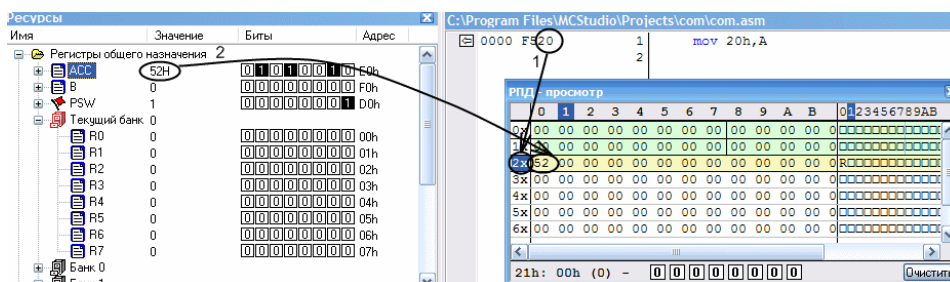


Рисунок 10.8 – Виконання команди mov 20h,A

Команда MOV ad, A виконує пересилання даних з акумулятора у внутрішню пам'ять. Адреса комірки пам'яті перебуває в другому байті команди. Пряма регістрова адресація (рис. 10.9).

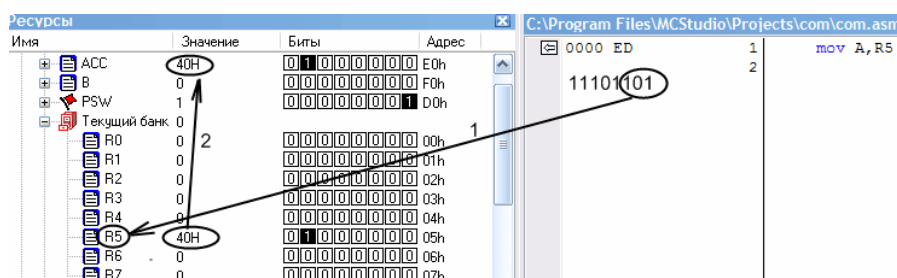


Рисунок 10.9 – Виконання команди mov A,R5

Команда MOV A,Rn, виконує пересилання значення регістру поточного банку в акумулятор. Адреса регістру зазначена в молодших трьох бітах коду операції (рис. 10.10).

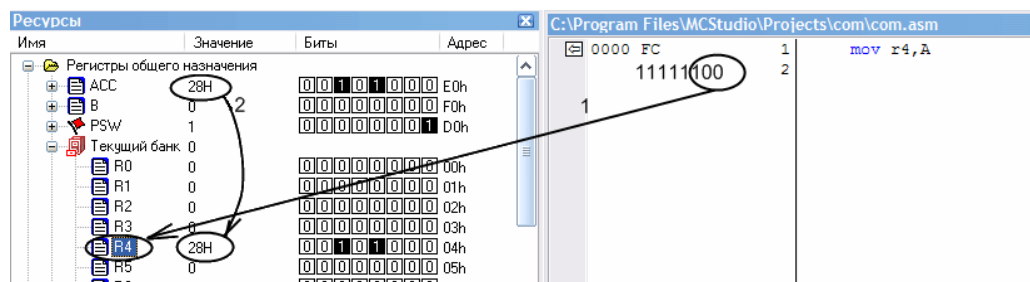


Рисунок 10.10 – Виконання команди mov R4, A

Команда MOV Rn,A виконує пересилання значення акумулятора в регістр поточного банку. Адреса регістру зазначена в молодших трьох бітах коду операції.

Задання лабораторної роботи. Написати програму роботи мікроконтролера МК51, яка виконує арифметичні й логічні дії над двома числами, поміщеними в регістри R2 і R3, і результати записує у внутрішню пам'ять даних з адреси 20H.

Текст програми мовою Асемблера

```
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 й результати виконання кожної операції.

10.3. Лабораторна робота № 2

Дослідження внутрішньої й зовнішньої пам'яті даних і пам'яті програм

Мета роботи: закріпити знання студентів при роботі з відлагоджувачем, вивчення команд пересилань із непрямою адресацією, укажчиків, роботу з масивами.

Теоретичні відомості. Пам'ять даних необхідна для приймання, зберігання й видачі інформації, використовуваної в процесі виконання програми. Пам'ять програм призначена для зберігання машинних кодів програми та констант, зберігання й 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, уміст 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 й вмісту `DPTR`. Тому в лабораторній роботі перед звертанням до ПП рекомендується скинути вміст акумулятора.

При асемблерному програмуванні мікроконтролерів сім'ї 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 відлагоджувачем однаково.

Нижче наведені фрагменти вікон робочого середовища MCStudio і пояснення виконання команд пересилань із непрямою адресацією.

Непряма адресація (рис. 10.11).

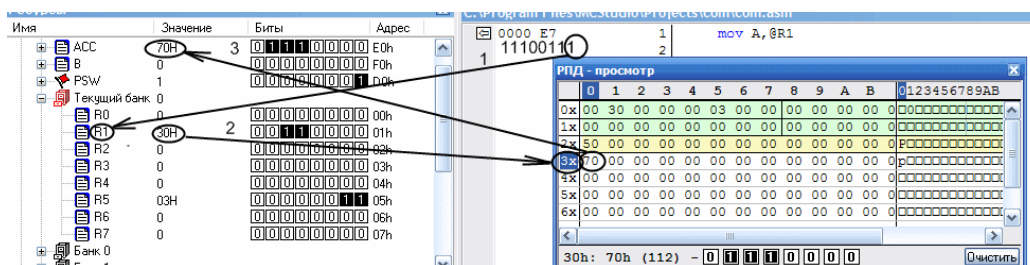


Рисунок 10.11 – Виконання команди `mov A,@R1`

Команда `MOV A,@Ri` виконує пересилання в акумулятор вмісту комірки внутрішньої пам'яті даних (рис. 10.12). Адреса регістра (стрілка 1) вказується в молодшому біті кода операції. Адреса комірки (стрілка 2) перебуває в регістрі-показчику. Регістром-показчиком можуть виступати тільки регістри R0 і R1.

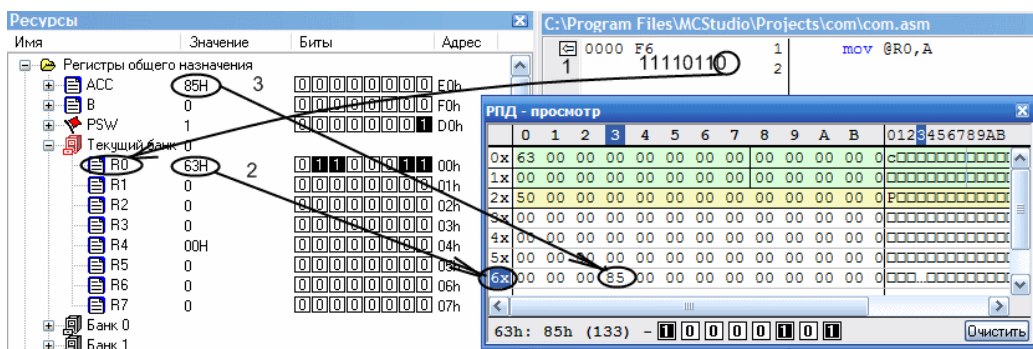


Рисунок 10.12 – Виконання команди `mov @r0,A`

Команда `MOV @Ri,A` виконує пересилання вмісту акумулятора в комірку внутрішньої пам'яті даних. Адреса комірки перебуває в регістрі-показчику. Регістром-показчиком можуть виступати тільки регістри R0 і R1.

Приклад задання. 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 з адреси 60h. Другий – у пам'яті програм контролера, починаючи з адреси 40h. Результат додавання за модулем масивів записується починаючи з 50h зовнішньої пам'яті. При роботі із зовнішньою пам'яттю використовується однобайтова адресація.

Примітка. При написанні програми задається лише один масив, що перебуває в пам'яті програм, починаючи з адреси 40H. Другий масив можна задати безпосередньо в процесі роботи з відлагоджувачем, записавши елементи масиву в заданій області внутрішньої пам'яті даних (у цьому випадку – з адреси 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 Асемблера

```

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 адрес 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 ; задання масиву в пам'яті програм
End

```

Звіт повинен містити текст програми, текст індивідуального завдання й блок-схему алгоритму, результат виконання програми.

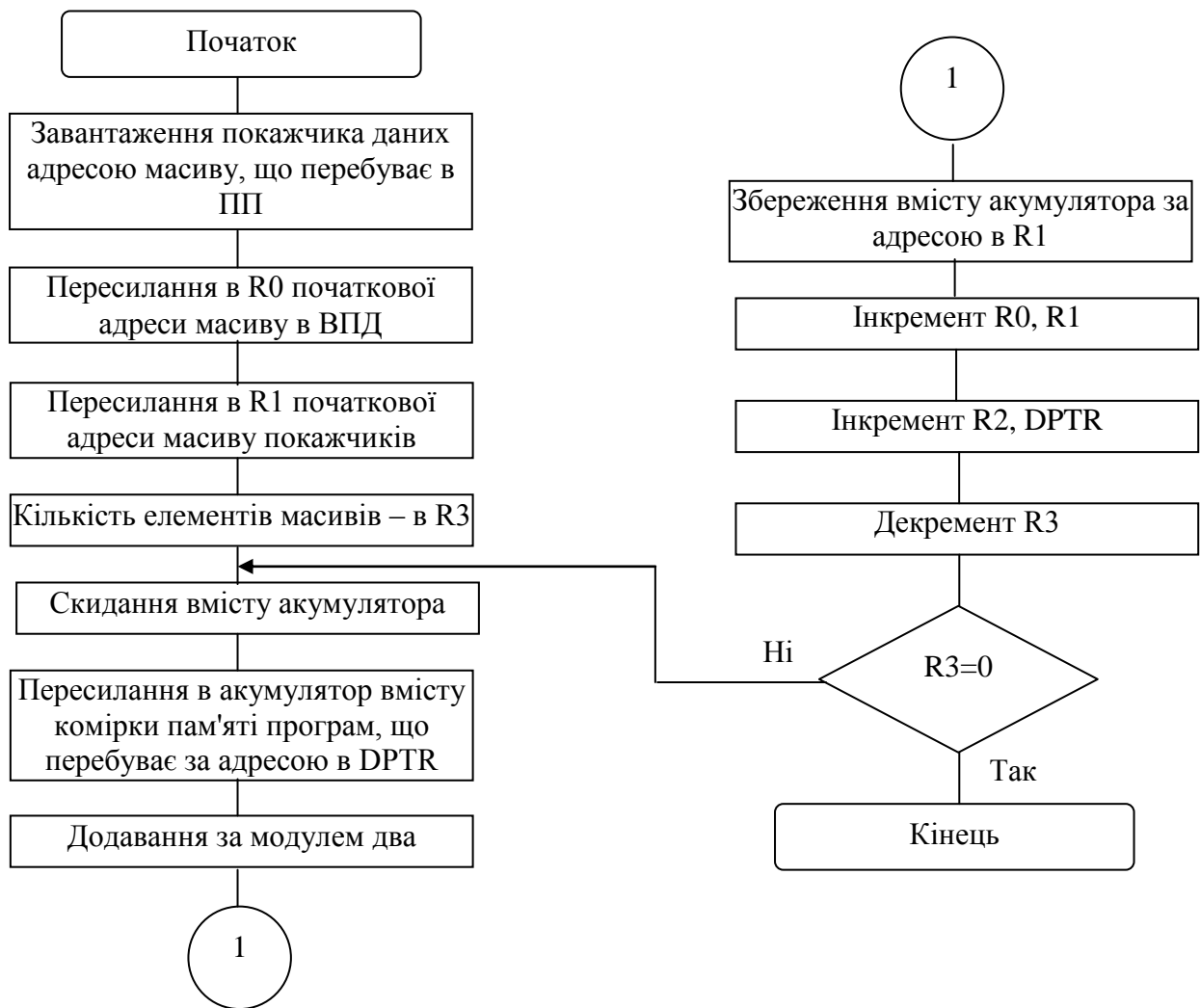


Рисунок 10.13 – Блок-схема програми складання за модулем двох масивів

Індивідуальні завдання

Варіант 1. Написати програму, яка обчислює двобайтову суму елементів масиву, розташованого в пам'яті програм з адреси 50H. Довжина вхідного масиву – 12 однобайтових елементів.

Варіант 2. Написати програму, яка виконує множення елементів масиву, розташованого в пам'яті програм з адреси 80H на 25, і двобайтовий добуток зберігає у вигляді масиву у внутрішній пам'яті даних з адреси 30H. Довжина вхідного масиву – 8 однобайтових елементів.

Варіант 3. Написати програму, яка виконує сортування елементів масиву, розташованого в пам'яті програм з адреси 70H. Парні елементи зберігаються у внутрішній пам'яті даних з адреси 30H, а непарні – в зовнішній пам'яті даних з адреси 50H. Довжина вхідного масиву – 10 однобайтових елементів.

Варіант 4. Написати програму, яка виконує ділення елементів масиву, розташованого в пам'яті програм з адреси 80H, на 3. Ціле від ділення

зберігається у внутрішній пам'яті даних з адреси 50Н, а залишок – у зовнішній пам'яті даних з адреси 20Н. Довжина вхідного масиву – 15 однобайтових елементів.

Варіант 5. Написати програму, яка виконує додавання за модулем 2 елементів масиву, розташованого в пам'яті програм з адреси 20Н із умістом комірки пам'яті за адресою 55Н. Довжина вхідного масиву – 12 однобайтових елементів. Результат розмістити за адресою 30Н у внутрішній пам'яті даних.

Варіант 6. Написати програму, яка виконує додавання елементів масиву, розташованого в пам'яті програм з адреси 100Н із умістом комірки пам'яті за адресою 25Н, і результат зберігає у вигляді масиву у внутрішній пам'яті даних з адреси 30Н. Довжина вхідного масиву – 10 однобайтових елементів.

Варіант 7. Написати програму, яка виконує сортування елементів масиву, розташованого в пам'яті програм з адреси 50Н. Негативні елементи зберігаються у внутрішній пам'яті даних з адреси 20Н, а позитивні – в зовнішній пам'яті даних з адреси 60Н. Довжина вхідного масиву 10 однобайтових елементів. Ознакою негативного числа є 1 в D7.

Варіант 8. Написати програму, яка виконує множення елементів масиву, розташованого в пам'яті програм з адреси 80Н, на 5. Результат зберігається у внутрішній пам'яті даних з адреси 50Н. Довжина вхідного масиву – 15 однобайтових елементів.

Варіант 9. Написати програму, яка виконує логічне додавання елементів масиву, розташованого в пам'яті програм з адреси 30Н із константою A5Н. Довжина вхідного масиву – 9 однобайтових елементів. Результат розмістити за адресою 40Н у зовнішній пам'яті даних.

Варіант 10. Написати програму, яка виконує сортування елементів масиву, розташованого в пам'яті програм з адреси 50Н. Елементи, що більше 30Н, зберігаються у внутрішній пам'яті даних з адреси 20Н, інші – в зовнішній пам'яті даних з адреси 60Н. 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 використання теорії цифрових автоматів для формалізації написання програм комбінаційних автоматів табличним способом.

Вихідні дані. Вхідні сигнали для семисегментного індикатору надходять на розряди порту P1.0-P1.3, а керування індикатором здійснюється з порту 2 через буферний підсилювач.

Написати програму перекодування двійкового значення, які надходить з порту P1, у значення, що відповідають цифрам шістнадцяткового алфавіту, котрі відображаються на семисегментному індикаторі. Підключення виходів порту P2 до індикатору зображено на рис. 3.3.

Теоретичні відомості. Дешифратори будуються на основі масивів вихідних сигналів, розташовуваних у пам'яті програм. Звертання до потрібного елемента масиву здійснюється заданням зсуву відносно до початкової адреси масиву.

Розв'язок: Запишемо таблицю істинності для дешифратора (табл. 10.1).

Таблиця 10.1 – Таблиця 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 табл. 10.1 (мітка BEGDIM), які потрібно вивести на індикатор для формування шістнадцяткових цифр від 0 до F. Текст підпрограми, яка реалізує дешифратор, наведено 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 й 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).

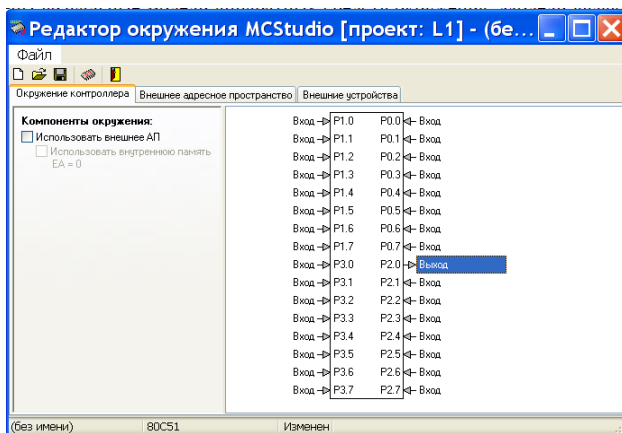


Рисунок 10.14 – Вікно налаштування оточення контролера

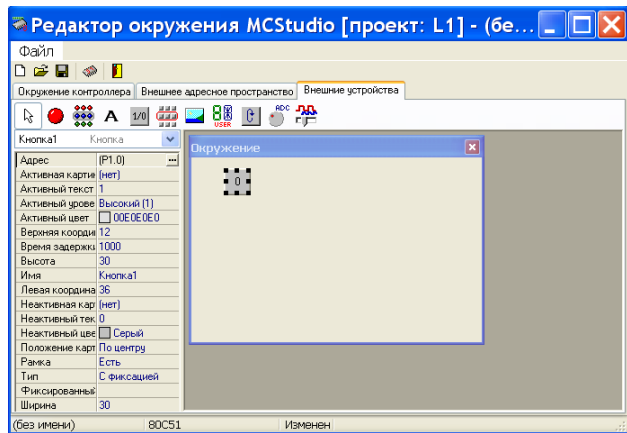


Рисунок 10.15 – Підключення зовнішніх пристроїв до контролера

3. Переходимо на вкладку «Зовнішні пристрої» і встановлюємо у вікні оточення чотири кнопки й семисегментний індикатор зі списку елементів.

4. Активізуємо по черзі кожний елемент, заповнюємо форму підключення його до контролера. Наприклад: клікнувши на кнопку в колонці, що з'явилася ліворуч, натискаємо на кнопку у 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 повинен містити текст індивідуального задання й текст програми.

Задання:

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 в 3 лабораторній роботі, має наступні обмеження: вхідні сигнали повинні бути підключені 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 нерационально використовується пам'ять програм. Тому пропонується інший спосіб розв'язку задання керування, заснований 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

Приклад задання алгоритму у вигляді таблиці істинності.

Таблиця 10.2 –

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.

$$P1.7 = !P1.2 \& P1.4 \& P2.1 \& P2.3 + P1.2 \& !P1.4 \& P2.1 \& !P2.3 + P1.2 \& !P1.4 \& P2.1 \& !P2.3 \quad (3.1)$$

Програмна реалізація для першої функції «І» рівняння має вигляд:

JV P1.2,M1

JNB P1.4,M1

JNB P2.1,M1

JNB P2.3,M1

SETV P1.7

LJMP EXIT

M1: * * * ;перевірка наступної умови

Схема алгоритму для функцій «АБО» наведена на рис. 4.12 у розділі 4.

Повна програма логічного рівняння, відповідного до таблиці істинності, має 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; установка виходу в одиницю
 LJMPLoop; 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; установка виходу в одиницю
 LJMPLoop; 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; установка виходу в одиницю
 LJMPLoop; і вихід
 M3: CLR P1.7; скинути вихід у нуль – не виконалася жодна з умов
 LJMPLoop; і вихід

Методика виконання:

1. Заносимо програму у вікно редактора програми.
2. Контроль над роботою програми зробимо за допомогою редактора оточення. Для цього підключаємо на входи відповідні до задання кнопки, а на виходи – світлодіоди.
3. Правильність складання умов появи вихідних сигналів перевіряється 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 автомату згідно варіанта заданого табл. 10.3. Відлагодження програми виконувати з використанням «Середовища оточення».

Звіт повинен містити текст програми, вхідні дані й блок-схему алгоритму.

Таблиця 10.3 –

Варіант 1	Комбінація натискання 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	Комбінація натискання 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	1	1	1	0	0
Кодова комбінація 2	0	1	1	0	0	1	1	0
Варіант 3	Комбінація натискання 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	Комбінація натискання 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	1	1	1
Кодова комбінація 2	0	1	0	0	0	1	1	0
Варіант 5	Комбінація натискання 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	1	1	0	0	1
Кодова комбінація 2	0	1	0	1	0	1	0	0
Варіант 6	Комбінація натискання 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	1	1	0	0	1
Кодова комбінація 2	0	1	1	0	0	1	1	0
Варіант 7	Комбінація натискання 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	1	0	1
Кодова комбінація 2	1	1	0	0	0	1	1	0
Варіант 8	Комбінація натискання 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	1	1	1	0	0
Кодова комбінація 2	0	1	1	0	0	1	1	0
Варіант 9	Комбінація натискання 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	1	0	1	0	0	1
Кодова комбінація 2	0	1	1	0	0	0	1	0
Варіант 10	Комбінація натискання 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	1	1	0	0	1
Кодова комбінація 2	0	1	0	1	0	1	1	0
Варіант 11	Комбінація натискання 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	1	1	0	0	1
Кодова комбінація 2	0	1	0	0	0	0	1	1

Варіант 12	Комбінація натискання 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	1	1	0	0	1
Кодова комбінація 2	0	1	1	0	0	0	1	0
Варіант 13	Комбінація натискання 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	0	0	1	0	1	1	1	0
Кодова комбінація 4	0	1	0	1	0	1	1	1
Варіант 14	Комбінація натискання 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	0	1	1	0	1	1	1	0
Кодова комбінація 4	1	0	0	1	0	1	1	1
Варіант 15	Комбінація натискання 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	Комбінація натискання 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	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 й розраховувати значення змінних для програмних тимчасових витримок.

Теоретичні відомості. Тимчасові витримки в мікропроцесорних системах легше всього організувати через програмні лічильники. У якості лічильників 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 1 ц

Mov r3,#47h ; завантаження старшого байта затримки 1 ц

Lcall zd1; виклик підпрограми затримки 2 ц

M1: ajmp M1; нескінченний цикл

zd1: mov a,r2; пересилання вхідного значення в акумулятор 1 ц вик. r3 раз

mov r4,a; пересилання вхідного значення в r4 з акумулятора 1 ц вик. . r3 раз

m2: djnz r4,h,m2 ; декремент молодшого байта затримки 2 ц вик. r2*r3 раз

djnz r3,zd1; декремент старшого байта затримки 2 ц вип. r3 раз

ret ;вихід з підпрограми 2 ц

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 наступною формулою

$$\begin{aligned} T_{зд} &= 1ц + 1ц + 2ц + 1ц * r_3 + 1ц * r_3 + 2ц * r_2 * r_3 + 2ц * r_3 + 2ц = \\ &= 1 + 1 + 2 + 71 + 71 + 2 * 37 * 71 + 2 * 71 + 2 = 5544 \text{ [мкс]}. \end{aligned}$$

Методика виконання

1. Контроль витримки затримки проводиться 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.

3. Для контролю виконання програми в автоматичному режимі використовувати крапку зупинки, встановлену на мітці M1. Для задання крапки зупинки необхідно маркер помістити на потрібний рядок програми й натиснути клавішу F5. Значення часу виконання можна дивитися за допомогою вікна Таймінг див. рис.1. Після першої зупинки програми на вікні Таймінг натискаємо кнопку Запуск, і після повторної зупинки бачимо кількість циклів та час виконання фрагменту кода.

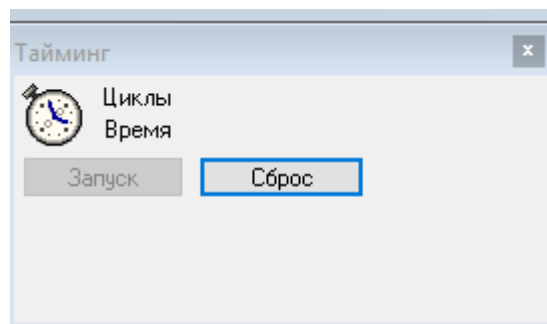


Рисунок 10.16 – Вікно Таймінг

4. Задається крапка зупинки. Програма запускається на симуляцію в автоматичному режимі після переривання виконання (в крапці зупинки), час виконання не повинен відрізнятися від встановленого в заданні більш ніж на 5 мкс.

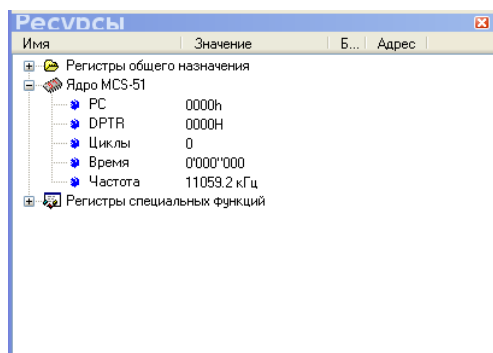


Рисунок 10.17 – Вікно ядра MCS-51

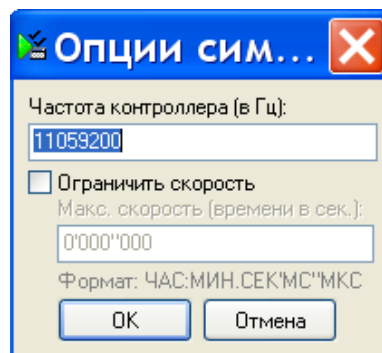


Рисунок 10.18 – Вікно установки частоты генератора микроконтроллера

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, додаємо ще один цикл у програмі. З урахуванням цього коректуємо формулу й знаходимо потрібні значення.

Варіанти задань

Час затримки повинен дорівнювати номеру за списком, що помножений на 27 мілісекунд.

Звіт повинен містити текст програми, вхідні дані й 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 відлагоджувачі 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). Задання режиму проводиться за допомогою занесення керуючого слова в регістр TMOD. За режим TMR0 відповідають біти D1, D0.

Для формування необхідного коефіцієнта ділення заносимо в регістри ТН0, ТL0 число, що дорівнює різниці 65535 і значенню необхідного коефіцієнта ділення.

Для дозволу роботи TMR0 встановлюємо в регістрі TCON.4 логічну «1».

Для дозволу проходження переривання від TMR0 у регістрі ІЕ.1 і ІЕ.7 необхідно встановити «1».

Для організації тимчасових затримок з використанням відмітника часу використовуємо програмно організовані лічильники, розташовувані в пам'яті даних. Для запуску затримки в комірку 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 Асемблера, що реалізує миготіння світлодіоду порту P1.0, яка здійснює керування двома затримками. Лічильники затримок організовано в 31 і 32 комірках пам'яті даних.

```

ljmp m0          ; обходимо вектор переривання
org 0bh         ; вектор переривання від таймера TMR0
push acc
mov th0,#3ch   }
mov tl0,#0b0h } ; налаштування таймера на період в 50 мс
djnz 30h,m2    }
mov 30h, #20  } ; організація постдільника
mov a,31h     }
jz m3         } ; організація першої затримки
dec a
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 ; установка таймера 0 у режим 1
mov th0,#0ffh }
mov tl0,#0e0h }
setb tcon.4   } ; дозвіл роботи TMR0 і проходження
setb ie.7     } ; переривання від TMR0
setb ie.1
mov 30h,#20
On: setb P1.0 ; включення світлодіоду

```

```

        mov a,#03h
        mov 31h,a ;запуск затримки 1 на 3з
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 включення
        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 проводиться перезавантаження таймера, яке забезпечує час до наступного переривання, що дорівнює 50 мс. У комірці 30h організовано програмний постдільник на 20. Таким чином, за допомогою постдільника одержуємо період відмітчика 1 с. Така змінна дуже зручна, якщо необхідно робити періодичні дії з періодом 1 с. У комірках 31 h і 32 h організовуємо затримки від 1 до 255 с.

В основній програмі (мітка m0) робимо ініціалізацію таймера й переривання від нього, після чого запалюємо світлодіод, підключений до біта 0 порту 1, і за допомогою затримки 1 залишаємо увімкненим його на 3 с. Після цього гасимо його й залишаємо вимкненим на 3 с за допомогою затримки 2. Командою LJMP On зациклюємо програму на миготіння.

Установивши крапки зупинки на рядках з мітками On і Off, у моменти зупинки програми в вікні Таймінг можна бачити, що зміна стану світлодіоду відбувається кожні 3000000 мкс.

Методика виконання

Заносимо програму у вікно її редактора.

Контроль над роботою програми здійснюємо за допомогою редактора оточення. Для цього підключаємо на вихід P1.0 світлодіод.

Установлюємо крапки зупинки відповідно до пояснень до програми.

Створюємо вікно для спостереження за змінними в процесі налагодження. Для цього в меню вибираємо пункти Вид – Перегляд значень. Ініціюємо правою кнопкою миші й додаємо у вікно реєстри, що цікавлять нас, і комірки пам'яті (th0, tl0, 30h, 31h, 32h) (рис. 10.20). Через вікно створення перегляду й конструктор виразів.

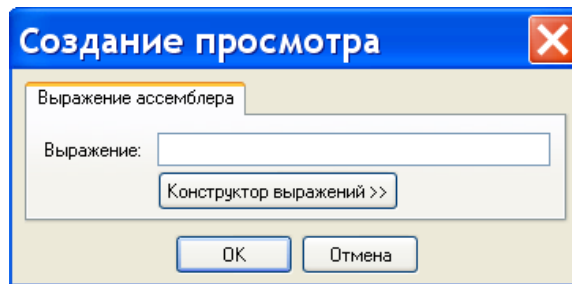


Рисунок 10.19 – Вікно для заповнення вікна налагодження

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, виконання програми до першого переривання робимо симуляцією в покроковому режимі. Для цього перший цикл переривання при ініціалізації виконаний укороченим.

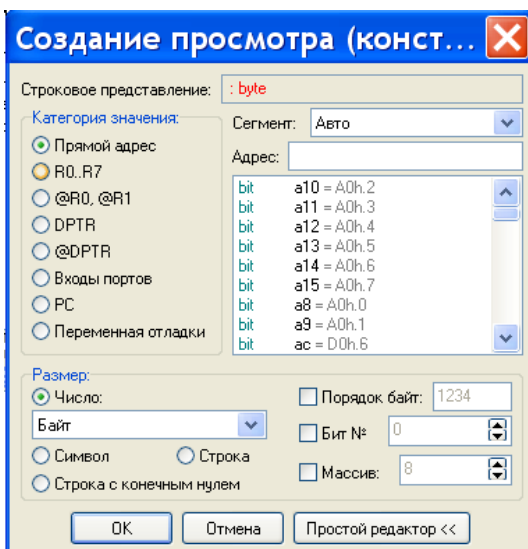


Рисунок 10.20 – Вікно конструктора виразів

Рисунок 10.21 – Вікно перегляду значень

Запускаємо симуляцію в автоматичному режимі й контролюємо час зупинки за часом виконання в ядрі MCS-51.

Звіт повинен містити: тексти програми та вихідного задання.

Задання

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 тині, що біжить, зліва направо.

Варіант 3. Реалізувати період миготіння 2 с результату комбінації, що спрацювала, для лабораторної роботи 4.

Варіант 4. Реалізувати рахунок 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 тині, що біжить, справа наліво.

Варіант 7. Реалізувати рахунок 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 натискання кнопки, а виключає, коли відпускають кнопку.

Варіант 9. Реалізувати програму, яка включає світлодіод, після натискання кнопки, а виключає через 5 с після відпускання кнопки.

Варіант 10. Реалізувати програму, яка блимає світлодіодом з періодом 3 с, доки натиснута кнопка, а виключає, коли відпускають кнопку.

10.8. Лабораторна робота № 7

Дослідження послідовного інтерфейсу в мікропроцесорних системах

Мета роботи: навчитись 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 восьми розрядній 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 регістру. Швидкість передачі задається за допомогою таймера 1, що працює в другому режимі без переривань.

Швидкість передачі визначається за формулою:

$$C = \left(2^{SMOD} / 32\right) (f_{osc} / 12) / (256 - (TH1)),$$

де f_{osc} – частота кварцового резонатора мікроконтролера.

Текст програми мовою Асемблера, що реалізує передачу масиву даних, які розташовуються у внутрішній пам'яті даних з адреси 30H довжиною 10 байтів. Після передачі проводиться приймання відповідних даних у масив, що розташовується у внутрішній пам'яті даних з адреси 50H довжиною 10 байтів. Швидкість обміну даними – 19200 біт/с, у режимі 9 бітів змінювана швидкість обміну.

```
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 проводиться в оброблювачі переривання. Зчитування прийнятих байтів із приймача виконується так само 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 приймання даних із прив'язкою до часу тактування мікроконтролера. Вікно симулятора УСАПП зображено на рис. 10.22. Крім цього, у вікні можна сформулювати повідомлення для приймача й 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».

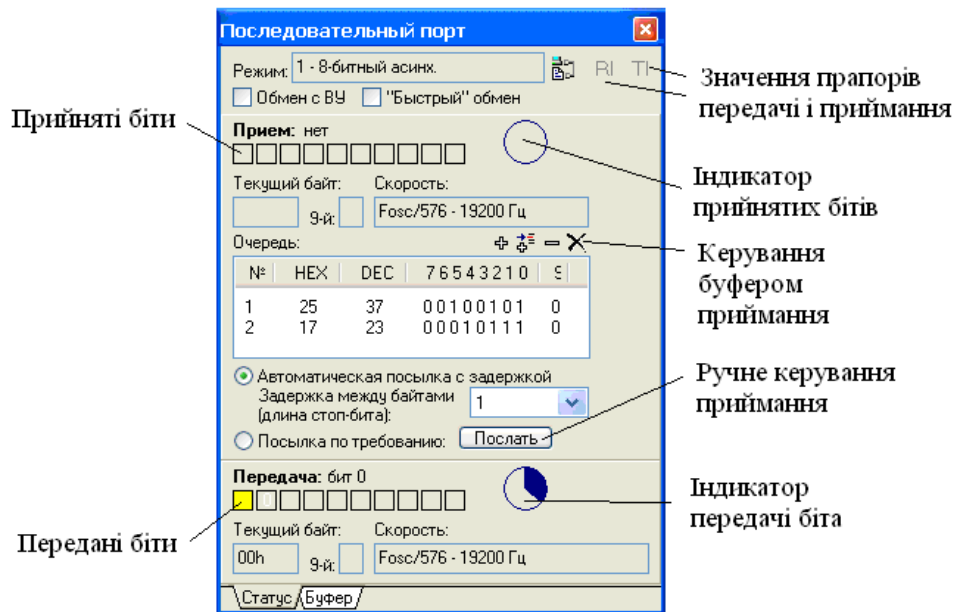


Рисунок 10.22 – Вікно симулятора послідовного обміну

Звіт повинен містити: текст програми, вихідні дані й блок-схему алгоритму.

Задання. Організувати обмін інформацією мікроконтролера з іншим мікроконтролером. Переданий масив розташовується в пам'яті програм з адреси й довжиною, що зазначено в таблиці, згідно варіанта. Прийняті дані треба розмістити у внутрішній пам'яті з адреси й довжиною, що зазначена в таблиці, згідно варіанта. В оброблювачі переривань треба використовувати банк регістрів загального призначення згідно варіанта. Швидкість обміну даними – згідно варіанта.

Таблиця 10.4 – Індивідуальні задання 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