
СПЕЦІАЛІЗОВАНІ МІКРОКОНТРОЛЕРНІ СИСТЕМИ Теорія і практика
СПЕЦІАЛІЗОВАНІ МІКРОКОНТРОЛЕРНІ СИСТЕМИ
Теорія і практика
Підручник

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Є.І. Сокол, І.Ф. Домнін, О.М. Рисований,

В.В. Замаруєв, О.В. Єресько

СПЕЦІАЛІЗОВАНІ МІКРОКОНТРОЛЕРНІ СИСТЕМИ. ТЕОРІЯ І ПРАКТИКА

Підручник

Затверджено Міністерством освіти і науки України
як підручник для студентів
вищих навчальних закладів

Харків НТУ «ХПІ» 2007

ББК 32.844.1
С71
УДК 621.38: 004.38

Рецензенти:
І.О. Фурман, д-р техн. наук, проф., академік АН Вищої школи України, зав. кафедри автоматизації й комп'ютерних технологій ХДТУСГ,
Л.Є. Бахнов, канд. техн. наук, зав. відділом НДІ ХЕМЗ

Гриф надано Міністерством освіти і науки України,
лист № 14/18/2-2089 від 20.09.2005 р.

Спеціалізовані мікроконтролерні системи. Теорія і практика : **С71** Підручник / Є. І. Сокол, І. Ф. Домнін, О. М. Рисований та ін. – Харків: НТУ "ХПІ", 2007. – 252 с.

ISBN 978-966-593-551-3

Розглянуто факти з історії виникнення сучасних обчислювальних приладів, наведено принципи організації мікроконтролерних систем, особливості проектування пристроїв та їх використання, розглянуто систему команд і засоби програмування мікроконтролерів сімейства MCS-51, організацію взаємодії мікроконтролерів з об'єктами управління, дано рекомендації щодо тестування та налагодження програм. Матеріал підручника ілюстровано великою кількістю прикладів програм, що написані мовою асемблера.

Призначено для студентів, які вивчають навчальні дисципліни "Спеціалізовані мікроконтролерні системи" та "Мікроконтролерні системи", а також може бути корисним усім, хто займається розробкою та експлуатацією сучасної електронної техніки.

Іл. 78. Табл. 32. Бібліогр. 46 назв.

ББК 32.844.1

ISBN 978-966-593-551-3

ВСТУП

Однокристальні мікроконтролери (ОМК) являють собою спеціалізовані мікро-ЕОМ, що розроблені для використання в управляючих пристроях, системах передачі даних і системах управління технологічними процесами та виготовлені на одному напівпровідниковому кристалі [1]. Для зв'язку із зовнішнім середовищем та підрахунку інтервалів часу ОМК мають вбудовані периферійні вузли і пристрої, набір яких багато в чому визначає їх функціональні можливості та галузі застосування. Більше двох третин світового ринку мікропроцесорних засобів в цей час складають саме ОМК.

Поява мікропроцесорів ознаменувала собою початок нової ери не тільки в обчислювальній техніці, але і в галузі систем вбудованого управління об'єктом, ери високопродуктивних і надійних мікроконтролерних (МК) систем управління, що інтегровані в об'єкт управління. Вони застосовуються в промислових роботах, новітніх перетворювачах електричної енергії, сучасних транспортних заходах, системах зв'язку, побутовій техніці тощо. Без спеціалізованих МК не зміг би працювати жоден з периферійних пристроїв персональної ЕОМ, таких, як клавіатура, "миша", жорсткий диск, принтер, сканер та інші.

Особливості розробки МК-систем та їх програмування пов'язані з обмеженими обчислювальними ресурсами МК, оскільки апаратура повинна коштувати якомога дешевше. Програміст повинен добре знати не тільки особливості архітектури МК, але й фізику роботи виробу, в якому він використовується, оскільки програми, що розробляються, мають працювати в реальному часі.

На сьогодні серед великої кількості типів систем управління МК-системи дістають все ширшого застосування. Розширення галузі застосування ОМК обумовлено рядом наявних переваг:

- можливістю реалізації різних задач управління;
- легкістю модифікації алгоритмів управління;
- мають малі габарити;
- низьким енергоспоживанням ;
- високою надійністю;
- низькою вартістю.

Для управляючих задач найчастіше використовуються 8-розрядні МК. Виробництвом МК цього класу зараз займається багато фірм: Atmel, Intel, Infineon, MicroChip, Motorola, Philips та ін., але на серйозну увагу заслуговують МК фірм: Atmel, Intel, MicroChip, Motorola, які мають відмінні від інших концепції в розробці контролерів.

Кожна з цих фірм має широку номенклатуру МК, адаптованих для задач управління з певним набором периферійних пристроїв, необхідних для ефективнішого вирішення цих задач. Але ні один з цих МК не набув такого поширення, як мікроконтролери на основі ядра Intel 8051 – сім'я MCS-51[1]. Нижче наводиться список фірм-виробників мікроконтролерів з 8051 сумісним ядром і мікроелектронних пристроїв на його основі. У дужках наведена кількість функціонально різних виробів, що випускаються фірмою на момент написання книги.

Acer Labs (2), Actel (1), Aeroflex UTMC (1), Altium (1), Analog Devices (14), AnchorChips (дивись Cypress Semiconductor), Atmel (92), Atmel Wireless & uC (дивись Atmel), Cast, Inc. (4), Chipcon (5), CML Microcircuits (1), Cybernetic Micro Systems (1), CybraTech (2), Cygnal Integrated Products (дивись Silicon Laboratories Inc.), Cypress Semiconductor (5), Daewoo (2), Dallas Semiconductor (26), Digital Core Design (12), Dolphin (4), Domosys (1), Easyplug (2), Evatronix (2), Genesis Microchip (2), Goal Semiconductor (15), Handshake Solutions (1), Honeywell (1), Hynix Semiconductor (39), Hyundai (дивись Hynix Semiconductor), Infineon (50), InnovASIC (2), Intel (55), ISSI (10), Maxim (2), Megawin (16), Mentor Graphics Co. (4), Micronas (4), MXIC (6), Myson Technology (8), Nordic Semiconductor (3), NXP (190), OKI (3), Oregano Systems (1), Philips (дивись NXP), RDC Semiconductor (5), Sanyo (1), Sharp (1), Siemens (дивись Infineon), Silicon Laboratories, Inc. (85), Siliconians (1), SMSC (17), SST (13), STMicroelectronics (25), SyncMOS (13), Synopsys (1), Syntek Semiconductor Co., Ltd. (2), TDK (дивись Teridian Semiconductor Corp.), Temic (дивись Atmel), Teridian Semiconductor Corp. (6), Tezzaron Semiconductor (3), TI (33), Triscend (4), Vitesse (1), Winbond (61), Zensys (2),

[1] MCS-51 та мнемонічні імена команд є власністю Intel Corporation. Інші назви є зарегістрованими торговими марками відповідних фірм.

Zylogic Semiconductor Corp. (4). Разом **57** фірм виробляють **868** функціонально різних пристроїв з використанням 8051 сумісного процесорного ядра. У цю кількість не входять мікроконтролери, що вироблені підприємствами СНД.

Сім'я 8-розрядних ОМК MCS-51 з'явилася на світовому ринку на початку восьмидесятих років минулого століття функціонально завершеними мікроЕОМ гарвардської архітектури. Один з основних принципів цієї архітектури полягає в логічному розділенні адресних просторів пам'яті програм і пам'яті даних. Перші модифікації кристалів були виконані за високоякісною n-МОП (HMOS) технологією. З розвитком напівпровідникової технології подальші версії мікросхем MCS-51 стали виготовляти за досконалішою і низькоспоживною КМОП (CHMOS) технологією (в активному режимі споживання кристалів було доведене до 10–50 мА). Функціональні можливості вбудованих периферійних пристроїв були достатніми для вирішення більшості задач управління:

- два 16-розрядних таймери-лічильники;
- апаратний послідовний дуплексний порт;
- дворівнева система переривань;
- чотири 8-бітових порти вводу-виводу;
- режими енергозбереження.

Система команд MCS-51 містить бітово-орієнтовані операції, команди множення та ділення, розширену групу команд передачі управління. Спільно з апаратною підтримкою доступу до одиничних бітів портів, регістрів, області в пам'яті даних це дає можливість говорити про реалізацію на кристалі бітового процесора і орієнтацію МК на виконання задач управління.

Сумарний розмір адресного простору зовнішньої пам'яті програм і даних становить 128 кбайт. 16-розрядні регістри лічильника команд (Program Counter) і покажчика даних (Data Pointer) дозволяють прямо звертатися до всього діапазону адрес, що дало розробникам можливість реалізації алгоритмів швидкої обробки великих масивів даних.

Біти керування всіма програмно-доступними вузлами МК було зведено в спеціальну область пам'яті даних – регістри спеціальних функцій (Special Function Register), що дозволило звертатися до них так само, як до звичайних даних у комірках резидентного оперативного запам'ятовуючого пристрою (ОЗП).

У пізніших модифікаціях МК, що випускаються багатьма фірмами, удосконалення йшло шляхом нарощення додаткових функціональних можливостей із збереженням повної програмної сумісності з попередніми версіями.

Особливостями останніх модифікацій МК сім'ї MCS-51 є:

- повністю статичний дизайн (мінімальна тактова частота 0 Гц);
- напруга живлення 1,5–5 вольт;
- широкий спектр вбудованих периферійних пристроїв;
- можливість оперативного перепрограмування;
- максимальна тактова частота – 40 МГц.

Для підготовки математичного забезпечення використовуються як мова асемблера, так і мови високого рівня - Сі, Паскаль, які полегшують сприйняття програми людиною, дозволяють спростити перенесення програм між МК різних виробників. За роки існування сім'ї MCS-51 для нього підготовлено безліч прикладних програм, створено інтегровані середовища розробки, ряд компіляторів, бібліотеки стандартних підпрограм. Іноземними і вітчизняними фірмами випускаються програмні емулятори і засоби макетування МК-систем.

Аналіз сьогоднішнього стану МК з ядром Intel 8051 (МК-51) переконливо доводить, що ці мікроконтролери динамічно розвиваються фірмами-виробниками: Intel продовжує їх випуск і вдосконалення, Philips виробляє декілька десятків найрізноманітніших мікроконтролерів з підвищеною швидкодією, Dallas Semiconductor має версії МК-51, у яких еквівалентна тактова частота (в порівнянні з стандартним МК Intel 8051) складає 99 МГц, Standard Micro (SMC) інтегрувала в одній мікросхемі для індустріальних споживачів ядро 8051 і повний контролер мережі Arcnet, МК фірм Cypress і Atmel підтримують інтерфейс USB, Triscend створила МК з вбудованою програмованою логічною матрицею, Analog Devices розширила можливості 8051 мало не до рівня цифрового сигнального процесора, Atmel об'єднала ядро 8051 і декодер MP3 файлів, НВО «Кристал» випускає МК з напругою живлення 1,5 В, виготовляються версії МК-51 для використання в спецапаратурі, з підвищеною температурною та радіаційною стійкістю.

Незважаючи на достатню "старовину" сім'ї (більше 25 років) і появу на світовому ринку за останні роки ОМК більшої продуктивності і вдосконаленої архітектури, контролери MCS-51 ще довго використовуватимуться в порівняно простих вбудованих системах управління.

Підручник укладено за матеріалами лекційних курсів з дисциплін, пов'язаних з вивченням мікроконтролерних систем та їх використанням, що читаються в Національному технічному університеті "Харківський політехнічний інститут". Матеріал може використовуватися при курсовому і дипломному проектуванні. Книга також може бути корисною для аспірантів і спеціалістів, що працюють у галузі електроніки.

1. КОРОТКА ІСТОРІЯ ВИНИКНЕННЯ СУЧАСНОГО КОМП'ЮТЕРА Й ДЕЯКИХ ПОВ'ЯЗАНИХ З НИМ ТЕРМІНІВ

Перші обчислювальні машини мали незмінні програми. У деяких дуже простих комп'ютерах все ще використовується цей підхід. Наприклад, калькулятор, у принципі, – це комп'ютер з фіксованою програмою. Він може виконувати дії елементарної математики, але не може виконати незаплановану операцію, використовуватись як текстовий процесор або управляти відеоіграми. Для зміни програми такої машини необхідно змінити взаємне з'єднання обчислювальних блоків, поміняти структуру або навіть перепроектувати систему. Перші комп'ютери були не запрограмовані, а скоріше розроблені для виконання конкретних завдань. "Перепрограмування", якщо це було взагалі можливо, було ручним процесом, що супроводжувався детальними інженерними розробками і потім – внесенням фізичних змін у конструкцію.

Пізніше процес обчислення був визначений як виконання ряду інструкцій – **програма** й створена **архітектура набору команд** – опис аспектів комп'ютерної архітектури, що використовуються програмістом, включаючи типи даних, інструкції, регістри, способи адресації, архітектуру пам'яті, обробку переривань і винятків, операції введення-виведення і т.д. Використання поняття «програма» дозволило зробити обчислювальну машину більш гнучкою.

1.1. Становлення комп'ютерної архітектури

Ера сучасних обчислювальних систем почалася до Другої світової війни, коли вакуумні лампи, реле й інші електронні кола замінили механічні

еквіваленти й аналогові обчислення були замінені цифровими. Комп'ютери, розроблені й створені в той час називають «першим поколінням» комп'ютерів. Спеціалісти з розробки комп'ютерів внесли великий вклад у розвиток архітектури обчислювальних систем: організацію пам'яті й системи адресації, засобів інтерфейсу користувача та операцій введення-виведення, системи команд і користувальницьких можливостей програмування.

У тридцяті – сорокові роки 20-го століття розвиток комп'ютерів йшов трьома паралельними потоками, інформація про два з них значною мірою ігнорувалася або була засекречена. Першим напрямком була робота німецького інженера Конрада Цузе (Konrad Zuse), який в 1941 році розробив комп'ютер **Z3**. В 1945 році Z3 був знищений під час бомбувань у Берліні. Це був перший електромеханічний комп'ютер загального застосування. Він використовував двійкову математику й був повністю програмованим за допомогою перфострічки, але використовував реле для виконання всіх функцій і тому не був електронною обчислювальною машиною.

Другим напрямком була секретна розробка на дослідницькій станції Поштового відомства Великобританії комп'ютера **Colossus**. Він виконував роботу з декодування шифрованих повідомлень в Блетчлі-Парк, графство Бакінгемпшир з 1943 року. Colossus був зруйнований в 1945 році за наказом Уїнстона Черчілля (Winston Churchill), і сам факт його існування залишався секретним до 70-х років 20-го століття.

Жоден з цих комп'ютерів не мав великого впливу на подальший розвиток обчислювальних пристроїв. Основний внесок у розробку першого покоління комп'ютерів зробили американські вчені.

7 серпня 1944 року в Гарвардському університеті був офіційно встановлений **IBM ASCC** (**Automatic Sequence Controlled Calculator – автоматичний обчислювач з послідовним управлінням**). Перший у США автоматичний комп'ютер був розроблений Говардом Айкіним (Howard H. Aiken) і створений фірмою IBM. В університеті його назвали Марк-1 (у дійсності на корпусі було написано: Aiken-IBM Automatic Sequence Controlled Calculator Mark I) [2]. Це був електромеханічний комп'ютер на основі реле, вимикачів, валів і муфт зчеплення, 16 м довжиною, 2,4 м висотою й 0,6 м шириною, його вага досягала 4500 кг. Для синхронізації «центрального процесора» використовувався механічний привід від 15-метрового вала, руху якому надавав електричний двигун потужністю 4 кВт. Уведення інструкцій здійснювалося з перфострічки – спочатку виконувалася одна, а потім зчитувалася наступна 24-бітна команда. Команди умовних переходів були відсутні, а цикли реалізовувалися буквально – повторним уведенням перфострічки. Марк-1 міг збе-

рігати 72 23-х розрядні десятеричні числа у фіксаторах (тригерах) на основі реле. Швидкодія становила 3 додавання (вирахування) у секунду, множення займало 6, а ділення – 15,2 секунди, обчислення тригонометричних функцій або логарифмів вимагало більше хвилини. Архітектура Марк-1 згодом була названа Гарвардською.

Термін **Гарвардська архітектура** стосується архітектури обчислювальної системи, що використовує фізично окрему пам'ять і шини сигналів для команд і даних [3].

Першим великомасштабним електронним комп'ютером з можливістю перепрограмування був **ENIAC (Electronic Numerical Integrator and Computer - Електронний Числовий Інтегратор і Комп'ютер)** [4]. Він був розроблений і побудований для науково-дослідної лабораторії балістики армії США для розрахунку балістичних таблиць для нових гармат. Рішення про будівництво комп'ютера в рамках цілком секретного проекту "PX" було прийнято 17 травня 1943 року. Розробка велася в Школі електротехніки Мура (Moore School of Electrical Engineering) Джоном Вільямом Маучлі (John William Mauchly) – розробка концепції й Джоном Проспером Еккертом (J. Presper Eckert) – розробка реалізації. ENIAC був уведений в експлуатацію 14 лютого 1946 року в університеті штату Пенсільванія, потім його було модернізовано й передано у дослідницький центр (Aberdeen Proving Ground) у штаті Меріленд. Постійно експлуатувався до 23:45 2 жовтня 1955 року. Перші розв'язані завдання стосувалися розробки водневої бомби.

Фізично ENIAC складався з 17468 вакуумних ламп, 7200 кристалічних діодів, 1500 реле, 70000 резисторів, 10000 конденсаторів і приблизно 5 мільйонів паяних з'єднань. Він важив 27 тонн і мав розміри 2,4 м на 0,9 м на 30 м, займав площу 167 квадратних метрів і споживав потужність 150 кВт. Введення і виведення інформації провадилося за допомогою перфокарт. Ці карти могли використовуватися для виводу даних на друк з автономним використанням рахункової машини IBM 405.

Деякі експерти пророкували, що відмови ламп будуть настільки частими, що машина ніколи не запрацює. Ці пророкування частково виправдалися: деякі лампи згоряли щодня, залишаючи машину непрацездатною половину часу. Більшість відмов траплялося під час вмикання або вимикання. Інженери дійшли висновку, що доцільно ніколи не виключати машину, це разом з появою більш надійних ламп знизило їх відмови до однієї за два дні. Найбільш тривалий інтервал роботи склав 116 годин.

Для зберігання цифр ENIAC використовував десятирозрядні кільцеві лічильники. Ідея арифметичних обчислень складалася в електронному повторенні числових коліс механічної рахункової машини.

Основний тактовий цикл комп’ютера дорівнював 200 мікросекунд, або 5000 циклів у секунду для операцій з десятизначними числами. За один цикл ENIAC міг записати число в регістр, прочитати число з регістра або скласти чи відняти два числа. Операції додавання (вирахування) могли виконуватися паралельно, що підвищувало швидкодію. Була передбачена можливість роботи з числами подвоєної точності. Множення числа з 10 цифрами на число з d-цифрами (для d до 10) використовувало d+4 цикли, таким чином, множення десятизначного числа на десятизначне займало 14 циклів, або 2800 мікросекунд - продуктивність 357 операцій у секунду. Ділення і добування квадратного кореня займало 13*(d+1) циклів, де d - число цифр у результаті (частка або квадратний корінь). Операція ділення або добування квадратного кореня виконувалася за час до 143 циклів, або 28600 мікросекунд, продуктивність - 35 операцій у секунду. Таку продуктивність забезпечували апаратні блоки виконання математичних операцій.

Під час розробки було ясно, що зчитування команд із перфокарт або перфострічки не буде досить швидким, у той час як виконання інструкцій передбачалося зі значно більшою швидкістю. Тому програма була зв’язана зі структурою комп’ютера, і для кожного нового завдання була необхідна зміна з’єднання обчислювальних блоків, що займало кілька днів. ENIAC міг бути запрограмований на виконання послідовних або циклічних математичних дій, операцій введення-виведення даних і умовних переходів. Хоча під час розробки ENIAC ідея комп’ютера із програмою, що зберігається (stored-program computer), уже була відома, але в його першій версії вона не була реалізована. Пріоритети воєнного часу вимагали швидку обчислювальну машину і 20 комірок пам’яті для зберігання даних і програми було замало.

При модифікації 1948 року в ENIAC увели найпростіший механізм зберігання програми, що допускав тільки її читання, програмний лічильник і деякі інші вдосконалення. Ці нововведення знизили швидкість роботи комп’ютера в шість разів, але прискорили процес його перепрограмування із днів до годин і були визнані корисними. З 1948 року ENIAC став комп’ютером з програмою, що зберігається.

Під час очікування створення конструкції ENIAC Маучлі й Еккерт, що знали про обмеження цієї обчислювальної машини, почали планувати створення EDVAC.

EDVAC (**Electronic Discrete Variable Automatic Computer – Електронний автоматичний комп’ютер з дискретними змінними**) – один з перших комп’ютерів, що, на відміну від ENIAC, оперував двійковими числами [5].

Архітектура EDVAC була розроблена ще до введення ENIAC в експлуатацію. Вона повинна була вирішити безліч проблем, виявлених при проектуванні й виготовленні ENIAC. До січня 1945 року був забезпечений контракт на проектування комп'ютера з програмою, що зберігається. Эккерт запропонував для зберігання програми й даних використати пам'ять із вживанням ліній ртутної затримки. Пізніше до Маучлі й Еккерта приєднався математик Джон фон Нейман (John von Neumann, за народженням Neumann János Lajos). Контракт на виробництво виробу з найменуванням Electronic Discrete Variable Automatic *Calculator* між Артилерійським відділом армії США й Електротехнічною школою Мура в університеті Пенсільванії був підписаний у квітні 1946 року. Основною ідеєю контракту був баланс надійності й економічності, однак кінцева вартість як EDVAC, так і ENIAC становила більше $500,000.

EDVAC був установлений у дослідницькій лабораторії балістики в серпні 1949 року. Два роки тривали роботи з доведення комп'ютера, і в 1951 році він був зданий в експлуатацію. Після модернізації пристроїв читання/запису перфокарт (1953 р.), установлення додаткової пам'яті на магнітних барабанах (1954 р.) і арифметичного пристрою для чисел з плаваючою точкою (1958 р.) він проробив до 1961 року по 20 годин на день із середнім часом безвідмовної роботи - 8 годин.

Комп'ютер мав більше 6000 електронних ламп, 12000 діодів і споживав потужність 56 кВт, займав площу 45,5 квадратних метрів і важив 7850 кг. Він був розрахований на операції з двійковими числами: апаратне додавання, вирахування, множення. Ємність пам'яті становила 1000 44-бітних слів. Операція додавання виконувалася за 864 мікросекунди, множення - за 2900 мікросекунд. Архітектура комп'ютера EDVAC стала відома як архітектура фон Неймана.

Термін **архітектура фон Неймана** стосується архітектури обчислювальної системи, що використовує загальну структуру зберігання інструкцій і даних [6].

Перший універсальний програмований комп'ютер у континентальній Європі був створений групою вчених під керівництвом Сергія Олексійовича Лебедєва в Київському Інституті електротехніки (Україна). Комп'ютер **МЭСМ (Малая электронная счётная машина)** почав експлуатуватися в 1950 році. Він складався з приблизно 6000 вакуумних ламп і споживав потужність 25 кВт, продуктивність дорівнювалася приблизно 3000 операціям у секунду.

Принципи комп’ютера зі збереженою програмою Джон фон Нейман [7] сформулював у проекті звіту «*First Draft of a Report on the EDVAC»*, датованому 30 червня 1945 року [8].

Це було перше повідомлення про комп’ютер загального застосування з програмою, що зберігається. Імена Маучлі й Еккерта у звіті були відсутні. Ідеї, закладені при розробці EDVAC, стали пов’язуватися співтовариством розроблювачів комп’ютерів з ім’ям фон Неймана.

Однак у той, час як робота фон Неймана була новаторською, використання терміна «архітектура фон Неймана» несправедливе по відношенню як до його сучасників, так і до попередників.

Німецький інженер Конрад Цузе – розробник повнофункціонального комп’ютера Z3, згадував цю концепцію комп’ютера, що зберігає програму, в патентах 1936 року.

Маучлі й Еккерт писали про зберігання програми в грудні 1943 року під час їхньої роботи над комп’ютером ENIAC. Людина, що запропонувала ідею в Електротехнічній школі Мура, невідома. У доповіді адміністратора проекту Криста Брейнерда (Grist Brainerd) про результати першого періоду розробки ENIAC (грудень 1943 року) розглянута концепція збереженої програми і відхилено її застосування в ENIAC. Брейнерд заявив, що "щоб мати найпростіший проект, а не ускладнювати справи" ENIAC буде побудований без будь-якого "автоматичного регулювання".

Останнім часом замість терміна «архітектура фон Неймана» все частіше використовується термін «**Принстонська архітектура»**, за назвою університету, де працював фон Нейман і де в 1952 році був розроблений комп’ютер IAS (від назви Institute for Advanced Study (IAS), Princeton, NJ, USA) [9]. Це був перший комп’ютер зі змішаним зберіганням інструкцій і даних в одній пам’яті. Схеми IAS поширювалися в зацікавлених організаціях, і незабаром налічувалося 15 похідних від IAS, але несумісних між собою комп’ютерів, у тому числі БЕСМ (Москва).

Основна ідея фон Неймана полягала в тому, що інструкції розглядаються в такий же спосіб, як і дані. У машині з Принстонською архітектурою програма може модифікувати сама себе. Подібна функціональність була пов’язана з необхідністю програмно змінювати адреси інструкцій, що раніше робилося вручну. З появою індексних регістрів і непрямої адресації це стало не важливо. Архітектура сучасного комп’ютера, що включає конвеєрну обробку й кешування, робить самомодифікацію коду програми непотрібною й неефективною. Самомодифікація призводить до складності розуміння й налагодження програми. Практика використання коду, що самомодифікується, тепер засуджується, а в ряді випадків - заборонена.

При міжпрограмній взаємодії здатність розглядати інструкції як дані – те, що уможливлює існування асемблерів, компіляторів і інших автоматизованих програмних інструментів. Можна "написати програми, які пишуть програми".

На рівні взаємодії програми з апаратними засобами, наприклад, формування зображення на дисплеї з побітовим відображенням, операції інтенсивного введення-виведення бітових масивів здійснювалися з використанням замовлених апаратних засобів. Згодом було показано, що ці операції можуть бути ефективно здійснені за допомогою технології "компіляції на лету", тобто використання програми, що генерує код.

В архітектурі фон Неймана є кілька недоліків. Модифікації програми можуть бути досить шкідливими або випадково, або цілеспрямовано. У деяких простих комп'ютерах з Принстонською архітектурою збій у роботі програми може ушкодити її, інші програми або операційну систему. Найбільш частий приклад такого збою – переповнення буфера. Здатність програм створювати й змінювати інші програми часто використовується шкідливим, у тому числі вірусним, програмним забезпеченням. Захист пам'яті й інші форми контролю доступу можуть допомогти в захисті й проти випадкової, й проти зловмисної модифікації програм.

У комп'ютері з Принстонською архітектурою центральний процесор може або читати команду, або читати (писати) дані з (в) пам'яті. Але він не може робити це в той же самий час, тому що команди і дані використовують ті самі шини сигналу і пам'ять. У комп'ютері з Гарвардською архітектурою центральний процесор може читати і команду, і дані з пам'яті в той же самий час. Комп'ютер з цією архітектурою може бути швидшим, тому що він здатний вибирати наступну команду в той же самий час, коли закінчує виконувати поточну команду. Швидкість досягається за рахунок більш складної електричної схеми.

Невелика швидкість передачі даних з пам'яті до центрального процесора відносно швидкості їх обробки спричиняє проблему, яка відома як вузьке місце архітектури фон Неймана. В останні роки швидкість центрального процесора в порівнянні зі швидкістю доступу до оперативної пам'яті зросла в багато разів. Для одержання високої швидкості виконання програми необхідно скоротити число звертань до оперативної пам'яті. Якщо, наприклад, кожна команда, виконана в центральному процесорі, вимагає доступу до пам'яті, то комп'ютер не одержує нічого від збільшеної швидкості процесора.

Пам'ять може бути зроблена значно швидшою, але це призведе до її подорожчання. Рішення полягає в тому, щоб використовувати невелику кіль-

кість дуже швидкодіючої пам’яті, відомої як кеш-пам’ять. Поки інформація, якої потребує центральний процесор, перебуває в кеші, часу виконання на це набагато менше, ніж коли необхідно одержати дані з оперативної пам’яті.

Сучасні процесори проектуються з урахуванням особливостей і Гарвардської, і Принстонської архітектур. На чипі процесора кеш-пам’ять розділена на кеш команд і кеш даних. Використовується Гарвардська архітектура, оскільки центральний процесор звертається до кешу. У випадку невдалого звертання до кешу дані одержують з оперативної пам’яті, що не розділена на пам’ять команд і пам’ять даних. У такий спосіб для доступу до пам’яті, розташованої поза кристалом, використовується Принстонська архітектура.

Гарвардська архітектура часто застосовується в спеціалізованих сигнальних процесорах, що звичайно використовуються в пристроях, які обробляють аудіо- та відеосигнали, вирішують складні завдання систем керування. Крім того, багато універсальних мікроконтролерів, що використовувались у найпростіших електронних пристроях, основані на Гарвардській архітектурі. Окрема пам’ять означає, що дані й команди можуть бути різної розрядності, це дозволяє оптимізувати їхню структуру. Пам’ять програм іноді використовується не тільки для збереження команд, але і незмінних даних - таблиць, строкових змінних тощо. Варто мати на увазі, що кілька комплектів шин для одночасної вибірки даних і команд з пам’яті даних і пам’яті програм використовуються тільки всередині кристала центрального процесора при роботі з внутрішньою пам’яттю процесора. Для звертання до зовнішньої пам’яті у всіх процесорах застосовується один комплект зовнішніх шин – шина адресу та шина даних. Це визначається обмеженнями, що накладаються технологією на кількість виводів. Тому розробники систем ЦОС прагнуть використовувати тільки внутрішню пам’ять, і процесори випускаються з великою внутрішньою пам’яттю як програм, так і даних. При використанні зовнішньої пам’яті, збільшується час, що витрачається на виконання операцій. Використання Гарвардської архітектури і скороченої системи команд гарантує, що більшість команд може виконуватися в межах одного машинного циклу.

1.2. Архітектура системи команд

Архітектура системи команд (АСК) служить межею між апаратурою і програмним забезпеченням та подає ту частину системи, яку видно програмісту або розробнику компіляторів.

Можливий поділ АСК за наступними категоріями:

1. Повнота системи команд:

- orthogonal instruction set - ортогональна система команд;
- CISC (Complex instruction set computer) – комп’ютер з повним набором команд;
- RISC (Reduced Instruction Set Computer) - комп’ютер зі скороченим набором команд;
- VLIW (Very long instruction word) - комп’ютер з довгим керуючим словом;
- MISC (Minimal Instruction Set Computer) - комп’ютер з мінімальним набором команд;
- ZISC (Zero Instruction Set Computer) - комп’ютер з нульовим набором команд.

2. Співвідношення команди – дані:

- SISD (Single Instruction, Single Data) - Єдина команда, єдині дані;
- SIMD (Single Instruction, Multiple Data) - Єдина команда, множинні дані;
- MISD (Multiple Instruction Single Data) - Множинні команди, єдині дані;
- MIMD (Multiple Instruction Multiple Data) - Множинні команди множинні дані.

3. Алгоритм виконання програми:

- Скалярний процесор;
- Vector processor (array processor) - векторний процесор;
- Суперскалярний процесор.

4. Спеціалізація процесора:

- ASIC - application-specific integrated circuit - спеціалізовані інтегральні мікросхеми;
- reconfigurable computing - процесор зі змінюваною конфігурацією;
- digital signal processor (DSP) - цифровий сигнальний процесор;
- graphics processing unit (GPU) - графічний процесор.

1.2.1. Ортогональна система команд

Ортогональна система команд - термін, використовуваний у теорії обчислювальних машин і систем. Наявність ортогональної системи команд - бажаний атрибут будь-якої комп’ютерної архітектури.

В обчислювальній системі будь-якої архітектури визначається набір команд, які повинні оброблятися. Ці команди обумовлюють основні операції, які комп’ютер може виконати і типи даних, які він може обробити.

Говорять, що система команд комп’ютера є ортогональною, якщо будь-яка команда може використовувати дані будь-якого типу за допомогою будь-якого способу адресації. Це визначення - наслідок розгляду комп’ютерної команди як вектора, чиї координати можуть бути розділені на кілька полів:

- код операції;
- тип даних, які обробляються;
- джерела даних і/або приймачі результату.

Як і в математиці, де для однозначного подання довільного вектора необхідне використання ортогональної системи координат, в обчислювальній техніці тільки ортогональна система команд може однозначно закодувати всі комбінації кодів операції, типів даних, регістрів і способів адресації.

Якщо поля команди можуть обумовити кожну з можливих команд комп’ютера, говорять, що система команд є "ортогональною". Термін походить з того, що можна розглядати ці кілька полів як координати в просторі; якщо кожна координата може змінюватися, не порушуючи інших координат, говорять, що осі координат є ортогональними одна до одної. Однак якщо виявляється, наприклад, що підсумовування чисел із плаваючою точкою може провадитися тільки тоді, коли вони містяться в Регістрах 0, 1 і 2, а підсумовування цілих чисел може відбуватися з будь-якого регістра, то система команд комп’ютера не є ортогональною.

У багатьох CISC комп’ютерах команда могла звернутися до регістрів або до пам’яті кількома різними способами. Це спростило програмування, тому що замість обов’язку пам’ятати тисячі індивідуальних кодів команд ортогональна система дозволила програмістові пам’ятати від тридцяти до ста кодів операцій ("підсумувати", "відняти", "помножити", "розділити" і т.д.) і три - десять способів адресації ("з регістра","з пам’яті" і т.д.).

У комп’ютерах DEC PDP-11 і Motorola 68000 використовувалися майже ортогональні системи команд.

За винятком команд із плаваючою точкою PDP-11 був строго ортогональним. Кожна команда могла працювати з 1-байтним або 2-байтним цілим числом і могла звернутися до даних, збережених у регістрах, збережених як частина команди, і зберегти дані в пам’яті, зберегти дані в пам’яті по покажчику адреси в регістрі. Навіть лічильник команд і покажчик стека могли застосовуватися у звичайних командах, використовуючи всі режими даних.

Проектувальники Фірми Motorola спробували зробити ортогональним асемблер, у той час як основна машинна мова була менш ортогональна. У порівнянні з PDP-11 MC68000 використовує окремі регістри для зберігання команди й адреси даних у пам’яті; асемблер сховав частину цього поділу від

програміста. На бінарному рівні, людина, яка пише асемблер (або налагоджує машинний код), ясно бачила би, що команди асемблера могли стати кожним з кількох різних кодів операції. Це був компроміс, що забезпечував таку ж зручність програмування, як дійсно ортогональна машина, і дозволяв проектувальникам процесора використовувати біти команд більш ефективно.

Система команд 8-бітного мікропроцесора Intel 8080 була дуже неортогональною. Програміст мовою асемблера повинен був постійно пам’ятати, які операції були допустимі для яких регістрів. Більшість операцій могли бути виконані тільки з даними в регістрі акумулятора, у той час як інші операції могли тільки бути виконані на парі регістрів H/L і т.д. Це було пов’язано з тим, що 8-бітне слово команди мікропроцесора 8080 могло закодувати в цілому тільки 256 команд машинної мови (у порівнянні з 65536 команд, доступними у PDP-11, або 4 мільярдами команд, доступних в MC68000).

Наприкінці 1970-х років дослідження в IBM та інших фірмах продемонстрували, що більшість "ортогональних" способів адресації не використовується більшістю програм.

RISC комп’ютери, залишаючись ортогональними щодо команд обробки даних, повернулися до архітектури "завантажити/зберегти". У цій архітектурі тількі спеціалізовані команди можуть звернутися до оперативної пам’яті й тільки для завантаження даних у регістри або збереження даних регістрів в оперативній пам’яті. Команди обробки можуть оперувати даними, що перебувають у регістрах. У RISC архітектурі це дозволило використати набагато більші набори регістрів, розширити віртуальну адресацію й допустити більш довгі безпосередні дані (дані, що збережені безпосередньо в межах комп’ютерної команди).

1.2.2. Комп’ютер з повним набором команд (CISC)

Архітектура процесора з повним набором команд – це архітектура, у якій кожна команда може виконувати декілька низькорівневих операцій, таких, як читання з пам’яті, арифметична операція, запис у пам’ять. Цим вона відрізняється від RISC архітектури. Причиною розробки CISC стало прагнення спроектувати систему команд, що підтримувала б мови програмування високого рівня, забезпечуючи команди типу виклику процедури та повернення, команди циклу "декремент і перехід, якщо ненульовий" і т.д. Використання повного набору команд спричиняє до зменшення розмірів програми і зниження вимог до оперативної пам’яті, що в той час (1960-і роки) приводило до значного зниження вартості комп’ютера.

Перетворення конструкцій мови високого рівня в меншу кількість команд не завжди приводило до поліпшення роботи - чим більш складна система команд, тим більший час декодування команди і складніша схема її реалізації.

Приклади процесорів команд, що використовують повний набір команд - CDC6600, System/360, VAX, PDP-11, сім'ї Motorola 68000 і Intel x86.

Сучасні CISC-процесори поки підтримують усі команди їхніх попередників, але максимально ефективно працюють з підмножиною команд подібних з RISC. Дійсно, багато процесорів з повним набором команд, типу сучасних x86 процесорів від Intel і AMD, декодують багато з x86 команд у ряд менших внутрішніх "мікрооперацій", які потім виконуються.

Для CISC-процесорів ***характерно***:

- порівняно невелика кількість регістрів загального призначення;
- велика кількість машинних команд, деякі з них завантажені семантично, аналогічно до операторів високорівневих мов програмування, і виконуються за декілька тактів;
- велика кількість методів адресації;
- велика кількість форматів команд різної розрядності; переважно двохадресного формату команд;
- наявність команд обробки типу регістр-пам'ять.

Важлива особливість CISC-процесорів – наявність блока мікропрограмного управління, що відпрацьовує послідовність мікрокоманд, записану в ПЗП усередині процесора, відповідну виконуваній команді.

Достоїнства CISC-процесорів:

- при використанні CISC-процесора об'єм пам'яті, необхідний для вирішення однієї і тієї ж задачі, істотно менший;
- мова асемблера CISC-процесора набагато легша для людини, отже, витрати часу на написання і відладку програм менші;
- у процесі виконання команд мова асемблера CISC-процесора значно менше завантажує системну шину, ніж RISC, що дозволяє іноді, навіть в мультипроцесорних системах, обходитися без кеш-пам'яті. Це досягається за рахунок того, що для реалізації операції, відпрацьованої CISC-процесором як одна команда, RISC повинен виконати цілу програму.

Недоліки CISC-процесорів:

- не у всіх програмах повністю використовується вся могутня система команд CISC-процесора, а це – невиправданий простій його дорогої апаратури;
- навіть найпростіші команди не можуть виконуватися за один такт (як

правило, потрібно два–три такти), оскільки вони реалізуються усередині процесора на мікропрограмному рівні;

- у CISC-процесорах програміст не може управляти процесом виконання складної команди. Отже, цей процес не може бути оптимізований для отримання максимально можливої швидкодії в кожному конкретному місці певної задачі. При програмуванні CISC-процесорів в критичних до швидкодії місцях програм іноді доцільно написати декілька простих команд замість однієї складної. У таких ситуаціях краще мати декілька додаткових регістрів загального призначення, як в RISC-процесорах;
- система команд CISC-процесорів, як правило, не відрізняється стрункістю. Складні команди мають обмежений набір режимів адресації і регістрів, з якими вони працюють. Усе це ускладнює автоматичну генерацію і оптимізацію програм, що може бути особливо актуальним для задач, формованих за допомогою мов логічного програмування.

1.2.3. Комп’ютер зі скороченим набором команд (RISC)

Зачатки цієї архітектури беруть свій початок від комп’ютера CDC6600, розробники якого Джим Торнтон (Jim Thornton) и Сеймур Крей (Seymour Cray) у 1964 році усвідомили важливість спрощення набору команд для побудови швидких обчислювальних машин (74 команди CDC6600 у порівняні з 400 командами процесора 8086). Однак остаточно поняття RISC в сучасному його розумінні сформувалося на базі трьох дослідницьких проектів комп’ютерів: процесора IBM 801 компанії IBM, процесора RISC університету Берклі і процесора MIPS Стенфордського університету.

Розробка експериментального проекту компанії IBM почалася у 1975 році, але його результати ніколи не публікувалися і комп’ютер на його основі в промислових масштабах не виготовлявся.

У 1980 році Д. Паттерсон (David Patterson) зі своїми колегами з Берклі почали свій проект і збудували дві машини, які отримали назви RISC-I (1982 рік, 32 інструкції) і RISC-II (1983 рік, 39 інструкцій). Головними ідеями цих машин було відділення повільної пам’яті від високошвидкісної (на регістрах). Характеристики обчислювальної машини покращувались за рахунок використання конвеєра та регістрових вікон. Маючи величезну кількість регістрів (128), програма мала доступ одночасно тільки до 8 з них. Програма, яка обмежувала себе використанням лише 8 регістрів на процедуру, могла дуже швидко зробити її виклик. Для цього було достатньо здвинути вікно на ті регістри, що використовуються процедурою. Перемикання між регістровими вікнами підвищувало швидкість виклику процедур за рахунок спрощення передачі параметрів та збереження даних. У звичайних процесорах

виклик процедури пов’язаний зі збереженням вмісту регістрів в оперативній пам’яті для звільнення місця та відновленні вмісту регістрів при поверненні з процедури.

У 1981 році Джон Хеннессі (John L. Hennessy) зі своїми колегами почав проект MIPS у Стенфордському університеті. Основним аспектом розробки була ефективна реалізація конвеєрної обробки. Конвеєр MIPS був швидший за інші за рахунок виконання всіх команд за один такт. Негативною стороною цього рішення було відкидання потенційно корисних команд, таких, як множення та ділення.

Ці три машини дотримувалися архітектури, яка відділяє команди обробки від команд роботи з пам’яттю, і звертали увагу на ефективну конвеєрну обробку. Система команд розроблялася таким чином, щоб виконання будь-якої команди займало невелику кількість машинних тактів (переважно один машинний такт). Сама логіка виконання команд з метою підвищення продуктивності орієнтувалася на апаратну, а не на мікропрограмну реалізацію. Щоб спростити логіку декодування команд, використовувалися команди фіксованої довжини і фіксованого формату.

Серед інших особливостей RISC-архітектури потрібно відзначити наявність досить великого регістрового файла (в типових RISC-процесорах реалізуються 32 або більше регістрів у порівнянні з 8–16 регістрами в CISC-архітектурі), що дозволяє більшому об’єму даних зберігатися в регістрах на процесорному кристалі більший час і спрощує роботу компілятора з розподілу регістрів під змінні.

З 1986 року почалася активна промислова реалізація архітектури RISC. До теперішнього часу ця архітектура міцно займає лідируючі позиції на світовому комп’ютерному ринку. Деякі з поширених розробок наведено нижче:

- сім’я MIPS використовується у більшості комп’ютерів SGI та ігрових консолях PlayStation і Nintendo 64;
- серія процесорів POWER фірми IBM використовується в їх власних суперкомп’ютерах та мейнфреймах;
- процесори PowerPC (підмножина архітектури POWER), що виготовляються Freescale (раніш Motorola) та IBM, застосовуються в ігрових консолях Nintendo Gamecube, Microsoft Xbox 360, Nintendo Revolution і Sony PlayStation 3, та у комп’ютерах Apple Macintosh;
- процесори фірми Sun - SPARC та UltraSPARC використовуються у всіх останніх обчислювальних машинах фірми Sun;
- процесори PA-RISC фірми Hewlett-Packard, також відомі як HP/PA;
- процесори Alpha фірми DEC;

- ARM — Palm, Inc. у своїх перших PDA застосовувала процесор Motorola 680x0 (CISC), але зараз використовує процесор ARM (RISC); Nintendo використовує процесор ARM7 в портативних ігрових системах Game Boy Advance та Nintendo DS. Багато стільникових телефонів, наприклад Nokia, використовують процесори ARM.

Розвиток архітектури RISC значною мірою визначався прогресом в галузі створення компіляторів, що оптимізують код програми. Саме сучасна техніка компіляції дозволяє ефективно використовувати переваги великого регістрового файла та конвеєрної організації.

Основним недоліком RISC-процесорів є те, що будь-який компілятор з мови високого рівня змушений генерувати мікрокод. Бібліотеки стандартних функцій для RISC-процесорів займають багато місця в пам'яті.

1.2.4. Комп'ютер з дуже довгим керуючим словом (VLIW)

Архітектура VLIW виникла у відповідь на вимоги щодо збільшення швидкодії обчислень. Вона використовує паралельне виконання інструкцій. Для цього процесор потрібен мати декілька виконавчих блоків: помножувачів, суматорів та ін., що дозволяє йому виконувати кілька інструкцій одночасно. Одна VLIW-інструкція кодує багато операцій, принаймні одну операцію на кожен виконавчий блок. Наприклад, якщо VLIW-пристрій має 5 виконавчих блоків, то VLIW-інструкція цього пристрою повинна мати 5 полів коду операції, кожне поле визначає операцію, що буде провадитись з відповідним блоком. Для розміщення полів коду операції VLIW-інструкція має велику ширину. Швидкість обробки даних визначається тільки внутрішніми особливостями самих виконавчих блоків.

Термін VLIW и поняття однойменної архітектури були запропоновані професором Джошем Фішером (Josh Fisher) в дослідницькій групі Йєльского університету на початку 1980-х. Ця архітектура бере початок від паралельного мікрокоду комп'ютерів CDC6600 (1965 рік) і IBM360/91, випущеного в 1972 році.

З початку розвитку процесорів деякі з них мали додаткові арифметичні і логічні модулі, що дозволяло виконувати паралельні операції. Для вирішення, які операції можуть виконуватись паралельно, були потрібні планувальники обчислень. Суперскалярні процесори використовують апаратні засоби, процесори VLIW - застосовують програмне забезпечення (компілятор).

Поняття передпланування функціональних модулів та паралелізм рівня команди в програмному забезпеченні були добре відомі в практиці розробки горизонтального мікрокоду. Новаціями Фішера були розробки компілятора, що міг одержати горизонтальну мікропрограму з програм, написаних звичай-

ною мовою програмування, та поняття, що архітектура VLIW-процесора повинна бути спроектована з урахуванням вимог компілятора - компілятор і архітектура VLIW повинні бути взаємозв'язані.

Перші справжні VLIW-машини були випущені в 1984 році компанією Multiflow, одним з організаторів якої був Фішер. Вони не мали комерційного успіху, проте використані в цих машинах планувальник обчислень і програмна конвеєризація, запропоновані Джошем Фішером і Бобом Рау (Bob Rau), є сьогодні основою технології VLIW.

VLIW-компілятор упаковує групи незалежних операцій в дуже довгі інструкції (від 256 до 1024 бітів) таким чином, щоб забезпечити їх ефективне виконання функціональними блоками за один машинний такт. Компілятор спочатку визначає всі залежності між даними, а потім визначає, як їх розв'язати. Найчастіше це робиться шляхом переупорядкування всієї програми – різні її блоки переміщуються з одного місця на інше. Цей підхід відрізняється від того, який застосовується в суперскалярних процесорах: вони використовують для визначення залежностей спеціальне апаратне рішення під час виконання програми. Суперскалярні процесори можуть визначати і планувати паралельну обробку тільки в середині базових програм до розгалуження. Деякі системи, що перепорядковують роботу (Pentium Pro і PA8000), поклали початок розширенню області сканування на всю програму.

При використанні VLIW-технології проводиться трасування – вибір найбільш оптимального маршруту для деякого набору вхідних даних, після чого ланки програми упаковуються в довгі інструкції і передаються на обчислення. Планувальник обчислень здійснює оптимізацію на рівні всієї програми. Компілятор прогнозує найбільш раціональний маршрут і планує його проходження як одного великого базового блока, після чого повторює цей процес для інших гілок розгалуження, і так до кінця програми.

Там, де RISC може тільки переглянути код вперед на виявлення розгалужень, VLIW-компілятор не тільки переміщує його з одного місця на інше до виявлення розгалуження (згідно з трасуванням), але може також здійснити крок назад до попереднього програмного положення. Плата за збільшення швидкодії таким шляхом, все ж набагато менша ціни компіляції.

Допоміжні функціональні блоки можуть збільшити продуктивність (за рахунок зменшення конфліктів за ресурси), не дуже ускладнюючи чип. Проте таке збільшення обмежується фізичними можливостями: кількістю портів читання-запису, необхідних для забезпечення одночасного доступу функціональних блоків до файлів регістрів, і взаємозв'язків, які зростають в геометричній прогресії при збільшенні кількості функціональних блоків. До того ж

компілятор повинен розпаралелити роботу до необхідного рівня, щоб забезпечити завантаження кожного блока.

Архітектурі VLIW властиві наступні недоліки:

- Великі об'єми пам'яті для запису програми з довгими командами та нераціональне використання цієї пам'яті.
- Трудність визначення незалежних частин програм. Це відбувається через те, що дані стають відомими тільки в динаміці обчислень.
- Вікно виконання VLIW-процесора не може бути дуже великим, зважаючи на відсутність у компілятора інформації про залежності, що формуються динамічно, в процесі виконання. Цей недолік перешкоджає можливості переупорядкування операцій у VLIW-процесорі. Наприклад, статично не може бути гарантовано правильне виконання операції завантаження у функції, що викликається паралельно з операцією запам'ятовування (особливо, якщо функція визначена динамічно). Крім того, VLIW-реалізація вимагає великого розміру пам'яті імен, багатовхідних регістрових файлів, великого числа перехресних зв'язків. Можливий також зупин, коли під час виконання виникла ситуація, що відрізняється від стану в момент генерації плану виконання (наприклад, під час виконання відбулося невдале звернення в кеш).
- Компілятор повинен знати в деталях внутрішні особливості архітектури процесора до функціональних модулів включно. Як наслідок, при створенні нових версій процесорів з іншою кількістю функціональних модулів або з іншою їх швидкодією все старе програмне забезпечення вимагає повної перекомпіляції. Тому процес компіляції поділяють на дві стадії: все програмне забезпечення готується в апаратно-незалежній формі з використанням проміжного коду, а потім вже транслюється в машинно-залежний код після установки в машині користувача.

1.2.5. Комп'ютер з мінімальним набором команд (MISC)

Архітектура процесора, що основана на використанні стека (стекові комп'ютери), на відміну від архітектури, що використовує регістри. Така архітектура більш проста, тому що всі команди працюють на верхніх входах стека. Результат цього - менший набір команд, менший і більш швидкий модуль декодування й більш швидке виконання операцій. Архітектура MISC має багато загального з мовою ФОРТ і Віртуальною машиною Java.

1.2.6. Комп'ютер з нульовим набором команд (ZISC)

Комп'ютер, оснований на технології порівняння даних зі зразком. Мікрокоманди, у звичайному розумінні, - відсутні. Технології комп'ютерів з

нульовим набором команд базуються на ідеях штучних нейронних мереж, розроблених доктором Паскалем Танхофом (Pascal Tannhof) з IBM і фірмою Silicon Recognition, Inc. Каліфорнія.

Перше покоління чипів ZISC містило 36 незалежних комірок-нейронів або паралельних процесорів. Кожна з них могла порівняти вхідний вектор з 64 байт з подібним вектором, що зберігається в пам'яті комірки. Якщо вектори збігаються, то комірка збуджується. Вихідний сигнал містить номер комірки, у якій відбулася відповідність, або індикатор "збігу не відбулося".

Паралелізм операцій є джерелом швидкості ZISC систем, які усувають крок послідовного завантаження й порівняння зразка для кожної комірки. Інший ключовий фактор - масштабованість. Мережа може бути розширена додаванням пристроїв ZISC без зменшення швидкості розпізнавання.

Сьогоднішній чип ZISC містить більше ніж 200 нейронів і може знайти відповідність серед 1000000 зразків через одну секунду, працюючи із частотою менше 50 Мгц.

Практичне використання технології ZISC зосереджено на розпізнаванні образів, інформаційному пошуку (аналізі інформації), безпеці й інших подібних завданнях.

1.2.7. SISD (Single Instruction, Single Data) - Єдина команда, єдині дані

Термін, що стосується архітектури, у якій один процесор виконує єдиний потік команд, працюючи з даними, збереженими в єдиній пам'яті. Відповідає архітектурі фон Неймана.

1.2.8. SIMD (Single Instruction, Multiple Data) - Єдина команда, множинні дані

В обчислювальній техніці - набір команд для ефективної паралельної обробки великої кількості даних.

Для обробки великої кількості даних застосовуються спеціалізовані процесори – цифрові сигнальні процесори (ЦСП). Основна різниця між SIMD і ЦСП полягає в тому, що ЦСП є повними процесорами з їх власною (часто важкою для використання) системою команд, тоді як розробки SIMD розраховані на виконання програми з використанням універсальних блоків процесора, а команди SIMD забезпечують тільки обробку даних. Команди ЦСП мають тенденцію до спеціалізації - обробки певного типу даних, наприклад звуку або відео, тоді як системи SIMD є більш універсальними.

Система команд звичайно містить повний набір векторних команд, включаючи множення, інвертування й трасування. Вони особливо корисні

для обробки тривимірної графічної інформації. Функції переупорядкування елементів у векторах використовуються для обробки даних і їхнього пакування (стиснення).

Уперше SIMD команди використовувалися у векторних суперкомп'ютерах в 1970-х роках. У теперішній час невеликі (64 або 128 біт) SIMD команди популярні в процесорах загального призначення: набір інструкцій MAX процесора PA-RISC (1994 рік); AltiVec - PowerPC; MMX, SSE, SSE2 і SSE3 - Intel; 3DNow! - AMD; VIS - SPARC; MDMX і MIPS-3D - MIPS.

1.2.9. MISD (Multiple Instruction Single Data) - Множинні команди, єдині дані

Архітектура з кількома потоками команд і одним потоком даних - тип архітектури паралельних обчислень, де багато функціональних модулів виконують різні операції на тих же самих даних. До цього типу належить конвеєрна архітектура. Існує небагато ілюстрацій цієї архітектури, оскільки MIMD і SIMD більше відповідають загальним методам паралельної обробки даних. Вони забезпечують краще масштабування й використання обчислювальних ресурсів, ніж **MISD**.

Можливо єдине відоме практичне застосування **MISD** - виявлення несправностей за допомогою надлишкових обчислень. Пристрої, які повинні досягти надзвичайно високих рівнів надійності, можуть здійснювати два або більше незалежних обчислювальних процесів й перевіряти результати на збіг, щоб гарантувати, що всі компоненти працюють правильно.

1.2.10. MIMD (Multiple Instruction Multiple Data) - Множинні команди множинні дані

MIMD - тип архітектури паралельних обчислень, де багато функціональних модулів виконують різні операції з різними даними. Приклади архітектури: багатопроцесорна система або трансп'ютер; мережа з робочих станцій.

transputer (офіційна форма написання – всі літери малі, від слів ***trans***istor і com***puter***) – альтернативна конструкція мікропроцесора, розроблена на початку 1980-х років фірмою INMOS, Брістоль, Великобританія. Комерційно безуспішний проект, практично забутий сьогодні. Однак архітектура трансп'ютера вплинула на появу нових ідей у комп'ютерній архітектурі, які реалізовані в сучасній обчислювальній техніці - кластерних обчисленнях, суперкомп'ютерах, системних шинах.

Трансп’ютер був першим універсальним мікропроцесором, спроектованим спеціально для використання в паралельних обчислювальних системах. Мета полягала у виробництві ряду мікросхем різних за продуктивністю й вартістю, які, з’єднані разом, утворять повний комп’ютер. Назва була вибрана для вказівки ролі, яку відіграють одиничні трансп’ютери: вони повинні використовуватися як основні будівельні блоки для комп’ютера, так само, як раніше транзистори, - для електронних схем.

Передбачалося використання трансп’ютерів у всіх блоках комп’ютера, від виконання функції центрального процесора до контролера каналу в тій же самій машині. Вільні цикли кожного з трансп’ютерів могли використовуватися для інших завдань, збільшуючи продуктивність машини. Трансп’ютер мав усі необхідні для самостійної роботи блоки, особливість, звичайно пов’язана з мікроконтролерами. Передбачалося спростити сполучення між трансп’ютерами без складної шини або системної плати, залишивши тільки тактовий сигнал. Оперативна пам’ять і її контролер, підтримка шини й навіть операційна система були вбудованими.

1.2.11. Скалярний процесор

Скалярні процесори становлять найпростіший клас процесорів. Скалярний процесор обробляє одночасно один елемент даних (типові дані - цілі числа або числа із плаваючою точкою).

1.2.12. Векторний процесор

Векторний процесор (vector processor або array processor) - процесор, що спроможний виконувати математичні операції на множині елементів даних одночасно. Це відрізняє його від скалярного процесора, що обробляє одночасно один елемент. Векторні процесори використовувалися для обчислень у науковій галузі, де лежали в основі більшості суперкомп’ютерів у 1980-ті й 1990-ті роки, але загальне збільшення продуктивності й поширення можливостей процесорів інших типів призвело до зникнення векторних процесорів як універсальних рішень.

Сьогодні майже всі вироблені процесори включають деякі команди обробки вектора, відомі як SIMD. Ігрові приставки й побутова комп’ютерна техніка, що оброблює графічну інформацію, у своїй архітектурі використовують ідеї векторних процесорів.

Середній процесор спроможний управляти одним або двома елементами даних одночасно. Наприклад, кожний процесор має команду, що говорить "додати A до B і помістити результат в C". Безпосередня форма передачі даних (дані для A, B і C закодовані безпосередньо в команду) використову-

ється рідко, майже завжди застосовуються "покажчики" на адресу пам’яті, що містить дані. Декодування цієї адреси й одержання даних з пам’яті займає якийсь час, що стає усе більш істотним на тлі росту швидкості процесора.

Для зменшення часу обробки команди сучасні процесори використовують методику, відому як конвеєрна обробка команд, у якій команди проходять через модулі. Перший модуль читає адресу й декодує її, наступний одержує значення, подальший робить математичну операцію. При конвеєрній обробці декодування наступної команди почнеться перш, ніж попередня покине процесор, дешифратор адреси використовується постійно. Будь-яка команда вимагає для свого завершення той самий час, відомий як *час очікування*, але процесор може обробити весь пакет команд набагато швидше, ніж у випадку обробки команд по одній.

У векторних процесорах разом з конвеєрною обробкою команд використовується й конвеєр даних. Вони застосовують команди, які говорять не тільки "додати А до В", але й додати всі числа "від .. до .." до всіх чисел "від .. до ..". Замість того, щоб постійно декодувати команди й потім вибирати дані для їхнього виконання, процесор читає єдину команду з пам’яті і "знає", що наступна адреса буде на одиницю більша, ніж попередня. Це істотно скорочує загальний час декодування й зменшує кількість програмного коду, що виконується. Векторна обробка може привести до значного підвищення ефективності виконання, особливо великих програм.

Звичайно векторний процесор має можливість суперскалярного виконання команд. Замість одного функціонального блоку, що складає 10 чисел, можлива наявність двох або чотирьох аналогічних. Векторна команда легко піддається паралельному виконанню, тому що не залежить від результатів інших команд, і два підсумовуючі блоки скорочують час виконання команди вдвічі. Найбільший ефект векторні команди приносять при обробці великої кількості однорідної інформації, наприклад при прогнозі погоди або у фізичних лабораторіях.

1.2.13. Суперскалярний процесор

Суперскалярна архітектура здійснює паралелізм на одному кристалі, підвищуючи продуктивність системи при даній тактовій частоті. Вона вибирає, виконує і повертає результат більш ніж однієї команди протягом одного ступеня конвеєра, звичайно це один тактовий цикл.

Найпростіші процесори - скалярні процесори, вони обробляють один елемент даних одночасно. У векторному процесорі одна команда працює одночасно з багатьма елементами даних. Різниця між ними аналогічна різниці між скалярною й векторною арифметикою. Суперскалярний процесор

поєднує риси обох. Кожна команда обробляє один елемент даних, але є множинні модулі обробки й багато команд можуть обробляти окремі елементи даних одночасно.

Суперскалярний процесор звичайно витримує норму виконання більше однієї команди за машинний цикл. Але тільки множинна обробка команд не робить архітектуру суперскалярною. Проста конвеєрна обробка, при якій процесор завантажує команду, виконуючи арифметичну дію для попередньої й зберігаючи результат останньої (виконуючи три команди за час однієї), - не суперскалярна обробка.

У суперскалярному процесорі є кілька функціональних модулів одного типу поряд зі схемою, що розподіляє команди по модулях. Звичайно існує більш ніж один модуль для оброки цілих чисел (АЛП - арифметико-логічний пристрій). Блок-диспетчер читає команди з пам'яті й вирішує, які з них можуть бути виконані паралельно, посилаючи їх модулям.

Робота диспетчера є ключовою в роботі суперскалярного процесора. Команди `a=b+c; d=e+f` можуть бути виконані паралельно, тому що жоден з результатів не залежить від інших обчислень. Однак команди `a=b+c; d=a+f` можуть або, можливо, не можуть виконатися паралельно, залежно від порядку, у якому виконаються команди при їхньому русі через модулі.

Значні зусилля розробників спрямовані на збільшення точності роботи диспетчера, що гарантує постійне використання всіх однотипних модулів. У сучасних процесорах число модулів збільшилося. Перші суперскалярні процесори мали два АЛП й один співпроцесор для операцій із плаваючою точкою, сучасний PowerPC 970 включає чотири АЛП, два співпроцесори для операцій з плаваючою точкою й два SIMD модулі. Якщо робота диспетчера по розподілу команд по модулях буде неефективна, то швидкодія системи в цілому значно постраждає.

Першим суперскалярним комп'ютером прийнято вважати CDC6600. Процесори Intel i960CA (1988 р.) і 29050 із сім'ї AMD 29000 (1990 р.) були першими комерційними однокристальними суперскалярними мікропроцесорами. RISC процесори, як більш прості, були першими, що застосовували суперскалярну архітектуру, але з поліпшенням технологічних процесів виготовлення кристала, навіть "складні" CISC розробки типу IA-32 змогли перейти в суперскалярні.

По суті всі універсальні процесори, розроблені приблизно з 1998, є суперскалярними.

Значне вдосконалення якості роботи блоку керування (диспетчера) у цей час малоймовірно, що обмежує перспективи збільшення швидкості базової суперскалярної архітектури. Одне з рішень цієї проблеми полягає в пере-

міщенні логіки диспетчера з процесора в компілятор, що може затратити значно більше часу на пошук кращих рішень. Це - основна передумова розвитку процесора з дуже довгим керуючим словом, що також відомий як *статичний суперскалярний* (*static superscalar*) або *планування в часі компіляції (compile time scheduling).*

1.2.14. Спеціалізовані інтегральні мікросхеми (ASIC)

ASIC - мікросхеми, що розроблені для використання в конкретних пристроях, - «замовлені мікросхеми», на відміну від мікросхем широкого застосування. Наприклад, мікросхеми, що виконують специфічні функції в стільникових телефонах або лазерних принтерах, на відміну від ТТЛ мікросхем 555 і інших серій, які використовуються для побудови логічних блоків, що знаходять застосування в різних галузях промисловості.

До складу ASIC входить від 5 тисяч до 100 мільйонів логічних елементів, які можуть бути об’єднані відповідно до вимог розробника, часто включають обчислювальне ядро 8-32 розрядного процесора з блоками оперативної і постійної пам’яті та інші блоки. Часто ASIC називають SoC (System-on-a-chip) - система на кристалі. Для опису необхідних законів функціонування логічного пристрою, розробники ASIC використовують мови, що визначають структуру розроблювальної мікросхеми (hardware description language (HDL)), такі, як Verilog або VHDL.

Програмувальні логічні інтегральні схеми (ПЛІС) - (FPGA - Field-programmable gate arrays) у теперішній час є такими ж інструментами, якими на зорі цифрової напівпровідникової техніки були мікросхеми 155 серії (аналог серії 7400), заміняючи ряд логічних мікросхем малого й середнього ступеня інтеграції одною ПЛІС. Основою ПЛІС є матриця логічних комірок, оточена блоками вводу-виводу, підключеними до виводів мікросхеми. Сучасні сім’ї ПЛІС містять спеціалізовані логічні блоки, що дозволяють оптимізувати розроблювальну структуру. До них належать трасувальні ресурси й вентилі для побудови помножувачів, блоки пам’яті й готові апаратні помножувачі, апаратно реалізовані процесорні ядра. З’єднання між логічними комірками, спеціалізованими блоками й блоками вводу-виводу задаються двійковим конфігураційним файлом (bitstream) після виконання розміщення й трасування, що дозволяє використовувати ту саму ПЛІС у різних пристроях. При дрібносерійному виробництві ПЛІС можуть бути економічно більш вигідними, ніж ASIC, що пов’язано зі значними (сотні тисяч доларів) однократними витратами на підготовку виробництва спеціалізованої мікросхеми.

Реконфігурувальний масив шин даних (reconfigurable data path array - rDPA) - напівпровідниковий пристрій, що містить реконфігурувальні модулі шин даних (reconfigurable data path units - rDPU) і програмувальні з'єднання. Кожний rDPU може бути сконфігурованим для виконання індивідуальної функції. Функції модулів і взаємних з'єднань можуть бути запрограмовані користувачем для виконання будь-якого складного обчислення. Відмінністю rDPA від FPGA є ширина шини даних логічного модуля: модулі шин даних шириною декілька біт (наприклад, 32 біта) в rDPA і однобітні логічні блоки в FPGA.

При розробці спеціалізованих інтегральних мікросхем можливе *проектування повністю замовленої мікросхеми.* В цьому разі ведеться розробка всіх рівнів дифузії та металізації розроблюваної мікросхеми.

Достоїнства повної розробки: зменшена площа кристала, поліпшення характеристик і можливість включення аналогових і інших, попередньо розроблених і повністю перевірених компонентів.

Недоліки повністю замовлених мікросхем: збільшений час проектування й виробництва, значні однократні витрати, застосування складних систем автоматизованого проектування (САПР або CAD – computer aided design) і високі вимоги до навичок розробників.

Проектування замовленої мікросхеми *на основі вентильної матриці* грунтується на використанні попередньо визначених виробником рівнів дифузії. Транзистори й інші активні елементи визначені на підкладці до металізації, інакше кажучи, не зв'язані. Розробники визначають взаємозв'язки вентилів кінцевого пристрою, що для більшості виготовлювачів спеціалізованих інтегральних схем становить від двох до п'яти рівнів металізації. Однократні витрати при цьому набагато нижчі, оскільки фотолітографські маски потрібні тільки для рівнів металізації. Використання стандартних технологічних наробітків скорочує цикл виробництва готового виробу.

Недоліком є неповне використання підкладки й можливі труднощі з трасуванням з'єднань, що призводить до збільшення використовуваної площі й підвищенню ціни пристрою.

У нинішній час розробка логічних пристроїв за цією технологією використовується рідко, вона витісняється ПЛІС на основі матриць логічних елементів, які можуть бути запрограмовані користувачем. Сьогодні вентильні матриці розвиваються в структуровані спеціалізовані інтегральні схеми (Structured ASICs), які включають заздалегідь розроблене ядро, до складу якого можуть входити: процесор, сигнальний процесор, периферійні пристрої, стандартні інтерфейси й блок реконфігурувальної нескомутованої логіки.

Кожен виготовлювач спеціалізованих інтегральних схем створює функціональні блоки з відомими електричними характеристиками, які подані в засобах розробки (САПР). Проектування *з використанням стандартних комірок* – застосування цих функціональних блоків для досягнення високої щільності компонування елементів і заданих електричних характеристик. Інструментальні засоби САПР перетворять описи HDL до рівня списку з'єднань стандартних логічних комірок. За своїми можливостями цей вид проектування розташовується між проектуванням на основі вентильної матриці й проектуванням повністю замовленої мікросхеми.

1.2.15. Система на чипі (SoC)

Система на чипі (System on chip) – реалізація ідеї інтегрувати всі компоненти комп'ютера або іншої електронної системи в одній інтегрованій мікросхемі (на одному чипі). Ця система може виконувати як цифрові, так і аналогові функції, які часто доповнюються комунікаційними можливостями в діапазоні радіочастот. Типовим об'єктом для використання SOC є вбудовані системи (embedded system).

Типова структура системи на чипі включає:

- одне чи більше ядро мікропроцесора, мікроконтролера або цифрового сигнального процесора;
- блоки пам'яті, що містять постійні запам'ятовуючі пристрої – ПЗП та оперативні запам'ятовуючі пристрої – ОЗП. У системі на чипі використовується декілька різновидів ПЗП: ROM – (Read-Only Memory, ПЗП з однократним програмуванням). EEPROM – (Electrically Erasable Programmable Read-Only Memory, перепрограмоване ПЗП з електричним стиранням). Пам'ять такого типу може стиратися і програмуватися сотні тисяч разів. Одним з різновидів EEPROM є флеш-пам'ять (Flash Memory).
- джерела синхронізації;
- периферійні пристрої: таймери-лічильники, таймери реального часу, генератори сигналу скидання;
- зовнішні стандартні інтерфейси: USB, FireWire, Ethernet, USART, SPI;
- Цифро-аналогові та аналогово-цифрові перетворювачі (DAC – Digital to analog converter, ADC – Analog to digital converter) та інші блоки.

Виготовляються системи на чипі за технологіями ASIC мікросхем і відзначаються гнучкими можливостями й ціною, що властива мікросхемам на замовлення.

1.2.16. Процесор зі змінюваною конфігурацією

Поняття реконфігурувального обчислення з'явилося в 1960-ті роки, коли Джеральд Естрін (Gerald Estrin) запропонував концепцію комп'ютера,

що складається зі стандартного процесора й масиву "реконфігурувальних" апаратних засобів. Основний процесор управляє поведінкою реконфігурувальних апаратних засобів, які настроюються для виконання певного завдання типу обробки зображення або розпізнавання образів. Це дозволяє вирішити завдання зі швидкістю, характерною для спеціалізованих апаратних рішень. Як тільки завдання вирішене, конфігурація апаратних засобів пристосовується для виконання наступного завдання. Таке рішення приводить до структури гібридного комп'ютера, що сполучає гнучкість програмного забезпечення зі швидкістю апаратних засобів. На жаль, ця ідея випередила свій час.

В 1990-ті ця галузь досліджень відродилася. Варіанти реконфігурувальної архітектури розроблялися в промисловості й академічних інститутах. Перший у світі комерційний реконфігурувальний комп'ютер Algotronix CHS2X4 був закінчений в 1991 році. Він не приніс комерційного успіху, але був досить багатообіцяльним, щоб Xilinx Inc (винахідник FPGA) купила технологію й розробників Algotronix.

У цей час для обробки даних використовуються реконфігурувальні пристрої типу FPGA або rDPA. Різний конфігураційний файл (bitstream) може бути завантажений протягом виконання програми. Реконфігурувальні комп'ютери з архітектурою Естріна включають звичайний процесор Принстонскої архітектури як головний або керуючий і один або більше реконфігурувальних пристроїв як співпроцесори. Нова архітектура, що базується на FPGA, усуває необхідність у ведучому процесорі, забезпечуючи механізми для конфігурування пристрою при завантаженні з флеш-пам'яті, і безпосередньо підтримує основні інтерфейси до пам'яті й мережних ресурсів через шину, сформовану в структурі пристрою. Забезпечення стійкості обчислювальної платформи в межах реконфігурувального пристрою вимагає можливості змінювати конфігурацію тільки тієї частини пристрою, що реалізує додаток, залишаючи незмінною ту частину пристрою, що реалізує платформу - пам'ять і мережні інтерфейси, драйвери пристрою і т.д. Сучасні пристрої FPGA дозволяють часткову реконфігурацію, але існуючі розробки, які можуть ефективно використовувати цю особливість, усе ще важко здійснити в реалізації «системи на чипі».

1.2.17. Цифровий сигнальний процесор (DSP)

Необхідність цифрової обробки аналогової інформації (ЦОС - цифрова обробка сигналів) у реальному масштабі часу (real-time) привела до створення спеціалізованих мікропроцесорів - цифрових сигнальних процесорів - ЦСП (DSP - digital signal processor).

Особливості архітектури цифрових сигнальних процесорів:

- роздільна пам’ять програм і даних (Гарвардська архітектура);
- спеціальні інструкції для SIMD (Single Instruction, Multiple Data) операцій;
- паралельна обробка даних без мультизадачності;
- можливість прямого доступу до пам’яті;
- одержання цифрових даних від аналогово-цифрового перетворювача (АЦП), обробка й передача даних з перетворенням в аналогову форму цифро-аналоговим перетворювачем (ЦАП).

Обробка цифрових сигналів може бути реалізована й у процесорі загального призначення, але в ЦСП введені модифіковані операції з числами, які спрощують процес обчислень. До таких змін належать:

- Арифметика з насиченням, у якій операції, що призводять до переповнення, залишають у регістрі результату максимальне (або мінімальне) значення, яке підтримується регістром замість того, щоб переповнити його (перейти через нуль). Максимум+1 не дорівнює мінімуму, як у багатьох універсальних процесорах, замість цього це значення числа залишається в максимумі. У деяких ЦСП доступні різні операції з бітом переносу з молодших розрядів.
- Операції множення з накопичуванням (MAC - multiply-accumulate), які корисні для матричних обчислень типу згортки, скалярного добутку або операцій з поліномами. Виконання множення з нагромадженням за один цикл реалізоване у багатьох ЦСП.
- Спеціалізовані команди для спрощення обчислення швидкого перетворення Фур’є - ШПФ (FFT - fast Fourier transform).

Виконання програми оптимізується за рахунок введення конвеєра, що зменшує повну тривалість обробки команд і збільшує пропускну здатність системи. Конвеєрна обробка призводить до значних втрат часу при неправильному передбаченні переходів. Тому в структурі ЦСП значне місце займає блок передбачення розгалужень.

Більшість сигнальних процесорів використовують арифметику з цілими числами (fixed-point arithmetic), що забезпечує достатній діапазон і переваги у швидкості обчислень. Однак у наукових приладах і пристроях, що вимагають підвищеної точності або розширеного діапазону подання чисел, застосовуються процесори, що підтримують обчислення з плаваючою точкою (floating point).

Процесори загального призначення зазнають впливу від ідей, що закладені у сигнальні процесори, і реалізують їх як MMX та ін. розширень команд архітектури Intel IA-32.

1.2.18. Графічний процесор (GPU)

Графічний процесор – пристрій, що забезпечує обробку графічної інформації і візуалізацію тривимірного зображення - рендерінг (rendering) у персональних комп'ютерах і ігрових приставках. Сучасні GPU ефективно обробляють і відображають графічну інформацію завдяки використанню спеціалізованих апаратних блоків для виконання алгоритмів обробки комплексних чисел. Графічний процесор використовує інструкції, що описують велику кількість найпростіших операцій з графічними примітивами, що забезпечує більш швидке функціонування, чим прямий вивід на екран з використанням центрального процесора комп'ютера. До таких операцій 2D (двомірної) комп'ютерної графіки належать: операції побудови трикутників, прямокутників і кіл, пересилання бітових блоків. У 3D графіці до них додалися операції *трансформації координат точок* або *вершин*, з яких складаються об'єкти, їхнього *освітлення*, а також *відсікання* невидимих ділянок сцени. Ці операції можуть виконуватися як програмно, так і апаратно (hardware transform and lighting (T&L)). Апаратна реалізація T&L блоку призводить до неможливості реалізації непередбачених виробником алгоритмів. Програмне рішення вимагає значних обчислювальних витрат центрального процесора. Для зняття обмеження на функціональність графічного процесора був використаний підхід, при якому програма користувача генерує код програми, безпосередньо керуючий роботою графічного процесора. Цей код складається з набору елементарних операцій, що часто застосовуються в 3D графіці. Сгенерована програма називається *шейдер* (*shader*). В GPU також використовуються команди обробки поверхонь, текстурування та ін.

2. ФОРМАЛІЗАЦІЯ ПРОЕКТУВАННЯ

На початку проектування мікроконтролерного пристрою (МКП) або системи (МКС) важко розробляти проект відразу з необхідним рівнем деталізації, тому звичайно проектування ведеться методом «зверху - вниз». Суть цього методу полягає в тому, що спочатку в майбутній системі виділяється невелике число (3-5) блоків, з яких буде складатися система, визначається призначення кожного з них і порядок їхньої взаємодії. Тим самим обумовлюється найбільш загальна структура системи. Надалі ступінь деталізації розгляду зростає, при цьому система розглядається не загалом, а окремими блоками. Такий підхід дозволяє на кожному рівні формулювати і вирішувати задачі допустимої складності, що піддаються усвідомленню й розумінню людиною та вирішенню за допомогою доступних засобів проектування.

Для визначення рівнів деталізації проектування «зверху - вниз» можна використати термін – «горизонтальний ієрархічний рівень проектування». Таким чином, **горизонтальні ієрархічні рівні проектування** являють собою рівні опису (моделі) об’єктів, які відрізняються ступенем деталізації відображення властивостей об’єкта. На всіх горизонтальних рівнях є групи задач, пов’язані з проектуванням схем, програм і конструкцій. Ці групи задач називаються **аспектами проектування** (таблиця 2.1)**.** Переваги такого підходу полягають в тому, що ***складна задача великої розмірності розбивається на групи задач малої розмірності, які вирішуються послідовно, причому всередині груп різні задачі можуть вирішуватися паралельно.***

На системному рівні загальну схему МКС можна подати у вигляді трьох послідовно виконуваних блоків:

1. Завдання (одержання) первинних даних.
2. Розв’язання поставленої задачі.
3. Видача результатів.

Таблиця 2.1 - Рівні та аспекти проектування МКС

Рівні	Аспекти		
	Схемотехнічний	Програмний	Конструкторський
1 Системний	Розробка структурної схеми	Розробка закону функціонування	Визначення конструкції МКС
2 Функціональний	Розробка функціональної схеми	Проектування схем алгоритмів	Розробка конструкції стояка, панелі, пульта, приладу
3 Схемний	Розробка принципової схеми	Поділ завдань на програмні та апаратні	Розробка конструкції ТЕЗ, модуля
4 Компонентний	Вибір елементної бази (з функціональної точки зору)	Програмування модулів	Вибір елементної бази (з конструктивної точки зору)

При аналізі блоку 1 приймається рішення про організацію взаємодії МК з об'єктом керування (одержання та ввід даних), оператором (завдання режимів роботи), з ЕОМ більш високого рівня (керуюча ЕОМ), визначаються вихідні дані для системи, що розроблюється, у якому вигляді вони повинні бути подані, вибираються інтерфейси передачі інформації. Під інтерфейсом розуміють засіб сполучення двох систем або частин однієї системи, у якому всі фізичні, електричні й логічні параметри відповідають попереднім згодам. Можуть бути використані унікальні (власної розробки) чи стандартні інтерфейси, використання останніх більш доцільно. Від ухвалених рішень залежить і характер системи, і її основна частина, у якій провадиться власне обробка даних - блок 2. На першому рівні деталізації цього блоку вибирається узагальнений спосіб рішення поставленої задачі, обумовлюються вихідні дані. У блоці 3 визначається список об'єктів, у які передаються дані: об'єкт керування, панель оператора, окремі МКП або ЕОМ більш високого рівня, вибираються інтерфейси передачі інформації.

На кожному рівні проектування кожен з аспектів може конкретизуватись шляхом виділення більш простих задач. Цей процес триває доти, поки досягнутий рівень деталізації дасть виконавцю повну ясність в алгоритмі розв'язання завдання.

Для вчасного виявлення допущених помилок й недоліків у розроблювальній системі необхідно перевіряти її правильність й оцінювати ефективність на кожному черговому кроці деталізації (рівні проектування), вчасно усуваючи виявлені помилки і недоліки. Переходити до наступного рівня деталізації має сенс лише тоді, коли є достатня впевненість у правильності й ефективності даного рівня деталізації. Незнайдені вчасно помилки або

неефективні рішення згодом можуть викликати більші витрати праці, часу й матеріальних ресурсів (наприклад машинного часу) на їхнє виявлення й усунення. Оскільки уточнення (деталізація) будь-якої задачі провадиться локально, незалежно від інших блоків, то й перевірка правильності проробленої деталізації також носить локальний характер, що значно спрощує виконання цієї роботи. При виявленні прорахунків у рішеннях, прийнятих на попередніх кроках, варто повернутися назад і усунути їх на відповідному рівні деталізації.

На кожному з рівнів проектування розробнику доводиться вирішувати задачі аналізу і синтезу. Метою вирішення задачі аналізу є вивчення вимог до системи на поточному рівні деталізації, її властивостей та їх оцінка; метою задачі синтезу - отримання конкретних варіантів апаратури, що проектується. Розрізняють структурний і параметричний синтез. Мета структурного синтезу – отримання структури пристрою, тобто складу елементів і способу зв’язку їх між собою. Задачу вибору оптимальної структури називають структурною оптимізацією. Мета параметричного синтезу – визначення числових значень параметрів елементів – параметрична оптимізація.

При розв’язанні задач аналізу використовують фізичні та математичні моделі системи, що проектується. Фізичними моделями є різного роду макети, математична модель – це сукупність математичних об’єктів (чисел, змінних, векторів, множин тощо) та відношення між ними, які адекватно відображують властивості об’єкта.

Підставою до розробки будь-якого об’єкта є технічне завдання (ТЗ), окреме ТЗ може оформлятися на розробку програми чи програмного виробу для засобів обчислювальної техніки [10]. У ТЗ на розробку окремих пристроїв МКС входять: перелік усіх функцій, що виконуються кожним пристроєм; вимоги до його вхідних і вихідних параметрів; дані про зміст і форму інформації, якою даний пристрій обмінюється з іншими пристроями апаратури; умови працездатності пристрою; склад конструкторської та програмної документації, що передається замовнику та ін. На рис. 2.1 зображена типова схема процесу проектування.

Розробка системи (об’єкта) починається з аналізу вимог ТЗ та аналізу можливостей їх реалізації. За результатами аналізу розробляють модель об’єкту для структурного і параметричного синтезу. Відповідно до розроблених моделей створюється початковий варіант МКС, параметри якого аналізуються з позицій відповідності вимогам ТЗ. У разі позитивного результату аналізу та забезпеченні умов працездатності із заздалегідь обумовленим запасом (з урахуванням допустимих відхилень параметрів елементів апаратури), задача синтезу на поточному рівні проектування вважається вирішеною. Виконується перехід до наступного рівня проектування та розробляється більш деталізована модель об’єкту. При завершенні останнього рівня проектування, отримані результати оформляють у вигляді необхідної технічної документації.

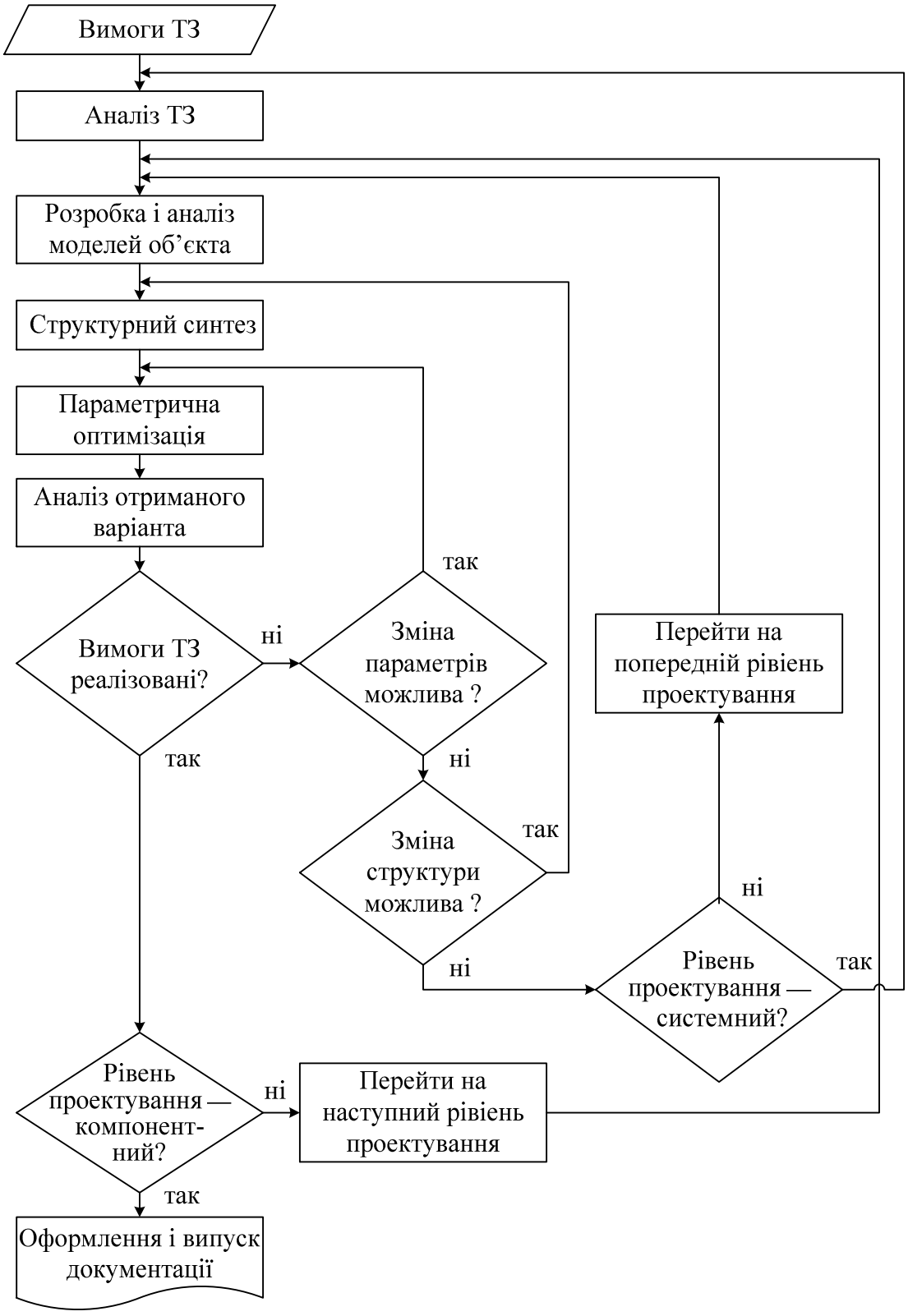


Рисунок 2.1 - Схема процесу проектування

У разі невиконання вимог ТЗ проводять аналіз можливості поліпшення характеристик синтезованого варіанта об’єкта шляхом зміни керованих параметрів або структури об’єкта в рамках моделей, що застосовуються на поточному рівні проектування. При неможливості на поточному рівні синтезувати варіант системи, що задовольняє вимоги ТЗ, повертаються на попередній рівень проектування і повторюють структурний та параметричний синтез урахуванням набутого досвіду. Неможливість синтезувати працездатну модель об’єкта на системному рівні проектування потребує поглибленого вивчення вимог ТЗ, проведення розширеного інформаційного пошуку, залучення до розробки фахівців зі споріднених галузей науки та виробництва.

Таким чином, процес проектування має ітераційний характер, ітерації можуть включати і більше одного рівня проектування, необхідність повернення до попереднього рівня може виявитися на будь-якому подальшому рівні проектування.

2.1. Типова структура МКС

Будь-яка система керування, до складу якої входить МКС, у процесі роботи виконує ряд узагальнених функцій для досягнення цілей, що закладають при проектуванні (функціональний рівень проектування). До таких функцій належать:

- збір інформації про керований об’єкт (процес), збурюючі впливи, стан зовнішнього середовища й апаратури, що входить до складу системи;
- перетворення отриманої інформації до вигляду, зручного для вводу в МК;
- використання різних каналів зв’язку для передачі інформації до МК від датчиків, що вимірюють стан та параметри процесів, і від МК до споживачів інформації;
- обчислення керуючих впливів за заданими алгоритмами, реалізоване у вигляді прикладного програмного забезпечення.

Відповідно до узагальнених функцій може бути подана й узагальнена структура МКС, що складається з об’єкта управління, МК та апаратури їх взаємного зв’язку (рис. 2.2). До складу МК входять постійний та оперативний запам’ятовуючі пристрої, ядро МК, котре об’єднує арифметико-логічний пристрій з регістрами та службовими блоками, порти вводу/виводу, блок стандартних інтерфейсів (БСІ), блок таймерів – лічильників (Т).

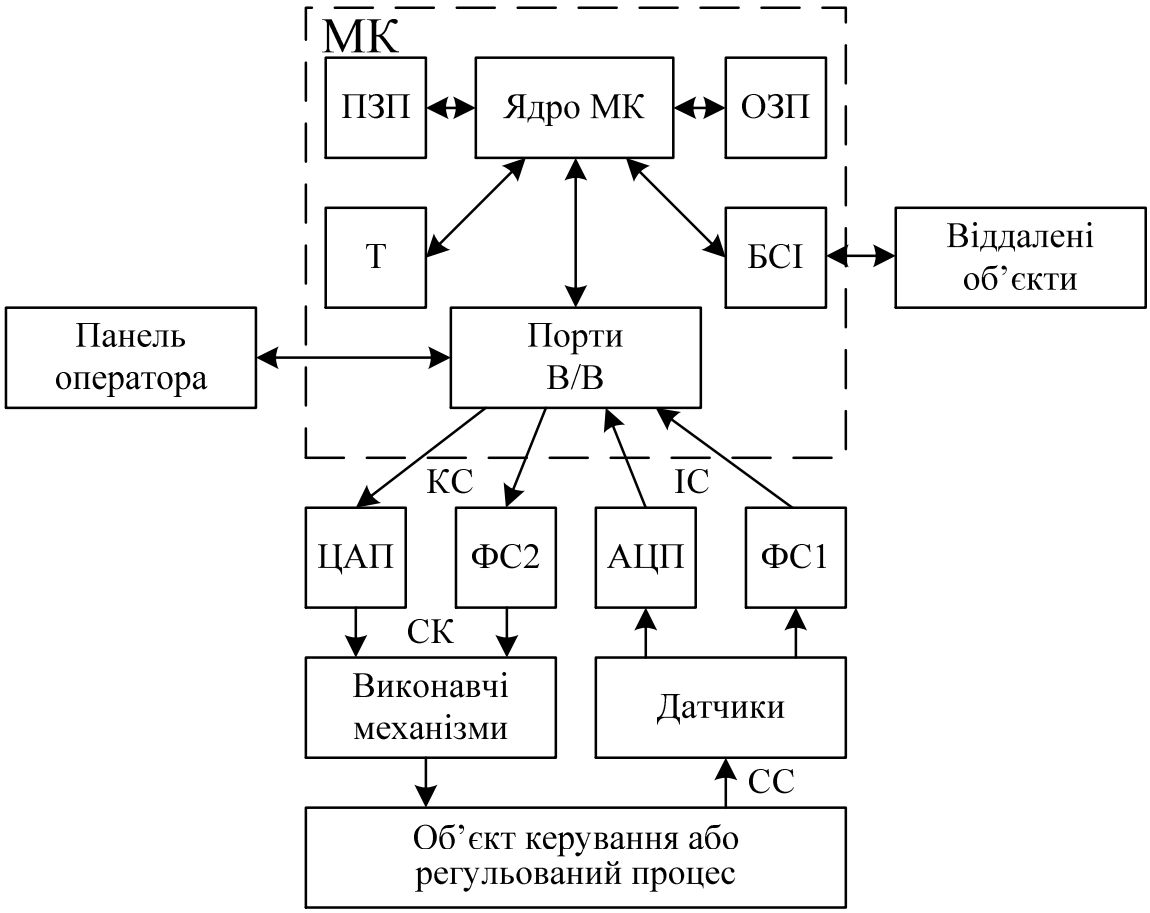


Рисунок 2.2 - Узагальнена структура МКС

Різноманітні за природою сигнали стану об’єкта керування (СС) надходять на датчики, які перетворюють їх на електричні сигнали, що мають як аналоговий, так і дискретний характер. Аналогові сигнали перетворюються аналого-цифровими перетворювачами (АЦП) до цифрового вигляду, імпульсні сигнали надходять на схеми формувачів сигналів (ФС), які виконують функції формування потрібних рівнів двійкових сигналів (найчастіше ТТЛ-рівня), гальванічної розв’язки тощо. Сигнали з виходів АЦП та ФС являють собою інформаційні слова (ІС). МК шляхом опитування інформаційних слів (періодичного чи за встановленим законом) отримує інформацію про об’єкт та генерує відповідно до алгоритму керування послідовності керуючих слів (КС). МК з необхідною періодичністю оновлює керуючі слова на своїх вихідних портах. Певна частина керуючого слова інтерпретується як сукупність прямих двійкових сигналів керування (СК), які через схеми формувачів сигналів (підсилювачі потужності, реле, оптрони тощо) надходять на виконавчі механізми. Інша частина керуючого слова являє собою двійкові коди, які через цифро-аналогові перетворювачі (ЦАП) впливають на виконавчі механізми аналогового типу. Як правило, до складу МКС входить панель опе-

ратора, на якій знаходяться пристрої вводу інформації та елементи її відображення. З віддаленими об’єктами, до яких належать інтелектуальні датчики, аналогічні МКС та керуючі ЕОМ, МКС спілкується за допомогою стандартних інтерфейсів: універсальних асинхронних RS232 та RS485, синхронних I2C і SPI, USB та ін., які апаратно підтримуються БСІ. Формування часових інтервалів апаратними таймерами-лічильниками дозволяє не займати ресурси ядра МК. До складу МК також можуть входити ЦАП, АЦП та деякі ФС.

2.2 Компроміс між програмними та технічними засобами

При розробці МКС необхідно визначити, яка частина функцій повинна бути реалізована програмними, а яка – апаратними засобами, тобто визначити архітектуру МКС.

Формування на виході МКС керуючих слів або сигналів стандартних інтерфейсів можливо трьома способами: програмним, апаратним і програмно-апаратним. Вибір оптимального способу залежить як від параметрів керуючих слів (тривалості імпульсів, частоти повторення й характеристик зміни параметрів), так і від архітектури МКС.

При реалізації програмного способу формування часових або частотних параметрів імпульсів ці параметри, а також їхня зміна формуються основною програмою. При цьому виникає задача розрахунку часу виконання операцій, що забезпечують формування потрібного набору імпульсів і зміну цього набору на наступний набір відповідно до алгоритму керування. Оскільки формування часових інтервалів залежить від тривалості виконання програми МК, то виникнення подій, що порушують очікуваний хід виконання програми, найчастіше призводить до непередбаченого перекручування часових інтервалів. До таких подій належать: обробка переривань, розгалуження алгоритму з різною тривалістю виконання, очікування готовності зовнішніх пристроїв і т.д.

При реалізації апаратного способу необхідна наявність відповідних апаратних засобів, що дозволяють сформувати сигнали з необхідними параметрами. У загальному випадку це можуть бути схеми на дискретних логічних елементах, лічильниках, регістрах зсуву і т.п. При цьому МК здійснює лише загальну синхронізацію роботи цих апаратних засобів. Апаратна частина може бути виконана як за рахунок внутрішніх ресурсів МК, так і за допомогою зовнішніх елементів.

Під реалізацією програмно-апаратного способу мають на увазі наявність апаратних засобів, що формують необхідні імпульсні послідовності на апаратному рівні, але мають спільний інтерфейс з обчислювальним ядром МК і безпосередньо працюючих під його керуванням. При цьому на програмному рівні здійснюється початкове завантаження та ініціалізація цих апаратних засобів, керування формуванням апаратною частиною послідовностей керуючих слів і реакцією на зовнішні, відносно МК, впливи. При програмно-апаратному способові припускається використання таких пристроїв, як таймери-лічильники, що програмуються, широтно-імпульсні модулятори, контролери стандартних інтерфейсів тощо.

Виробництво обчислювальних пристроїв для систем, що вбудовуються, а саме до них належать МКС, підпорядковується загальним економічним законам і орієнтується на великі обсяги випуску продукції. Це обумовлює економічну доцільність випуску мікропроцесорів загального призначення з різною продуктивністю й мінімальним набором вбудованих апаратних засобів; виробництво мікроконтролерів з високою функціональною універсальністю, що приводить до збільшення апаратних і обчислювальних ресурсів і, отже, до підвищення вартості й складності пристроїв; розробку вузькоспеціалізованих МК; випуск сімей МК, основаних на популярних обчислювальних ядрах з додаванням тих або інших апаратних засобів, що дозволяють спростити розв’язання типових завдань.

Для проектувальника МКС необхідно, як правило, мінімізувати вартість системи при заданих показниках функціональності, надійності, енергоспоживання, масогабаритних характеристик. При цьому функціональна надмірність може бути виправдана, якщо вона істотно не погіршує відзначених показників системи.

Застосування мікропроцесорів загального призначення або призводить до програмної реалізації функцій МКС і пов’язаного з цим ростом необхідної швидкодії мікропроцесора та обсягу програмного забезпечення, або вимагає розробки зовнішніх апаратних засобів. Обидва шляхи призводять до значного збільшення вартості проекту.

Для вирішення типових або критичних до вартості кінцевого продукту завдань найбільш підходить використання вузькоспеціалізованих МК. Недоліком такого рішення є обмежена (закладена виробником) функціональність, складність подальшої модернізації виробу.

При проектуванні МКС необхідно на підставі розробленої функціональної схеми описати характеристики внутрішньої і зовнішньої організації потоків даних і керуючої інформації, обґрунтувати вимоги до інтерфейсів, провести аналіз функціональної схеми МКС з позицій фізичної реалізації

потрібних пристроїв та збалансованості програмних і апаратних рішень, максимально переклавши рішення стандартних завдань і формування інтервалів часу на апаратний рівень. На підставі отриманих даних проводиться пошук МК з вбудованими апаратними засобами, що найбільш повно відповідають поставленим вимогам. Особливу увагу варто приділяти наявності апаратної підтримки необхідних стандартних інтерфейсів, формувачам часових інтервалів, апаратній підтримці складних обчислень. Для забезпечення спадковості рішень при модернізації МКС при виборі МК аналізується склад сім'ї, динаміка розробки нових МК, їх технічна підтримка, наявність МК з більшою функціональністю. Правильний вибір архітектури і сім'ї МК дає можливість оптимізувати обчислювальний процес, спростити подальшу модернізацію МКС. Після вибору МК приймається рішення про розробку необхідних, зовнішніх до МК апаратних засобів.

2.3. Обґрунтування застосування та вибору сім'ї ОМК для систем, що проектуються

На системному і функціональному рівнях проектування МКС і МКП завжди вирішується задача вибору ОМК. Вибір ОМК для конкретного застосування є найменш вирішеною з численних проблем проектування МКС і МКП. Це обумовлюється постійним зростанням кількості ОМК та їх функціональності, розширенням галузі застосування, а також відсутністю чіткої методики, яка дозволила б зробити однозначний вибір ОМК.

ОМК є функціонально складним, програмно керованим пристроєм, виконаним у вигляді великої інтегральної схеми (ВІС), і характеризується великою кількістю параметрів. Тому вибір оптимального з технічної і економічної точок зору ОМК для конкретної галузі застосування перетворюється в багатокритеріальну оптимізаційну задачу. При виборі ОМК важливим є формування основних вимог, які висуваються до пристроїв, що створюються. Критерієм оптимізації може бути швидкодія, відповідність наявних функціональних блоків вимогам задачі, що вирішується, простота виготовлення, ціна тощо.

Пристрої з вбудованим ОМК, як правило, мають такі характерні риси:

- робота в реальному часі;
- наявність фіксованого набору задач, що багато разів вирішується протягом усього терміну служби пристрою;
- підвищена надійність і завадозахищеність;
- економічна ефективність.

При виборі ОМК його звичайно розглядають під різними кутами зору:

1) ***з точки зору системного проектування*** треба аналізувати: тип архітектури, швидкодію, наявність необхідних апаратних засобів тощо;

2) ***з точки зору розробки математичного забезпечення*** потрібно аналізувати: набір команд і способи адресації, розрядність даних і команд, організацію пам'яті, наявність уже створеного програмного забезпечення, як власного, так і третьої сторони тощо;

3) ***з конструктивної та схемотехнічної точки зору*** необхідно враховувати: електричну сумісність з іншими ІМС, енергоспоживання і наявність режимів енергозбереження, габарити, тип корпусу та кількість виводів, діапазон робочих температур тощо;

4) ***з економічної точки зору*** визначальним параметром є сукупна вартість ОМК, доступність постачальника (виробника) ОМК, наявність засобів розробки, тестування і налагодження програмного забезпечення, перспективи використання придбаного досвіду, устаткування і програмних продуктів в подальшій роботі.

Під архітектурою обчислювальної системи розуміють її опис на загальному рівні, включаючи опис користувальницьких можливостей програмування, системи команд і засобів інтерфейсу користувача, організації пам'яті і системи адресації, операцій вводу – виводу, керування тощо (абстрактна уява ЕОМ, що відображає її структурну, схемотехнічну та логічну організації) [1, 11]. Реалізація конкретної архітектури в різних обчислювальних системах може відрізнятися як на рівні фізичних компонентів апаратних засобів, так і на рівні способів реалізації підсистем. Різні реалізації однієї архітектури можуть сильно відрізнятися як за продуктивністю, так і за вартістю. У контексті розробки МКС і проектування її апаратних засобів термін «архітектура» використовується для опису принципу дії, конфігурації й взаємного з'єднання основних логічних пристроїв МКС.

Швидкодія МК обумовлюється здатністю АЛУ виконувати інструкції і залежить від тактової частоти МК та довжини машинного циклу (при однаковій частоті продуктивність може відрізнятися в 4–12 разів), набору команд (для виконання однієї і тієї ж дії в RISC МК може знадобитися в 3 рази більше команд, ніж в CISC), наявності апаратних засобів виконання математичних операцій (множення, ділення тощо).

Однією з основних характеристик, що відображають функціональні можливості ОМК, може служити його розрядність (розрядність даних і команд). Діапазон необхідної розрядності даних в МКП досить широкий і залежить від їх функціонального призначення. Для переважної більшості

застосувань використовуються 8-розрядні ОМК, однак у ряді випадків необхідно використовувати 16-розрядні (прецизійні контрольно-вимірювальні системи, системи збору і обробки даних) або 32-розрядні ОМК (цифрові фільтри, спектральні аналізатори, системи керування з великими обсягами обчислень). В останньому випадку разом з ОМК можуть використовуватися і МК з меншою розрядністю, на які покладаються задачі з керування і введення-виведення інформації.

При оцінці апаратного забезпечення, як критерію вибору ОМК, потрібно враховувати альтернативні рішення. Відомо, що пристрої і системи, які використовують апаратні засоби розв'язання, є більш швидкодіючими порівняно з МКП з програмною реалізацією їх рішення, але останні більш гнучкі (тобто можуть бути перепрограмовані). Крім того, створення МКС з апаратними засобами рішення завдань має низькі витрати на розробку і високі – на виробництво; при програмному рішенні спостерігається протилежна картина.

Вибір сімейства МК здійснюється з урахуванням результатів вирішення компромісу "програмні – апаратні засоби" та наявності мінімально необхідної апаратної підтримки внутрішніх та зовнішніх перериванння, необхідних портів вводу-виводу, таймерів-лічильників, вбудованих ОЗП і ПЗП (ППЗП).

Системи команд ОМК відрізняються не тільки типом: RISC, CISC та ін., кількістю і розрядністю (8, 16 і 32 біти), але й можливостями організації компактних ланцюжків команд при програмуванні різних алгоритмів задач, що розв'язуються. Наявність розвинених способів адресації операндів дозволяє створювати більш ефективні програми. Важливе значення має організація пам'яті МК. Загальний адресний простір пам'яті даних і програм дозволяє оперативно перерозподіляти ресурси, підвищує швидкість виконання програм, які знаходяться в ОЗП, але потребує більш дисциплінованої і кваліфікованої роботи програміста і схемотехніка. Сторінкова організація пам'яті ускладнює роботу з структурами даних, розмір яких перевищує сторінку пам'яті. Ці фактори повинні враховуватись при виборі ОМК. Важливим фактором, що впливає на вибір ОМК, є наявність у розроблювача вже створеного програмного забезпечення, а також бібліотек підпрограм стандартних функцій: математичних, вводу – виводу тощо. Підпрограми рішення типових завдань розроблюються як виробниками МК і програмних продуктів, так і світовою спільнотою програмістів, вони входять до складу і комерційних пакетів програм, і пакетів програм, що вільно розповсюджуються. Після визначення допустимого класу ОМК, що задовольняють поставлені вимоги, оцінюються програмні можливості ОМК.

При цьому потрібно враховувати наявність засобів автоматизованого програмування і налагодження програмного забезпечення (інтегровані середовища і компілятори, програмні симулятори, внутрішньосхемні емулятори, програматори та ін.). При виборі ОМК важливо враховувати складність програмування, яке визначається системою команд та архітектурою ОМК. Для оцінки МК часто використовується пробне програмування. Воно виконується для заздалегідь визначеного набору задач, що відображають специфіку галузі застосування. Знання галузі застосування ОМК дозволяє виділити найбільш специфічні та принципові частини алгоритмів і скласти програми для всіх допустимих типів ОМК з метою отримання експлуатаційних характеристик МК. З них найбільш важливі такі, як загальний час виконання програми та її критичних фрагментів, необхідна ємність ПЗП (ППЗП) і ОЗП, час реакції МК на зовнішні сигнали тощо. Внаслідок пробного програмування одержуються реальні часові характеристики застосування конкретного ОМК в реальній системі. Отримані дані, як правило, достатні для оцінки ОМК.

Проектування систем з МК не обмежується розробкою схемних рішень і програмного забезпечення. При визначенні конструкції МКС, що використовуються на відкритому повітрі, на транспорті чи спеціальних об’єктах, важливу роль відіграє діапазон робочих температур МК. Недостатній діапазон може потребувати підігріву чи примусового охолодження МК, що збільшує вартість МКС, зменшує її надійність і може не відповідати вимогам ТЗ. В системах з живленням від автономних джерел живлення (сонячних, гальванічних чи акумуляторних батарей) суттєвим стає енергоспоживання МК. Для його зменшення потрібно вибирати МК з меншим показником відносного струму споживання мА/МГц, з напругою живлення 3.3В і менше, використовувати режими енергозбереження: відключення периферійних пристроїв та ядра МК при очікуванні запланованих подій. У більшості випадків МК мають ТТЛ сумісні виводи, однак при використанні МК з низьковольтним живленням або при підключенні великої кількості периферійних ІМС, ІМС з неузгодженою або двополярною напругою живлення необхідно переконатися в допустимості електричних режимів як МК, так і периферійних ІМС. Перевірка узгодження часових параметрів сигналів МК і ІМС необхідна при формуванні на виводах МК високочастотного сигналу і використанні CMOS логіки. Габарити і тип корпуса МК важливі при розробці друкованих плат і підготовці процесу виготовлення МКС. Монтаж деяких типів корпусів ІМС вимагає застосування спеціалізованого технологічного обладнання й не може бути виконаний на застарілому устаткуванні, це вимагає поновлення технічного

оздоблення виробництва або передачу монтажу виробу на інше підприємство.

Вартість системи на основі МК складається з ціни власне МК, вартості розробки програмного забезпечення, ціни необхідних систем розробки, наладки і тестування програм. Ціна МК складає менше 1% від ціни систем розробки (IDE – integrated development environment). Невеликі проекти можна реалізувати з використанням мови Assembler, що для більшості МК розповсюджується виробниками безкоштовно. Програмування великого проекту ведеться мовою високого рівня: C, C++, Pascal, Basic. Для зменшення вартості розробки МКС можна використовувати програми-компілятори з цих мов, що вільно розповсюджуються на підставі ліцензії GNU GPL [12], наприклад, компілятор мови C – GCC. Деякі комерційні систем розробки також використовують GCC для компіляції програм. Доступність постачальника (виробника) МК визначає безперебійність придбання і використання необхідних МК з можливістю отримання технічної та інформаційної підтримки: документації на МК, прикладів програмного та апаратного вирішення типових проблем (application note), переліку знайдених помилок та шляхи боротьби з ними (errata) і зменшення пов’язаних з цим витрат. Використання однієї з сімей МК приводить до купівлі чи розробки і виготовлення спеціалізованої апаратури, яка призначена для програмування та налагодження МКС при функціонуванні в режимі реального часу, наприклад, програматорів, адаптерів інтерфейсу JTAG та ін. Ця апаратура, як і розроблене програмне забезпечення, не завжди може адаптуватись до МК іншої сім’ї, тому з економічної точки зору доцільно зберігати спадкоємність сімей МК, а при її зміні враховувати можливість використання нової сім’ї в подальших розробках.

2.4. Особливості розробки апаратних засобів і прикладного програмного забезпечення МКС

Для МКС характерне розв’язання заздалегідь визначених завдань, які не змінюються протягом періоду експлуатації системи. Часові цикли рішення цих завдань жорстко пов’язані з параметрами руху вимірюваної і оброблюваної інформації. Запізнювання видачі керуючих впливів через збільшення часу розрахунків може призвести до нестійкості системи регулювання, збільшення інтервалу дискретизації вхідної інформації - до її перекручування й невірогідності результатів подальших обчислень. Спостерігається взаємозв’язок динамічних параметрів об’єкта керування, апаратних засобів МКС і ефективності (швидкодії) програм обробки інформації і обчислення керу-

ючих впливів. Склад функціональних завдань, розв'язуваних МКС, відповідає її узагальненим функціям.

Збір інформації про об'єкт.

Інформація про вимірювані, контрольовані й керовані параметри процесів міститься в первинних сигналах, отриманих від датчиків. Корисна інформація відповідає різним параметрам первинного фізичного сигналу: амплітуді, фазі, формі імпульсу, частоті, тривалості й т.п. Датчики повинні забезпечувати задані динамічні характеристики, узгодження рівнів електричних сигналів, їхню гальванічну розв'язку, виключати проникнення перешкод по вимірювальних колах.

Перетворення інформації.

Однією з тенденцій застосування МКП є їх наближення до місць одержання первинної інформації і попередня обробка її безпосередньо в цих місцях. Поруч з датчиками виконується аналогово-цифрове перетворення інформації. Інтервал дискретизації вхідного впливу визначається на основі теореми Котельникова для випадкових процесів з обмеженим спектром. Для детермінованих процесів він звичайно вибирається виходячи з умови перевищення порогом квантування за рівнем приросту вимірюваної величини за час, що дорівнює інтервалу дискретизації.

Передача отриманої інформації.

Отримана в цифровій формі інформація може передаватися в спеціалізований МКП по каналу зв'язку для подальшої обробки. Як правило, МКС працюють з багатьма датчиками в середовищі з підвищеним рівнем електромагнітних перешкод. Характеристики фізичного і логічного рівня інтерфейсу каналу передачі інформації повинні забезпечувати необхідну швидкість передачі, механізм адресації і усунення колізій, засоби виявлення й виправлення помилок.

Обробка інформації.

Фізичні сигнали, як правило, спотворені перешкодами, тому велике значення в МКС мають цифрові методи і засоби виділення корисної інформації з первинного сигналу. Від ефективності виділення залежить збереження точності обмірюваних інформативних параметрів сигналів, з огляду на те, що при подальших їхніх перетвореннях і обробці похибки виміру можуть істотно спотворити результати обчислень. Алгоритми розв'язання завдань, пов'язаних з цифровою фільтрацією і спектральною обробкою, ставлять дуже високі вимоги до обчислювальних засобів у частині необхідних інформаційних об'ємів пам'яті й швидкодії, для їхньої реалізації можливе застосування спеціалізованих МК.

Обчислення керуючих впливів.

Обчислення керуючих впливів з математичної точки зору зводиться до перетворення систем координат, пов'язаних з різними функціональними просторами; визначення поточних параметрів обмірюваних процесів, поданих у цифровій формі; спільного розв'язання лінійних і нелінійних алгебраїчних рівнянь, у тому числі й рекурентного типу; розв'язання систем диференціальних рівнянь; прогнозування ходу керованих процесів; інтерполяції й екстраполювання функцій, що обчислюються; пошуку даних у таблицях, цифроаналогових перетворень і т. п. Теоретичною основою для розв'язання цих завдань є числові методи. На їхній основі розробляються алгоритми й програми розв'язання для конкретних МК. Критеріями для вибору тих або інших числових методів є такі показники: необхідні об'єми оперативної й постійної пам'яті; кількість елементарних обчислювальних операцій (типу додавання, множення); швидкість збіжності, що залежить від кількості ітераційних циклів розв'язання; інструментальні похибки обчислень (через округлення, обмеженості розрядної сітки ЕОМ) [13].

До допоміжних завдань належить контроль правильності функціонування апаратних і програмних засобів МКС. Ця група завдань реалізується у вигляді контрольних і діагностичних тестів; програм, що забезпечують генерацію стандартних впливів на керуючі підсистеми та ін.

Розробка програм проходить кілька стадій, починаючи від розробки технічного завдання і закінчуючи її впровадженням [14]. Узагальнено можна подати наступний список виконуваних робіт:

- розробка ТЗ;
- аналіз вимог ТЗ;
- розробка алгоритмів;
- програмування;
- тестування й налагодження;
- випробування;
- впровадження.

Під час розробки ТЗ на розробку програми (прикладного програмного забезпечення - ПЗ) [10] у числі іншого визначається структура вхідних і вихідних даних, провадиться попередній вибір методів розв'язання завдання, визначаються вимоги до технічних засобів, вибирається мова програмування.

Починаючи з цього етапу, над розробкою ПЗ поряд з програмістами починають працювати фахівці в галузі, де планується використовувати МКС, їх мета - формалізувати поставлене завдання.

На етапі аналізу вимог ТЗ уточнюються цілі розробки; визначаються ресурси проектування, тобто кількість і кваліфікація фахівців, терміни і ко-

шти, виділені на розробку програм, технічні і програмні засоби технологічного забезпечення етапів програмування і налагодження; здійснюється вибір типу МК.

При розробці алгоритмів вибирають математичні методи розв'язання завдань, що забезпечують необхідну точність при заданих похибках вихідних величин. Визначають усі вхідні і вихідні змінні для кожного завдання, діапазони і швидкості їхньої зміни. Проектування алгоритмів супроводжується моделюванням алгоритмів на ЕОМ загального призначення з визначенням часових витрат на їхнє виконання в МК. За результатами моделювання уточнюються вимоги до продуктивності і об'ємів пам'яті МК. Підвищення швидкодії розроблювального ПЗ можна домогтися виконанням фрагментів програми мовою асемблера, виключенням виклику підпрограм і застосування циклів – використання лінійних програм (in-line program), обмеженням розрядності чисел, що беруть участь в обчисленнях, використанням «швидких» алгоритмів. Всі ці заходи мають і негативні сторони: використання асемблера підвищує трудомісткість програмування і збільшує вимоги до кваліфікації програміста; перехід від циклів і підпрограм до лінійного запису коду призводить до значного росту обсягу програми; обмеження розрядності обчислень ускладнює можливу модифікацію програми і вимагає оцінки внесеної похибки; «швидкі» алгоритми можуть бути складними для розуміння, що є додатковим джерелом помилок, або використовувати табличні обчислення, що збільшує розмір коду програми і ускладнює її модифікацію.

Виконання цих робіт вимагає залучення кваліфікованих фахівців різних профілів: математиків-програмістів, системо- і схемотехніків.

На етапі програмування алгоритми рішення завдань подаються у вигляді схем алгоритмів, що дозволяють здійснювати кодування програми обраною мовою програмування (асемблер або мова високого рівня).

Під тестуванням розуміють виконання програми з метою виявлення помилок, а під налагодженням – пошук і усунення причин виявлених помилок. Обидві дії тісно пов'язані між собою й взаємно доповнюють одна одну. Проведення тестування й налагодження розпадається на етапи статичного налагодження і тестування та динамічного налагодження і тестування. Статичному налагодженню і тестуванню піддаються окремі програмні модулі і програма в цілому. Як правило, статичне тестування і налагодження проводяться на ЕОМ загального призначення з інтерпретатором команд МК (програмним симулятором). Найбільш складними є проведення динамічного налагодження і тестування, які здійснюються на реальній МКС з імітацією зовнішнього середовища технологічною ЕОМ або з підключенням блоків і пристроїв реальної апаратури. На всіх перерахованих етапах вирішується про-

блема генерації тестів, масивів вихідних даних, контрольних статичних і динамічних завдань, фіксації результатів роботи комплексу програм і МКС, що налагоджуються. Після проведення етапу тестування й налагодження випускається погоджений із замовником комплект документів.

Найбільша кількість помилок, які важко усуваються, пов'язана з помилками при формалізації завдання на початкових етапах розробки ПЗ. Помилки формалізації звичайно викликаються двома причинами. Перша з них - недостатня кваліфікація фахівця в заданій предметній галузі. Тонкощі процесів, що відбуваються в системі, сприймаються ним на інтуїтивному рівні і при спробі їхнього формального опису виникають помилки. Друга - ведеться розробка нової системи, закони протікання процесів у якій раніше не описувалися. У цьому випадку формалізація завдання ведеться методом послідовного наближення. Здійснення розробки МКС невеликими колективами, у яких відсутні або професійні програмісти, або фахівці з предметної галузі, приводить до появи розробників-універсалів, що освоюють додаткову спеціальність. Їхня кваліфікація за непрофільним фахом достатня для роботи над нескладними проектами.

Перед впровадженням - завершальним етапом розробки - проводиться випробування програм. Ці випробування поєднуються, як правило, з випробуваннями МКС у цілому, здійснюваними в реальних умовах її функціонування. Після випробувань вносяться необхідні зміни в програми і документацію.

Після передачі замовнику фіксація помилок у роботі МКС здійснюється в процесі її реального функціонування. Однак усунення виявлених помилок у роботі провадиться розробником. Ним вносяться необхідні зміни в експлуатаційну і робочу документацію, провадиться доробка випущених раніше виробів. Цей процес називається супроводом виробу (програми).

3. ОРГАНІЗАЦІЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ

3.1. Загальна характеристика МК

Мікроконтролер є складним програмно-керованим цифровим пристроєм в мікроелектронному виконанні. Тому він описується множиною ***параметрів***, властивих як електронним приладам:

- функціональність вбудованих пристроїв;
- робоча частота;
- кількість рівнів та значення напруги живлення;
- споживана потужність;
- тип корпусу;
- робочий температурний діапазон;
- габарити і маса;
- надійність, вартість тощо,

так і обчислювальним засобам:

- архітектура процесорного ядра;
- кількість внутрішніх регістрів;
- кількість джерел переривань;
- розрядність команд і даних;
- довжина циклу виконання команд;
- продуктивність;
- тип та об’єм пам’яті;
- склад програмного забезпечення тощо.

Для цільової сфери застосування МК з перерахованих характеристик виділяють найбільш істотні, за якими і порівнюють різні ОМК.

Однокристальні мікроконтролери дозволяють істотно розширити інтелектуальні можливості різного роду пристроїв і систем. ОМК став сьогодні одним з найпоширеніших елементів програмованої логіки. Велика продуктивність МК дозволяє реалізувати різні пристрої, що працюють в реальному масштабі часу. Ось тільки деякі **приклади застосування МК:**

- **Комп'ютери і периферія:** принтери, плотери, мережні картки, «миші», сканери, накопичувачі на гнучких і жорстких магнітних дисках, CD та DVD дисководи, мультимедійні пристрої тощо.
- **Радіотехніка:** модулі телетексту, синтезатори частоти, пристрої цифрової обробки сигналів.
- **Техніка зв'язку:** звичайні та радіомодеми, мікро-АТС, автовідповідачі, мобільні телефони, пейджери, факс-апарати тощо.
- **Промислові контролери:** пристрої попередньої обробки даних та дистанційного управління (наприклад, електродвигунами), промислові роботи, інтелектуальні датчики, регулятори температури, вологості, тиску тощо.
- **Автомобільна електроніка:** системи управління запалюванням, мікрокліматом і упорскуванням палива, панелі приладів, радарні детектори, автомобільні сигналізації, комбіновані вимірювальні та керуючі пристрої.
- **Побутова техніка:** картки електронної платіжної системи, системи обробки звуку і синтезу мовних повідомлень, блоки дистанційного управління, відеоігри, системи сигналізації, лічильники води, газу та електроенергії, іграшки тощо.

МК 8051 - базовий представник сім'ї мікроконтролерів MCS-51 фірми INTEL і його ядро є основою для всіх МК сім'ї MCS-51. Основними особливостями ядра 8051 є:

- 8-бітний процесор (CPU), оптимізований для завдань управління;
- вичерпні можливості булевої обробки даних (однобітна логіка);
- апаратні блоки множення, ділення 8-бітних чисел;
- 64К адресного простору пам'яті програм;
- 64К адресного простору пам'яті даних;
- 4К пам'яті програм, розміщених на кристалі;
- 128 байт пам'яті даних, розміщених на кристалі;
- 32 двонаправлених лінії вводу/виводу, що адресуються індивідуально;
- два 16-бітних таймери/лічильника;

- дуплексний універсальний синхронно-асинхронний приймач-передавач - УСАПП (USART - universal synchronous/asynchronous receiver/transmitter);
- 5 векторів переривань від 6 джерел з двома рівнями пріоритету;
- генератор тактових імпульсів, який працює з частотою від 0 до 12 МГц.

Управління вбудованими пристроями здійснюється через регістри спеціальних функцій - РСФ (SFR – special functions register).

Класичний МК сім'ї MCS-51 при тактовій частоті 12 МГц має мінімальний цикл виконання команди - 1 мкс, а його продуктивність дорівнює одному мільйону коротких операцій за секунду; потребує одного джерела живлення 5 В та в активному режимі споживає потужність 0,15 Вт; кожна лінія вводу/виводу може бути підключена до одного логічного елемента ТТЛ або до восьми – ТТЛШ з малим споживанням, випускається в корпусі DIP-40 (dual-in-line package - корпус с дворядним розташуванням виводів). Передбачені два режими енергозбереження [15, 16, 17]. Більш детальні відомості про електричні параметри МК наведено в Додатку А.

Розглядаючи функціональні особливості сім'ї МК, об'єднаних під загальною назвою MCS-51, необхідно виділити ***дві групи МК***. До однієї входять класичні МК, відмінні від базового зразка INTEL 8051 незначними удосконаленнями. До другої групи входять МК на базі ядра 8051 зі значними змінами, внесеними в периферійні пристрої. Всі розглянуті МК сумісні з 8051 за системою команд. Введення нового додаткового обладнання впливає тільки на кількість регістрів спеціальних функцій.

До ***першої групи*** удосконалень можна віднести:

- розширення внутрішньої пам'яті програм до 64 кбайт або її зменшення до 1 кбайта;
- розширення внутрішньої пам'яті даних до 256 байт – 8 кбайт;
- зміну кількості таймерів/лічильників;
- підвищення робочої частоти до 40 МГц;
- розширення діапазону допустимої напруги живлення до 2,7 - 6 В;
- використання різної технології виготовлення пам'яті програм;
- забезпечення працездатності в наступних температурних діапазонах:
  - комерційний (0 °С...70 °С);
  - промисловий (– 40 °С...85 °С);
  - автомобільний (– 40 °С...125 °С);
  - військового призначення (– 55 °С...125 °С);
- використання різних типів корпусів МК:
  - PDIP/CerDIP – plastic/ceramic dual-in-line package (пластмасовий/керамічний дворядний корпус);

– PLCC/LCC - plastic leaded chip carrier (пластмасовий кристалотримач з виводам);

– PQFP/TQFP – Plastic/Thin Quad Flat Package (пластмасовий/тонкий квадратний корпус);

- зменшення або збільшення кількості виводів (портів вводу/виводу) тощо.

До ***другої групи*** належать структурні вдосконалення МК:

- скорочення машинного циклу;
- введення конвеєрної обробки команд;
- впровадження блоку завантаження (boot block), що розширює функціональні можливості програми користувача;
- використання нових алгоритмів програмування МК:

  – ISP – In-System Programming. Програмування МК без вилучення його з системи. Програмування відбувається за послідовним інтерфейсом;

  – IAP – In-Application Programming. Програмування, перепрограмування та селективне стирання FLASH пам’яті та конфігураційних бітів МК здійснюється програмою користувача;
- розширення внутрішньої пам’яті програм понад 64 кбайт;
- впровадження енергонезалежної внутрішньої пам’яті даних;
- вбудовування ЦАП, АЦП;
- розширення функцій таймерів/лічильників за рахунок нових режимів порівняння, захвату, ШІМ;
- розширення кількості вбудованих послідовних інтерфейсів: SPI, I2C, USB, CAN тощо;
- розробка вбудованих апаратних блоків, що виконують складні математичні перетворення: множення, ділення, кодування та декодування даних, наприклад MP3 Decoder;
- об’єднання ядра 8051 з пристроями управління електроприводами постійного та змінного струму, приймачами/передавачами діапазонів 433/868/915 MHz, криптографічною системою тощо.

Залежно від технології виготовлення постійного запам’ятовуючого пристрою (ПЗП) всі типи МК поділяються на п’ять груп:

1. Мікроконтролери, які можуть бути багаторазово запрограмовані користувачем. Ці МК оптимальні для експериментальних розробок і налагодження програм. Вони, в свою чергу, можуть бути поділені також на групи:

1) Мікроконтролери з ПЗП з ультрафіолетовим стиранням.

Родоначальники МК з ПЗП, що може програмуватись десятки разів. Для стирання інформації використовується ультрафіолетове проміння. Цикл стирання-запис триває десятки хвилин. Зараз майже не використовується.

2) Мікроконтролери, ПЗП програм і даних які можуть бути багаторазово перепрограмовані користувачем.

ПЗП виготовлені за різними технологіями (FLASH, EEPROM), але дозволяють електричне стирання-запис інформації. Ці МК дають можливість легко змінювати програму і дані, що спрощує та прискорює процес налагодження програми та може бути використано для занесення калібрувальних даних вже після остаточного тестування розробленого мікроконтролерного пристрою.

2. Мікроконтролери, які одноразово програмуються (OTP – One-Time Programmable memory).

МК цього типу застосовуються в тих випадках, коли немає потреби змінювати зміст програми або конфігурацію МК в розробленому МКП. Використання OTP МК зменшує вартість виробу.

3. Мікроконтролери, які програмуються на підприємстві, що їх виготовляє (OTP).

Ці МК замовляються і повністю програмуються на підприємстві, що їх виготовляє, за заздалегідь наданою користувачем інформацією. Тобто це також МК, що програмуються одноразово (OTP), з тією різницею, що програмування здійснюється не користувачем. Програмування на підприємстві передбачає великий об’єм випуску МК.

4. Мікроконтролери, які послідовно програмуються на підприємстві, що їх виготовляє (SOTP - Serial OTP).

Це також замовні МК типу OTP. Декілька комірок, що задаються користувачем, в кожному МК програмуються різними серійними номерами. Причому ці номери можуть бути випадковими, псевдовипадковими або послідовними. Послідовне програмування дозволяє кожному МК мати власний номер, що може бути використаний як пароль, код доступу чи ідентифікація.

5. Маскові мікроконтролери (ROM).

Ці МК також є замовними і забезпечують максимально низьку вартість при великих серійних замовленнях.

3.2. Структурна організація МК 8051

3.2.1. Структурна схема МК51

Основу структурної схеми МК51 (рис. 3.1) утворює внутрішня двоспрямована 8-бітна шина, що пов'язує всі основні вузли і пристрої: резидентну пам'ять, АЛП, блок регістрів спеціальних функцій, пристрої управління та порти вводу-виводу.

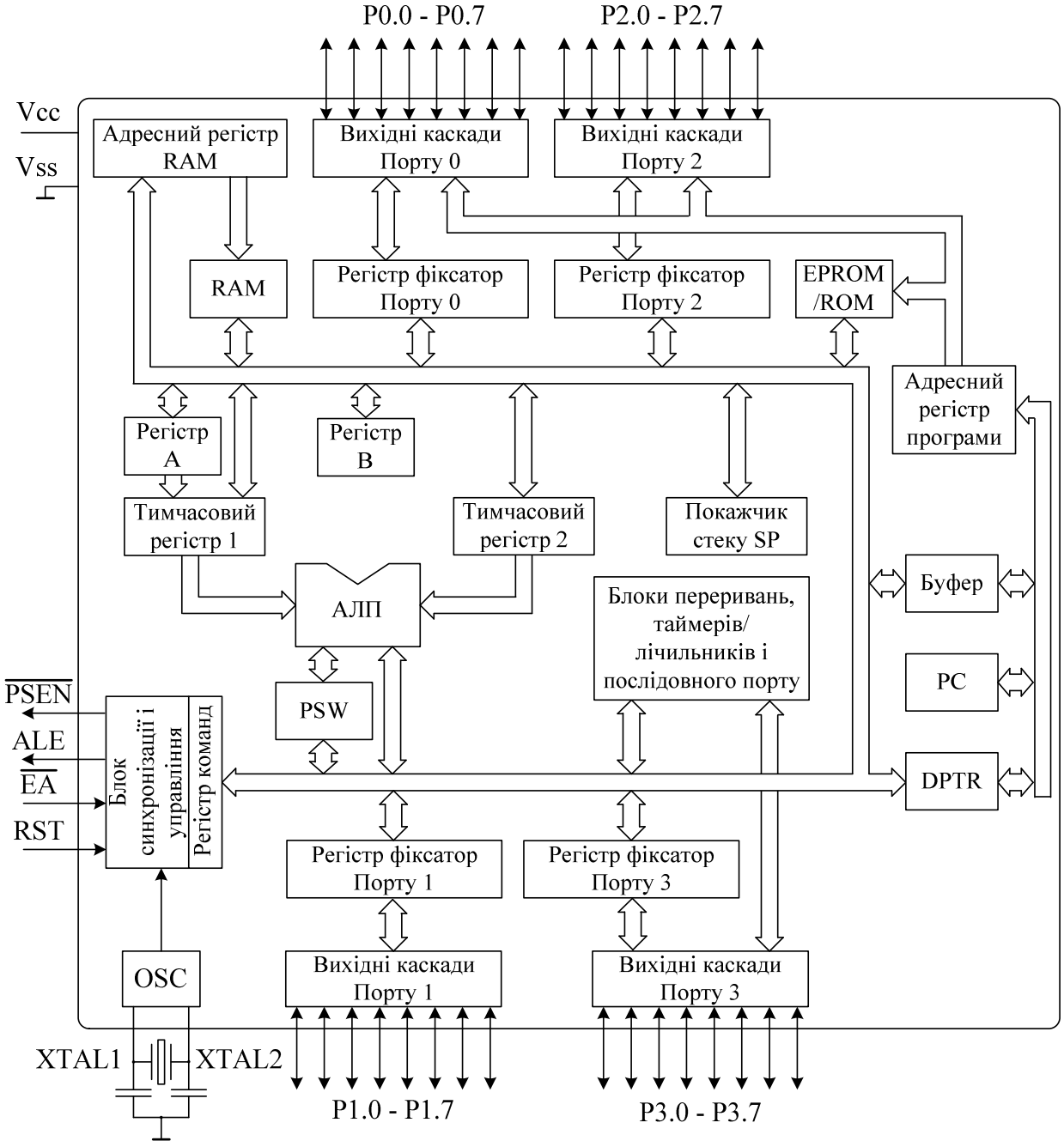


Рисунок 3.1 - Структурна схема мікроконтролера 80C51 фірми INTEL

Джерело живлення підключається до виводів Vss («земля») і Vcc (напруга живлення +5 В). Для управління роботою всіх пристроїв в МК використовується генератор імпульсів, який може працювати від зовнішнього джерела або автономно (у режимі самозбудження). В останньому випадку до виводів XTAL1 і XTAL2 підключається кварцовий резонатор або LC контур. Після ввімкнення живлення необхідно встановити внутрішні пристрої МК в початковий стан подачею імпульсу високого рівня на вивід RST. МК починає роботу з виконання команди, записаної за нульовою адресою.

Паралельні порти мають по 8 виводів. Порти P0 і P2 використовуються також для видачі адреси та обміну даними при зверненні до зовнішньої пам'яті. Крім того, для роботи із зовнішніми ЗП використовуються виводи $\overline{PSEN}$, ALE та $\overline{EA}$.

Виводи, до яких приєднані шини порту P3, можуть використовуватись:

- як приймальна RxD та передавальна TxD лінії послідовного порту;
- для введення сигналів зовнішніх переривань $\overline{INT0}$ і $\overline{INT1}$;
- для підрахунку зовнішніх імпульсів T0 і T1;
- для видачі сигналів запису $\overline{WR}$ і зчитування $\overline{RD}$ на зовнішні запам'ятовуючі пристрої.

Програма і константи записуються в ПЗП (постійний ЗП, ROM - Read Only Memory). Для зберігання змінних даних використовується ОЗП (оперативний ЗП або оперативна пам'ять; RAM - Random Access Memory).

Для спілкування з пристроями пам'яті МК51 має два 16-розрядні регістри: PC (Program Counter - програмний лічильник) і DPTR (Data PoinTeR - покажчик даних). Перший з них використовується тільки для зчитування команд з ПЗП, а другий – для зчитування даних з зовнішніх ПЗП і ОЗП і для запису в ОЗП. Таким чином, МК51 може використовувати адресний простір до 65536 байтів.

Внутрішній ОЗП МК51 має ємність всього 128 байтів. Але адресний простір у нього 8-розрядний, тобто 256 байтів. Старші адреси призначені для звернення до РСФ МК51.

Конфігурація виводів та умовне графічне позначення МК51 наведені на рис. 3.2.

Найменування виводів МК51:

- **Vcc** – джерело живлення;
- **Vss** – земля;
- **P0** – двонаправлений 8-бітний порт вводу/виводу з відкритим стоком вихідних транзисторів. Як вихідний порт він здатен прийняти струм до восьми входів логічних ТТЛШ мікросхем з малим споживанням. При запису

у порт P0 логічних одиниць виводи порту можуть бути використані як входи з високим вхідним опором.

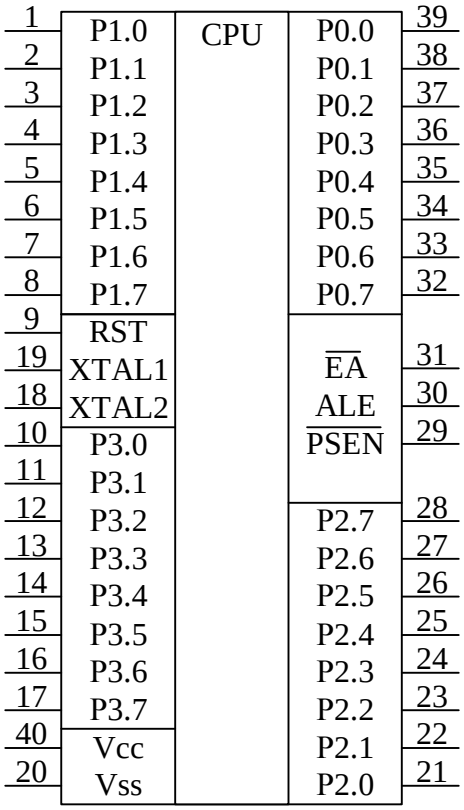


Рисунок 3.2 - Умовне графічне позначення мікроконтролера 8051

При роботі з зовнішньою пам’яттю порт P0 мультиплексує шину адреси (молодший байт) та шину даних. У цьому режимі виводи P0 мають внутрішні резистори, що підтягують їх до джерела живлення (pullup).

При програмуванні внутрішньої пам’яті порт P0 приймає байти даних, що будуть запрограмовані, та через нього читають зміст комірок пам’яті при її верифікації. В режимі верифікації потрібні зовнішні підтягуючі резистори.

- **P1** – двонаправлений 8-бітний порт вводу/виводу, що має внутрішні підтягуючі резистори. Коли до розрядів порту записані логічні одиниці, ці виводи підтягуються до високого логічного рівня внутрішніми резисторами. У цьому режимі вони можуть бути використані як входи для вводу інформації. Якщо входи підключити до низького логічного рівня, то вони виступають джерелом струму I_{IL}, значення якого наведено в технічній документації.

При програмуванні та верифікації внутрішньої пам’яті порт P1 приймає молодший байт адреси.

- **P2** – двонаправлений 8-бітний порт вводу/виводу, що має внутрішні підтягуючі резистори. Коли до розрядів порту записані логічні одиниці, ці виводи підтягуються до високого логічного рівня внутрішніми резисторами. У цьому режимі вони можуть бути використані як входи для вводу інформації. Якщо входи підключити до низького логічного рівня, то вони виступають джерелом струму I_{IL}.

При роботі з зовнішньою пам'яттю і використанні 16-розрядної адреси через порт P2 передається старший байт адреси. У цьому режимі виводи P2 мають внутрішні резистори, що підтягують їх до джерела живлення.

При програмуванні та верифікації внутрішньої пам'яті порт P2 приймає контрольні сигнали та старшу частину адреси.

- **P3** – двонаправлений 8-бітний порт вводу/виводу, що має внутрішні підтягуючі резистори. Коли до розрядів порту записані логічні одиниці, ці виводи підтягуються до високого логічного рівня внутрішніми резисторами. У цьому режимі вони можуть бути використані як входи для вводу інформації. Якщо входи підключити до низького логічного рівня, то вони виступають джерелом струму I_{IL}.

Виводи порту P3 підтримують додаткові функції, що властиві сім'ї MCS-51:

– P3.0 - RxD – вхід послідовних даних УСАПП;
– P3.1 - TxD – вихід послідовних даних УСАПП;
– P3.2 - $\overline{INT0}$ – вхід зовнішнього переривання 0;
– P3.3 - $\overline{INT1}$ – вхід зовнішнього переривання 1;
– P3.4 - T0 – вхід лічильника T/C0;
– P3.5 - T1 – вхід лічильника T/C1;
– P3.6 - $\overline{WR}$ – вихід сигналу синхронізації при запису до внутрішньої пам'яті даних;
– P3.7 - $\overline{RD}$ – вихід сигналу синхронізації при зчитуванні із зовнішньої пам'яті даних.

При програмуванні та верифікації внутрішньої пам'яті порт P3 приймає контрольні сигнали.

- **RST** – сигнал загального скидання. Подача на вивід скидання сигналу високого логічного рівня протягом двох машинних циклів при працюючому тактовому генераторі приводить до скидання мікроконтролера. Внутрішній резистор, що підтягує вивід скидання до «землі» (pulldown), дозволяє використовувати для формування сигналу скидання при подачі напруги живлення лише один зовнішній конденсатор, підключений до джерела живлення.

- **ALE/**$\overline{PROG}$ – (Address Latch Enable) вихідний сигнал, що дозволяє фіксування молодшого байта адреси при звертанні до зовнішньої пам'яті. На цей вивід подається імпульс програмування при програмуванні внутрішньої пам'яті.

Сигнал ALE формується з частотою 1/6 від частоти генератора і може бути використаний для синхронізації зовнішніх пристроїв. При звертанні до зовнішньої пам'яті мікроконтролером використовується кожен другий імпульс ALE.

- $\overline{PSEN}$ – (Program Store Enable) вихідний сигнал, що синхронізує читання зовнішньої пам'яті програм (ПЗП). Якщо мікроконтролер працює з внутрішньою пам'яттю програм, то сигнал $\overline{PSEN}$ неактивний – високого рівня;
- $\overline{EA}$**/Vpp** – (External Access enable) вхідний сигнал, що дозволяє використання зовнішньої пам'яті програм (ПЗП). При $\overline{EA}$ = 0 весь діапазон адрес від 0000H до 0FFFFH належать до зовнішньої пам'яті. Для виконання програми, що записана до внутрішньої пам'яті програм, потрібно забезпечити $\overline{EA}$=1.
- При програмуванні внутрішньої пам'яті на цей вивід подається напруга програмування Vpp = 12.75 В.
- **XTAL1** – вивід для підключення кварцового резонатора; вхід підсилювача генератора та вхід внутрішніх кіл синхронізації.
- **XTAL2** – вивід для підключення кварцового резонатора; вихід підсилювача генератора (внутрішнє підключення виводів XTAL може змінюватись залежно від технології виробництва мікросхеми).

3.2.2. Арифметично-логічний пристрій

Арифметично-логічний пристрій (АЛП) може виконувати арифметичні та логічні операції з 8-бітними операндами: додавання і віднімання, множення і ділення; логічні операції І, АБО, виключне АБО, операції циклічного зсуву, зсуву, скидання та доповнення тощо. АЛП приймає рішення щодо умовних переходів, забезпечує тимчасові регістри для збереження даних та шляхи їх передачі усередині системи. Інші операції будуються на підставі розглянутих примітивів. Найпростіша операція додавання використовується в АЛП для інкрементування вмісту регістрів, автоматичного обчислення адреси наступної інструкції, що буде виконуватись у програмі. Найпростіша операція віднімання використовується в АЛП для декрементування регістрів і порівняння змінних. Найпростіші операції автоматично об'єднуються для виконання в АЛП таких операцій, як, наприклад, інкрементування 16-бітних

регістрових пар. В АЛП реалізується механізм каскадного виконання найпростіших операцій для реалізації складних команд.

Так, наприклад, при виконанні однієї з команд умовної передачі керування за результатом порівняння в АЛП тричі інкрементується програмний лічильник, тричі зчитуються байти з пам'яті команд, виконується арифметичне порівняння двох змінних, формується 16-бітна адреса переходу і приймається рішення про те, виконувати чи не виконувати перехід. Всі перераховані операції виконуються в АЛП усього лише за два машинні цикли.

Важливою особливістю АЛП є його здатність оперувати не тільки байтами, але і бітами. Окремі біти можуть бути встановлені, скинуті, інвертовані, передані, перевірені та використані в логічних операціях. Здатність АЛП оперувати бітами робить його придатним для рішення завдань управління. Алгоритми, що при цьому використовуються, по суті, оперують логічними змінними (істина/хибність), і їх реалізація засобами звичайних мікропроцесорів пов'язана з певними труднощами. Всі ці особливості дали розробникам MCS-51 підставу говорити про наявність в ньому "булевого або логічного процесора".

Завдяки потужним АЛП та набору інструкцій мікроконтролери MCS-51 придатні для управління в реальному часі та для обробки даних. Набор інструкцій підтримує 51 операцію з трьома типами інформаційних об'єктів: логічними (булевськими) (1 біт), байтами (8 біт) і адресами (16 біт). Використовується 11 режимів адресації (7 для даних і 4 для управління послідовністю виконання програми). Оскільки більшість інструкцій підтримує декілька режимів адресації, то шляхом комбінування "операція/режим адресації" базова кількість команд з 111 зростає до 255 інструкцій з 256 можливих при однобайтному коді операції [18].

3.2.3. Організація пам'яті

Пам'ять програм і пам'ять даних МК 51 фізично і логічно поділені, вони мають різні адресні простори, керуються різними сигналами, доступ до них відбувається за допомогою різних команд. Карта пам'яті мікроконтролера МК51 наведена на рис. 3.3.

Пам'ять програм, що розміщена на кристалі МК51, має ємність 4 кбайти і призначена для зберігання команд, констант, таблиць перекодування вхідних та вихідних змінних тощо. Використання 16-бітної шини адреси дозволяє збільшувати пам'ять програм до 64 кбайт шляхом підключення зовнішніх мікросхем. При одночасному використанні внутрішньої та зовнішньої пам'яті, робоча зона зовнішньої починається з адреси 1000H.

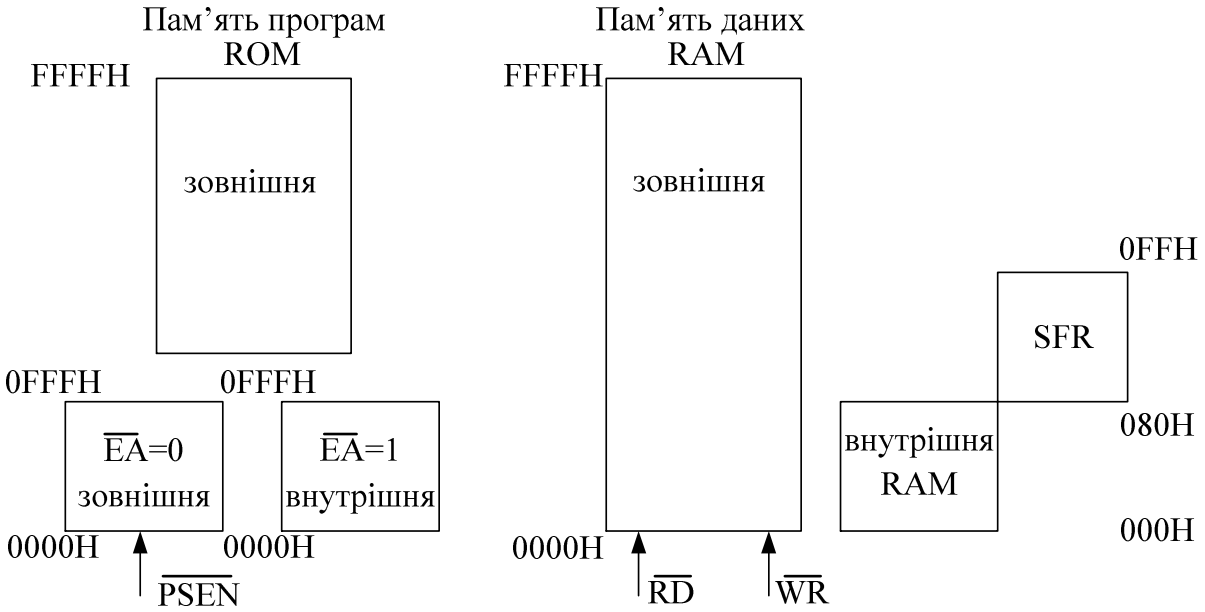


Рисунок 3.3 - Карта пам’яті мікроконтролера МК51

Пам’ять даних (ОЗП) призначена для зберігання змінних у процесі виконання прикладної програми. Пряму або непряму адресацію допускають 128 байт пам’яті даних, що розміщено на кристалі МК51. Ця пам’ять може бути розділена на три сегменти, як зображено на рис. 3.4.

Рисунок 3.4 - Молодші 128 байтів внутрішньої пам’яті даних

Початкові комірки оперативної пам’яті з адресами від 00H до 1FH (32 байти) використовуються під чотири регістрові банки: bank0 – bank3.

Після скидання мікроконтролер автоматично обирає bank0, переключитися на інші банки користувач може програмно. До складу кожного банку входить вісім однобайтних регістрів загального призначення: R0 - R7. До перших 32 байтів можна звертатись як за абсолютною адресою, так і за мнемонічним ім'ям, попередньо вибравши відповідний регістровий банк. Розподіл адрес регістрів у банках 0, 1, 2 і 3, наведений в табл. 3.1.

Таблиця 3.1 - Адреси регістрів R0 - R7 у різних банках

	R0	**Rl**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
Банк 0	00h	01h	02h	03h	04h	05h	06h	07h
Банк 1	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh
Банк 2	10h	11h	12h	13h	14h	15h	16h	17h
Банк 3	18h	19h	1Ah	1Bh	1Ch	1Dh	1Eh	1Fh

16 байтів з адресами від 20H до 2FH створюють ділянку з 128 бітів, що мають власні адреси з 00H до 7FH. Всі 128 бітів можуть бути прямо адресовані. Кожен з 16 байтів також може адресуватись як байт.

Байти з адресами від 30H до 7FH створюють основний робочий простір внутрішньої пам'яті даних.

Пам'ять програм, так само як і пам'ять даних, може бути розширена до 64 кбайт шляхом підключення зовнішніх мікросхем. Адресні простори внутрішньої та зовнішньої пам'яті даних не пересікаються.

3.2.4. Регістри спеціальних функцій

До адресного простору внутрішньої пам'яті даних приєднуються адреси регістрів спеціальних функцій, що займають старші 128 байт пам'яті даних. Мнемонічні імена РСФ, їх адреси та початковий стан після скидання наведено в табл. 3.2. РСФ допускають тільки пряму адресацію.

Програмний лічильник **PC** (Program Counter) ніяк не відображається на адресному просторі ОЗП, щоб ненавмисно не зіпсувати його вміст, хоча його вміст може використовуватися при програмуванні.

Арифметичні і логічні дії виконуються за допомогою регістра акумулятора з мнемонічним ім'ям **A** (Accumulator). Мнемонічні імена з таблиці 3.1 підтримуються всіма програмними засобами, але синоніми, що використовуються програмістами, наприклад, акумулятор – ACC, потребують попереднього визначення перед першим використанням у програмі.

Під час операцій множення і ділення використовується регістр **B**. Для інших операцій це регістр загального застосовування.

Регістр слова стану програми **PSW** (Program Status Word) зберігає

інформацію про результат виконання команди та стан програми.

Восьмирозрядний регістр - покажчик стека **SP** (Stack Pointer) може розміщатися будь-де в пам’яті даних на кристалі мікроконтролера, але після скидання автоматично ініціалізується адресою 07H, що забезпечує збереження даних у комірці внутрішньої пам’яті даних з адресою 08H. Тіло стека зростає в бік збільшення адреси. Вміст покажчика стека збільшується раніше збереження даних при виконанні інструкцій PUSH або CALL. Зменшення вмісту покажчика стека відбувається після виконання команд POP та RET. Подібний спосіб адресації елементів стека називають переддекрементним / постдекрементним.

Таблиця 3.2 - Регістри спеціальних функцій

Символ		Найменування регістра	Адреса	Початковий стан
*	A	Акумулятор	0E0H	000H
*	B	Регістр B	0F0H	000H
*	PSW	Регістр слова стану програми	0D0H	000H
	SP	Покажчик стека	81H	007H
	DPTR	Покажчик даних: старший байт (DPH)	83H	000H
		молодший байт (DPL)	82H	000H
*	P0	Порт 0	80H	0FFH
*	P1	Порт 1	90H	0FFH
*	P2	Порт 2	0A0H	0FFH
*	P3	Порт 3	0B0H	0FFH
*	IP	Регістр пріоритетів переривань	0B8H	xxx00000B
*	IE	Регістр маски переривань	0A8H	0xx00000B
	TM0D	Регістр режиму таймера/лічильника	89H	000H
*	TC0N	Регістр керування/статусу таймера	88H	000H
	TH0	Таймер 0 (старший байт)	8CH	000H
	TL0	Таймер 0 (молодший байт)	8AH	000H
	TH1	Таймер 1 (старший байт)	8DH	000H
	TL1	Таймер 1 (молодший байт)	8BH	000H
*	SCON	Регістр керування УСАПП	98H	000H
	SBUF	Буфер послідовних даних УСАПП	99H	xxxxxxxxB
	PCON	Регістр керування потужністю	87H	0xxx0000B

Знак ***** показує, що даний регістр допускає адресацію окремих бітів.
Буквою **x** позначено довільне значення біта.
Символи **H** і **B** відповідають шістнадцятковому і двійковому кодам.

Покажчик даних **DPTR** складається з двох байтів: старшого **DPH** і молодшого **DPL** (H і L – від слів High і Low відповідно). Його призначення – зберігання 16-розрядної адреси при звертанні до 64 кбайт пам'яті даних або програм. Можливе використання регістра покажчика даних як одного 16-розрядного або двох незалежних 8-розрядних регістрів.

Мнемонічні імена РСФ паралельних портів збігаються з позначеннями їх виводів: **P0**, **P1**, **P2** і **P3**.

Таймери/лічильники Т/С (Timer/Counter) з номерами 0 і 1, які використовуються як для підрахунку тактових імпульсів (режим таймера), так і для рахунку імпульсів на зовнішніх входах, працюють з використанням 16-розрядних регістрів, що будуються з регістрових пар TH0 - старший, TL0 - молодший байти та TH1, TL1 відповідно нульового та першого таймера-лічильника.

Буфер послідовних даних **SBUF** (Serial BUFfer) обслуговує приймач і передавач, що входять до складу УСАПП. Фізично SBUF складається з двох різних буферних регістрів – передавача та приймача. Коли дані завантажуються до SBUF, вони поміщаються до буфера передавача, де знаходяться до завершення послідовної передачі. Передача байта до SBUF ініціює початок передачі. Якщо дані читаються з SBUF, вони читаються з буфера приймача.

Регістри керування. РСФ **IP**, **IE**, **TMOD**, **TCON** і **PCON** містять біти керування та стану системи переривань, таймерів – лічильників, послідовного порту та системи енергозбереження, значення бітів можуть бути програмно змінені. Призначення окремих бітів цих регістрів буде розглянуто у відповідних розділах книги.

Регістр слова стану програми **PSW**.

При виконанні багатьох інструкцій в АЛП формується ряд ознак результату операції (прапори), що фіксуються в регістрі PSW, або виконання інструкції залежить від стану одного з прапорів. У табл. 3.3 наводиться перелік бітів PSW, подаються їхні мнемонічні імена і умови їхнього формування.

Точка, що використана в імені позиції біта, відділяє найменування регістра від номера позиції біта в ньому. Ім'я PSW.1 відповідає першому біту регістра PSW. Ця нотація спрощує сприйняття інформації і часто зустрічається в технічній документації. Програмні засоби, що використовуються для розробки програм для мікроконтролерів, не завжди дозволяють використання точки в імені змінної чи константи, тому в тексті програм замість символу точки «.» доцільно використовувати символ підкреслення «_», наприклад PSW_1, що забезпечить універсальність програмного коду.

Таблиця 3.3 - Формат слова стану програми

Символ	Позиція	Адреса	Ім’я і призначення
C	PSW.7	0D7H	Прапор переносу. Установлюється/скидається апаратно чи програмно при виконанні арифметичних та логічних операцій
AC	PSW.6	0D6H	Прапор додаткового переносу. Установлюється тільки апаратно. Сигналізує про перенос чи позику в біті 3. Використовується у двійково-десятиричній (BCD - Binary Coded Decimal) арифметиці
F0	PSW.5	0D5H	Прапор F0. Установлюється/скидається та перевіряється програмно, як прапор користувача
RS1	PSW.4	0D4H	Біти 1 та 0 вибору банку регістрів. Установлюються/скидаються програмно для вибору робочого банку регістрів (див. примітку)
RS0	PSW.3	0D3H	
OV	PSW.2	0D2H	Прапор переповнення. Установлюється/скидається при виконанні арифметичних операцій. Сигналізує про переповнення
-	PSW.1	0DlH	Не використовуються
P	PSW.0	0D0H	Прапор парності. Установлюється/скидається апаратно в кожному циклі команди для індикації парної/непарної кількості одиничних бітів в акумуляторі – доповнює кількість одиничних бітів акумулятора до парного числа

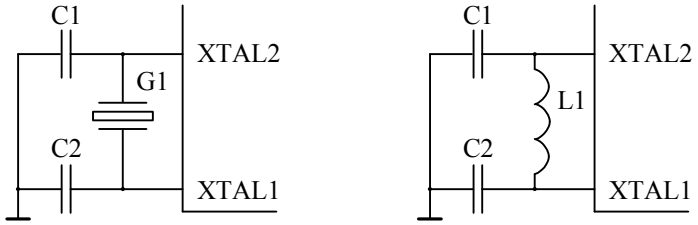
Примітка. Вибір робочого банку регістрів.

Біти		Банк	Адреса
RS1	RS0		
0	0	Банк 0	00H - 07H
0	1	Банк 1	08H - 0FH
1	0	Банк 2	10H - 17H
1	1	Банк 3	18H - 1FH

АЛП самостійно не управляє бітами вибору банку регістрів і їх значення повністю визначається прикладною програмою.

3.2.5 Пристрій управління і синхронізації

Кварцовий резонатор, що підключається до зовнішніх виводів XTAL1 і XTAL2 корпуса МК51 (рис. 3.5, *а*), управляє роботою внутрішнього генератора, який в свою чергу формує сигнали синхронізації. Конденсатори С1, С2 мають ємність 30 пФ ± 10 пФ.



а *б*

Рисунок 3.5 - Підключення пристроїв синхронізації

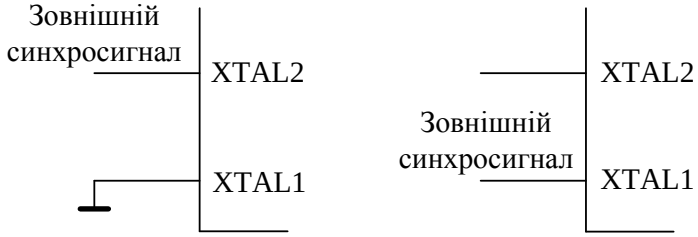
Замість кварцового резонатора можна використовувати LC-контур (рис. 3.6, *б*). Ємність конденсаторів С1 та С2 дорівнює *С*. Частота синхронізації *F* знаходиться за формулою

$$F = 1/(2\pi(LC')^{1/2}),$$

де $C' = (C + 3C_{pp})/2$, C_{pp}= 10 пФ – ємність виводу.

Стабільність частоти у цьому випадку зменшується, але й ціна деталей, що використовуються, дещо менша.

В багатопроцесорних системах, та системах що вимагають загальної системи синхронізації, МК може синхронізуватись від зовнішнього джерела синхронізуючих імпульсів. Схема підключення зовнішнього джерела залежить від типу МК. Для вітчизняних МК К1816ВЕ51 (Intel HMOS 8051) підключення відбувається за схемою, наведеною на рис. 3.6, а; для МК К1830ВЕ51 (Intel CHMOS 80C51) - за схемою, наведеною на рис. 3.6, б.



а *б*

Рисунок 3.6 - Підключення зовнішньої синхронізації

Цикл виконання команди.

Залежно від складності виконуваних дій цикл виконання команди займає від одного до чотирьох тактів (машинних циклів), тривалість кожного з яких дорівнює дванадцяти періодам коливань кварцового резонатора. Якщо

використовується кварцовий резонатор на 12 МГц, то тривалість такту дорівнює 1 мкс. Цикл виконання чергової команди починається зі зчитування її коду з пам'яті програм за адресою, записаною в регістр PC (програмний лічильник). Після читання кожного з байтів команди вміст цього регістра збільшується на 1, таким чином, після читання останнього байта команди програмний лічильник містить адресу першого байта наступної команди. Потім здійснюються читання операндів, обробка одержаної інформації і запам'ятовування результату операції. Залежно від коду операції виконуваної команди ті чи інші перераховані дії можуть не виконуватися. У системі команд МК є такі, які можуть змінити вміст програмного лічильника. Ці команди називаються керуючими. Одні з них здійснюють це залежно від результатів поточної або попередніх операцій, інші – незалежно від виконання будь-яких умов. За рахунок використання керуючих команд при виконанні програми мікроконтролер не тільки здійснює обчислення, але й виконує логічні дії, наприклад, вибір варіанта або циклічного повторення одних і тих же дій.

Виконання наступної команди починається тільки після завершення поточної, тобто одночасне виконання хоча б двох команд неможливе. Проте під час виконання програми окремі пристрої МК можуть працювати автономно.

Пристрій управління МК51 на основі сигналів синхронізації формує машинний цикл фіксованої тривалості, що складається з послідовності шести станів (S1 – S6), кожний тривалістю два періоди кварцового резонатора. Кожний стан підрозділяється на дві фази (P1, P2), як показано на рис. 3.7, *а*. Весь машинний цикл складається з 12 фаз, починаючи з фази S1P1 і закінчуючи фазою S6P2. Зовнішніми сигналами, що спостерігаються, є тільки сигнали синхронізації і фіксування молодшого байта адреси зовнішньої пам'яті. Як видно з часової діаграми на рис. 3.7, *б*, сигнал ALE формується двічі за один машинний цикл (S1P2-S2P1; S4P2-S5P1) і використовується для управління процесом звернення до зовнішньої пам'яті.

Часові діаграми на рис. 3.7, *в-е* ілюструють роботу МК51 при вибірці і виконанні команд різноманітного ступеня складності. Більшість команд МК51 виконується за один машинний цикл. Деякі команди, що оперують з 2-байтовими словами або пов'язані зі звертанням до зовнішньої пам'яті, виконуються за 2 машинних цикли. Тільки команди ділення і множення вимагають чотири машинні цикли. На основі цих особливостей роботи пристрою управління МК51 здійснюється розрахунок часу виконання прикладних програм.

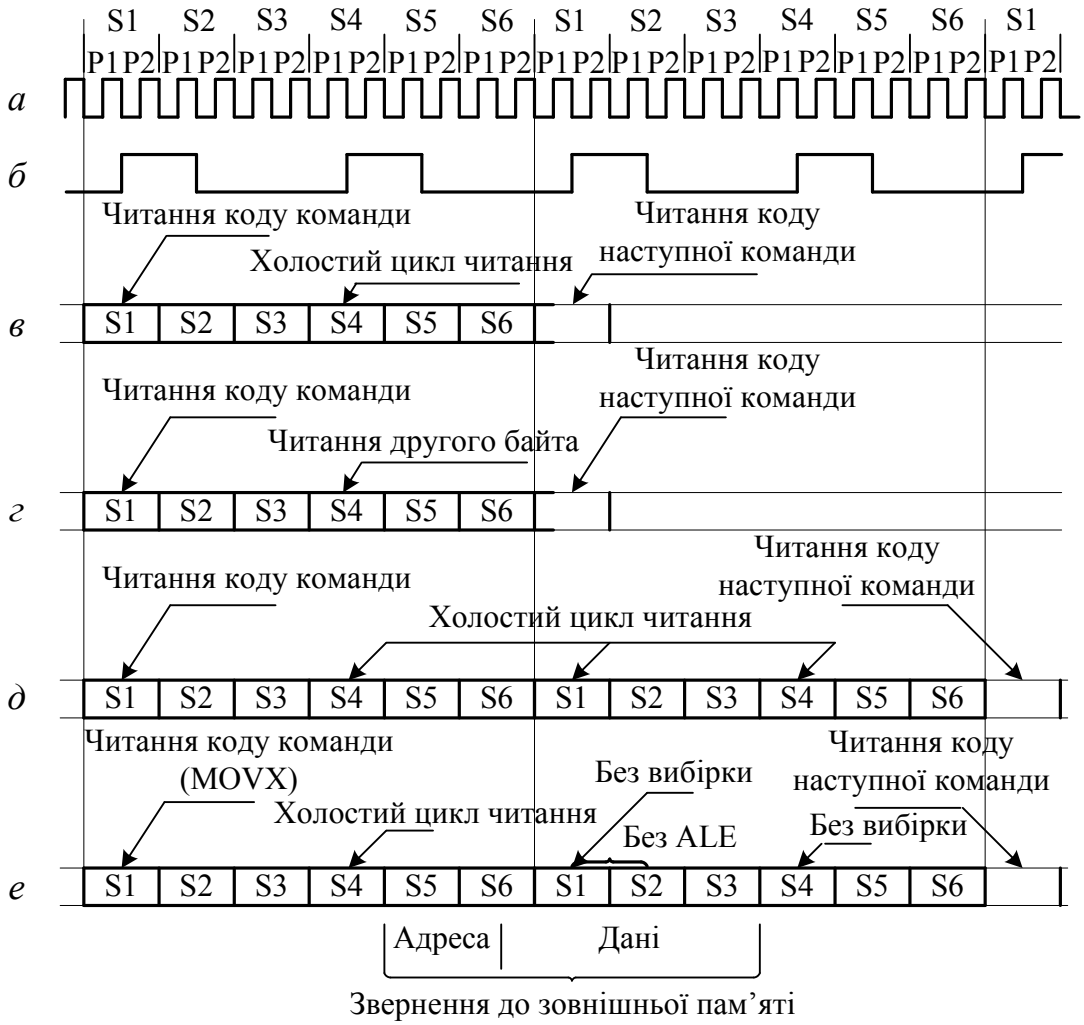


Рисунок 3.7 - Послідовність вибірки та виконання команд в МК51:

а – сигнал синхронізації;

б – сигнал фіксування адреси ALE;

в – команда 1 байт/1 цикл, наприклад INC A;

г – команда 2 байти/1 цикл, наприклад ADD A, #d;

д – команда 1 байт/2 цикли, наприклад INC DPTR;

е – команда 1 байт/2 цикли, наприклад MOVX

3.2.6. Порти вводу-виводу інформації

Всі чотири порти МК51 призначені для вводу та виводу інформації. Кожен з портів містить восьмирозрядний регістр-фіксатор (РСФ Р0 – Р3) і має можливість адресації як байта, так і окремого біта для установки (скидання) і перевірки розрядів порту за допомогою програмного забезпечення.

Фізичні адреси фіксаторів Р0, Р1, Р2, Р3 складають:

Р0 – 80Н, адреси бітів порту 80Н – 87Н;

Р1 – 90Н, адреси бітів порту 90Н – 97Н;

Р2 – 0А0Н, адреси бітів порту 0А0Н – 0А7Н;

Р3 – 0В0Н, адреси бітів порту 0В0Н – 0В7Н.

Схема кожного розряду порту містить регістр-фіксатор, вхідний буфер і вихідний драйвер. Вихідні драйвери портів Р0 і Р2, а також вхідний буфер порту Р0 використовуються у разі звертання до зовнішньої пам’яті. При цьому через порт Р0 в режимі мультиплексування спочатку виводиться молодший байт адреси зовнішньої пам’яті, а після цього видається або приймається байт даних. Через порт Р2 виводиться старший байт адреси в тих випадках, коли розрядність адреси дорівнює 16 бітам, в інший час на виводи порту Р2 продовжують подаватись дані з РСФ Р2. Всі виводи порту Р3 – багатофункціональні, вони не тільки служать виводами порту, але й можуть використовуватись для виконання додаткових (альтернативних) функцій. Додаткові функції можуть використовуватись тільки, якщо у відповідному біті РСФ Р3 записана логічна одиниця, в іншому випадку розряд порту останеться в стані логічного нуля. Спрощені функціональні схеми одного розряду портів наведено на рис. 3.8 - рис. 3.11. Регістр – фіксатор виконано на D-тригері, що синхронізується імпульсом з внутрішньої шини у відповідь на сигнал «записати у фіксатор» від АЛП. Вивід Q фіксатора підключається до внутрішньої шини мікроконтролера по сигналу АЛП - «читати фіксатор», виводи розрядів порту - за сигналом АЛП - «читати вивід».

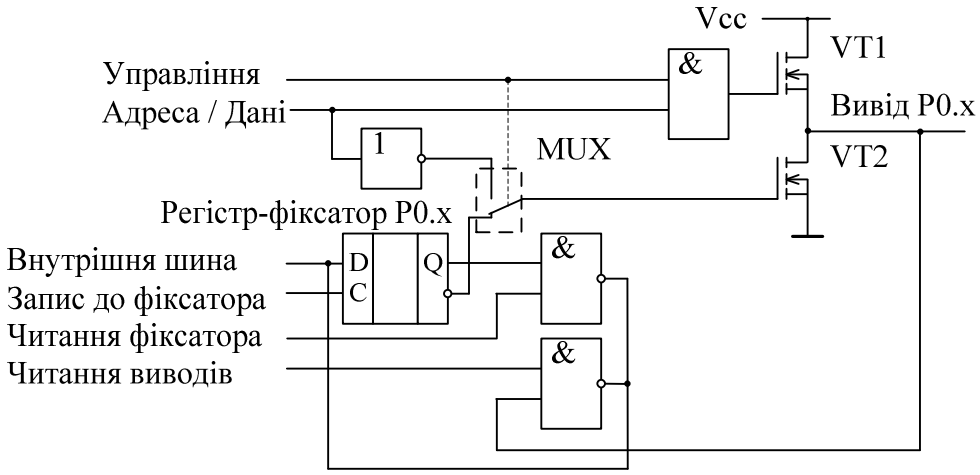


Рисунок 3.8 - Схема порту Р0

Для доступу до зовнішньої пам’яті, вихідні драйвери портів Р0 і Р2 (рис. 3.8, рис. 3.10) можуть внутрішнім сигналом *Управління* підключатись до внутрішніх шин *Адреса* та *Адреса/Дані*. Протягом доступу до пам’яті стан РСФ Р2 залишається незмінним, а до РСФ Р0 записуються одиниці.

Порти Р1, Р2 та Р3 мають внутрішні блоки, що підтягують їх виводи до потенціалу джерела живлення. Порт Р0 має вихід з відкритим стоком. Кожна лінія вводу-виводу може бути незалежно використана як вхід чи як вихід, за

виключенням портів P0 і P2 в разі їх використання для доступу до зовнішньої пам'яті. Для використання виводу як вхід, до фіксатора відповідного біта повинна бути записана одиниця, що вимкне транзистор вихідного драйвера порту, при цьому виводи портів P1, P2 та P3 лишаються підтягнутими до потенціалу логічної одиниці, однак підключення зовнішнього джерела з напругою логічного нуля може встановити на них відповідний вхідний сигнал.

Порт P0 не має внутрішніх джерел, що підтягують його виводи. Верхній транзистор вихідного драйвера включається лише при формуванні логічної одиниці при звертанні до зовнішньої пам'яті, в інших випадках він вимкнений. Тому вихідні лінії порту P0 використовуються як виходи з відкритим стоком. Запис одиниці у фіксатор приводить до відключення обох транзисторів вихідного драйвера і переходу лінії до третього стану, що дає можливість використовувати її як вхід з високим вхідним опором.

Наявність внутрішніх блоків, які підтягують виводи портів P1, P2 та P3 до потенціалу джерела живлення, дає привід говорити, що вони «квазідвоспрямовані». При конфігуруванні виводу як входу він має потенціал логічної одиниці без зовнішнього підключення і стає джерелом струму, що випливає, при прикладенні ззовні потенціалу логічного нуля. Вимогою до зовнішнього джерела сигналу є здатність пропустити цей струм без підвищення своєї напруги.

Після скидання мікроконтролера в регістри-фіксатори портів автоматично записуються одиниці, що переводить їх в режим вводу і встановлює на виводах високий логічний рівень.

Для отримання сигналу логічної одиниці на виводах порту P0 необхідно використовувати зовнішні резистори, підключені до джерела живлення.

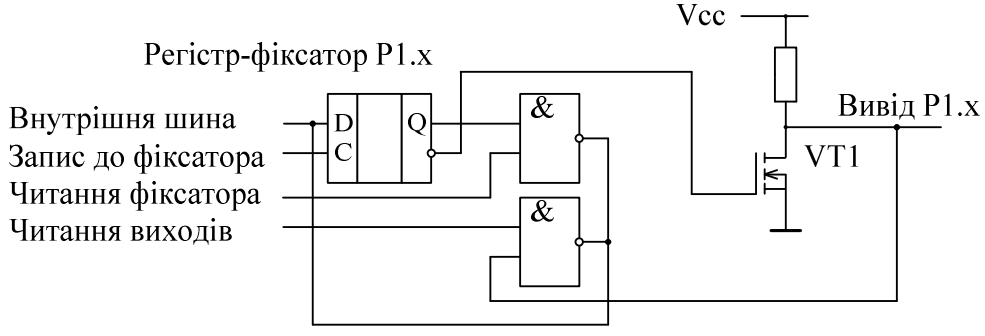


Рисунок 3.9 - Схема порту P1

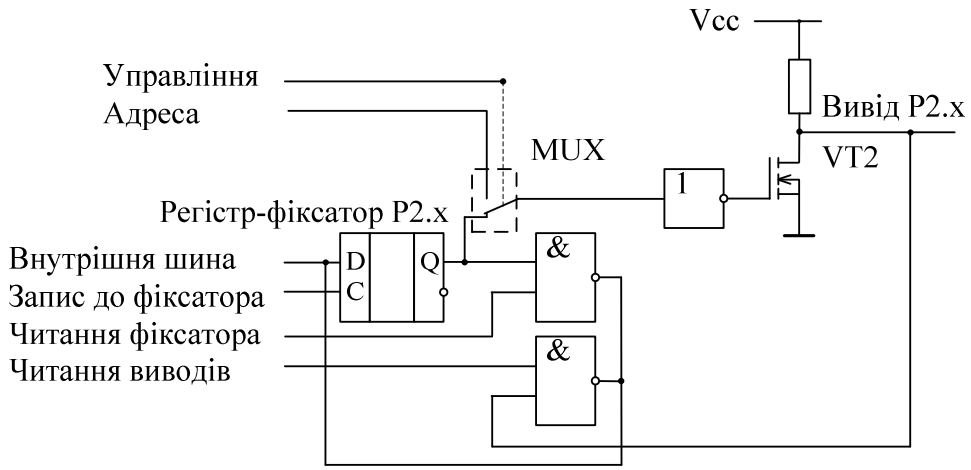


Рисунок 3.10 - Схема порту P2

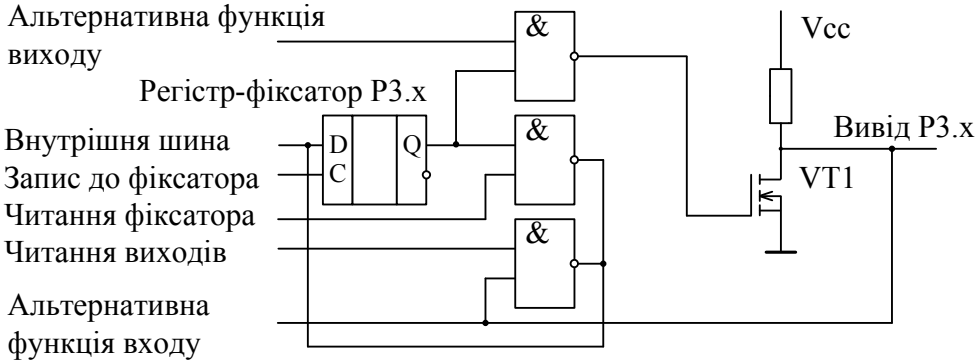


Рисунок 3.11 - Схема порту P3

Як видно з рис. 3.11, якщо фіксатор біта порту P3 містить одиницю, то вихідний стан виводу контролюється сигналом «альтернативна функція виводу». Реальне значення логічного сигналу виводу порту завжди доступне альтернативній функції вводу інформації. Якщо користувач не використовує альтернативних функцій, то порт P3 працює як і інші. Альтернативні функції виводів P3 перераховані в табл. 3.4.

Запис в порт. При виконанні команди, що змінює вміст регістра-фіксатора порту, нове значення надходить на вхід регістра в момент S6P2 останнього циклу команди, однак передача вмісту регістра-фіксатора на його вихід здійснюється під час фази P1 (протягом фази P2 вихідний буфер зберігає значення, що було у попередній фазі P1), і, отже, новий вміст регістра-фіксатора з’являється на вихідних контактах порту тільки в момент S1P1 наступного машинного циклу. Графічно це подано на рис. 3.23.

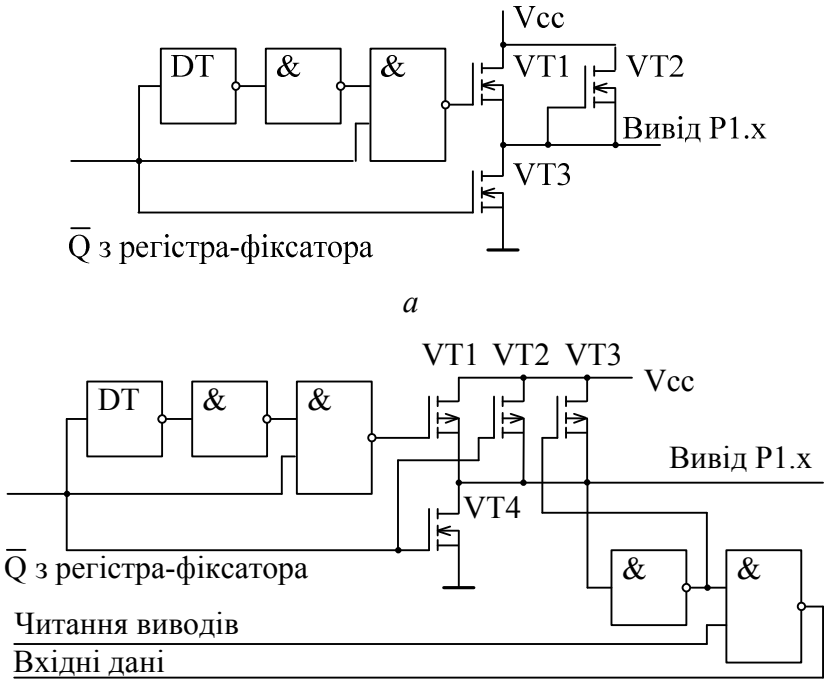
Якщо відбувається зміна стану розряду портів P1- P3 з логічного нуля до логічної одиниці, то у фазах S1P1 та S1P2, коли цей перехід відбувся, до виводів порту підключаються додаткові джерела енергії, що дозволяє при-

скорити перехідний процес. Струм додаткових джерел майже в 100 разів перевищує сталий струм виводу порту.

Таблиця 3.4 - Альтернативні функції виводів порту Р3

Символ	Позиція	Ім’я і призначення
$\overline{RD}$	Р3.7	Синхронізація зчитування зовнішньої пам’яті даних. Активний сигнал низького рівня формується апаратно
$\overline{WR}$	Р3.6	Синхронізація запису зовнішньої пам’яті даних. Активний сигнал низького рівня формується апаратно
Т1	Р3.5	Вхід таймера/лічильника 1 чи тест-вхід
Т0	Р3.4	Вхід таймера/лічильника 0 чи тест-вхід
$\overline{INT1}$	Р3.3	Вхід запиту переривання 1. Сприймається сигнал низького рівня чи зріз
$\overline{INT0}$	Р3.2	Вхід запиту переривання 0. Сприймається сигнал низького рівня чи зріз
TXD	Р3.1	Вихід передавача послідовного порту в режимі УАПП. Вихід синхронізації в синхронному режимі УСАПП
RXD	Р3.0	Вхід приймача послідовного порту в режимі УАПП. Ввід-вивід даних в синхронному режимі УСАПП

Схеми вихідних каскадів портів Р1 - Р3 наведені на рисунках 3.12, *а, б*. Для старших мікроконтролерів, виготовлених за HMOS технологією (рис. 3.12, *а*), транзистор VT2, що включено в режимі джерела струму, при підключенні виводу до нульового потенціалу забезпечує струм до 0,25 мА. При переході «0» - «1» паралельно до VT2 підключається транзистор VT1, що вмикається на інтервалі S1 і забезпечує струм до 30 мА протягом двох періодів синхронізації [15]. Часові параметри забезпечуються елементом часової затримки DT. В CHMOS версії мікроконтролера (рис. 3.12, б) до схеми вихідного каскаду входить три польових транзистори р-типу. Вимикання транзистора VT4 проходить одночасно з вмиканням транзистора VT2 та транзистора VT1 на інтервалі S1. Вмикання транзистора VT1 забезпечує протікання струму, що форсує перемикання виводу порту. Після появи на виході порту логічної одиниці через інвертор DD1 вмикається транзистор VT3. Інвертор і транзистор створюють фіксатор, що підтримує на виході порту стан логічної одиниці. При надходженні на вивід негативної перешкоди, стан інвертора DD1 і транзистора VT3 змінюється на протилежний. Для відновлення вихідного стану виводу використовується малопотужний транзистор VT2, що управляється регістром-фіксатором порту.



а

б

Рисунок 3.12 - Схема вихідного каскаду портів P1 - P3:
а - HMOS мікроконтролерів; *б* - CHMOS мікроконтролерів

Навантажувальна здатність портів. Вихідні лінії портів 1, 2 і 3 можуть працювати на один вхід мікросхеми ТТЛ. Лінії порту 0 можуть бути безпосередньо навантажені на два входи мікросхем ТТЛ кожна. При підключенні до ліній порту 0 мікросхем, що виготовлені за польовою технологією, необхідно використовувати зовнішні резистори, що підтягують лінії порту 0 до джерела живлення, за винятком випадку, коли шина порту 0 використовується як шина адреси/даних зовнішньої пам'яті.

Вхідні сигнали для МК51 можуть формуватися ТТЛ або n-МОН - мікросхемами. Допустимо використовувати як джерела сигналів для МК51 схеми з відкритим колектором або відкритим стоком. Однак при цьому час зміни вхідного сигналу при переході з 0 в 1 виявиться надто затягнутим, що пояснюється особливостями роботи вихідних драйверів портів.

Навантажувальна здатність портів різних мікроконтролерів сім'ї MCS-51 може сильно відрізнятися від наведеної.

Особливості роботи портів. Звернення до портів вводу-виводу можливе з використанням команд, що оперують з байтом або окремим бітом. При цьому в тих випадках, коли порт є водночас джерелом інформації і місцем призначення результату, пристрій управління автоматично реалізує спеціальний режим, що називається "зчитування-модифікація-запис". Цей режим звернення припускає ввід сигналів не з зовнішніх виводів порту, а з його

регістра-фіксатора, що дозволяє виключити неправильне зчитування раніше виведеної інформації.

Подібний механізм звернення до портів реалізований в таких командах:

ANL – логічне І, наприклад ANL P1, A;

ORL –логічне АБО, наприклад ORL P2, A;

XRL – виключне АБО, наприклад XRL P3, A;

JBC – перехід, якщо біт встановлено, і наступне скидання біта, наприклад JBC P1.1, LABEL;

CPL – інверсія біта, наприклад CPL P3. 3;

INC – збільшення, наприклад INC P2;

DEC – зменшення, наприклад DEC P2;

DJNZ – зменшення і перехід, якщо не нуль, наприклад DJNZ P3, LABEL;

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

SET PX.Y – встановлення біта Y порту X;

CLR PX.Y – скидання біта Y порту X.

Зовсім не очевидно, що останні три команди з наведеного списку є командами “зчитування-модифікація-запис”. Однак це саме так. За цими командами байт спочатку зчитується, а після цього новий байт записується в регістр-фіксатор. Причиною, з якої команди “зчитування-модифікація-запис” забезпечують розподілений доступ до регістра-фіксатора порту і до зовнішніх виводів порту, є необхідність виключити можливість неправильного читання рівнів сигналів на зовнішніх виводах. Припустимо для прикладу, що лінія Y порту X з’єднується з базою біполярного транзистора і вихідний сигнал на ній призначений для його управління. Коли в даний біт записана 1, то транзистор вмикається. Якщо для перевірки стану виконавчого механізму (в нашому випадку - транзистора) прикладній програмі потрібно прочитати стан вихідного сигналу в тому ж біті порту, то зчитування сигналу із зовнішнього виводу порту, а не з D-тригера регістра-фіксатора порту призведе до неправильного результату: одиничний сигнал на базі транзистора має відносно низький рівень і буде інтерпретований МК як сигнал «0». Команди “зчитування-модифікація-запис” реалізують зчитування з регістра-фіксатора, а не із зовнішнього виводу порту, що забезпечує отримання правильного значення «1».

3.2.7. Доступ до зовнішньої пам’яті

У МКС, побудованих на основі МК51, можливе використання двох типів зовнішньої пам’яті: постійної пам’яті програм і оперативної пам’яті даних. Ємність кожного типу пам’яті сягає 64 кбайт.

Доступ до зовнішньої пам’яті програм здійснюється за допомогою керуючого сигналу $\overline{PSEN}$, що виконує функцію синхронізації читання.

Доступ до зовнішньої пам’яті даних забезпечується керуючими сигналами $\overline{RD}$ і $\overline{WR}$, що формуються на лініях P3.7 і P3.6 при виконанні портом P3 альтернативних функцій.

У разі звертання до пам’яті програм завжди використовується 16-бітова адреса. Доступ до пам’яті даних можливий з використанням:

- 16-бітної адреси (MOVX A, @DPTR);
- 8-бітної адреси (MOVX A, @Ri).

У будь-яких випадках використання 16-бітної адреси старший байт адреси зберігається незмінним на виводах порту P2 протягом одного циклу запису або читання, вміст регістра-фіксатора не змінюється. Якщо використовується 8-бітна адреса (MOVX A, @Ri), то сигнали на зовнішніх виводах P2 відповідають регістру-фіксатору і не змінюються протягом усього циклу зовнішньої пам’яті.

Через порт P0 в режимі мультиплексування здійснюється видавання молодшого байта адреси і передача байта даних. Для синхронізації запису молодшого байта адреси в зовнішній регістр використовується сигнал ALE. Адреса дійсна на спаді сигналу ALE. В циклі запису (рис. 3.13) байт даних, що записується, з’являється на виводах порту P0 перед появою сигналу $\overline{WR}$ і дійсний як по передньому, так і по задньому фронтах сигналу. У циклі читання (рис. 3.14) байт даних, що читається, приймається в порт P0 по наростанню сигналу $\overline{RD}$.

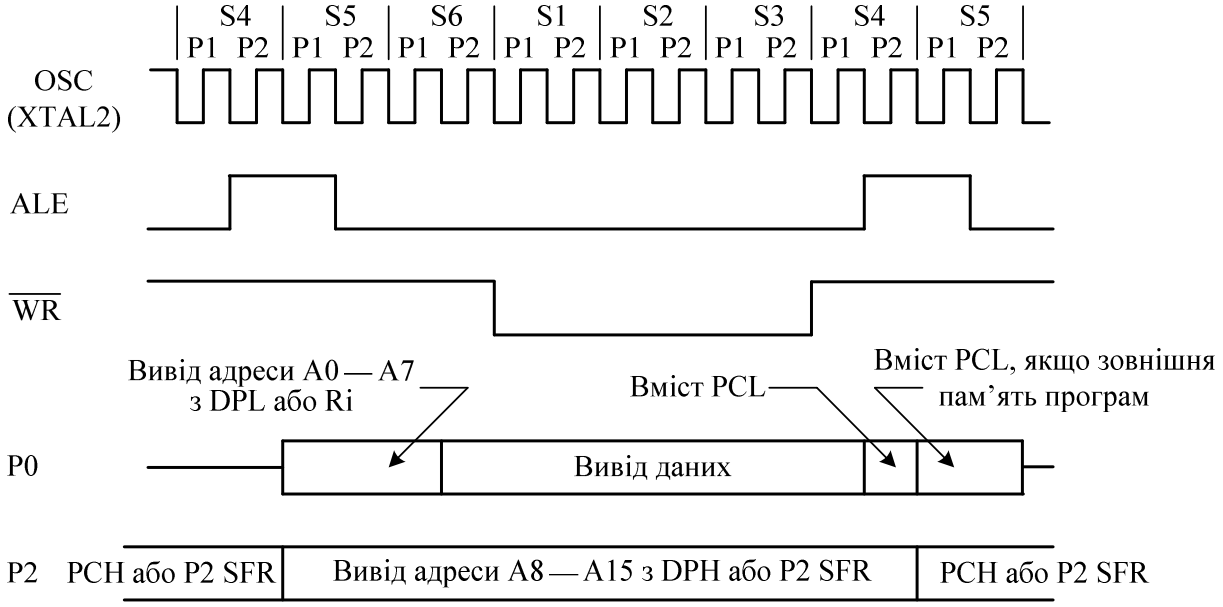


Рисунок 3.13 - Цикл запису зовнішньої пам’яті даних

При будь-якому зверненні до зовнішньої пам’яті пристрій управління МК51 завантажує в регістр-фіксатор порту P0 код 0FFH, стираючи інформацію, що в ньому зберігалась.

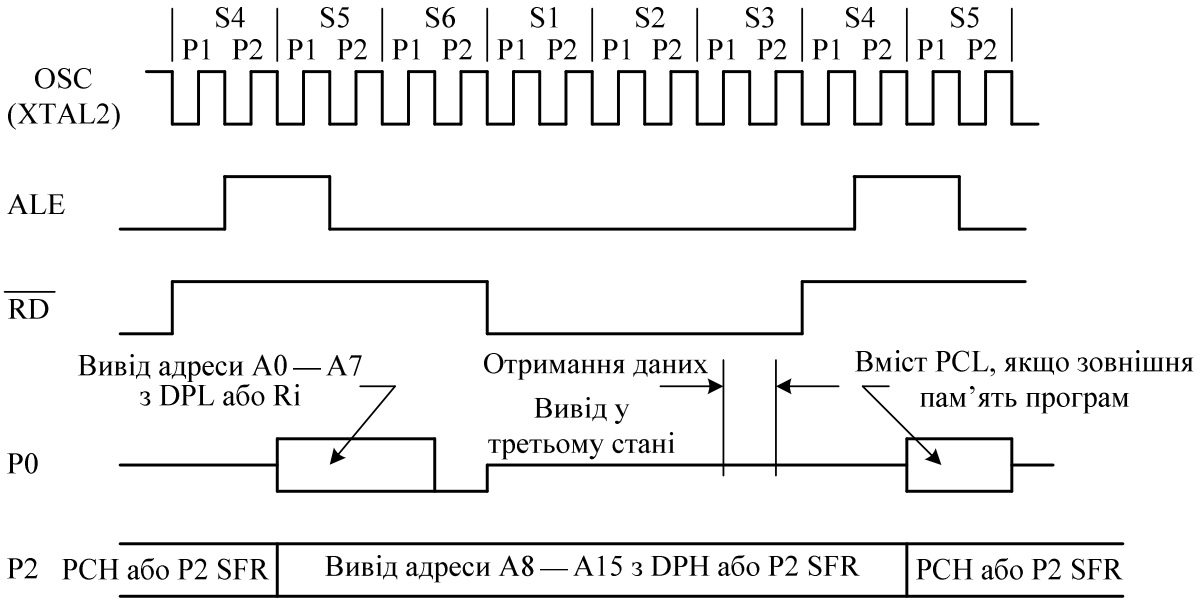


Рисунок 3.14 - Цикл читання зовнішньої пам'яті даних

Доступ до зовнішньої пам'яті програм можливий з використанням команд MOVC A, @A+DPTR, MOVC A, @A+PC при виконанні двох умов: або сигнал $\overline{EA}$ - активний, або вміст лічильника команд перевищує значення 0FFFH. Наявність сигналу $\overline{EA}$ необхідна для забезпечення доступу до молодших 4 кбайт адресного простору зовнішньої пам'яті програм при використанні МК31 (МК без внутрішньої пам'яті програм). На рисунку 3.15 наведено цикл читання зовнішньої пам'яті програм.

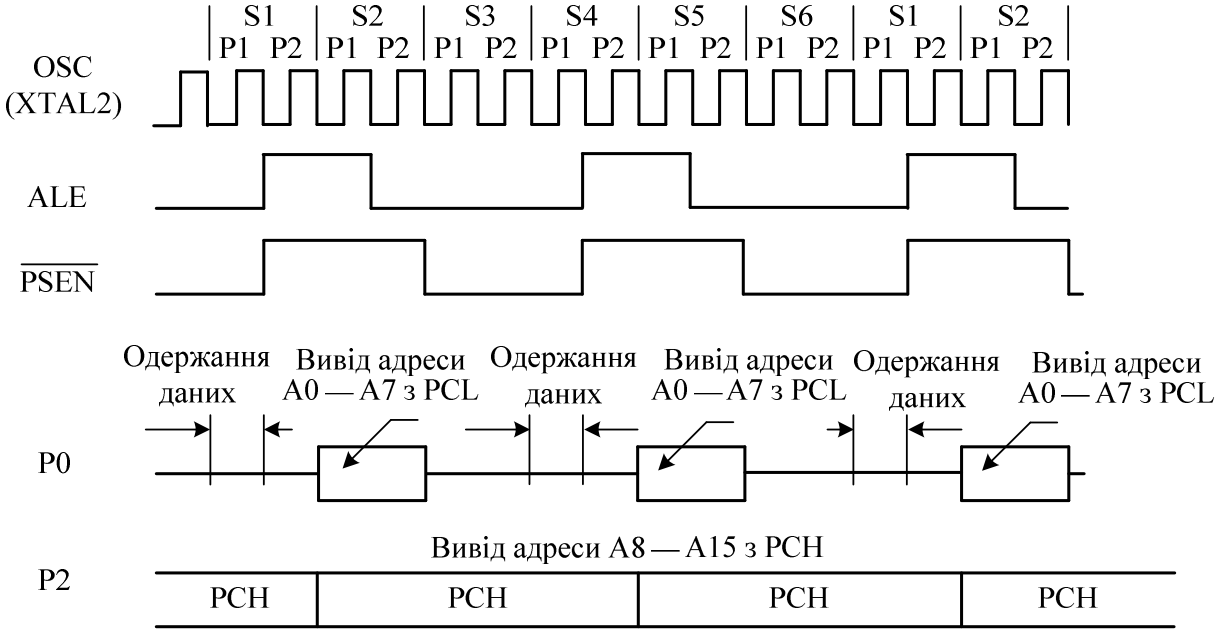


Рисунок 3.15 - Цикл читання зовнішньої пам'яті програм

3.2.8. Таймер/лічильник

Два 16-бітних таймери/лічильники, що програмуються, (Т/С0 і Т/С1) можуть бути використані як таймери або лічильники зовнішніх подій. У режимі таймера вміст Т/С збільшується в кожному машинному циклі, тобто через кожні 12 періодів резонатора, а в режимі лічильника вміст Т/С збільшується під впливом переходу з «1» в «0» зовнішнього сигналу, що подається на відповідний (Т0 або Т1) вивід МК51.

Опитування значення зовнішнього вхідного сигналу виконується в момент часу S5P2 кожного машинного циклу. Вміст лічильника буде збільшений на 1 в тому випадку, якщо в попередньому циклі був зчитаний вхідний сигнал високого рівня - «1», а в наступному – низького - «0». Нове (збільшене) значення лічильника буде сформоване в момент S3P1 в циклі, наступному за тим, в якому був виявлений перехід сигналу з «1» в «0». Оскільки на розпізнання переходу потрібно 2 машинних цикли, то максимальна частота підрахунку вхідних сигналів дорівнює 1/24 частоти резонатора. На тривалість періоду вхідних сигналів обмежень зверху немає. Для гарантованого розпізнання імпульсів вхідного сигналу він повинен утримувати логічний стан протягом, як мінімум, одного машинного циклу МК51.

Для управління режимами роботи Т/С і для організації взаємодії таймерів з системою переривання використовуються 2 регістри спеціальних функцій (TMOD і TCON), опис яких наведено у табл. 3.5 та 3.6.

Режими 0, 1 і 2 нульового та першого таймера/лічильника однакові.

Режим 0. Режим сумісності з MCS-48, 13-бітний таймер/лічильник. У цьому режимі значення мають регістр TH та молодші 5 бітів регістра TL, що створюють регістр таймера з розрядністю 13 бітів. При перевищенні вмістом TL значення 1FH збільшується вміст TH і при переході зі стану “1111111100011111B” до стану “всі нулі” (переповненні таймера/лічильника) встановлюється прапор переривання по переповненню таймера TF.

Режим 1. 16-бітний таймер/лічильник. Регістр таймера складається з регістрової пари TH, TL і має розрядність 16 бітів. При переповненні 16-розрядного регістра - переході зі стану 0FFFFH до стану 0000H - встановлюється прапор переривання по переповненню таймера TF. На рис. 3.16 наведена схема таймера/лічильника в режимі 1.

Режим 2. У режимі 2 робота організована таким чином, що переповнення 8-бітового лічильника TL призводить не тільки до встановлення прапора TF, але і автоматично перевантажує в TL вміст старшого байта (TH) регістра таймера. Вміст TH попередньо задається програмно і не змінюється при перевантаженні. На рис. 3.17 наведена схема таймера/лічильника в режимі 2.

Таблиця 3.5 – TMOD регістр режиму роботи таймера/лічильника

Символ	Позиція		Ім’я та призначення
GATE	TMOD.7	таймер/лічильник 1	Вибір управління. Якщо біт GATE=1, то робота таймера/лічильника Т/С1 дозволена, доки на вході $\overline{INT1}$ високий рівень і біт управління TR1 встановлений. Якщо GATE=0, то робота Т/С1 дозволяється, як тільки встановлюється біт управління TR1
$C/\overline{T}$	TMOD.6		Біт вибору режиму - таймер або лічильник подій. Якщо біт $C/\overline{T}$=0, то працює таймер від внутрішнього джерела сигналів синхронізації. Якщо біт $C/\overline{T}$=1, то працює лічильник зовнішніх сигналів на вході Т1
M1.1	TMOD.5		Режим роботи таймера/лічильника (див. примітку)
M0.1	TMOD.4		
GATE	TMOD.3	таймер/лічильник 0	Вибір управління. Якщо біт встановлений, то робота таймера/лічильника Т/С0 дозволена, доки на вході $\overline{INT0}$ високий рівень і біт управління TR0 встановлений. Якщо біт скинутий, то робота Т/С0 дозволяється, як тільки встановлюється біт управління TR0
$C/\overline{T}$	TMOD.2		Біт вибору режиму - таймер або лічильник подій. Якщо біт скинутий, то працює таймер від внутрішнього джерела сигналів синхронізації. Якщо біт встановлений, то працює лічильник зовнішніх сигналів на вході Т0
M1.0	TMOD.1		Режим роботи таймера/лічильника (див. примітку)
M0.0	TMOD.0		

Примітка. Значення бітів вибору режиму роботи таймера/лічильника

M1	M0	Режим роботи
0	0	13-бітний таймер/лічильник
0	1	16-бітний таймер/лічильник
1	0	8-бітний таймер/лічильник, що автоматично перевантажується. THx зберігає значення, яке перевантажується в TLx після його переповнення
1	1	Таймер/лічильник 1 зупинено. Таймер/лічильник 0: TL0 – 8-бітний таймер/лічильник, що контролюється керуючими бітами таймера 0. TH0 - 8-бітний таймер, його режим визначається керуючими бітами таймера 1

Встановлення біта GATE дозволяє використати таймер для вимірювання тривалості імпульсного сигналу, що подається на вхід запиту переривання.

Таблиця 3.6 – TCON регістр управління/стану таймера

Символ	Позиція	Ім’я та призначення
TF1	TCON.7	Прапор переповнення таймера T/C1. Встановлюється апаратно при переповненні таймера/лічильника. Скидається апаратно при обслуговуванні переривання.
TR1	TCON.6	Біт управління таймера 1. Встановлюється/скидається програмою для пуску/зупинки таймера/лічильника T/C1
TF0	TCON.5	Прапор переповнення таймера T/C0. Встановлюється апаратно при переповненні таймера/лічильника. Скидається апаратно при обслуговуванні переривання
TR0	TCON.4	Біт управління таймера T/C0. Встановлюється/скидається програмою для пуску/зупинки таймера/ лічильника T/C0
IE1	TCON.3	Прапор переривання 1. Встановлюється апаратно при виконанні умов переривання, що встановлені бітом IT1 (зріз/низький рівень). Скидається при обслуговуванні переривання (автоматично/програмно)
IT1	TCON.2	Біт управління типом переривання 1. Встановлюється/скидається програмно для специфікації запиту $\overline{INT1}$ (зріз/низький рівень)
IE0	TCON.1	Прапор переривання 0. Встановлюється апаратно при виконанні умов переривання, що встановлені бітом IT0 (зріз/низький рівень). Скидається при обслуговуванні переривання (автоматично/програмно)
IT0	TCON.0	Біт управління типом переривання 0. Встановлюється/скидається програмно для специфікації запиту $\overline{INT0}$ (зріз/низький рівень)

Примітка: всі біти прапорів регістра TCON можуть встановлюватись та скидатись програмно, що дозволяє програмно ініціювати обслуговування переривань.

Режим 3. У режимі 3 таймери/лічильники працюють по-різному. T/C1 зберігає незмінним свій поточний вміст (зупинений). T/C0 використовує TL0 і TH0 як два незалежні 8-бітні лічильники. На рис. 3.18 наведена схема таймера/лічильника 0 в режимі 3. Таймер/лічильник на основі TL0 використ-

товує біти управління T/C0: C/$\overline{T}$, GATE, TR0, $\overline{INT0}$ і TF0, його переривання по переповненню. Лічильник з TH0 може тільки рахувати машинні цикли – працювати таймером. Він використовує біти управління TR1 і TF1 першого таймера та його переривання по переповненню.

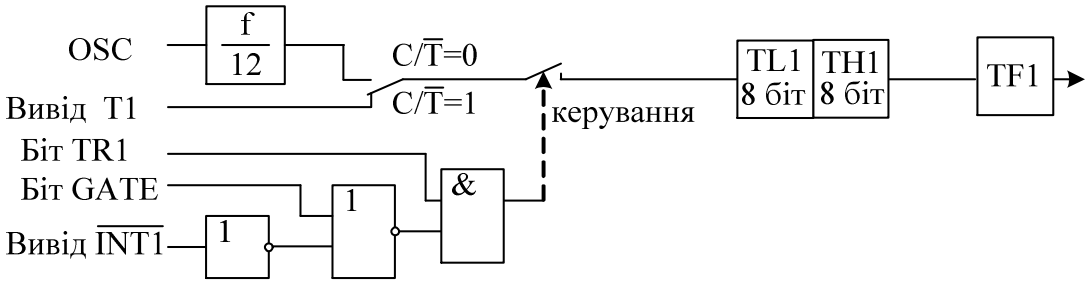


Рисунок 3.16 - Таймер/лічильник 1 в режимі 1

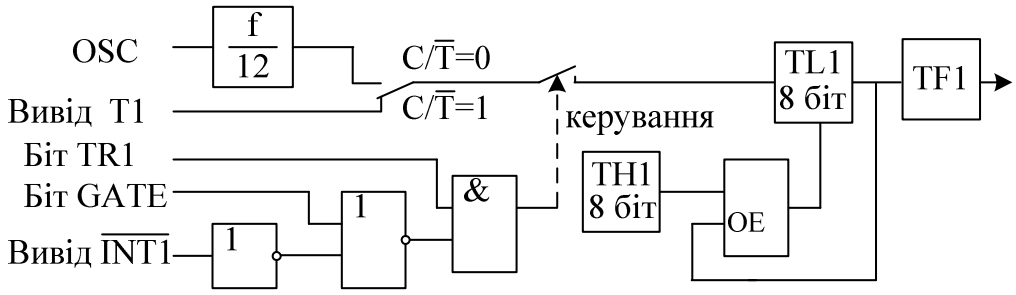


Рисунок 3.17 - Таймер/лічильник 1 в режимі 2

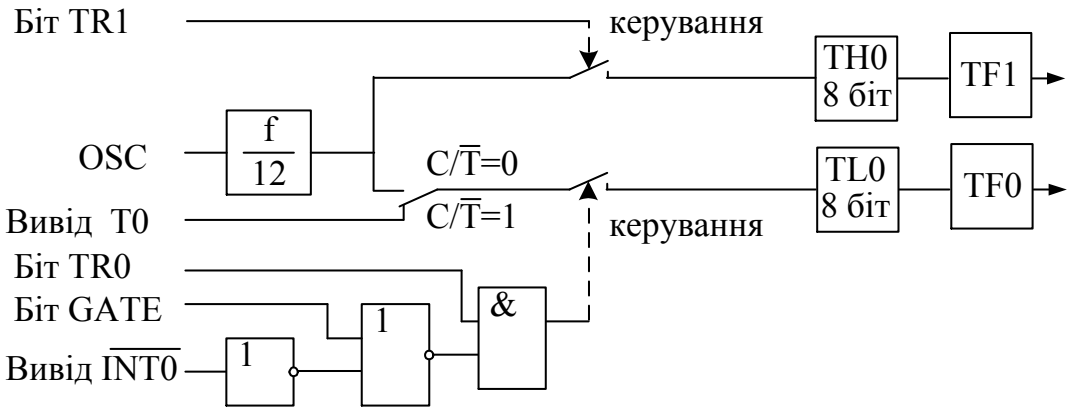


Рисунок 3.18 - Таймер/лічильник 0 в режимі 3: два 8-бітові лічильники

Третій режим таймера використовується при нестачі таймерів чи лічильників. При роботі T/C0 в режимі 3 зупинка таймера T/C1 здійснюється переводом його в режим 3, пуск – в будь-який інший режим. T/C1 може продовжувати використовуватись для завдання швидкості передачі УСАПП та в прикладних програмах, що не потребують переривань.

3.2.9. Послідовний інтерфейс

Через універсальний синхронно-асинхронний приймач-передавач (УСАПП) здійснюється прийом і передача інформації у послідовному коді, в дуплексному режимі обміну. УСАПП може одночасно виконувати прийом та передачу інформації.

До складу УСАПП, що часто називається послідовним інтерфейсом, входять регістри зсуву, які приймають і передають інформацію, а також спеціальний буферний регістр (SBUF) приймача-передавача. Запис байта в буфер приводить до його автоматичного перезапису в регістр зсуву передавача та ініціює початок передачі. Наявність буферного регістра приймача дозволяє сполучати операцію читання раніше прийнятого байта з прийомом чергового байта. Якщо на час закінчення прийому байта попередній байт не був зчитаний з SBUF, то він буде втрачений.

Послідовний інтерфейс МК51 може працювати в чотирьох режимах.

Режим 0. У цьому режимі інформація і передається, і приймається через зовнішній вивід входу приймача (RxD). Приймаються або передаються 8 бітів даних (молодшим бітом вперед). Через зовнішній вивід виходу передавача (TxD) видаються імпульси зсуву, що супроводжують кожен біт. Частота передачі бітів інформації дорівнює 1/12 частоти кварцового резонатора. Часові діаграми роботи послідовного інтерфейсу в режимі 0 наведено на рис. 3.19.

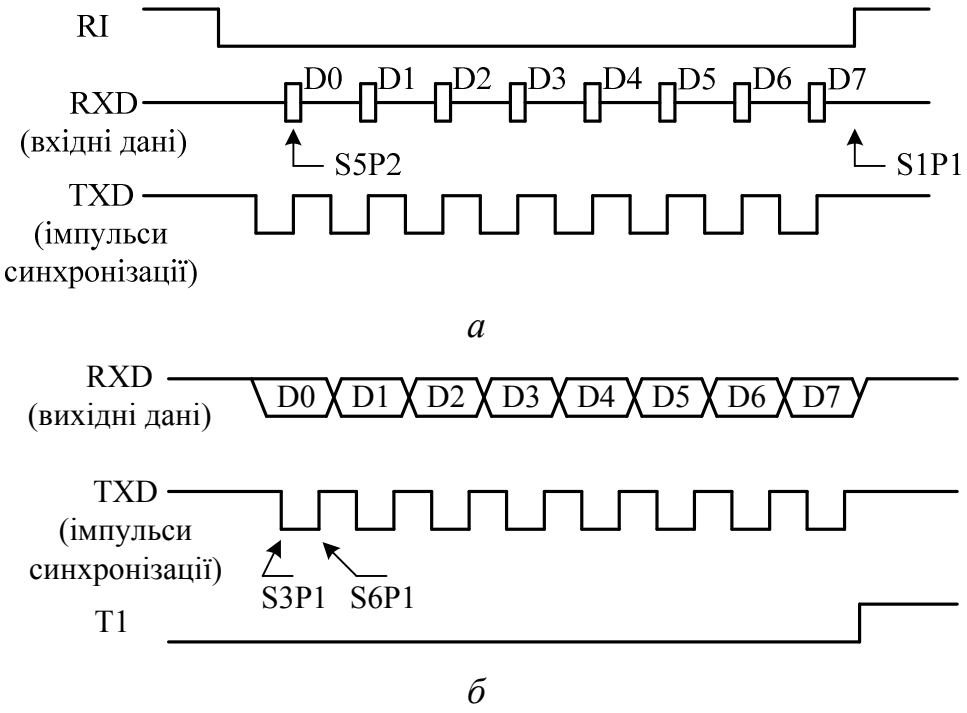


Рисунок 3.19 - Часові діаграми роботи послідовного інтерфейсу в режимі 0:
а - прийом байту; *б* - передача байту

Прийом даних ініціюється встановленням біта дозволу прийому REN = 1 та скиданням прапора приймача RI = 0 в регістрі SCON. Опитування вхідних даних відбувається в момент S5P2 машинного циклу по фронту імпульсу синхронізації на виводі TxD. У момент S1P1 десятого машинного циклу після запису в регістр SCON даних, що скинули біт RI, він апаратно встановлюється.

Передача даних ініціюється записом біта в регістр SBUF. Зсув біта даних, що передається, відбувається в момент S6P2 машинного циклу, тому він дійсний як за зрізом (момент S3P1), так і за фронтом (момент S6P1) імпульсу синхронізації.

Режим 1. У цьому режимі через TxD передаються або з RxD приймаються 10 бітів інформації: старт-біт (0), 8 бітів даних (молодшим бітом вперед) і стоп-біт (1). При прийомі стоп-біт передається до біта RB8 регістра SCON. Швидкість прийому/передачі - величина змінна і задається таймером. Часові діаграми роботи послідовного інтерфейсу в першому режимі наведено на рис. 3.20. Опитування стану виводу RxD проводиться три рази посередині часового інтервалу, що відповідає тривалості біту на обраній швидкості. Рішення про стан виводу приймається за більшістю опитувань.

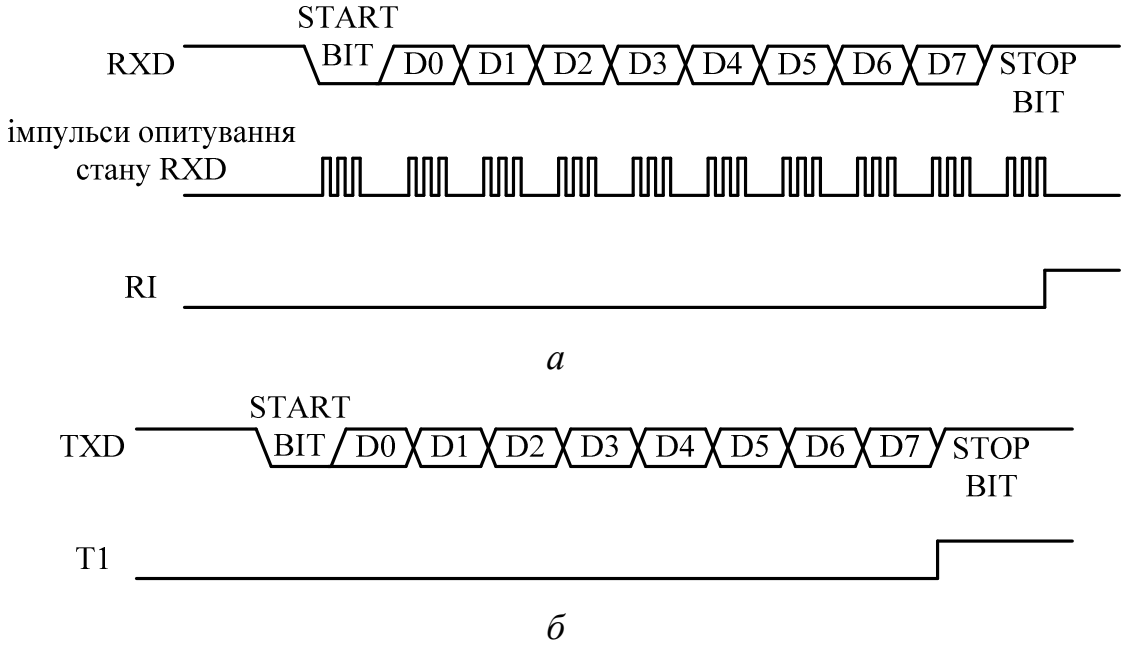


Рисунок 3.20 - Часові діаграми роботи послідовного інтерфейсу в режимі *1*: *а* - прийом байта; *б* - передача байта

Режим 2. У цьому режимі через TxD передаються або з RxD приймаються 11 бітів інформації: старт-біт, 8 бітів даних (молодшим бітом вперед), 9-й біт, що програмується, і стоп-біт. При передачі 9-й біт даних може приймати значення «0» або «1» або, наприклад, для підвищення надійності

передачі шляхом контролю парності в нього може бути поміщене значення ознаки парності з регістра стану програми (PSW.0). Частота прийому/передачі вибирається програмою і може дорівнювати або 1/32, або 1/64 частоти кварцового резонатора залежно від керуючого біта SMOD регістра PCON. Часові діаграми роботи послідовного інтерфейсу в режимі *2* наведені на рис. 3.21.

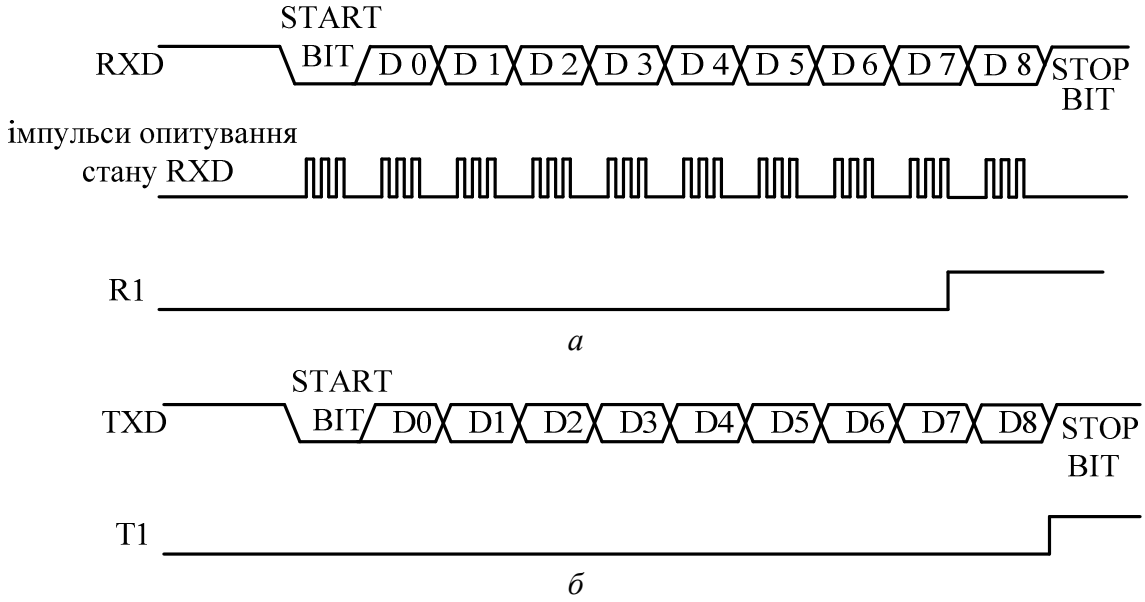


Рисунок 3.21 - Часові діаграми роботи послідовного інтерфейсу в режимі *2*: *а* - прийом байта; *б* - передача байта

Режим 3. Режим *3* збігається з режимом *2* в усіх деталях, за винятком частоти прийому/передачі, що є величиною змінною і задається таймером.

Передача даних ініціюється записом байта в регістр SBUF. Прийом даних в режимі 0 починається при виконанні умов RI = 0 та REN = 1. В інших режимах прийом даних починається при REN = 1.

Регістр управління/стану УСАПП. Управління режимом роботи УСАПП здійснюється через спеціальний регістр спеціальних функцій SCON. Цей регістр містить не тільки керуючі біти, які визначають режим роботи послідовного інтерфейсу, але і дев’ятий біт даних, що приймаються або передаються, і прапори переривання приймача-передавача. Функціональне призначення бітів SCON наведено в табл. 3.7.

Швидкість прийому/передачі. В режимі 0 швидкість прийому/передачі фіксована і становить 1/12 частоти кварцового резонатора:

$$F_0 = \frac{1}{12} f_{OSC} .$$

Таблиця 3.7 - SCON - регістр керування/стану УСАПП

<table>
<tr><td>Символ</td><td>Позиція</td><td colspan="5">Ім’я та призначення</td></tr>
<tr><td rowspan="5">SM0</td><td rowspan="5">SCON.7</td><td rowspan="5">Біти управління режимом роботи УСАПП. Встановлюються/скидаються програмно</td><td>SM0</td><td>SM1</td><td>Режим роботи</td><td>Швидкість передачі</td></tr>
<tr><td>0</td><td>0</td><td>0 - регістр зсуву</td><td>$f_{osc}/12$</td></tr>
<tr><td rowspan="3">SM1</td><td rowspan="3">SCON.6</td><td>0</td><td>1</td><td>1 – 8-бітний УАПП</td><td>змінна</td></tr>
<tr><td>1</td><td>0</td><td>2 – 9-бітний УАПП</td><td>$f_{osc}/64$ або $f_{osc}/32$</td></tr>
<tr><td>1</td><td>1</td><td>3 – 9-бітний УАПП</td><td>змінна</td></tr>
<tr><td>SM2</td><td>SCON.5</td><td colspan="5">У режимі 0 біт SM2 повинен бути скинутим.
У режимі 1, якщо SM2=1, то прапор RI не встановлюється, якщо не прийнято припустимий стоп біт.
У режимі 2 та 3, якщо SM2=1, то прапор RI не встановлюється, якщо 9-й біт прийнятого повідомлення дорівнює 0.</td></tr>
<tr><td>REN</td><td>SCON.4</td><td colspan="5">Біт дозволу прийому. Встановлюється/скидається програмно для дозволу/заборони прийому послідовних даних</td></tr>
<tr><td>TB8</td><td>SCON.3</td><td colspan="5">9-й біт даних, що передається в режимах 2 і 3. Встановлюється/скидається програмно при необхідності</td></tr>
<tr><td>RB8</td><td>SCON.2</td><td colspan="5">В режимі 2 та 3 - 9-й біт даних, що приймаються. В режимі 1, якщо біт SM2=0, до RB8 записується стоповий біт, що був прийнятий. В режимі 1 біт RB8 не використовується</td></tr>
<tr><td>TI</td><td>SCON.1</td><td colspan="5">Прапор переривання передавача. Встановлюється апаратно після закінчення передачі байта. Скидається програмно після обслуговування переривання</td></tr>
<tr><td>RI</td><td>SCON.0</td><td colspan="5">Прапор переривання приймача. Встановлюється апаратно при прийомі байта. Скидається програмно після обслуговування переривання</td></tr>
</table>

У режимі 2 швидкість прийому/передачі залежить від значення біта SMOD регістра PCON. Якщо SMOD=0 (значення після скидання), то швидкість становить 1/64 частоти кварцового резонатора, при встановленні SMOD=1, швидкість підвищується до 1/32 f_{osc}:

$$F_2 = \frac{2^{SMOD}}{64} f_{OSC}.$$

У режимах 1 і 3 швидкість прийому/передачі визначається часом переповнення таймера 1. В цьому випадку переривання Т/С1 повинно бути заборонено, а таймер може працювати в режимі таймера чи лічильника в одному з трьох режимів. Режим лічильника дозволяє використовувати зовнішнє джерело тактового сигналу. Найбільш доцільно використовувати режим таймера в режимі 2 – з автоматичним перевантаженням.

$$F_{1,3} = \frac{2^{SMOD}}{32} \cdot \frac{f_{OSC}}{12 \cdot (256 - (TH1))}.$$

Для отримання низьких швидкостей прийому/передачі можна перевести Т/С1 в режим 16-бітного таймера, дозволити переривання при його переповненні та здійснювати перевантаження регістрів таймера під час обробки переривання. В цьому випадку

$$F_{1,3} = \frac{2^{SMOD}}{32} \cdot \frac{f_{OSC}}{12 \cdot (65536 - (TH1, TL1))}.$$

3.2.10. Система переривань

Мікроконтролер МК51 підтримує 5 джерел переривань з двома рівнями пріоритету. Система переривань з умовним позначенням бітів керування та порядку опитування показана на рис. 3.22.

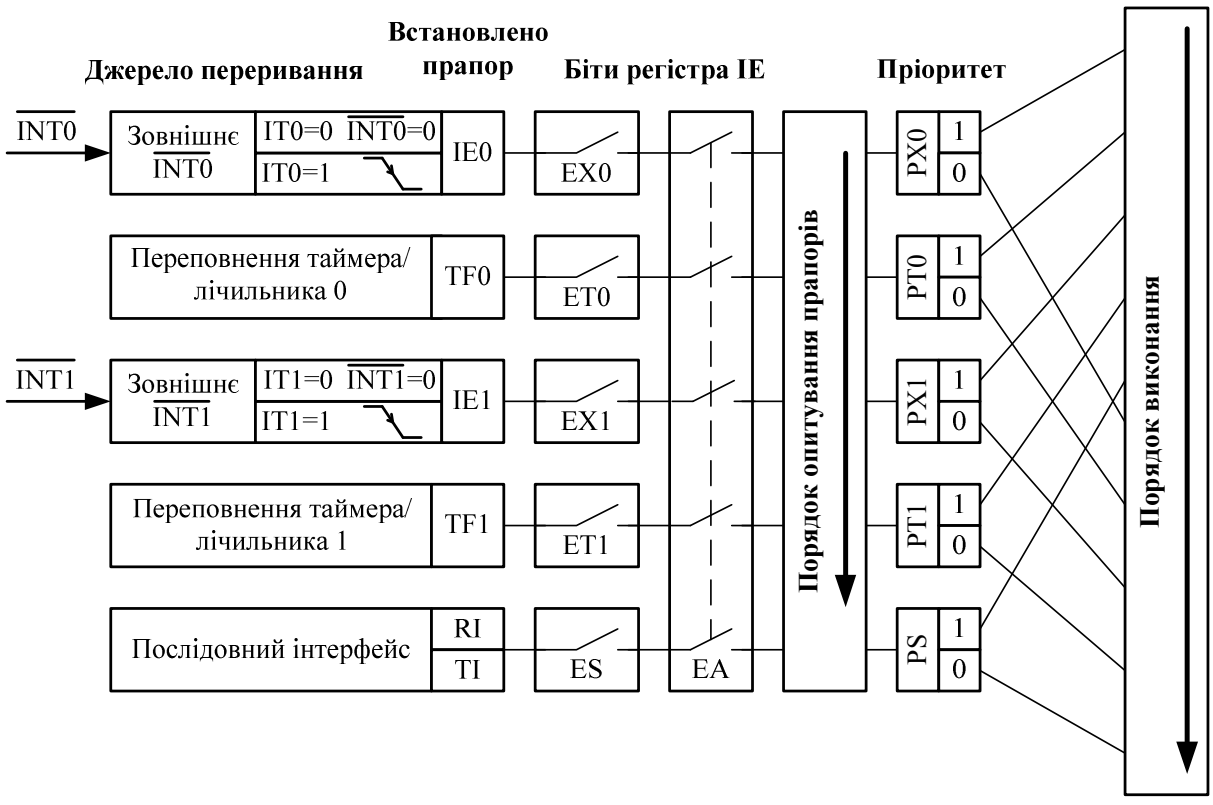


Рисунок 3.22 - Система переривань МК51

Зовнішні переривання **INT0** і **INT1** залежно від значень керуючих бітів IT0 і IT1 в регістрі TCON можуть бути викликані або низьким рівнем, або переходом сигналу з «1» в «0» на входах $\overline{INT0}$ та $\overline{INT1}$ МК51. При виконанні умов зовнішніх переривань встановлюються прапори IE0 і IE1 в регістрі TCON, що ініціює виклик відповідної підпрограми обслуговування переривання, якщо воно дозволене. Скидання цих прапорів виконується апаратно тільки в тому випадку, якщо переривання дозволене і було викликане по переходу (зрізу) сигналу. Якщо ж переривання викликане рівнем вхідного сигналу, то відповідна підпрограма обслуговування переривання повинна спочатку вплинути на джерело переривання з метою зняття ним запиту, а потім – скинути прапор IE. Опитування виводів зовнішніх переривань здійснюється один раз за машинний цикл. Для надійного визначення стану сигналів, що привели до переривання, високий або низький рівні сигналів повинні утримуватись незмінними не менше одного машинного циклу.

Прапори запиту переривання від таймерів/лічильників TF0, TF1 встановлюються при переповненні регістрів таймерів і скидаються автоматично при передачі управління підпрограмі обслуговування.

Прапори запитів переривання послідовного інтерфейсу RI, TI встановлюються блоком управління УСАПП апаратно, але скидатися повинні програмою користувача.

Прапори запитів переривання встановлюються при виконанні умов виникнення переривань незалежно від їх дозволу.

Переривання можуть бути викликані або скасовані програмою користувача, бо всі перераховані прапори програмно-доступні і можуть бути встановлені/скинуті програмою з тим же результатом, як би вони були встановлені/скинуті апаратними засобами.

Серед регістрів спеціальних функцій є два регістри, що призначені для управління дозволом переривань і рівнями їх пріоритету. Призначення бітів цих регістрів, що мають символічні імена IE (Interrupt Enable – дозвіл переривань), IP (Interrupt Priority – пріоритет переривань), описані в табл. 3.8 і 3.9 відповідно.

Кожне з джерел переривань може бути індивідуально дозволено чи заборонено встановленням/скиданням відповідного біта в РСФ IE. Але для зручності програмування можна заборонити виконання всіх переривань, не змінюючи їх власних параметрів.

Для кожного з джерел переривань може бути індивідуально встановлено один з двох рівнів пріоритету. Встановлення пріоритету здійснюється встановленням/скиданням відповідного біта в РСФ IP. Переривання з низьким пріоритетом (біт пріоритету скинуто) може само бути перервано перери-

ванням з високим пріоритетом (біт пріоритету встановлено). Виконання переривання з високим пріоритетом не може бути перервано ніяким чином.

Таблиця 3.8 - Регістр дозволу переривань IE

Символ	Позиція	Ім’я та призначення
EA	IE.7	Заборона переривань. При EA=0 всі переривання заборонені, незалежно від стану IE.4 – IE.0. При EA=1 кожне джерело переривань дозволяється/забороняється індивідуально відповідними бітами управління IE.4 – IE.0
–	IE.6	Не використовуються
–	IE.5	
ES	IE.4	Біт дозволу переривання від УСАПП. Встановлюється/скидається програмно для дозволу/заборони переривань при встановленні прапорів TI або RI
ET1	IE.3	Біт дозволу переривання від таймера 1. Встановлюється/скидається програмно для дозволу/заборони переривань від таймера 1
EX1	IE.2	Біт дозволу зовнішнього переривання INT1. Встановлюється/скидається програмно для дозволу/заборони зовнішнього переривання INT1
ET0	IE.1	Біт дозволу переривання від таймера 0. Працює аналогічно IE.3
EX0	IE.0	Біт дозволу зовнішнього переривання INT0. Працює аналогічно IE.2

Таблиця 3.9 - Регістр пріоритетів переривань IP

Символ	Позиція	Ім`я та призначення
–	IP.7	Не використовуються
–	IP.6	
–	IP.5	
PS	IP.4	Біт пріоритету УСАПП
PT1	IP.3	Біт пріоритету таймера 1
PX1	IP.2	Біт пріоритету зовнішнього переривання 1
PT0	IP.1	Біт пріоритету таймера 0
PX0	IP.0	Біт пріоритету зовнішнього переривання 0

Опитування прапорів переривань відбувається в момент S5P2 кожного машинного циклу. Якщо одночасно надійшли два переривання з різним рівнем пріоритету, то першим виконується переривання з вищим пріоритетом. При одночасному надходженні переривань з рівним пріоритетом вони виконуються за порядком опитування – IE0, IT0, IE1, IT1, TI або RI. Система переривань сформує апаратно виклик (LCALL) відповідної підпрограми обслуговування, якщо вона не заблокована однією з таких умов:

1) на цей момент обслуговується запит переривання рівного або більш високого рівня пріоритету;

2) поточний машинний цикл – не останній в циклі команди, що виконується;

3) виконується команда RETI або будь-яка команда, пов’язана зі звертанням до регістрів IE або IP.

Після завершення виконання умови 3 перед викликом підпрограми обслуговування переривання має виконатись одна команда, що йде за RETI або за командою доступу до регістрів IE або IP.

Якщо прапор запиту переривання був встановлений, але за однією з вище перерахованих умов не отримав обслуговування і на час закінчення блокування вже був скинутий, то запит переривання втрачається і ніде не запам’ятовується.

Обробка переривання починається з виконання коду LCALL, що апаратно сформувався. Система переривань поміщає в стек тільки вміст лічильника команд (PC) і завантажує в лічильник команд адресу вектора відповідного переривання. Вектор переривань наведений в таблиці 3.10.

Таблиця 3.10 - Вектор переривань

Джерело переривання	Адреса вектору
IE0	03H
IT0	0BH
IE1	13H
IT1	1BH
TI або RI	23H

За адресою вектора повинна бути розміщена команда безумовної передачі управління до початкової адреси підпрограми обслуговування переривання. Підпрограма обслуговування у випадку необхідності повинна починатися командами збереження в стеку або пам’яті даних слова стану програми, акумулятора, регістрів, що використовуються та ін. і закінчуватися командами відновлення цих даних. Виконання підпрограми обслуговування пере-

ривання продовжується до команди RETI, що інформує процесор про завершення обробки переривання, перевантажує в лічильник команд зі стека два байти збереженої адреса повернення до основної програми. Команда RET також повертає управління основній програмі, але система обробки переривань вважає при цьому, що переривання ще в стадії обробки, це призводить до блокування виконання переривань цього і нижчого рівня пріоритету.

3.2.11. Скидання, режим холостого ходу і режим зниженого енергоспоживання

Скидання. Скидання МК51 здійснюється шляхом подачі на вхід RST сигналу логічної одиниці. Для впевненого скидання МК51 цей сигнал повинен бути утриманий на вході RST щонайменше протягом двох машинних циклів (24 періоди резонатора). Сигнал скидання асинхронний до внутрішнього генератора. Опитування стану входу RST здійснюється в момент S5P2 машинного циклу, як зображено на рис. 3.23.

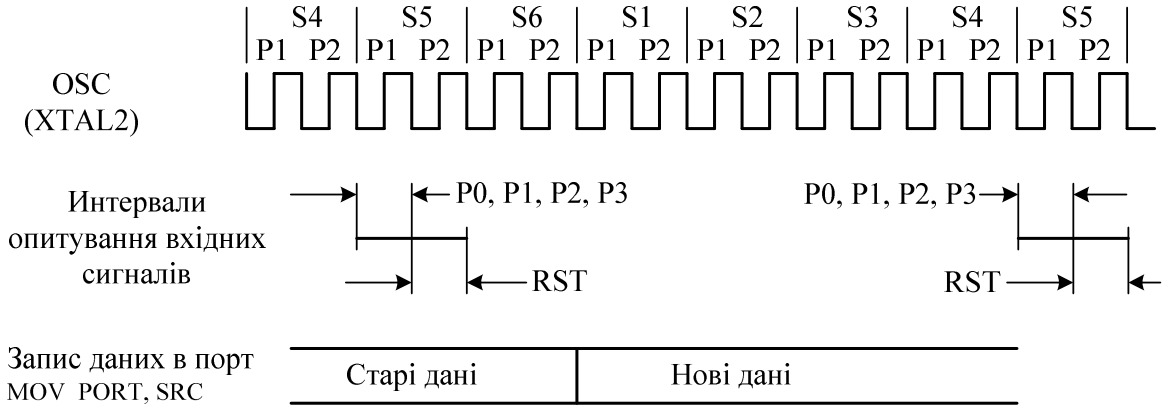


Рисунок 3.23 - Часова діаграма опитування стану виводів портів та виводу скидання RST, виведення даних в порти

Під впливом сигналу RST скидається вміст регістрів: PC, ACC, B, PSW, DPTR, TMOD, TCON, T/C0, T/C1, IE, IP і SCON, в регістрі PCON скидається тільки старший біт, в регістр-покажчик стека завантажується код 07H, а в порти P0-P3 – коди 0FFH. Стан регістра SBUF невизначений. Сигнал RST не впливає на вміст комірок внутрішньої пам’яті даних.

На рисунку 3.24 показана схема автоматичного формування сигналу RST при вмиканні електроживлення. Час, необхідний для повного заряду ємності, забезпечує упевнений запуск резонатора і його роботу протягом більш ніж двох машинних циклів. Рекомендоване значення ємності конденсатора C1 10 мкФ. Опір резистора R1 складає 10 кОм. В МК, виготовлених за CHMOS технологією (вітчизняні МК К1830ВЕ51 та ін.), резистор R1 вбудо-

вано до схеми МК, і в схемі формування сигналу RST елемент R1 відсутній, ємність конденсатора С1 може бути зменшена до 1 мкФ. Час запуску резонатора залежить від його робочої частоти і для кварцових резонаторів з частотою 1 МГц становить 10 мс. У цей час стан виводів портів невизначений.

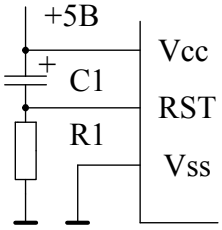


Рисунок 3.24 - Схема формування сигналу скидання

Режим зниженого енергоспоживання. У тих застосуваннях МКС, де споживання електроенергії, а отже, габаритні розміри і маса джерела електроживлення є одними з основних показників якості виробу, можливе використання МК51 в режимі зниженого енергоспоживання. Цей режим присутній в мікроконтролерах, що виготовлені за CHMOS технологією, старші – HMOS контролери - його не мають. Керування споживаною потужністю мікроконтролера відбувається через регістр керування потужністю PCON (Power CONtrol register), значення бітів якого наведене в табл. 3.11.

Таблиця 3.11 - Регістр керування потужністю PCON

Символ	Позиція	Ім’я та призначення
SMOD	PCON.7	Подвійна швидкість передачі. Якщо біт SMOD встановлений в «1», то швидкість передачі вдвічі більша, ніж при SMOD = 0
-	PCON.6	Не використовуються
-	PCON.5	
-	PCON.4	
GF1 GF0	PCON.3 PCON.2	Прапори, що специфікуються користувачем (прапори загального призначення)
PD	PCON.1	Біт зниженої потужності. При встановленні біта в «1» МК переходить в режим зниженої споживаної потужності
IDL	PCON.0	Біт холостого ходу. Якщо біт встановлений в «1», то МК переходить в режим холостого ходу

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

Режим холостого ходу. Будь-яка команда, за якою встановиться керуючий біт IDL в регістрі PCON, переведе МК51 в режим холостого ходу. При цьому продовжує роботу внутрішній генератор, але його сигнал не надходить до АЛП, зберігають працездатність всі джерела переривань, всі регістри зберігають своє значення, на виводах всіх портів утримується логічний стан, який на них був у момент переходу в режим холостого ходу, на виводах ALE і $\overline{PSEN}$ формується рівень «1».

Вийти з режиму холостого ходу можна або за сигналом скидання, або за перериванням. Будь-який з дозволених сигналів переривання приведе до апаратного скидання біта PCON.0 і припинить таким чином режим холостого ходу. Після виконання команди RETI (вихід з підпрограми обслуговування переривання) буде виконана команда, яка йде в програмі за командою, що перевела МК51 в режим холостого ходу. Сигнал скидання безпосередньо скидає біт IDL. Виконання програми починається з наступної команди за тією, що встановила режим холостого ходу. Внутрішній алгоритм скидання займає 2-3 машинні цикли. На цей час блокується доступ до внутрішньої пам’яті даних, але дозволяється доступ до портів. Для зменшення вірогідності неочікуваного виводу до порту команда, що виконується першою після повернення з режиму холостого ходу, не повинна звертатись до пам’яті даних або записувати інформацію до портів.

Режим зниженої потужності. Переведення МК51 в цей режим можливо при встановленні біта PD в регістрі PCON. При цьому зупиняється внутрішній генератор, вміст пам’яті даних і регістрів спеціальних функцій зберігається, а на вихідних контактах портів утримуються значення, відповідні вмісту їх регістрів-фіксаторів. Виходи сигналів ALE і $\overline{PSEN}$ скидаються.

Після встановлення режиму зниженої потужності напруга електроживлення VCC може бути знижена до 2 В. Перед виходом з режиму вона повинна бути відновлена до номінального значення.

Вихід з режиму зниженої потужності можливий тільки за сигналом скидання. При цьому всі регістри спеціальних функцій отримують початковий вміст, але вміст пам’яті даних не змінюється.

4. ЗАСОБИ ПРОГРАМУВАННЯ МК СІМ'Ї MCS-51

4.1. Засоби програмування МК. Асемблер

Всі обчислювальні пристрої мають подібну структурну організацію, у яку входять: процесор, пристрої зберігання програм, даних, вводу-виводу інформації. Процесор працює під управлінням програми, що написана машинною мовою й складається з інструкцій, що безпосередньо сприймаються процесором. Машинна мова відбиває всі тонкощі архітектури ЕОМ і індивідуальна для кожної сім'ї обчислювальних пристроїв. Складність і незручність використання числових послідовностей, що являють собою код операції й необхідні операнди, привело до появи мови асемблера (assembly language), яка пропонує мнемонічний (символьний) запис машинних команд, і програми Асемблера, що перекладає текст із мови асемблера в інструкції процесора. Подання інформації в машинних командах (шістнадцяткове подання) і мовою асемблера МК MCS-51 істотно відрізняються. Наприклад, запис у машинних командах 8980 відповідає запису мовою асемблера MOV P0, R1 – помістити (**MOV**e) у порт P0 вміст регістра R1; запис 9A відповідає SUBB A, R2 – відняти (**SUB**tract) з акумулятора A вміст регістра R2. Команди мови асемблера являють собою легко пізнавані скорочення відповідних англійських термінів. Мнемонічний запис команд мікроконтролера MCS-51 захищений авторським правом корпорації INTEL. Мова асемблера є найбільш старою мовою програмування, що використовується дотепер.

Для програмування МК сім'ї 8051 можна використовувати мови асемблер, Бейсік, Паскаль, Сі. Перетворення тексту програми, написаної на одній з цих мов, в машинні коди здійснюється по-різному. Для асемблера проводиться безпосередня підстановка відповідних кодів операції і необхідних

даних. Мови високого рівня використовують перетворення операторів мови в машинні коди підстановкою заготівок, які містять стандартні для даного оператора рішення. Використання стандартних рішень призводить до надмірної функціональності машинного коду, збільшення його об'єму і зниження швидкодії. Після етапу компіляції може здійснюватись оптимізація коду, при якій видаляються формально невживані ділянки, змінюється порядок проходження команд. Однією з особливостей мов високого рівня є активне використання ОЗП МК. Невелика ємність ОЗП, типово – 128 байт, ускладнює отримання ефективного коду. Незважаючи на оптимізацію, програми, розроблені на мовах високого рівня, не дозволяють одержати таку ж швидкодію і об'єм, як при програмуванні на асемблері. З іншого боку, мови високого рівня полегшують програмування задач з складними обчисленнями і розвиненим інтерфейсом користувача, які не критичні за часом виконання.

Достоїнствами програм, написаних мовою асемблера, є: максимальна швидкість виконання обчислень, мінімальний об'єм, займаний програмою, найбільш повне використання можливостей процесора. Компілятори мов високого рівня мають засоби інтегрування своїх модулів і модулів, написаних мовою асемблера, або підтримують асемблерні вставки в текст програми. Файл, що виконується, або бінарний файл програми (програма в машинних командах) може бути переведений на мову асемблера за допомогою програми дізасемблера.

До недоліків програмування мовою асемблера варто віднести підвищену складність і трудомісткість програмування, що стає особливо помітно при розробці великих проектів. Для зниження трудомісткості рекомендується використання бібліотек стандартних функцій. Програми на асемблері не мають властивості перенесення, що пояснюється повним використанням можливостей конкретного процесора. Для програм, написаних мовами високого рівня, властивість перенесення супроводжується зниженням швидкодії й збільшенням розміру, однак для мікроконтролерів різних сімей із сильними розходженнями в архітектурі перенесення програм найчастіше неможливе.

Програма перекладається з мови асемблера в машинні коди МК за допомогою програм, які називаються транслятором, бібліотекарем і редактором зв'язків. На сьогодні всі вони об'єднуються в одну програму – Асемблер. Необхідно розуміти різницю між мовою асемблера – системою позначень і мнемонічними кодами операцій і програмою Асемблер або транслятор – програмою для перетворення початкової програми на мові асемблера в машинний код.

4.2. Способи адресації операндів МК MCS-51

Система команд мікроконтролерів сім'ї MCS-51 проектувалася з прицілом на додання їй максимально ортогонального характеру при обмеженій кількості команд, що має на увазі можливість оперувати змінними з використанням різних способів їхньої адресації. Повний список команд асемблера наведений у Додатку В. Кожна команда містить мнемонічне позначення операції та до трьох операндів, розділених комами. При наявності двох операндів місце призначення результату наводиться першим, а джерело змінної -- другим. Багато команд, що оперують зі змінною - байтом, використовують акумулятор як джерело змінної або приймач результату операції. У цьому випадку для ясності акумулятор позначається буквою «А», другий операнд може адресуватися декількома способами. Перед мнемонічним позначенням операції може знаходитися символьна мітка, а після знака «;» - коментар.

При визначенні способу адресації операндів у команді [15] необхідно враховувати, що адресація для кожного операнда команди своя. У загальному випадку вони можуть не збігатися.

Регістрова адресація використовується для звертання до восьми робочих регістрів R0 – R7 обраного банку робочих регістрів. У мові асемблера регістрова адресація позначається символом Rn (де n набуває значення від 0 до 7). Номер регістра записується в молодші три біти коду операції.

Наприклад, для додавання вмісту регістрів R6 і R7 може використовуватися наступний фрагмент програми з регістровою адресацією:

```
MOV A, R6       ;
ADD A, R7       ; Результат додавання перебуває в акумуляторі.
```

Залежно від обраного банку регістрів дана операція може провадитися із вмістом комірок пам'яті з адресами 06h і 07h для банку 0, 0Eh і 0Fh для банку 1, 16h і 17h для банку 2 і 1Eh і 1Fh для банку 3.

Пряма адресація використовується для звертання до комірок внутрішньої пам'яті даних з адресами від 0 до 127, до портів вводу-виводу й регістрів спеціальних функцій, яким надані адреси від 128 до 256. Адреса комірки пам'яті поміщається в другий байт команди. Наприклад:

```
MOV A, 20h      ;запис в акумулятор вмісту комірки з адресою 20h,
                ;машинний код операції E5 20.
```

Використання при програмуванні абсолютної адресації призводить до труднощів написання, читання й модифікації програм. Мова асемблера дозволяє надавати адресам символічні імена й надалі використовувати їх при програмуванні. Наприклад, для запису в порт 1 вмісту акумулятора можна скористатися командою

MOV 90h, A

або, попередньо надавши імені P1 значення 90h,

MOV P1, A ;цей варіант більш наочний.

Команди роботи зі стеком PUSH і POP допускають тільки пряму адресацію операнда. Неприпустиме використання за операндів вбудованих символьних імен A або R4. Однак при підстановці в команду PUSH відповідних адрес: акумулятора - 0E0h або R4 - 05h їхній вміст буде поміщено в стек.

Для операцій з окремими бітами, що допускають індивідуальну адресацію (128 біт з адресами від 0 до 7Fh, розташованих в комірках внутрішньої пам'яті даних з адресами 20H-2FH і біти регістрів спеціальних функцій з адресами від 80h до 0FFh), також використовується пряма адресація. Наприклад:

SETB 20h ;установити біт

CLR 81h ;скинути біт з адресою 81h - перший біт порту P0

Непряма регістрова адресація використовується для звертання до комірок внутрішньої пам'яті даних. Як регістри-покажчики адреси використовуються регістри R0, R1 обраного банку регістрів. Їхній вміст визначає адресу комірки пам'яті. У мові асемблера непряма регістрова адресація показується символом @ (комерційне "at"), попереднім до R0, R1. Наприклад:

```
MOV A, @R0    ;записати в акумулятор байт,
              ;адреса якого перебуває в R0
MOV @R1, A    ;записати за адресою, що перебуває в R1 байт, з
              ;акумулятора
```

При звертанні до зовнішньої пам'яті даних як регістри-покажчики на 8-розрядну адресу використовуються регістри R0 і R1 (поточного банку регістрів). Наприклад:

MOVX A, @R1

Якщо як регістр-покажчик використовується 16-розрядний покажчик даних (DPTR), то можна вибрати будь-яку комірку зовнішньої пам'яті даних об'ємом до 64 Кбайт.

MOVX A, @DPTR

При читанні інформації з пам'яті програм використовується непряма регістрова адресація по сумі вмісту двох регістрів.

Виконавча адреса утвориться підсумовуванням вмісту акумулятора й регістра покажчика даних або програмного лічильника (PC). Наприклад:

```
MOVC A, @A+PC     ;дозволяє прочитати байт щодо поточної
                  ;позиції в програмі
MOVC A, @A+DPTR   ; дозволяє прочитати будь-який байт
```

Використання непрямої регістрової адресації дозволяє використовувати адресу, що обчислюється, при роботі з таблицями, рядками символів і інших даних, що перебувають у зовнішній пам'яті даних або пам'яті програм.

При виконанні команд PUSH <адреса> і POP <адреса> дані, що перебувають за указаною адресою, записуються / читаються за адресою, що міститься в регістрі покажчика стека SP. Формально операції зі стеком, еквівалентні операціям передачі даних з непрямою адресацією:

PUSH <адреса> - "INC SP"
"MOV @SP, <адреса>"
POP <адреса> - "MOV <адреса>, @SP"
"DEC SP"

Приклади еквівалентних операцій наведені тільки для демонстрації дії команд PUSH, POP і неприпустимі в мові асемблера MCS-51.

Безпосередня адресація дозволяє оперувати з константами, явно зазначеними в команді. У мові асемблера константа вказується символом # попередньому її значенню. Константа може бути подана числовим значенням, символьною змінною або арифметичним виразом, що використовує константи. Програма асемблера обчислює арифметичні вираження під час асемблювання. Наприклад:

```
MOV A, #(1+4)        ;Завантажити в акумулятор число 5=1+4
MOV DPTR, #1245h     ;Завантажити в регістр покажчик даних
                     ;шістнадцятирічне число 1245h
```

Неявна адресація При неявній адресації регістр-джерело або регістр-приймач мається на увазі в самому коді операції. Наприклад:

```
RR A                 ;Зрушити вміст акумулятора вправо
```

4.3 Система команд

Машинні команди можуть займати від одного до трьох байтів. Розмір команди визначається кодом операції, записаним у першому байті. Додаткові байти можуть містити адреси і/або дані, частина інформації, необхідної для адресації, може міститися в коді операції. За допомогою байта коду операції можна закодувати 256 команд, але для сім'ї MCS-51 використовується тільки 255. Код 0A5h зарезервований для подальшого розвитку. Команди виконуються за один, два або чотири (множення та ділення) машинних циклів [15].

Список команд поділено на 5 функціональних груп: команди пересилання даних, арифметичні команди, логічні операції, операції з бітами і ко-

манди передачі керування. Існує одна команда, яку не можна віднести ні до однієї з груп, оскільки вона не робить нічого протягом одного такту: NOP. Однак ця команда (No OPeration – немає операції) необхідна для роботи в реальному масштабі часу, щоб забезпечити короткочасну затримку перед виконанням наступної команди.

При описі команд використовуються такі позначення:

dat – число зі значенням dat;

src, dest – адреса операнда – джерела, операнда - призначення;

<src>, <dest> – вміст комірки з відповідною адресою;

@ src – непряма адресація з джерела src (Ri (i=0, 1), DPTR, A + DPTR, A + PC);

bit – адреса біта;

rel – відносна 8-бітна адреса;

adr11 – 11-бітна адреса;

adr16 – 16-бітна адреса.

4.3.1. Команди пересилання даних

Незважаючи на передачу даних у незмінному вигляді, ці команди здійснюють один із способів обробки інформації. Як приклад такої обробки можна навести сортування. У командах пересилання використовуються всі варіанти адресації даних. Пересилання даних може здійснюватися у форматах байта, половини байта, двох байтів і біта.

Для маніпуляції вмістом внутрішньої пам’яті даних використовуються команди, список яких наведений у таблиці 4.1.

Команда **MOV** (MOVe з англійської “пересунути”) копіює вміст джерела в приймач (при виконанні цієї команди первісний вміст приймача губиться):

```
MOV   R1, 02H
```

Для занесення нуля в акумулятор простіше використовувати команду очищення **CLR** (CleaR – очистити):

```
CLR   A
```

Читання і запис даних байтового формату при звертанні до зовнішнього ОЗП здійснюється за допомогою команд **MOVX**, де літера X означає e**X**ternal (зовнішня пам’ять):

```
MOVX   A, @Ri          MOVX   @Ri, A
MOVX   A, @DPTR        MOVX   @DPTR, A
```

Перед виконанням цієї команди у відповідний регістр потрібно записати адресу комірки пам’яті, з якою здійснюється обмін даними.

Таблиця 4.1 - Список команд пересилання даних, що мають доступ до внутрішньої пам’яті даних мікроконтролера

Мнемонічне позначення	Виконувана операція	Спосіб адресації[1]				Час виконання (такти)
		Пр	Непр	Рег	Безп	
MOV A, srs	A = <srs>	+	+	+	+	1
MOV dest, A	<dest> = A	+	+	+		1
MOV dest, srs	<dest> = <srs>	+	+	+	+	2
MOV DPTR, #dat16	в DPTR записується 16-розрядна константа				+	2
PUSH src	SP=SP+1 MOV @SP, <src> [2]	+				2
POP dest	MOV <dest>, @SP [2] SP=SP-1	+				2
XCH A, src	Акумулятор і джерело обмінюються вмістом	+	+	+		1
XCHD A, @Rn (n=0,1)	Акумулятор і байт, що непрямо адресується, обмінюються молодшими тетрадами		+			1

[1] – спосіб адресації операнда: Пр – пряма; Непр – непряма; Рег – регістрова; Безп – безпосередня.
[2] – псевдооперація.

Читання даних із ПЗП здійснюється за допомогою команди **MOVC**, при цьому літера C означає **C**ode (програма):

MOVC A, @A + DPTR

MOVC A, @A + PC

Ці команди дуже зручні для читання з таблиць, записаних у пам’ять програм.

Запис в ОЗП і читання з нього за допомогою стекового способу адресації здійснюється командами:

PUSH src

POP dest

Мнемонічний запис стекових команд відповідають дієсловам “заштовхнути” і “виштовхнути”.

Команда копіювання **XCH** (eXCHange означає “обміняти”) здійснює обмін вмісту джерела і приймача. Такий обмін можна здійснити за допомогою трьох команд пересилання MOV. Команди, наведені нижче, роблять це втричі швидше, займають менше місця в ПЗП і не вимагають використання

додаткової комірки ОЗП. Вони використовують прямий, регістровий та непрямий способи адресації:

XCH A, src
XCH A, Rn
XCH A, @Ri

Для обміну молодших половин байтів використовують команду:

XCHD A, @Ri

Тут D означає Digit (чотири біти – молодша тетрада, використовуються для двійкового подання десяткової цифри).

Одна з команд пересилання даних записує два байти в регістр покажчика даних:

MOV DPTR, #16dat

Інших команд для явного пересилання двобайтних даних немає.

Команда пересилання не впливає на вміст слова стану програми, за винятком випадків пересилання інформації в цей регістр.

4.3.2. Арифметичні команди

Усі команди арифметичних операцій призначені для роботи з додатними цілими числами байтового формату, хоча команди додавання і віднімання у випадку відсутності переповнення забезпечують одержання коректного результату при спеціальному способі кодування від'ємних чисел. При необхідності роботи з числами, що не можуть бути подані в форматі байта, необхідно розробляти відповідні підпрограми. Виконання операцій множення і ділення з числами, що мають довільний знак, можливе, якщо подати їх через знак і модуль та при використанні відповідних підпрограм власної розробки чи запозичені з бібліотек інших розробників.

Команда додавання **ADD** працює з даними у форматі байта, при цьому як приймач завжди використовується тільки акумулятор:

ADD A, #dat
ADD A, Rn
ADD A, @Ri
ADD A, src

Мнемоніка цієї команди відповідає слову ADDition (додавання).

Для роботи з числами, що не можуть бути подані у вигляді одного байта, використовується команда додавання, яка враховує біт переносу, отриманий при додаванні попередньої пари байтів – **ADDC**:

ADDC A, #dat
ADDC A, Rn
ADDC A, @Ri

ADDC A, src

Літера C у позначенні команди вказує на використання біта переносу (ADDition with Carrier).

Існує також команда збільшення **INC**, за допомогою якої здійснюється збільшення операнда на одиницю (INCrement):

INC A
INC Rn
INC @Ri
INC src

Така ж команда працює з 16-бітним регістром покажчика даних:

INC DPTR

За допомогою цієї команди можна змінювати вміст покажчика для читання послідовності байтів із ПЗП.

Команда десяткової корекції акумулятора – **DA** використовується при арифметичних операціях з числами, що подані двійково - десятковими кодами. Після операції додавання таких чисел потрібно використовувати команду десяткової корекції суми

DA A

Десяткова корекція не конвертує двійкове число до двійково-десяткового уявлення.

Набір команд для віднімання набагато менший. Команда обчислення різниці – **SUBB** існує тільки у варіанті з відніманням вмісту біта переносу:

SUBB A, #dat
SUBB A, Rn
SUBB A, @Ri
SUBB A, src

Мнемоніка цієї команди відповідає словам SUBtraction with Borrow (тобто віднімання з урахуванням позики, тому що при відніманні утворюється позика, а не перенос). З цієї причини перед обчисленням різниці молодших байтів потрібно обов'язково очищати біт переносу, якщо немає впевненості в його вмісті. При обчисленні різниці старших байтів цього робити не потрібно.

Існує також команда зменшення – **DEC**, за допомогою якої значення заданого операнда зменшується на одиницю (DECrement):

DEC A
DEC Rn
DEC src
DEC @Ri

Команди зменшення для роботи з двобайтовим форматом даних немає.

Результати виконання команд додавання і віднімання впливають на вміст бітів переносу, додаткового переносу і переповнення в слові стану програми.

Команди множення (**MUL** – MULtiplication) і ділення (**DIV** – DIVision) працюють при записі операндів в акумулятор і регістр B. Для команди множення порядок запису співмножників у ці регістри неважливий.

MUL AB

Добуток має двобайтний формат. Молодший байт добутку записується в акумулятор, а старший – у регістр B.

Для команди ділення ділене повинно бути записане в акумулятор, а дільник – у регістр B:

DIV AB

Після виконання команди в акумуляторі знаходиться частка, а в регістрі B – залишок. Після виконання команд множення і ділення в біт переносу заноситься 0. Якщо старший байт добутку не дорівнює нулю, то в біт переповнення заноситься 1. При діленні на 0 у біт переповнення також заноситься 1.

4.3.3. Логічні команди

Система команд мікроконтролера MCS-51 дозволяє реалізувати логічні операції:

- І **ANL** - ANd Logical,
- АБО **ORL** - OR Logical,
- ВИКЛЮЧНЕ АБО **XRL** - eXclusive oR Logical.

Усі логічні команди не впливають на вміст слова стану програми, за винятком випадків безпосереднього запису результату в цей регістр.

Команди для обчислення функції І використовують різноманітні способи адресації.

```
ANL   A, #dat
ANL   A, Rn
ANL   A, @Ri
ANL   A, src
ANL   dest, A
ANL   dest, #dat
```

Ці команди можуть використовуватися, наприклад, для очищення окремих бітів двійкового коду або для перевірки наявності «1» у деякому наборі бітів.

Аналогічний набір команд існує і для функції АБО:

```
ORL   A, #dat
ORL   A, Rn
```

```
ORL   A, @Ri
ORL   A, src
ORL   dest, A
ORL   dest, #dat
```

Ці команди можуть використовуватися, наприклад, для установлення окремих бітів двійкового коду в «1».

Для функції ВИКЛЮЧНЕ АБО формат команд аналогічний.

```
XRL   A, #dat
XRL   A, Rn
XRL   A, src
XRL   A, @Ri
XRL   dst, A
XRL   dst, #dat
```

Ці команди можуть використовуватися, наприклад, для зміни значення окремих бітів двійкового коду на зворотне (toggle) або для перевірки кодів на збіг.

Існують логічні операції, що виконуються лише над вмістом акумулятора. Для інвертування його вмісту призначена команда **CPL**

```
CPL   A
```

Мнемоніка цієї команди ComPLement означає “доповнення”, хоча в результаті її виконання виходить не додатковий, а зворотний код.

До логічних команд обробки інформації можуть належати команди циклічного зсуву вліво - **RL** і вправо - **RR**, що працюють з 8 бітами (акумулятор) чи з 9 бітами (акумулятор та біт переносу) – **RLC** та **RRC**:

```
RL   A
RR   A
RLC   A
RRC   A
```

Перша літера в мнемокодах цих команд означає Rotate (повертати), друга вказує на напрямок (Left чи Right), а третя – на участь біта переносу. При зсуві вліво в усі біти акумулятора, крім самого молодшого, записується старий вміст сусіднього правого біта. При зсуві вправо в усі біти акумулятора, крім самого старшого, записується старий вміст сусіднього лівого біта. Якщо біт переносу не бере участі в операції циклічного зсуву, то при зсуві вліво в самий молодший біт записується старий вміст самого старшого біта, а при зсуві вправо в самий старший біт записується старий вміст самого молодшого біта. При участі біта переносу його вміст включається в ланцюжок циклічного переносу, що дозволяє здійснювати зсув багатобайтових кодів.

Якщо перед виконанням команди зсуву очистити біт переносу, то зсув за участю цього біта може використовуватися як команда арифметичної операції. Зсув вправо відповідає поділу додатного числа на 2, при цьому у біт переносу записується залишок від ділення. Зсув вліво відповідає множенню додатного числа на 2, при цьому “1” у біті переносу сигналізує про переповнення.

До операції циклічного зсуву на 4 розряди без участі біта переносу можна віднести команду **SWAP**:

```
SWAP   A
```

Вона здійснює обмін тетрад вмісту акумулятора, так що її можна інтерпретувати як циклічний зсув вмісту акумулятора на 4 розряди вліво.

4.3.4. Бітові команди

Команди порозрядної обробки інформації відрізняються від команд арифметичних операцій тим, що працюють з окремими бітами незалежно від вмісту байта в цілому. Їх також називають булевими командами, тому що з їх допомогою можна обчислювати функції алгебри логіки І, АБО, НІ та ВИКЛЮЧНЕ АБО.

Для обчислення функції І між бітом переносу і бітом, що адресується, існує дві команди:

```
ANL   C, bit
ANL   C, /bit
```

Символ косої риски в другій команді означає, що для обчислення логічної функції використовується інвертоване значення біта.

Дві аналогічні команди обчислюють функцію АБО:

```
ORL   C, bit
ORL   C, /bit
```

Інвертування окремих бітів виконується командами:

```
CPL   C
CPL   bit
```

Для запису констант “0” і “1” використовуються команди очищення **CLR** і установлення **SETB** (SET Bit означає “установити біт”):

```
CLR   C          SETB   C
CLR   bit        SETB   bit
```

Кілька команд пересилання інформації працюють з бітовими змінними. У команді MOV джерелом або приймачем повинен бути біт переносу С:

```
MOV   C, bit
MOV   bit, C
```

4.3.5. Команди передачі керування

Адресація команд (тобто їх виконання) одна за одною, їх відповідне розташування в пам'яті програм називається *природною*. При завершенні читання чергової команди вміст програмного лічильника містить адресу коду операції наступної команди. Команди, у результаті виконання яких може бути змінений природний порядок виконання команд, називаються *керуючими*. Передача керування може відбуватися залежно від виконання деяких умов. Тоді це називається умовною передачею керування (conditional jump). Якщо команда завжди передає керування в іншу частину програми, то це називається *безумовною передачею керування* (unconditional jump). Деякі з умовних керуючих команд використовують інформацію, що міститься в слові стану програми, а інші самі порівнюють байти чи перевіряють стан окремих бітів. В операндах керуючої команди міститься інформація для зміни вмісту програмного лічильника. Способи адресації керуючих команд розрізняються за дальністю переходу на короткі (short), абсолютні (absolute) і довгі (long). Для передачі керування програмісту досить вказати символічну адресу переходу у відповідному операнді команди асемблера.

При використанні короткого способу адресації в останньому байті команди міститься різниця між адресою тієї команди, якій передається керування, і адресою команди, що йде за керуючою командою. Ця різниця може складати від –128 до +127. Такий спосіб часто називають *відносним* (relative).

Назва абсолютного переходу успадкована від попередньої моделі мікроконтролера, у якого обсяг пам'яті програм був обмежений двома кілобайтами. При переході до використання 64 кбайт пам'яті об'єм старого адресного простору стали називати *сторінкою* (page), тому цей спосіб забезпечує адресацію в межах однієї сторінки пам'яті програм у 2 кбайти. При абсолютному способу адресації 11 молодших розрядів вмісту програмного лічильника замінюється вмістом адресної частини команди.

Для довгого переходу адресна частина команди складається з двох байтів, вміст яких заноситься в програмний лічильник, тобто виконується перехід в межах 64 кбайт пам'яті програм.

Короткий, абсолютний і довгий безумовні переходи (JuMP) позначаються в мнемонічних кодах команд початковими літерами S (Short), A (Absolute) і L (Long) відповідно – **SJMP**, **AJMP**, **LJMP**.

Команди безумовного переходу можна розділити на команди без запам'ятовування адреси повернення, команди із запам'ятовуванням адреси повернення і команди повернення. В останньому різновиді команд безумовного переходу адресна частина відсутня. Є також команда безумовного переходу, що використовує індексну адресацію.

Команди безумовного переходу без повернення використовують адресацію всіх трьох довжин:

SJMP rel

AJMP adr11

LJMP adr16

Команда безумовного переходу **JMP** з індексною адресацією дозволяє змінювати адресу переходу за вмістом акумулятора:

JMP @A + DPTR

За допомогою цієї команди можна здійснювати переходи за кожною з 256 адрес щодо вмісту регістра покажчика. Оскільки адреса передачі керування залежить від вмісту акумулятора, цей варіант команди, по суті, не є командою безумовної передачі керування. Ця команда може використовуватися як перемикач, якщо в заданій області програми записати команди безумовного переходу на різні блоки програми. Оскільки ці команди будуть займати більше одного байта, розглянута команда може використовуватися для передачі керування не більше ніж за 128 адресами.

Команди безумовного переходу з поверненням не використовують коротку адресацію – **ACALL** та **LCALL**:

ACALL adr11

LCALL adr16

CALL перекладається як виклик. Команди застосовуються для виклику підпрограм, після виконання яких керування повинне повертатися команді, яка йде за командою виклику. Для цього перед занесенням адресної частини команди виклику в програмний лічильник старе значення програмного лічильника записується в два байти ОЗП, які адресуються покажчиком стека. Спочатку в стек заноситься молодший байт програмного лічильника, а потім старший байт.

Типовий спосіб повернення до програми передбачає, що останньою виконуваною командою підпрограми повинна бути команда **RET**.

Мнемоніка цієї команди відповідає слову RETurn (повернутися). Команда не має адресної частини, тому що при її виконанні в програмний лічильник записуються два байти, які адресуються покажчиком стека (першим записується старший байт, а другим – молодший). Програміст повинен забезпечити правильний вміст покажчика стека на момент виходу з підпрограми (значення покажчика при вході та виході з підпрограми - однакові). Ця вимога означає, що при виконанні підпрограми кількість команд запису в стек повинна дорівнювати кількості команд зчитування зі стека або покажчик стека коригується примусово. Оскільки адресний простір стека розміщується в ОЗП, програміст повинен подбати про те, щоб команди, які використову-

ють інші способи адресації, не руйнували інформації в адресному просторі стека.

Для повернення до програми, що виконувалася в момент апаратного переривання з підпрограми його обробки, використовується команда **RETI**. Літера I відповідає слову Interrupt (переривання). Апаратні переривання передають керування певним адресам із збереженням адреси повернення, одночасно фіксується факт обробки переривання з визначеним пріоритетом.

Команда RETI працює аналогічно RET, але крім повернення вона повідомляє мікроконтролер про кінець обробки переривання.

Команди умовного переходу використовують тільки відносний спосіб адресації в межах 128 байт. Для кожної умови існує пара команд, одна з яких здійснює передачу керування при її дотриманні, а інша – при недотриманні. У полі коментарів наводиться розшифровка мнемонічного позначення цих команд.

Передача керування залежить від того, дорівнює чи не дорівнює нулю вміст акумулятора. Перехід за відносною адресою (на мітку), якщо нуль – **JZ**:

JZ rel

Перехід за відносною адресою (на мітку), якщо не нуль – **JNZ**:

JNZ rel

Як умова переходу використовується рівність біта переносу одиниці чи нулю. Перехід, біт переносу встановлено – **JC**:

JC rel

Перехід, біт переносу скинуто – **JNC**:

JNC rel

Існують команди, що використовують як умову переходу рівність одиниці (**JB**) чи нулю (**JNB**) будь-якого біта в регістрі спеціальних функцій чи адресованого біта в ОЗП:

JB bit, rel

JNB bit, rel

Команда передачі керування за рівністю біта одиниці має варіант з очищенням вмісту цього біта – **JBC**:

JBC flag, rel

Комбінація різних команд передачі керування дозволяє обійти обмеження, пов’язані зі способом адресації. Якщо адреса команди, на яку потрібно передати керування, відрізняється від адреси наступної команди на значну величину (додатну чи від’ємну), то можна використовувати пару команд, перша з якої при дотриманні зворотної умови передає керування через один рядок вихідного тексту програми, а друга є командою безумовного переходу з абсолютною чи далекою адресацією.

Перераховані команди здійснюють перехід залежно від результатів попередніх обчислень. Однак є керуючі команди, які самі здійснюють обчислення для одержання умов передачі керування. Мнемонічний код першої з таких команд – **CJNE** (Compare and Jump if Not Equal означає “порівняти і перейти, якщо не дорівнює”). Це єдина команда мікроконтролера, що має 3 операнди. Її чотири різновиди відрізняються способами адресації джерела і приймача:

```
CJNE   A, src,  rel
CJNE   A, #dat,  rel
CJNE   Rn, #dat,  rel
CJNE   @Ri, #dat,  rel
```

Команда обчислює різницю першого і другого операндів. Передача керування за зазначеною адресою здійснюється при нерівності операндів. Операнди не змінюються, і результат віднімання нікуди не записується, за винятком біта переносу. При порівнянні додатних чисел біт переносу встановлюється в «1», якщо перший операнд менший другого. Якщо за адресою переходу записати команду передачі керування за вмістом біта переносу, то в результаті одного порівняння можна виконати три різні блоки програми.

Другим використанням цієї команди є реалізація “case” структури – послідовного порівняння отриманих даних з наперед заданими і відповідною реакцією у випадку їх збігу.

Інша команда – **DJNZ** (Decrement and Jump if Not Zero) означає “зменшити і перейти, якщо не дорівнює нулю”) зменшує вміст першого операнда на одиницю. Якщо операнд не дорівнює 0, то керування передається за зазначеною адресою:

```
DJNZ   Rn, rel
DJNZ   dest, rel
```

Ця команда зручна для програмування циклу за лічильником. Перед початком циклу за адресою приймача треба записати число, яке дорівнює кількості повторень циклу. Якщо в другому операнді записати адресу початку циклу і не змінювати вміст першого операнда іншими командами в циклі, то задана ділянка програми буде повторена задану кількість разів.

4.4. Мова програмування асемблер

Мова асемблера призначена для подання в зручному для сприйняття вигляді програм, написаних у машинних кодах. Вона дозволяє програмістові:

- користуватися мнемонічними позначеннями кодів машинних команд;

- присвоювати символьні імена регістрам мікроконтролера, коміркам пам’яті й константам;
- використовувати для подання числових констант різні системи числення: шістнадцяткову, десяткову, вісімкову, двійкову;
- позначати спеціальними мітками рядків програми, для того щоб до них могли звертатися за символьними іменами міток інші частини програми при передачі керування.

Програма Асемблера перетворить вихідний текст програми в об’єктний або завантажувальний модуль. В об’єктний модуль поміщаються результати трансляції вихідного тексту програми, налагоджувальна й службова інформація, що дозволяє об’єднувати модулі в єдину програму, провадити налагодження програми з використанням символьних імен, визначених програмістом. Завантажувальний модуль являє собою програму в машинних кодах, підготовлену для завантаження в пам’ять мікроконтролера. Дані в завантажувальному модулі можуть перебувати в бінарному подані (такому ж, як і в пам’яті мікроконтролера) або в hex форматі фірми Intel для використання програматорами пам’яті мікроконтролерів.

У той час, як мнемонічні позначення кодів машинних команд конкретного мікроконтролера визначаються його виробником і залишаються незмінними, інші атрибути мови асемблера: алфавіт мови, службові команди асемблера – директиви й правила їх запису змінюються від однієї реалізації мови до іншої. Автори намагаються розширити функціональність і зручність використання свого варіанта асемблера й при цьому дотримуються устояного набору правил і директив, що спрощує перехід програміста з одного асемблера на інший. Як приклад розглянемо вільно розповсюджуваний асемблер ASEM-51 [19], що набув широкого застосування серед виробників програмного забезпечення. Асемблер ASEM-51 являє собою двопрохідний асемблер для мікроконтролерів сім’ї MCS-51. Під час першого проходу асемблер перевіряє дотримання правил мови; видаляє коментарі; визначає значення міток і символьних змінних, генерує таблицю символів модуля; обробляє директиви (службові інструкції), зустрінуті в тексті програми. Під час другого проходу генеруються об’єктний або завантажувальний модулі програми.

4.4.1. Оператори мови

Вихідний текст програми може містити оператори трьох видів:

[symbol:] [instruction [arguments]] [;comment]
symbol instruction argument [;comment]
$control [(argument)] [;comment]

де symbol – символьне ім’я;

instruction – мнемонічне позначення команди або директиви;
arguments – аргументи (операнди) команди;
control – директива асемблера;
comment – коментарі.

Всі елементи, що поміщені у квадратні дужки, можуть бути відсутні.

В одному рядку може бути записаний тільки один оператор, ознакою кінця оператора є символ "повернення каретки".

Довжина рядка не повинна перевищувати 255 символів.

Лексичні елементи оператора можуть розділятися пропусками або символами табуляції, операнди команди відділяються один від одного комами.

Всі дані наступні за символом «;», як і порожній рядок вважаються коментарем.

У символьних іменах символи верхнього й нижнього регістрів - еквівалентні.

```
Load:       MOV A, #0FFH   ;мітка Load стосується рядка MOV A,
#0FFH

L12:                       ;мітка L12 належить до наступного рядка
            PUSH  12       ;

time        EQU      20    ;присвоювання символьному імені time
                           ;значення 20

$INCLUDE    (8051.MCU)     ;включення в текст програми файла
                           ;визначення регістрів - 8051.MCU
```

4.4.2. Алфавіт мови

Символьні імена вихідної програми являють собою підмножину таблиць символів ASCII для DOS і ANSI для WINDOWS. У вихідному тексті програми, написаному на ASEM-51, допустиме використання наступних символів:

– символи інтервалу;
– літери;
– знаки;
– цифри.

До *символів інтервалу* належать "пропуск" і "табуляція".

Як *літери* сприймаються латинські літери верхнього й нижнього регістру:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z,

a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.

Нижче наведений перелік *цифр*:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Найменування знаків і їхнє позначення наведено в таблиці 4.2.

Таблиця 4.2 -Найменування знаків і їхнє позначення

Найменування	Позначення
Номер	#
Знак грошової одиниці	$
Апостроф	‘
Кругла дужка ліва	(
Кругла дужка права	)
Зірочка	*
Плюс	+
Кома	,
Мінус	-
Крапка	.
Дробова риса	/
Двокрапка	:
Крапка з комою	;
Менше	<
Дорівнює	=
Більше	>
Знак питання	?
Комерційне ет (at)	@
Підкреслення	_

Знаки, комбінації знаків (<>, >=, <=), а також символи інтервалу є роздільниками конструкцій мови. До і після знака-роздільника в будь-якій конструкції мови можуть бути вставлені символи інтервалу.

Із символів формуються ідентифікатори й числа.

4.4.3. Ідентифікатори

Ідентифікатор - це символьне ім’я об’єкта програми – адреси, константи та ін. Як ідентифікатор може бути використана будь-яка послідовність літер і цифр алфавіту мови, знак питання «?» і знак "підкреслення" « _ ».

Ідентифікатор може починатися тільки з літери, що дозволяє відрізняти його від числа. Кількість символів в ідентифікаторі обмежено довжиною рядка (255 символів), але значущими є перший 31 символ.

Приклади ідентифікаторів:

ADR5, FFFFh, Port_number_1.

У мові програмування ASEM-51 є три категорії ідентифікаторів:

– ключові слова;
– вбудовані імена;
– обумовлені імена.

Зарезервовані (ключові) слова

Ключове слово є визначальною частиною оператора мови асемблера. Значення ключових слів мови асемблера ASEM-51 не можуть бути змінені або перевизначені.

У мові ASEM-51 є наступні категорії ключових слів:

– інструкції;
– директиви;
– операції.

Інструкції з форми запису збігаються із мнемонічними позначеннями команд мікроконтролерів сім'ї MCS-51 і разом з операндами становлять команди мікроконтролера. Список інструкцій:

ACALL, ADD, ADDC, AJMP, ANL, CALL, CJNE, CLR, CPL, DA, DEC, DIV, DJNZ, INC, JB, JBC, JC, JMP, JNB, JNC, JNZ, JZ, LCALL, LJMP, MOV, MOVC, MOVX, MUL, NOP, ORL, POP, PUSH, RET, RETI, RL, RLC, RR, RRC, SETB, SJMP, SUBB, SWAP, XCH, XCHD, XRL.

Директиви визначають дії в програмі, які повинні бути виконані асемблером у процесі перетворення вихідного тексту програми в об'єктний код. У мові програмування ASEM-51 дозволені наступні директиви:

AT, BIT, BSEG, CODE, CSEG, DATA, DB, DBIT, DS, DSEG, DW, END, EQU, IDATA, ISEG, NAME, ORG, SET, USING, XDATA, XSEG.

Оператори виконуються асемблером у процесі обчислення виразів на етапі трансляції вихідного тексту програми для визначення конкретного числа, що використовується в команді. Перелік операторів, що мають символьні імена:

AND, EQ, GE, GT, HIGH, LE, LOW, LT, MOD, NE, NOT, OR, SHL, SHR, XOR.

4.4.4. Вбудовані імена

Вбудовані імена привласнені акумулятору A, робочим регістрам R0 - R7 поточного банку регістрів і їхнім адресам - AR0 - AR7, регістрам покаж-

чика даних DPTR і лічильника команд PC, регістровій парі, що бере участь в операціях ділення/множення AB, а також прапору переносу C.

4.4.5. Обумовлені імена

Обумовлені імена оголошуються користувачем. У мові програмування ASEM-51 є наступні категорії обумовлених ідентифікаторів:

- мітки;
- змінні адресного типу;
- змінні числового типу;
- імена сегментів;
- назви програмних модулів.

Для простоти доступу до регістрів спеціальних функцій можливе використання визначених ідентифікаторів, що збігаються з іменами РСФ, наведеними в інструкції користувача відповідного мікроконтролера. Визначені ідентифікатори для кожного з підтримуваних типів процесорів описуються у файлах *.MCU, де символьному імені ставиться у відповідність відповідне числове значення. Характер ідентифікатора вказується при описі – байт (DATA), біт (BIT) або адреса в пам’яті програм (CODE). Для завдання ідентифікаторів порту P0, біта дозволу переривання IE0 і адреси точки скидання RESET застосовуються наступні описи:

```
P0      DATA    080H
IE0     BIT     089H
RESET   CODE    000H
```

4.4.6. Константи

Числові константи в мові програмування ASEM-51 подані цілими числами без знака у двійковій, вісімковій, десятковій і шістнадцятковій формах запису. Для визначення основи системи числення використовується суфікс (літера, що іде за числом):

- B - двійкове число (дозволені цифри 0, 1);
- Q\O - вісімкове число (дозволені цифри 0, 1, 2, 3, 4, 5, 6, 7);
- D - десяткове число (дозволені цифри 0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
- H - шістнадцяткове число (дозволені цифри 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Для десяткового числа суфікс може бути відсутнім. Діапазон значень чисел перебуває в межах від 0 до 65535.

Приклади запису чисел:

101b, 11001001B, 753Q, 210o, 48765, 0ah, 0BD61H, 1FH.

Число завжди починається з цифри. Це необхідно для того, щоб відрізняти шістнадцяткове число від ідентифікатора.
ABCDH - ідентифікатор
0ABCDH - число
Скрізь, де допустиме використання числових констант, можливе використання символьних констант - символів, поміщених в апострофи, наприклад:
‘X’ – 8-бітна константа 58H (ASCII код символу «X»)
‘a5’ – 16-бітна константа 6135H (ASCII коди символів «a» і «5»)
Символьні константи зручні для збереження в програмі текстової інформації, у тому числі рядків символів. У цьому випадку кожний символ у пам’яті програм буде замінений відповідним байтом - номером символу в кодовій таблиці символів. Використання символьних констант підвищує читабільність тексту програми.

```
MESSAGE:    DB    ‘Block error!’      ; символьна константа
MESSAGE:    DB    42h, 6ch, 6fh, 63h, 6bh, 20h, 65h, 72h, 72h, 6fh, 72h, 21h
            ;той самий текст у числовому поданні
```

4.4.7. Використання виразів в операндах команд асемблера

Арифметичні вирази складаються з операндів, операторів (знаків операцій) і круглих дужок. Операндами можуть бути числа, ідентифікатори й константи, обумовлені програмістами, або спеціальні символи асемблера. Всі операнди розглядаються як двобайтні числа без знака. Спеціальні символи, які можуть використовуватися як операнди:

AR0, ... , AR7 - адреси регістрів R0, ... , R7;

$ - адреса поточної команди асемблера.

В арифметичних виразах застосовуються унарні й бінарні операції. Їхній список наведений у таблицях 4.3 і 4.4.

Таблиця 4.3 -Унарні операції

Оператор	Операція	Визначення
+	Підтвердження знака	+x = x
-	Інверсія (доповнення до двох)	-x = 0-x
NOT	Побітова інверсія (доповнення до одиниці)	NOT x = 0FFFFH-x
HIGH	Старший байт двобайтного числа	-
LOW	Молодший байт двобайтного числа	-

Оператори, які є не спеціальними символами, а ключовими словами, наприклад SHR або AND, повинні бути відділені від їх операндів, принаймні, одним символом інтервалу. (M SHL N) – число M зсувається вліво на N по-

зицій. У виразах оператори обчислюються зліва направо відповідно до пріоритету операцій, що наведені у таблиці 4.5. Порядок обчислень може бути змінений круглими дужками. Вкладеність круглих дужок не обмежується. Результат обчислень завжди приводиться до 16-бітного числа без знака, переповнення ігнорується.

Таблиця 4.4 - Бінарні операції

Оператор	Операція	Результат
+	Додавання	
-	Віднімання	
*	Множення	
/	Ділення	
MOD	Ділення за модулем	
SHL	Логічний зсув вліво	
SHR	Логічний зсув вправо	
AND	Логічне І	
OR	Логічне АБО	
XOR	Виняткове АБО	
.	Бітова операція, використовувана у випадку бітів, що адресуються у байті	
EQ або =	Дорівнює	0, якщо НЕПРАВДА, FFFFH, якщо ІСТИНА
NE або <>	Не дорівнює	
LT або <	Менше	
LE або <=	Менше або дорівнює	
GT або >	Більше	
GE або >=	Більше або дорівнює	

Таблиця 4.5 -Пріоритет операцій

Оператор	Тип	Пріоритет
( )		Вищий
+ - NOT HIGH LOW	Унарний	
.	Бінарний	
* / MOD		
SHL SHR		
+ -		
EQ = NE <> LT < LE <= GT > GE >=		
AND		
OR XOR		Нижчий

Приклади:

Вихідний вираз: P1.((87+3) /10 AND - 1 SHR 0DH)

Покрокове обчислення виразу:

1. P1.(90/10 AND - 1 SHR 0DH)
2. P1.(90/10 AND 0FFFEH SHR 0DH)
3. P1.(9 AND 0FFFEH SHR 0DH)
4. P1.(9 AND 7)
5. P1.1
   P1.1 - перший біт байта P1
   P1 - визначений символ P1=90H
   P1.1=91H
6. 91H

При використанні окремих бітів операції логічного зсуву спрощують запис операндів. Зсув здійснюється з нульової позиції.

MOV A, #(1 SHL 3) еквівалентне MOV A, #001000b

Можна пересувати декілька одиниць одночасно:

MOV A, #((3 SHL 5) OR (1 SHL 0)) еквівалентне MOV A, #01100001b.

Замість цифр можуть використовуватися ідентифікатори, яким привласнені числові значення. Якщо в акумуляторі необхідно встановити біт BST (BST = 6), то можна використовувати команду:

ORL A, #(1 SHL BST), що еквівалентно ORL A, 01000000b.

4.4.8. Директиви асемблера ASEM-51

При розгляді директив асемблера необхідно відзначити, що ключові слова будуть написані прописними літерами, аргументи директив подані у вигляді <arg1>, а числові вирази - <expr1>. У випадку списку з будь-яким числом членів використовуються крапки «...», а елементи, укладені у квадратні дужки, можуть бути відсутніми. Директиви наводяться як у порядку їхнього використання в програмі, так і відповідно до їхньої значимості й застосовності в інших асемблерах.

Типи та використання сегментів

На відміну від процесорів, для МК сегментація пов’язана не зі способом обчислення адреси, а з форматом оброблюваної інформації. З одного боку, в програмі присутні два сегменти: сегмент коду програми і сегмент даних. З іншого боку - мікроконтролер має три види пам’яті: пам’ять програм, внутрішню оперативну пам’ять з регістрами спеціальних функцій і зовнішню оперативну пам’ять. У пам’яті програм, крім кодів команд, зберігаються константи, тобто розташовуються принаймні два сегменти: коду й даних. Чергування коду програми й констант може привести до небажаних

наслідків. Внаслідок помилок адресації дані можуть бути випадково виконані як програма або, навпаки - код програми може бути сприйнятий і оброблений як дані. Вбудована оперативна пам’ять має адресний простір для даних у форматах байтів і бітів, тобто ще два сегменти.

Приклад розміщення різних типів даних у пам’яті мікроконтролера наведено на рисунку 4.1.

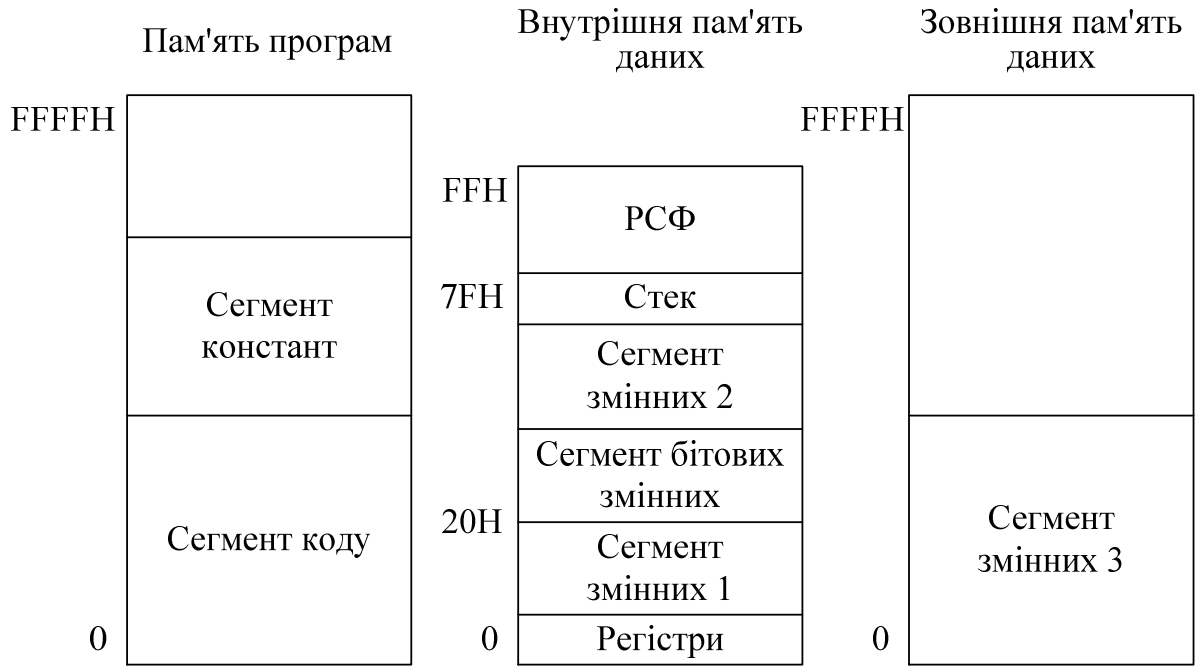


Рисунок 4.1 - Розподіл пам’яті мікроконтролера на сегменти

Для зручності програмування бажано явно виділити принаймні п’ять сегментів:

1) програми (коду);
2) констант;
3) змінних;
4) бітових змінних;
5) змінних у зовнішній пам’яті даних.

Найбільш простий спосіб визначення сегментів - це використання абсолютних сегментів пам’яті. При цьому способі розподіл пам’яті робиться вручну. Початкова адреса сегмента задається програмістом, і він же стежить за тим, щоб сегменти не перекривалися один з одним. Використання абсолютних сегментів дозволяє більш гнучко працювати з пам’яттю даних, тому що тепер байтові змінні в пам’яті даних можуть бути призначені за допомогою директиви резервування пам’яті DS, а бітові змінні - за допомогою директиви резервування бітів DBIT. Під сегмент стека звичайно виділяється вся

область внутрішньої пам’яті, не зайнята змінними. Це дозволяє створювати програми з максимальним рівнем вкладеності підпрограм.

Директиви **CODE, DATA, IDATA, BIT, XDATA** визначають символьні імена для адрес у п’яти можливих адресних просторах:

```
<symbol>  CODE    <expr>    ;визначення адреси в пам’яті програм
<symbol>  DATA    <expr>    ;визначення адреси в пам’яті даних
<symbol>  IDATA   <expr>    ;визначення непрямої адреси в пам’яті
                            ;даних
<symbol>  BIT     <expr>    ;визначення адреси біта
<symbol>  XDATA   <expr>    ;визначення адреси в зовнішній пам’яті
                            ;даних
```

Для імен у просторах DATA, IDATA і BIT значення <expr> не повинні перевершувати 0FFH і повинні бути відомі на першому проході асемблера. Діапазон адрес 0 - 0FFH для сегмента IDATA доступний для представників сім’ї MCS-51 з вбудованою пам’яттю даних 256 байт (верхні 128 байт ОЗП доступні в непрямому режимі адресації). Мікроконтролери з 128 байтами пам’яті не мають різниці у використанні директив DATA і IDATA. Привласнені імена не можуть бути перевизначені в іншому місці програми.

Приклад застосування:

```
EPROM     CODE     08000H
STACK     DATA     7
V24BUF    IDATA    080H
REDLED    BIT      P1.5
SAMPLER   XDATA    0100H
```

Для визначення абсолютних сегментів пам’яті використовуються директиви **CSEG, DSEG, ISEG, BSEG, XSEG.** Ці директиви не призначають ім’я сегменту:

```
CSEG [AT <expr>]    ;визначення абсолютного сегмента CODE [за
адресою]
DSEG [AT <expr>]    ;визначення абсолютного сегмента DATA [за
адресою]
ISEG [AT <expr>]    ;визначення абсолютного сегмента IDATA [за
адресою]
BSEG [AT <expr>]    ;визначення абсолютного сегмента BIT [за адресою]
XSEG [AT <expr>]    ;визначення абсолютного сегмента XDATA [за
адресою]
```

Якщо базова адреса сегмента визначена виразом "AT <expr>", то починається новий абсолютний сегмент за зазначеною адресою. Базова адреса повинна бути відома на першому проході асемблера. Якщо вираз "AT <expr>"

опущено, то при першому використанні директиви встановлюється нульова базова адреса, а надалі зберігається попереднє значення покажчика місцезнаходження в поточному сегменті.

При старті програми встановлюється за замовчуванням сегмент CODE, базовим адресам й покажчикам місцезнаходження присвоюються **нульові** значення.

Приклад застосування:

```
DSEG              ;перехід до попереднього сегмента DATA
CSEG  AT  8000h   ;початок нового сегмента CODE за адресою 8000H
XSEG  AT  0       ; початок нового сегмента XDATA за адресою 0
```

Директива **BSEG** дозволяє визначити абсолютний сегмент у внутрішній пам'яті даних з бітовою адресацією за заданою адресою. Використання бітових змінних дозволяє значно заощаджувати внутрішню пам'ять програм мікроконтролера за рахунок використання однорозрядних прапорів.

Приклад використання:

```
BSEG          AT    8     ;Сегмент починається з восьмого біта
Indicator     DBIT  1     ;Прапор режиму індикації
Knopka        DBIT  1     ;Прапор режиму
Power         DBIT  1     ;Прапор режиму живлення
```

Директива **CSEG** дозволяє визначити абсолютний сегмент у пам'яті програм за заданою адресою.

Приклад використання:

```
;Реакція на переривання
CSEG  AT  03h ;Вектор переривання від зовнішнього переривання INT0
Int0:
SETB  P0.0    ;У відповідь на переривання по спаду
NOP           ;формується вузький імпульс
CLR   P0.0    ;
Reti
```

Директива **DSEG** дозволяє визначити абсолютний сегмент у внутрішній пам'яті даних за заданою адресою. Передбачається, що до цього сегмента будуть звертатися команди з прямою адресацією.

Приклад використання:.

```
DSEG  AT  20H     ;Розмістити сегмент за адресою, що збігається з
                  ; бітовим адресним простором мікроконтролера для
                  ;можливості одночасно бітової й байтової адресації
Prog    DS    1   ;Змінна, що відображає стан програми
Robota  DS    1   ;Змінна, що відображає режими роботи
```

Розглядаючи даний приклад визначення сегмента й резервування простору для змінних разом із прикладом використання директиви BSEG, можна помітити, що для змінної Robota резервується та ж адреса, що й для бітів Indicator, Knopka і Power. Зміна зазначених бітів приводить до зміни змінної і навпаки. Такий підхід дозволяє підвищити ефективність роботи програмного забезпечення.

Директива **ISEG** дозволяє визначити абсолютний сегмент у внутрішній пам'яті даних за заданою адресою.

Приклад використання:

```
ISEG    AT      80H     ;Розмістити сегмент у діапазоні адрес,
                        ;сполучених з SFR (МК із 256 байт пам'яті)
Buffer  DS      10      ;масив з 10 байт
```

Директива **XSEG** дозволяє визначити абсолютний сегмент у зовнішній пам'яті даних за заданою адресою. Використання цієї директиви не відрізняється від використання директиви DSEG.

Директиви резервування пам'яті та ініціалізації даних

Директива **DB** – define bytes – визначення байтів.

DB <arg1> [,<arg2> [,<arg3>…]]

Директива DB резервує й ініціює байти, розташовані в пам'яті програм, значеннями, які визначаються аргументами. Аргументи можуть бути виразами, які приводяться до 8-бітного значення або рядком символів будь-якої довжини.

DB 0A3H, 'b', 'Version 1.3', 100, 2+3*15, 1001001B

Застосування директиви DB допускається тільки в сегменті CODE.

Директива **DW** – define words – визначення слів.

DW < expr 1> [,<expr 2> [,<expr 3>…]]

Директива DW працює аналогічно директиві DB, але ініціює двобайтові слова. Аргументи можуть бути довільними вираженнями, що вимагають для розміщення двох байт. Ініціалізація окремих значень або списків значень здійснюється тільки в числовому форматі.

Оскільки система команд MCS-51 не працює з даними у форматі слова, то розташування старших і молодших байтів може задаватися програмістом довільно. Для тих, хто працює з системою команд IBM PC, звично записувати старший байт за старшою адресою, притому адреса молодшого байта повинна бути парною. Наведені директиви ініціалізації даних у форматі слова розташовують старший байт за молодшою адресою без дотримання парності адрес.

DW 0, 0f101h, 2006

Директива **DS** – define space – визначення простору.

DS <expr>

Директива DS резервує для подальшого використання кількість байт, обумовлена <expr>, але не ініціалізує їх. Значення <expr> повинне бути відомим на першому проході асемблера. Директива не застосовується в BIT сегменті.

DS 200H

Директива **DBIT** – define bits – визначення простору бітів.

DBIT <expr>

Директива DBIT працює аналогічно директиві DS, але резервує деяку кількість бітів і застосовується тільки в BIT сегменті.

DBIT 16

Директива **USING** – виділення банку регістрів

USING <expr>

Установлює використовуваний банк регістрів відповідно до значення <expr>, що повинне лежати в інтервалі 0-3. Директива USING впливає тільки на значення вбудованих імен AR0, ... , AR7, що надають безпосередні адреси регістрів R0, ... , R7 поточного банку. Значення <expr> повинне бути відомим на першому проході асемблера, значення за замовчуванням – 0.

Приклад використання:

```
USING 2
```

Директива **ORG** – origin – адреса початку блока в поточному сегменті.

ORG <expr>

Директива ORG встановлює покажчик місцезнаходження в поточному сегменті на значення, яке визначає <expr>. Значення <expr> повинно бути відомим при першому проході асемблера та дорівнювати чи бути більшим за базову адресу сегмента. Початкове значення всіх покажчиків місцезнаходження дорівнює нулю.

```
org 0               ;за адресою 0 розміщення
ajmp  Start         ; команди передачі управління на мітку Start
org   100h          ; за адресою 100h буде поміщена
Start:              ;мітка Start
mov   dptr, #$      ;поточне значення PC = 100h
mov   dptr, #Start  ;значення #Start =100h
```

Директива **END** – кінець програми

```
END                 ;кінець програми
```

За цією директивою транслятор припиняє трансляцію. У наступній за цим рядком частині вихідного тексту програми можуть бути коментарі або порожні рядки.

Директиви підстановок

Директива **EQU** – equal – визначає символьне ім’я числовій константі або регістру.

<symbol> EQU <expr> ;присвоєння імені константі
<symbol> EQU <reg> ;присвоєння імені регістру

Якщо імені <symbol> привласнене числове значення <expr>, то надалі воно має тип - число. Якщо регістрове значення - <reg>, то тип - регістр. Як <reg> можуть використовуватися вбудовані імена регістрів - A, R0...R7.

Ім’я, один раз визначене директивою EQU, не може бути змінене.

Директива **SET** – присвоює значення змінної або регістру.

<symbol> SET <expr> ;присвоєння імені числовій змінній
<symbol> SET <reg> ;присвоєння імені регістру

Директива SET працює аналогічно директиві EQU, але імена можуть бути перевизначені наступними директивами SET. Значення <expr> і <reg> повинні бути відомими на першому проході асемблера.

Імена, визначені директивою EQU, не можуть бути перевизначені директивою SET і навпаки.

Приклад використання:

```
MAXMONTH        EQU   12
OCTOBER         EQU   MAXMONTH-2
REG_COUNTER     EQU   R5

TIMER       SET   1
TIMER       SET   TIMER +1
TIMER       SET   A
```

Директиви управління

Асемблери мають велику кількість директив, що впливають на створення файла лістингу програми. У лістингу наводяться повні дані про розроблювану програму. Ставляться у відповідність номери та зміст рядків вихідного тексту програми, адреси пам’яті та машинні коди, що згенеровані асемблером. Фрагмент лістингу програми наведено в таблиці 4.6. З лістингу видно, що директиви початку блока org не генерують машинних кодів (рядок 10 та 14 таблиці 4.6), а дають асемблеру інструкції розмістити машинні команди, що йдуть за нею, за вказаними адресами (рядок 11 та 16). Порожні рядки 12 та 15, як і рядок 13 з коментарем не впливають на розподіл адрес

пам’яті мікроконтролера.

Таблиця 4.6 Фрагмент лістингу програми

Рядок програми	Адреса пам’яті	Машинні коди	Текст програми	Коментар
10			org 0000h	
11	0000	02 01 00	jmp start	
12				
13				;Start prg
14			org 0100h	
15				
16	0100	74 30	start: mov A, #030h	
17	0102	12 08 A9	call wrcmd	
18	0105	74 0C	mov A, #lcd_setvisible+4	

Наприкінці лістингу програми знаходиться таблиця символів, яка вміщує всі символьні імена, що використані в програмі або наведені в файлах, які були підключені до вихідного файла програми. Фрагмент таблиці символів наведено в таблиці 4.7.

Таблиця 4.7 - Фрагмент таблиці символів

Символ	Тип	Значення	Рядок, де був визначений
ACC	DATA	E0	
IE	DATA	A8	
IE0	BIT	89	
LCD_SETVISIBLE	NUMBER	0008	1886
WRCMD	CODE	08A9	1944

Ім’я WRCMD, що використовується як аргумент команди call (рядок програми 17, табл. 4.6), в таблиці символів описане як адреса в пам’яті програм (мітка), константа LCD_SETVISIBLE – як число. Результат обчислення асемблером виразу з константами підставляється до машинної команди (рядок програми 18, табл. 4.6). В таблиці символів описуються й визначені ідентифікатори, наприклад ACC, IE, IE0.

При трансляції програміст може управляти форматом видачі лістингу та змістом інформації, яка буде до нього включена.

Однією з найбільш використовуваних директив управління є $INCLUDE, яка при трансляції розміщує в місці свого знаходження вміст вказаного текстового файла.

$INCLUDE (filename)

Приклад використання:

```
org    200
$INCLUDE (table.asm)   ;за адресою 200 буде розміщено вміст файла table.asm.
```

Згідно з цією директивою вказаний файл включається в задане місце тексту програми до початку трансляції. Якщо файл знаходиться в каталозі за умовчанням, то достатньо вказати ім’я файла і його розширення. Інакше потрібно також додати до операнда шлях до файла.

4.4.10. Макроси

Макроси дозволяють об’єднувати декілька команд асемблера в один макрос – суперкоманду. Макрос визначається як блок коду з присвоєнням символьного імені. Виклик макроса за символьним ім’ям призводить до підстановки в місце виклику блока команд з параметрами, що були задані при виклику. Параметри, що передаються макросу, повинні бути відомі при першому проході асемблера. Використання макросів полегшує читання тексту програми, але підвищує вірогідність появи помилок через відсутність контролю за даними, що передаються та повертаються.

4.4.11. Відмінності Асемблерів

З різних причин у програміста може виникнути необхідність використання різних Асемблерів, чи програмного забезпечення, що було написане з використанням іншого Асемблера. До таких причин, насамперед, відноситься використання підпрограм сторонніх розробників і передача власних вихідних текстів. При зовнішній схожості, Асемблери мають безліч розходжень, які тією чи іншою мірою утрудняють роботу з «нерідними» файлами. Розходження можуть бути несуттєві (наприклад, різний набір директив керування) й більш серйозні. Проблема розбіжності імен деяких директив вирішується пошуком директив з аналогічною функціональністю. Відсутність реалізації, як правило, ставиться до функцій малої значущості, які легко можна виключити з програми. Більш трудомісткі рішення доводиться застосовувати у випадку відсутності деяких операцій. Часто спостерігаються розходження в реалізації операцій HIGH, LOW, NOT, бітова крапка, позначення та послідовності виконання операцій в арифметичних виразах. Існують розбіжності в позначенні систем числення та в запису символьних констант. Ці розходження стосуються обчислення арифметичних виразів під час першого проходу асемблера, тому необхідно коректувати текст програми відповідно до нових правил. Такі ж дії застосовуються й при зміні правил запису дирек-

тив: ім’я директиви починається з крапки (.org), ознакою керуючої директиви є знак «#» (#include “table.asm”) і т.д.

При відсутності або відмінності підтримки макросів вирішення проблеми полягає в заміні макросів фрагментами коду з підстановкою параметрів. Незначні на перший погляд відмінності у правилах запису та реалізації макросів можуть призвести до помилок в роботі програми, що важко знаходяться.

Деякі Асемблери використовують регістрозалежні імена змінних і констант (ABC, Abc і abc - різні імена) з різним числом розпізнаваних символів – 6, 11 та ін. Для усунення плутанини і можливих помилок такі імена для різних змінних використовувати не рекомендується, необхідно використовувати осмислені імена (price, а не ppp), при необхідності додаючи до них лічильник (led_1). Визначені імена регістрів і іменованих бітів мікроконтролера так само можуть бути регістрозалежними.

Для спрощення аналізу тексту програми в ряді Асемблерів символьне ім’я, що починається з першої позиції рядка, вважається міткою. Використання шаблонів, в яких за позицією в рядку визначали тип імені, має своїм джерелом перфокарти старих обчислювальних машин, але й у сучасних мовах програмування такий підхід продовжує залишатися актуальним (у мові Python блоки операторів обумовлюються відступами). Використання табуляції в тексті програми дозволяє більш наочно показати її структуру, зменшує можливість помилок. Звичка до формалізації тексту програми: мітка починається з першої позиції, команда - через інтервал табуляції дозволяє не тільки підвищити читаність програми, але й спрощує перехід на Асемблер іншого автора.

Приклади вирішення деяких проблем несумісності Асемблерів

Багато Асемблерів підтримують список операторів арифметичних і логічних обчислень, що повторює стандартний набір операторів мови С (табл. 4.8).

Для Асемблерів, які не підтримують директив HIGH та LOW, замість запису

MOV R3, #HIGH(0ABCDh) і MOV R3, #LOW(0ABCDh)

необхідно використовувати команди

MOV R3, #(0ABCDh SHR 8) і MOV R3, #(0ABCDh)

або MOV R3, #(0ABCDh >> 8) і MOV R3, #(0ABCDh).

Якщо правила мови дозволяють символьні константи, що містять один символ, взятий в лапки (наприклад, “s”, “W”) та повертають ASCII значення цього символу, то рядок символів переписують до еквівалентного вигляду:

DB “Ver. 123” еквівалентно DB “V”,”e”,”r”,”.”,” “,”1”,”2”,”3”

Таблиця 4.8 - Арифметичні і логічні оператори асемблера

Оператор	Тип	Опис
+	Арифметичні	Додавання
-		Віднімання
*		Множення
/		Ділення
%		Ділення за модулем
<<		Логічний зсув вліво
>>		Логічний зсув вправо
-	Унарні	Інверсія (доповнення до двох)
~		Побітова інверсія (доповнення до одиниці)
==	Відносини	Дорівнює
!=		Не дорівнює
<		Менше
>		Більше
<=		Менше або дорівнює
>=		Більше або дорівнює
&	Бінарні	Двійкове “І”
|		Двійкове “АБО”
^		Двійкове “виключне АБО”

Для позначення системи числення, в якій записано число, використовується система префіксів і суфіксів (табл. 4.9).

Таблиця 4.9 - Префікси і суфікси систем числення

Основа системи числення	Префікс	Суфікс
2	%	B або b
8	@	Q або q, O або o
10 (за умовчанням)	немає	D або d або нічого
16	$ або 0x	H або h

Таке зображення чисел еквівалентне:

1234H	або	$1234
	або	0x1234
100d	або	100
177400O	або	@177400
01011000b	або	%01011000

5. ПРИКЛАДИ ПРОГРАМУВАННЯ МК51

5.1. Приклади використання команд передачі даних

Приклад 5.1. Передати вміст буфера УАПП за непрямою адресою з R0:

```
MOV   @R0, SBUF      ;передача прийнятого по послідовному каналу
                     ;байта у вбудований ОЗП
```

Приклад 5.2. Завантажити в покажчик даних адресу 7F00H масиву даних, розташованого у ВПД:

```
MOV   DPTR, #7F00H  ; завантаження початкового значення
                    ; покажчика даних
```

Приклад 5.3. Завантажити керуюче слово в регістр управління таймером (встановити біти IT1 і IT0).

- Зі знищенням інформації, що була в регістрі,

```
MOV TCON, #00000101B      ;важко зрозуміти суть дії
```

або

```
b_IT0       EQU       0      ; значення номера біта IT0
b_IT1       EQU       2      ; значення номера біта IT1
. . .
MOV   TCON, #((1 SHL b_IT1) OR (1 SHL b_IT0))
```

зрозумілі дії, одночасна зміна бітів, потребує опису бітів

- Зі збереженням інформації, що була в регістрі, вважаємо, що ідентифікатори бітів IT0 і IT1 визначені, як адреси

```
SETB        IT0              ; зрозумілі дії, не одночасна зміна
SETB        IT1              ; бітів, в пам'яті займає 2 байти
```

або

```
ORL   TCON, #((1 SHL b_IT1) OR (1 SHL b_IT0))      ; зрозумілі дії,
```

одночасна зміна бітів, потребує опису бітів, в пам'яті займає 3 байти

Приклад 5.4. Скинути всі прапори користувача (бітова область у вбудованому ОЗП з адресами 20H – 2FH):

```
        MOV R0, #20H     ; значення початкової адреси області прапорів
        MOV R1, #10H     ; лічильник (довжина області прапорів)
LOOP:   MOV @R0, #0      ; скидання одного байта  (8 прапорів)
        INC  R0          ; перехід до наступного байта
        DJNZ  Rl,  LOOP  ; цикл з лічильником Rl R1 = R1 – 1,  якщо R1 > 0
                         ; то перейти на мітку LOOP
```

Приклад 5.5. Запам'ятати у зовнішній пам'яті даних, починаючи з адреси 5000H, вміст регістрів банку 0. Скористаємось адресами регістрів банку 0 у вбудованому ОЗП (0H … 7H) і непрямою адресацією комірки пам'яті.

```
        MOV  PSW,  #01000b    ; вибір для роботи першого банку регістрів
                              ; вмісту регістрів банку 0 не змінюється
        MOV  R0, #8           ; встановлюємо лічильник на 8
                              ; запис початкових адрес:
        MOV  DPTR, #5000H     ; комірки у зовнішній пам'яті
        MOV  Rl, #0           ; регістра R0 нульового банку регістрів
LOOP:   MOV  A, @R1           ; читання вмісту комірки пам'яті
                              ; за адресою @R1
        MOVX  @DPTR, A        ; передача з акумулятора у зовнішню пам'ять
        INC  R1               ; перехід до наступного регістра
        INC  DPTR             ; приріст покажчика адреси
        DJNZ  R0, LOOP        ; цикл з лічильником R0
```

Приклад 5.6. Звернення до таблиці у пам'яті програм. Часто необхідно мати в пам'яті програм таблиці готових рішень (lookup table – довідкова таблиця). Для можливості роботи з такими таблицями, що зберігаються в пам'яті програм, використовується команда звернення до пам'яті програм – MOVC. Пояснимо використання цих команд на прикладі підпрограми обчислення синуса кута *X* (*X* змінюється в межах від 0 до 89° з дискретністю 1°). Найшвидше обчислення функції можна одержати шляхом вибірки готового значення синуса з таблиці. Така таблиця для діапазону 0 – 89° займе 90 байтів при похибці 0,4%. Кожен байт таблиці містить дробову частину двійкового подання синуса. Діапазон відносних значень 0 – 255, що зберігаються в таблиці TAB_SIN, відповідає значенням синуса 0 – 0,999. Параметром для підпрограми служить значення кута *X*, що знаходиться в акумуляторі. Значення синуса після виконання підпрограми SINX також знаходиться в акумуляторі. Декілька з можливих варіантів програми розглянуто нижче.

Варіант 1. Таблиця розташована безпосередньо за фрагментом програми, який її використовує.

```
SINX:
        INC   A                 ; корекція значення акумулятора
        MOVC  A, @A + PC        ; завантаження значення синуса з таблиці
        RET                     ; повернення
        ; Таблиця значень синуса
TAB_SIN:
        DB   00000000B          ; SIN(0) = 0
        DB   00000100B          ; SIN(1)  = 0.017
        DB   00001001B          ; SIN(2)  = 0.035
        . . .
        DB   11111111B          ; SIN(89) = 0.999
```

Для отримання правильних результатів дії оператора MOVC A, @A + PC необхідно провести корекцію значення акумулятора, яке відповідає номеру елемента таблиці, на величину різниць адрес оператора MOVC й першого елемента таблиці. Оператори MOVC и MOVX займають лише 1 байт пам’яті, тому до значення акумулятора потрібно додати розмір пам’яті, що займають оператори між MOVC і таблицею – RET займає один байт. Сума розміру таблиці і величини корекції не може бути більше 255 елементів.

Варіант 2. Таблиця розташована в довільному місці пам’яті програм, розмір таблиці не перевищує 256 елементів.

Для отримання даних необхідно використовувати команду MOVC A, @A + DPTR. У цьому випадку адреса елемента таблиці визначається сумою значень регістрів (A + DPTR).

```
SINX:
        MOV DPTR, #TAB_SIN      ; передача в регістр-покажчик даних
                                ; адреси таблиці
        MOVC A, @A + DPTR       ; отримання значення синуса з таблиці
        RET                     ; повернення
        . . .
        . . .
; Таблиця значень синуса
TAB_SIN:
        DB   00000000B          ; SIN(0) = 0
        . . .
        DB   11111111B          ; SIN(89) = 0.999
```

Після виконання операції отримання з таблиці, в акумуляторі знаходиться значення синуса, а не номер елемента таблиці! Для отримання наступного значення з таблиці необхідно збільшити номер елемента, який треба зберегти у проміжному місці зберігання.

Варіант 3. Таблиця розміщена в перших 255 байтах пам’яті програм, розмір таблиці обмежений доступною ємністю пам’яті програм.

```
ORG        40H              ; таблиця починається з адреси 40H сегмента CODE
; Таблиця значень синуса
TAB_SIN:
DB  00000000B               ; SIN(0) = 0
. . .
DB  11111111B               ; SIN(89) = 0.999
    . . .
SINX:           ;
    MOV A, #TAB_SIN     ; передача в акумулятор адреси таблиці
    MOVC A, @A + DPTR ; завантаження значення синуса з таблиці
    RET                 ; повернення
```

Для отримання наступного значення з таблиці необхідно збільшити номер комірки (INC DPTR)

Приклад 5.7. Операції зі стеком. Механізм доступу до стека МК51: перед завантаженням в стек вміст регістра-покажчика стека (SP) інкрементується, а після витягання зі стека декрементується. Покажчик стека зберігає адресу останньої зайнятої комірки структури стека.

За сигналом системного скидання в SP заноситься початкове значення 07H. Перший елемент даних зберігатиметься за адресою 08H. Для двобайтових адрес молодший байт зберігається за молодшою адресою.

Перевизначити SP на адресу addr можна командою MOV SP, #addr.

Стек може розташовуватися в будь-якому місці вбудованої пам’яті даних, враховуючи те, що він зростає в сторону великих адрес. Звичайно стек використовується для організації звернень до підпрограм, в цьому випадку в ньому зберігається адреса, на яку перейде програма після виконання підпрограми.

При обробці переривань стек може бути використаний для передачі параметрів підпрограмам і для тимчасового зберігання вмісту регістрів спеціальних функцій та ін. Підпрограма повинна зберегти в стеку вміст тих регістрів, які вона сама використовуватиме, а перед поверненням в перервану програму повинна відновити їх значення.

Підпрограма обробки зовнішнього переривання може, наприклад, мати таку структуру:

```
ORG         3               ; встановлення адреси вектора INT0
    SJMP    SUBINO          ; перехід на підпрограми  обробки
ORG         30H
```

```
SUBINO:
        PUSH    PSW             ; збереження в стеку PSW
        PUSH    ACC             ; збереження в стеку акумулятора
        PUSH    B               ; збереження в стеку B
        PUSH    DPL             ; збереження в стеку DPL
        PUSH    DPH             ; збереження в стеку DPH
        MOV     PSW, #1000B     ; використання регістру PSW
TEST:
        . . .                   ; використання регістрів A, B, DPTR …
        POP     DPH             ; відновлення значення регістрів
        POP     DPL
        POP     B
        POP     ACC
        POP     PSW
        RETI                    ; повернення
```

При значенні SP, яке до переходу на виконання підпрограми обробки переривання було 1FH, вміст пам’яті, зайнятої стеком, наведений в табл. 5.1, де PC – значення програмного лічильника на момент виникнення переривання.

Таблиця 5.1 - Вміст пам’яті, зайнятої стеком

Адреса пам’яті	Після входу в підпрограму обробки	Після виконання підпрограми до мітки TEST:
26 H	Не визначено	DPH
25 H	Не визначено	DPL
24H	Не визначено	B
23H	Не визначено	ACC
22H	Не визначено	PSW
21H	PCH	PCH
20H	PCL	PCL
1FH		

Іноді при завершенні підпрограми необхідно повернутися не в те місце програми, звідки вона була викликана (найчастіше це потрібно в разі повернення з підпрограми обробки переривання). Для вирішення цієї задачі використовується зміна вмісту стека. У прикладі фрагмента програми наведено умовні адреси, за якими розміщуються команди. Виклик підпрограми ST здійснювався з адреси 0001, повернення здійснюється на адресу 0003. Для

повернення в інше місце програми встановлюємо мітку. Необхідно відзначити, що мітка не займає елемента пам'яті, а отже, її адреса збігається з адресою команди, на яку вказує мітка.

```
0001        ACALL ST
0003        NOP
0004        NOP
0005        NOP
0006  M2:
0006        NOP
0007        NOP
            . . .
      ST:
            NOP                 ; в SP адреса повернення 0003
            . . .               ;
            DEC SP              ; зменшуємо покажчик стека
            DEC SP              ; (видаляємо із стека адресу повернення)
            MOV DPTR, #M2       ; завантажуємо нову адресу повернення
            PUSH DPL            ; поміщаємо її в стек
            PUSH DPH            ;
            RET                 ; повертаємося за адресою мітки M2
                                ; (адреса 0006)
```

5.2. Приклади використання команд арифметичних операцій

Приклад 5.8. Скласти два двійкових багатобайтових числа. Обидва доданки розташовуються у вбудованій пам'яті даних, починаючи з молодшого байта. Початкові адреси доданків задані в R0 і R1. Довжина доданків в байтах задана в R2, результат зберігається на місці другого доданка:

```
            CLR C           ; скидання біта переносу
LOOP:       MOV  A, @R0     ; завантаження в акумулятор поточного байта
                            ; першого доданка
            ADDC  A, @R1    ; складання з урахуванням перенесення
            MOV @R0, A      ; збереження результату
            INC  R0         ; адреса наступного байта доданка 1
            INC  Rl         ; адреса наступного байта доданка 2
            DJNZ  R2, LOOP  ; цикл, якщо додавання не закінчено
```

Приклад 5.9. Програма віднімання двох двобайтових чисел, перше зберігається в регістрах R0 R1, молодший байт в R0, друге – в регістрах R1

R2, молодший байт в R1. Результат розміщується в регістрах R0 R1, молодший байт в R0:

```
CLR   C          ; очищення біта переносу
MOV   A, R0      ; запис молодшого байта першого числа в акумулятор
SUBB  A, R2      ; віднімання молодшого байта другого числа
MOV   R0, A      ; збереження молодшого байта різниці
MOV   A, R1      ; запис старшого байта першого числа в акумулятор
SUBB  A, R3      ; віднімання старшого байта другого числа і позики
MOV   R1, A      ; збереження старшого байта різниці
```

Різниця між діями додавання і віднімання полягає ще і в тому, що при додаванні потрібно враховувати перенос, а при відніманні – позику. Але для збереження позики при відніманні використовується той же самий біт переносу.

Приклад 5.10. Множення. Команда MUL обчислює добуток двох цілих чисел без знака, що зберігаються в регістрах A і B. Молодша частина результату розміщується в A, а старша – в регістрі-розширювачі B. Якщо вміст B дорівнює нулю, то прапор переповнення OV скидається, інакше – встановлюється. Прапор переносу завжди скидається. Наприклад, якщо акумулятор містив число 200 (0C8H), а розширювач 160 (0A0H), то в результаті виконання команди MUL AB одержимо результат 32 000 (7D00H). Акумулятор міститиме нуль, а розширювач – 7DH, прапор OV буде встановлений, а прапор C – скинутий.

Для отримання добутку двох чисел довільного формату використовуються підпрограми множення.

Приклад 5.11. Ділення. Команда DIV здійснює ділення вмісту акумулятора на вміст регістра-розширювача. Після ділення акумулятор містить цілу частину частки, а розширювач залишок. Прапори C і OV скидаються. При діленні на нуль встановлюється прапор переповнення, а частка залишається невизначеною. Команда ділення може бути використана для швидкого перетворення двійкових чисел на десятиричні двійково-кодовані (BCD) числа.

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

```
MOV   B, #100    ; дільник дорівнює 100 для обчислення числа сотень
DIV   AB         ;
MOV   R0, A      ; збереження числа сотень
```

```
XCH   A, B        ; в акумуляторі частка від ділення на 100
MOV   B, #10      ; дільник дорівнює 10 для обчислення числа десятків
DIV   AB          ;
SWAP  A           ; розміщення числа десятків
                  ; в старшій тетраді акумулятора
ADD   A, B        ; додавання числа одиниць
```

Перетворення займає 16 машинних тактів.

Операція ділення двох чисел довільного формату вимагає використання підпрограми ділення.

Слід зазначити, що множення чи ділення двійкового числа, що складається з декількох байтів, на цілий ступінь двійки здійснюється за допомогою зсуву всіх байтів цього числа вліво чи вправо. Оскільки вага двійкової цифри 1 зростає вдвічі при переході в сусідній лівий розряд, то для множення на 2 потрібен зсув вліво на 1 розряд, для множення на 4 — на 2 розряди і так далі. Для передачі бітів коду числа з одного байта в інший потрібно використовувати операцію циклічного зсуву вліво за участю біта переносу. Оскільки перед черговим зсувом молодшого байта значення біта переносу може бути довільним, треба встановлювати його нульове значення. Операція ділення здійснюється аналогічно, з використанням зсуву вправо, починаючи зі старшого байта. Множення на заздалегідь задану константу, у коді якої міститься невелика кількість одиниць, також може здійснюватись послідовністю операцій зсувів і додавання. Однак у тому випадку, коли значення множника заздалегідь невідоме, потрібно використовувати операції множення.

5.3. Приклади використання команд логічних операцій

Приклад 5.12. Вибрати нульовий регістровий банк:

```
ANL   PSW, #11100111B      ; скидання бітів RS1 і RS0 регістра PSW
```

або

```
CLR   RS1
CLR   RS0
```

Приклад 5.13. Встановити біти 0 – 3 порту 1:

```
ORL   P1, #00001111B
```

Приклад 5.14. Знайти збіжні біти в акумуляторі і регістрі R7:

```
ANL   A, R7               ; логічне І акумулятора і регістра R7
```

Приклад 5.15. Інвертувати біти порту P1, відповідні одиничним бітам акумулятора:

```
XRL   P1, A              ; виключне АБО порту 1 та акумулятора
```

Приклад 5.16. Інвертувати біти 7, 6, 5 порту 0:

```
XRL   P0, #11100000B   ;виключне АБО порту 0 та константи
```

Приклад 5.17. Інвертувати біти 0 – 3 акумулятора:

```
XRL   A, #0FH            ; виключне АБО  акумулятора та константи
```

Приклад 5.18. Управління групою бітів порту. У вбудованій пам'яті даних знаходиться масив розпакованих десяткових цифр. Треба передати масив зовнішньому пристрою. Для передачі чотирьох бітів даних використовуються молодші лінії порту P1. Лінії P1.4 і P1.5 використовуються як сигнали підтвердження. Передачу даних на вихід МК супроводжує сигнал на лінії P1.4. Зовнішній пристрій, прийнявши дані, повідомляє про це в МК сигналом на вході P1.5. Графічне зображення протоколу наведено на рис. 5.1.

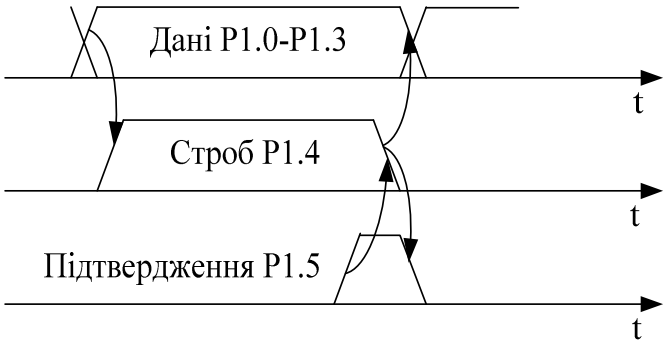


Рисунок 5.1 - Протокол передачі даних

Біти P1.6 – P1.7 в процесі передачі даних не повинні змінювати свого значення. Початковими параметрами для програми є початкова адреса масиву, занесена в R0, і довжина масиву, що зберігається в R1:

```
        ORL   P1, #00100000B  ; настроювання P1.5 на введення інформації
        ANL   P1, #11100000B  ; скидання старих даних і строба
LOOP:
        MOV   A, @R0          ; завантаження байта в акумулятор
        ANL   A, #00001111B   ; виділення даних
        ORL   P1, A           ; видача даних
        SETB  P1.4            ; видача строба
        JNB   P1.5, $         ; очікування відповіді, перехід на
                              ; поточне значення лічильника команд
        INC   R0              ; збільшення покажчика адреси
        CLR   P1.4            ; скидання строба
        DJNZ  R1, LOOP        ; цикл, якщо не всі дані передані
```

5.4. Приклади операцій з бітами

Приклад 5.19. Обчислити логічну функцію трьох змінних.

Для позначення інверсії сигналу використовуватимемо підкреслення символу

$$Y = X \wedge \underline{V} \vee W \wedge (X \vee V).$$

Змінні X, V і W надходять на виводи 2, 1 і 0 порту 1 відповідно. Результат необхідно вивести на вивід 3 порту 1:

```
TMP  EQU  20h    ; адреса комірки пам'яті тимчасового збереження змінних
P1.3 EQU  93h    ; адреса виводу 3 порту P1
F0   EQU  0D5h ; адреса біта тимчасової логічної змінної
                 ; визначення символьних імен:
Y    EQU  P1.3   ; виводу P1.3 порту
X    EQU  2      ; другого біта
V    EQU  1      ; першого біта
W    EQU  0      ; нульового біта
     MOV  R1, P1          ;одночасне отримання логічних змінних
     MOV  C, X            ; C = X
     ORL  C, V            ; C = X ∨ V
     MOV  F0, C           ; збереження результату в F0
     MOV  C, X            ; C = X
     ANL  C, /V           ; C = X ∧ V
     ORL  C, W            ; C = X ∧ V ∨ W
     ORL  C, F0           ; C = X ∧ V ∨ W ∧ (X ∨ V)
     MOV  Y, C            ; виведення функції
```

Тимчасова змінна TMP використовується для одночасного отримання всіх логічних змінних. Її адреса дозволяє адресацію окремих бітів прочитаного байта. Біт F0 з регістра PSW використовується для проміжного зберігання логічної змінної.

Приклад 5.20. Зчитування вмісту порту P1 і занесення зчитаних даних у внутрішній ОЗП за адресами 31, 32:

```
     MOV  31, P1      ;запис даних читання виводів порту до пам'яті
     MOV  32, 31      ;дублювання запису
                      ;(читання виводів може дати інші дані)
```

Приклад 5.21. Інвертувати старшу тетраду порту P2, якщо на виводі P1.4 логічна одиниця

```
     JNB  P1.4, NON_INV   ;перевірка біта
```

```
        XRL   P2,#11110000B   ; інвертування старшої тетради порту
NON_INV:                      ;
```

Інвертування стосується вмісту тригерів на виході порту, а не його виводів.

Приклад 5.22. Передача даних з акумулятора послідовним кодом через лінію P1.0. Кожен біт, що передається, синхронізується позитивним імпульсом на лінії P1.1:

```
        MOV   R7, #8      ; ініціалізація лічильника циклів
LOOP:
        RRC   A           ; передача молодшого біта акумулятора до біта
переносу C
        MOV   P1.0, C     ; передача біта
        SETB  P1.1        ;
        NOP               ;
        CLR   P1.1        ;
        DJNZ  R7, LOOP    ; цикл, якщо не всі біти передані
```

Приклад 5.23. Послідовна передача даних з акумулятора через лінію P2.0. Передача ведеться в манчестерському коді (кожен біт передається двома інтервалами: перший інтервал містить інверсію біта, інший – пряме значення). Тип інтерфейсу – асинхронний, тривалість інтервалів повинна бути однаковою:

```
        MOV   R0, #8      ; лічильник бітів
LOOP:
        RRC   A           ; передача молодшого біта ACC.0 до біта переносу C
        CPL   C           ; інверсія біта
        MOV   P2.0, C     ; передача інверсного біта
        CPL   C           ; відновлення прямого значення біта
        NOP               ; три команди NOP для вирівнювання
        NOP               ; тривалості інтервалів
        NOP               ;
        MOV   P2.0, C     ; передача прямого значення біта
        DJNZ  R0, LOOP    ; цикл, якщо лічильник не нульовий
```

Передача виконується, починаючи з молодших бітів. Тривалість одного інтервалу дорівнює шести машинним циклам.

6. ОРГАНІЗАЦІЯ ВЗАЄМОДІЇ МК51 З ОБ'ЄКТАМИ УПРАВЛІННЯ

6.1. Підключення зовнішньої пам'яті

Можливість використання до 64К зовнішньої пам'яті програм та даних виділило мікроконтролери сімейства MCS-51 серед аналогічних виробів. Особливості підключення зовнішньої пам'яті визначаються протоколом обміну, розглянутому у розділі 3.2.7.

Підключення мікросхеми ПЗП зовнішньої пам'яті програм наведено на рис. 6.1. На початку виробництва МК, зовнішня пам'ять програм дозволяла здешевити виробництво мікроконтролерних систем, бо МК з вбудованою пам'яттю коштували дорого. В теперішній час, коли у продажу присутні МК з різними обсягами вбудованої пам'яті, використання зовнішньої пам'яті програм доцільне при настроюванні системи, коли в мікросхемі ПЗП знаходиться тестова програма, а перепрограмування мікроконтролера недоцільне, при використанні масивів даних великого обсягу з необхідністю їх оперативної заміни тощо.

Доступ до зовнішньої пам'яті програм завжди виконується з використанням 16-розрядної адреси, тому порт P2 бере участь в операції звернення до пам'яті, і його використання для виведення даних, не пов'язаних з пам'яттю програм, ускладнене.

Порт P0 використовується для видачі молодшого байта адреси команди в пам'яті програм і для прийому з неї байта даних. Молодший байт адреси фіксується в регістрі DD2 по спаду сигналу ALE (Address Latch Enable – дозволена фіксація адреси, рис. 3.15), тому для фіксації адреси використовується регістр К555ИР22 (аналог 74LS373). Синхронізація отримання даних відбувається сигналом $\overline{PME}$. Від встановлення нульового рівня сигналу до

читання даних з ПЗП проходить деякий час, що відповідає часу доступу ПЗП. Формування всіх сигналів, що беруть участь у обміні даними з пам'яті програм, проводиться апаратно.

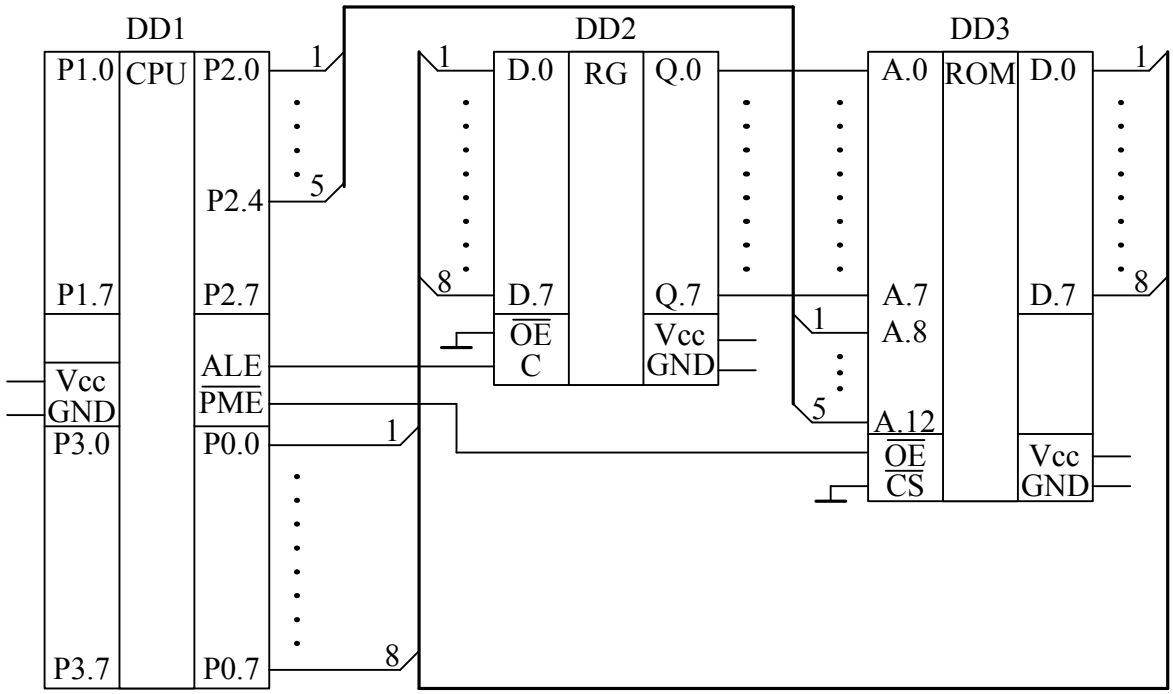


Рисунок 6.1 - Схема підключення зовнішньої пам'яті програм до МК

Частіше, ніж зовнішня пам'ять програм, використовується зовнішня пам'ять даних. Зовнішній ОЗП використовують при обчисленнях, що потребують зберігання великої кількості проміжних результатів, при формуванні зображень на текстовому та, особливо, графічному індикаторах для зберігання бінарної копії екрана як відеопам'ять. Підключення мікросхеми ОЗП зовнішньої пам'яті даних наведено на рис. 6.2. Залежно від розміру зовнішньої пам'яті даних, доступ до неї може відбуватись з використанням як 16-розрядної адреси, так і 8-розрядної адреси, спосіб доступу обумовлює участь порту Р2 в обміні даними.

Як і при звертанні до зовнішньої пам'яті програм, порт Р0 використовується для виведення молодшого байта адреси комірки пам'яті даних і для виведення/введення в/з неї байта даних. Молодший байт адреси фіксується в регістрі DD2 за спадом сигналу ALE (рис. 3.13, 3.14). Синхронізація отримання даних відбувається сигналом $\overline{RD}$ (Р3.7), запис даних синхронізується сигналом $\overline{WR}$ (Р3.6). Від встановлення нульового рівня сигналу до операції з даними ОЗП проходить деякий час, що відповідає часу доступу ОЗП. Сигнали, що беруть участь у обміні з пам'яттю даних, формуються апаратно і використовують два виводи порту Р3.

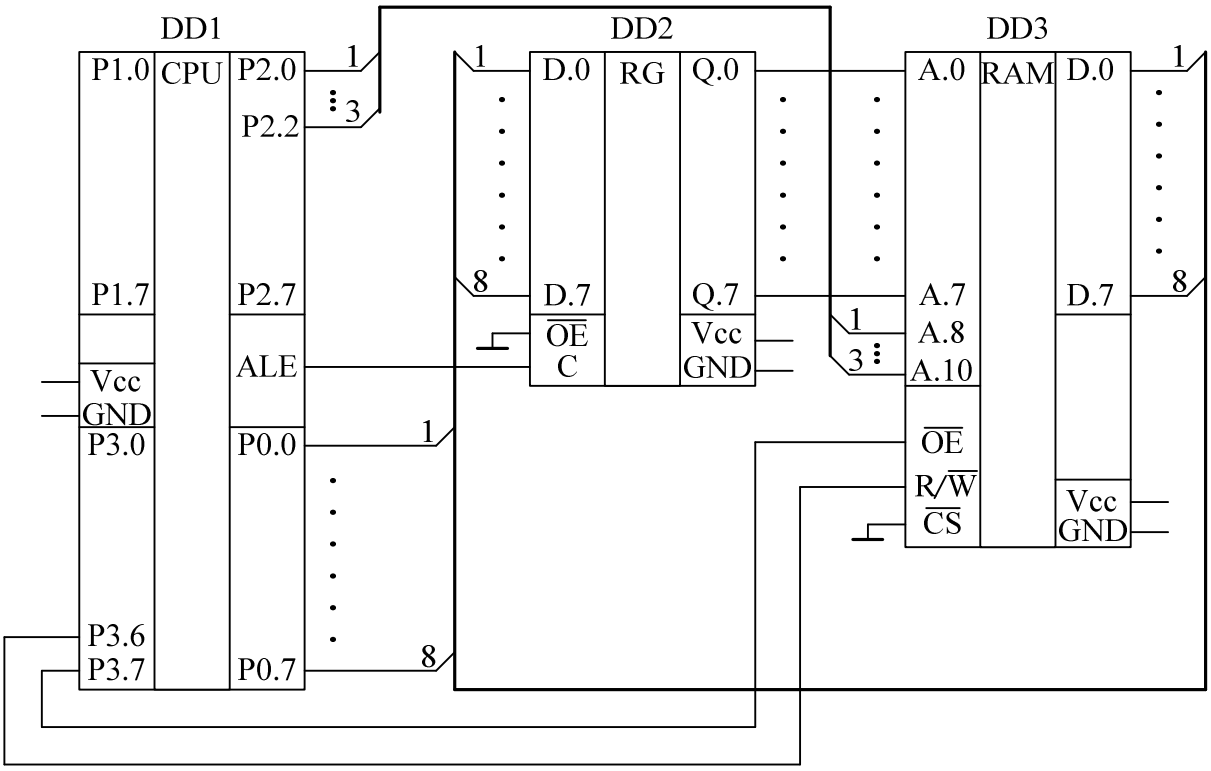


Рисунок 6.2 - Схема підключення зовнішньої пам’яті даних до МК

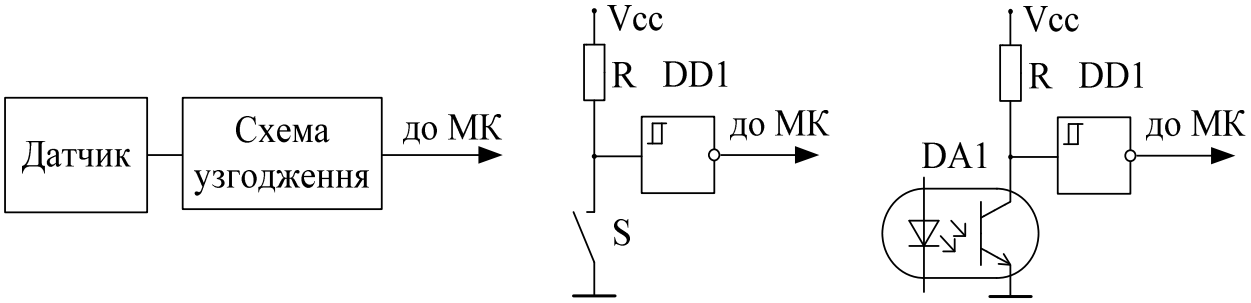
Одночасно з підключенням до МК зовнішньої пам’яті, до системної шини МК, утвореної портами P0, P2, сигналами ALE, $\overline{PME}$, $\overline{RD}$ і $\overline{WR}$, можуть бути підключені зовнішні пристрої, що потребують формування імпульсів синхронізації. Використання системної шини спрощує програмування обміну даними, дозволяє підключити декілька пристроїв з використанням дешифратора їх адреси. Приклад підключення до МК рідкокристалічного індикатора наведено на рис. 6.29 і 6.30.

6.2. Введення інформації з датчиків

6.2.1. Опитування двійкового датчика. Очікування події

У пристроях і системах логічного управління об’єктами події в об’єкті керування фіксуються з використанням різноманітних датчиків цифрового й аналогового типів. Найбільшого поширення набули двійкові датчики типу так/ні, наприклад кнопкові вимикачі, що підключаються до МК з використанням схем узгодження (рис. 6.3, а). Залежно від вимог до мікропроцесорного пристрою, схеми узгодження забезпечують узгодження рівнів напруги, гальванічну розв’язку електричних кіл датчика та мікроконтролера, підвищують завадостійкість кіл обробки вхідних сигналів. Найчастіше двійкові

датчики підключаються до мікроконтролера з використанням логічних елементів з функцією тригера Шмідта (рис. 6.3, б), які не тільки підвищують завадостійкість вхідних кіл, але й забезпечують захист мікроконтролера при виникненні аварійних ситуацій в колах датчика. Замість механічного датчика S можуть підключатись оптронні пристрої (рис. 6.3, в).



а *б* *в*

Рисунок 6.3 - Схема двійкового датчика

Очікування статичного сигналу. Типова процедура очікування події (WAIT) складається з одержання сигналу від датчика (читання біта порту), аналізу значення сигналу і передачі керування залежно від стану датчика. На рис. 6.4 наведено схему алгоритму процедури очікування події, яка фіксується замиканням контакту двійкового датчика.

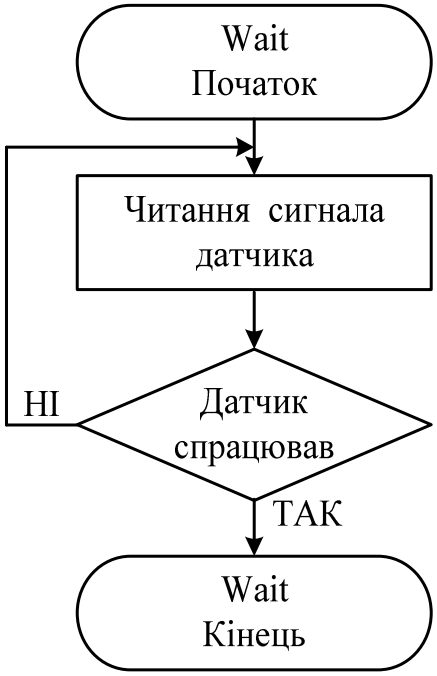


Рисунок 6.4 - Схема процедури Wait очікування події

Конкретна програмна реалізація процедури залежить від того, яким чином датчик підключений до МК. Наприклад, при підключенні датчика (за

схемою рис. 6.3, б) до лінії біта 3 порту 1 програма реалізації процедури очікування замикання контакту буде мати вигляд

```
WAIT1:      JNB P1.3, WAIT1     ; очікування «1» на P1.3
                                ; замикання контакту датчика
```

Іншим випадком типової процедури очікування події є процедура очікування розмикання контакту, що може бути реалізована таким способом:

```
WAIT0:      JB  P1.3, WAIT0     ; очікування «0» на P1.3
                                ; розмикання контакту датчика.
```

Очікування імпульсного сигналу. Схема підключення датчика імпульсного сигналу аналогічна схемі на рисунку 6.3. Особливість процедури очікування імпульсного сигналу полягає в тому, що МК повинен виявити не тільки факт появи, але і факт закінчення сигналу.

Для програмування цієї процедури зручно скористатися розглянутими вище прикладами очікування події, змонтувавши їх послідовно в лінійну програму. Оформляти процедури WAIT1 і WAIT0 у вигляді підпрограм недоцільно, тому що це збільшує час виконання програми, а час виконання програми визначає мінімальну тривалість імпульсу, що може бути виявлений програмою.

Послідовність виконання процедур WAIT1 і WAIT0 залежить від форми імпульсу. Для “негативного” імпульсу (1 – 0 – 1) процедура WAIT0 випереджає процедуру WAIT1, для “позитивного” (0 – 1 – 0) – йде за нею.

Нижче наведено приклад програмної реалізації процедури очікування “негативного” імпульсного сигналу при підключенні датчика до біта 3 порту 1 за умови, що початковий стан входу – одиничний.

```
WAIT0:      JB   P1.3, WAIT0    ; очікування P1.3 = 0
WAIT1:      JNB  P1.3, WAIT1    ; очікування P1.3 = 1
```

Для виявлення короткочасних позитивних імпульсів можна використовувати спосіб фіксації імпульсу з допомогою зовнішнього тригера [20] (рис. 6.5). Вхідний, навіть короткий, імпульс встановлює вихідний сигнал тригера. Після фіксації надходження імпульсу сигнал з мікроконтролера скидає тригер.

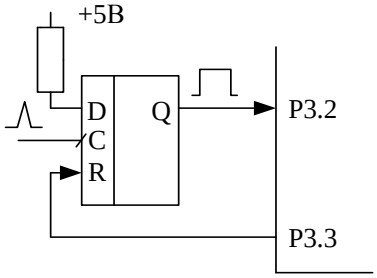


Рисунок 6.5 - Схема фіксації імпульсу з допомогою зовнішнього тригера

На вхід МК у цьому випадку надходить не короткочасний сигнал з датчика, а сигнал, який формується тригером. Тригер встановлюється по фронту імпульсу, а скидається програмним шляхом — видачею сигналу на вхід скидання тригера. Тривалість вхідного імпульсу знизу при цьому обмежена тільки швидкодією тригера.

6.2.2. Усунення деренчання контактів

Замикання та розмикання механічних контактів завжди супроводжується деренчанням, яке викликане конструкцією і шорсткістю поверхні контактів, пружністю їх матеріалу, та призводить до появи короткочасних імпульсів напруги під час комутації. Діаграму напруги *u* на контактах наведено на рис. 6.6, де розмикання контакту починається в момент часу t_1, а замикання – в момент часу t_3. На інтервалах часу t_1 - t_2 та t_3 - t_4 система може зробити хибний висновок про багаторазове замикання-розмикання контактів.

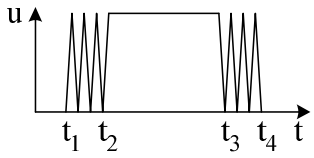


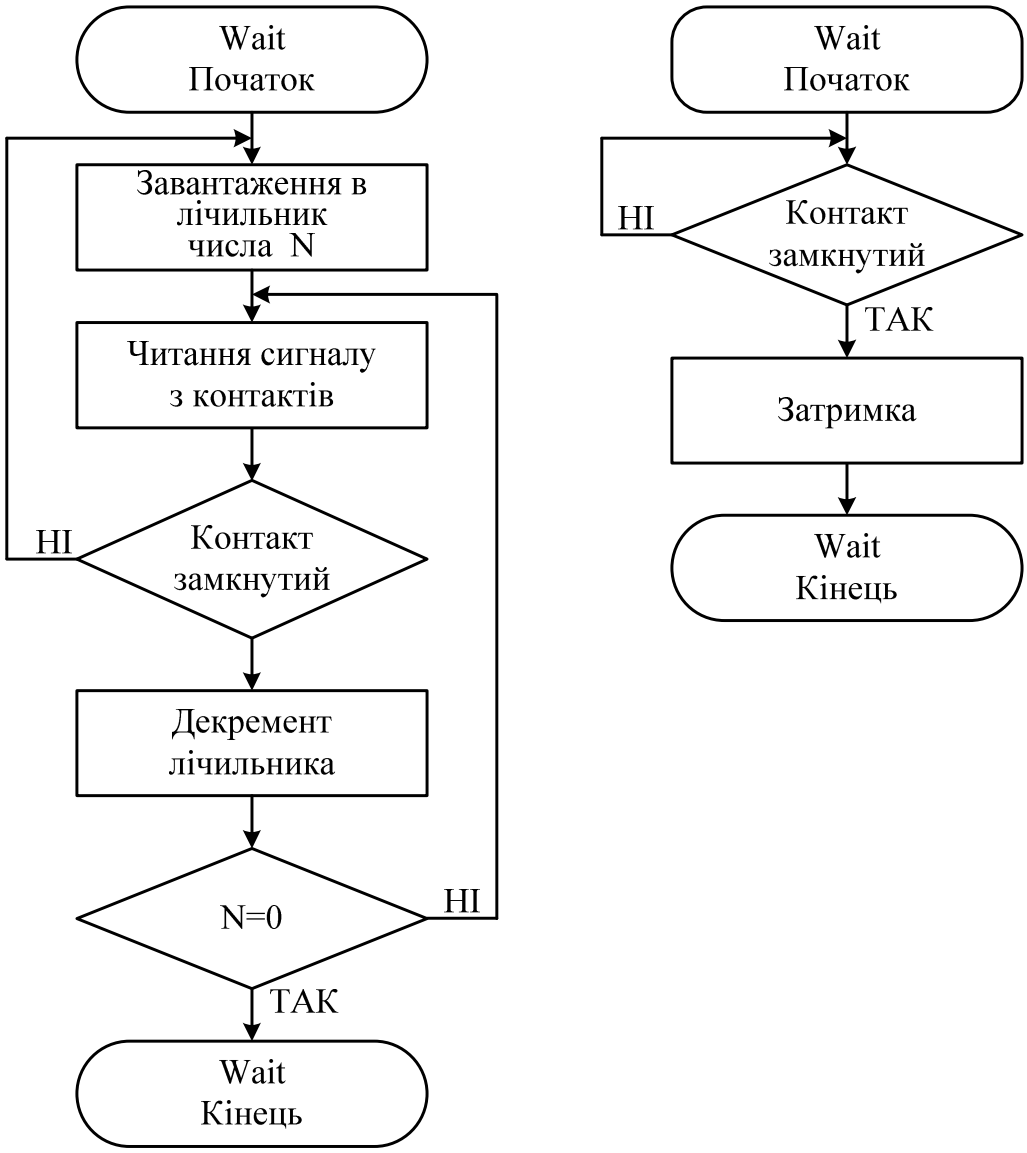
Рисунок - 6.6 Сигнал з контактів, що деренчать

Для усунення неоднозначності сигналу з контактів в момент комутації, потрібно вживати схемотехнічних чи програмних заходів. Найпростіше фільтрування сигналу сповільнює фронт перемикання напруги і для підвищення завадостійкості потребує використання додаткових логічних елементів з функцією тригера Шмідта. Найбільшого поширення набули два програмних способи очіування сталого сигналу:

1) підрахунок заданої кількості збіжних значень сигналу;

2) часова затримка.

Схеми двох процедур усунення перешкод від деренчання контактів при очікуванні сигналу «0» (замикання контакту) показано на рис. 6.7. Суть першого способу полягає в багаторазовому зчитуванні сигналу з контакту. Підрахунок вдалих опитувань (тобто опитувань, які визначили, що контакт стійко замкнутий) ведеться програмним лічильником. Якщо після серії вдалих опитувань зустрічається невдале, то підрахунок починається спочатку (продовжується деренчання контактів). Контакт вважається стійко замкнутим, якщо надійшло N вдалих опитувань. Число N підбирається експериментально для кожного типу використовуваних датчиків і звичайно знаходиться у межах від 5 до 50.



а *б*

Рисунок 6.7 - Схеми процедур усунення деренчання контактів шляхом: *а* – багаторазового зчитування; *б* – використання часової затримки

Приклад програмного усунення деренчання контактів [20] наводиться для випадку, коли датчик імпульсного сигналу підключений до входу P3.4; рахунок вдалих опитувань ведеться в регістрі R3, N = 20.

```
BRZKT:     MOV R3, #20       ; ініціалізація лічильника
BRZKT_1:   JB P3.4, BRZKT    ; якщо контакт розімкнутий, то почати відлік
                             ; опитувань с початку
           DJNZ R3, BRZKT_1  ; повторювати, поки R3 не буде дорівнювати 0
```

Усунення деренчання контактів другим способом - шляхом введення часової затримки - здійснюється таким чином: програма, знайшовши замикання контакту, забороняє опитування стану цього контакту на час, свідомо

більший тривалості перехідного процесу комутації. Недоліком цього способу є можливість отримання хибного уявлення про стан контакту при надходженні короткої перешкоди. До переваг застосування часової затримки відносяться простота реалізації та можливість не використовувати спеціальні процедури затримки, якщо опитування контактів відбувається рідше ніж 10 разів на секунду.

6.2.3. Введення інформації з клавіатури

У пристроях введення інформації найбільшого поширення набули клавіатури. За складністю вони поділяються на прості (менше 8 клавіш) та складні (більше 8 клавіш).

Для отримання коду натиснутої клавіші в простій клавіатурі можна використовувати безпосереднє підключення клавіш до порту МК (рис. 6.8). Таке підключення клавіатури дозволяє визначити коди декількох одночасно натиснутих клавіш.

Код натиснутої клавіші одержуємо безпосереднім читанням значення порту:

```
MOV A, P1          ; читання коду натиснутих клавіш.
```

Використання порту для введення інформації можливе при запису одиниць в його регістри-фіксатори. Вхідні лінії порту при цьому підтягнуті до джерела живлення, і їх читання повертає значення логічної одиниці; при натисненні клавіші відповідний їй біт дорівнює 0.

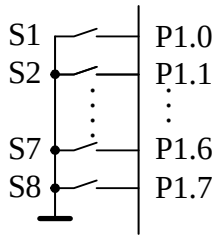


Рисунок 6.8 - Безпосереднє підключення клавіш до порту МК

Для подальшої обробки, часто зручніше використовувати код, при якому натиснутій клавіші відповідає біт, що дорівнює 1:

```
CPL A          ; перетворення коду.
```

Безпосереднє підключення клавіш до МК вимагає використання великої кількості його вхідних ліній. Для зменшення апаратних вимог, можуть використовуватися пріоритетні шифратори. Спрощену схему використання мікросхеми шифратора К555ИВ1 (аналог 74LS148) для зменшення кількості необхідних вхідних ліній порту наведено на рис. 6.9.

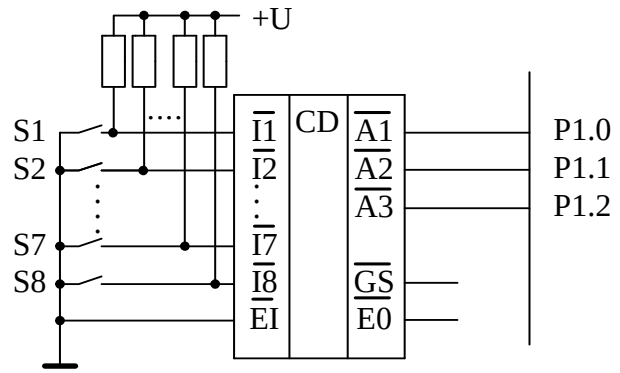


Рисунок 6.9 - Використання шифратора при підключенні клавіатури

Особливістю цього шифратора є пріоритетність – якщо натиснуто декілька клавіш, на виходах з'являється сигнал, який відповідає старшому номеру. У табл. 6.1 наведено таблицю істинності шифратора К555ИВ1.

Таблиця 6.1 - Таблиця істинності шифратора К555ИВ1

Вхід									Вихід				
^EI	^I1	^I2	^I3	^I4	^I5	^I6	^I7	^I8	^GS	^A0	^A1	^A2	^E0
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	x	x	x	x	x	x	x	0	0	0	0	0	1
0	x	x	x	x	x	x	0	1	0	1	0	0	1
0	x	x	x	x	x	0	1	1	0	0	1	0	1
0	x	x	x	x	0	1	1	1	0	1	1	0	1
0	x	x	x	0	1	1	1	1	0	0	0	1	1
0	x	x	0	1	1	1	1	1	0	1	0	1	1
0	x	0	1	1	1	1	1	1	0	0	1	1	1
0	0	1	1	1	1	1	1	1	0	1	1	1	1

Примітка. Символ ^ позначає інверсний характер змінної.

Незважаючи на можливість каскадного нарощування шифраторів, їх застосування обмежене. Для великих клавіатур використовується матрична схема включення вимикачів.

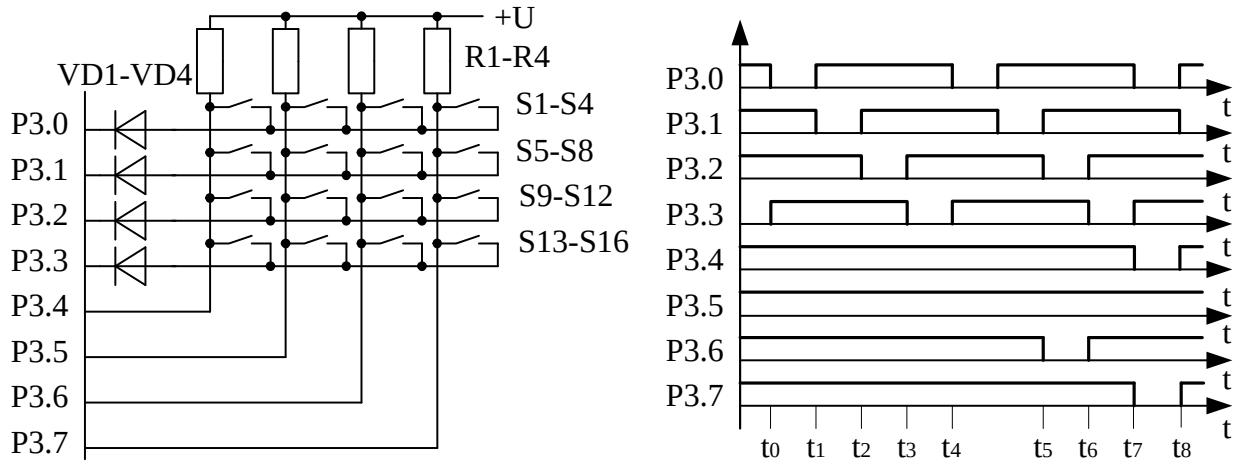
На рис. 6.10, а наведено типову схему включення клавіатури. Для отримання коду натиснутої клавіші використовується процедура сканування клавіатури, в якій біти P3.0 – P3.3 використовуються для формування сигналу, що сканує, а біти P3.4 – P3.7 – для читання коду натиснутої клавіші. Резистори R1 – R4 забезпечують рівень логічної одиниці, якщо жодна з клавіш не натиснута. Опір резисторів вибирається відповідно до навантажувальної здатності виводів порту. Для мікроконтролерів сімейства MCS-51 вико-

ристання резисторів R1 – R4 не обов'язкове, бо порти P1 – P3 мають у своєму складі джерела струму, що забезпечують струм до 60 мкА й еквівалентні резисторам з опором 80 кОм.

Сканування клавіатури полягає в послідовному формуванні на виводах порту P3.0 – P3.3 сигналу логічного нуля (рис. 6.10, б). За відсутності натиснутих клавіш читання старших бітів порту P3 дає число 0Fh (інтервал t0 – t4 на рис. 6.10, *б*). Припустимо, що у момент часу t4 буде натиснута клавіша S11, тоді на інтервалі t5 – t6 читання бітів P3.4 – P3.7 дасть число 1011b. При одночасному натисканні клавіш S1 і S4 читання бітів P3.4 – P3.7 на інтервалі t7 – t8 дасть число 0110b.

Для зменшення кількості виводів мікроконтролера, що використовуються для сканування рядків клавіатури, можливе застосування дешифратора. Це дозволяє, наприклад, проводити сканування 8 рядків при використанні трьох розрядів порту. Якщо сканування рядків проводиться логічною мікросхемою, необхідно використовувати діоди VD1 – VD4, що виключають перевантаження її вихідних каскадів при одночасному натисненні декількох клавіш в одному стовпці. Без них натискання клавіш, наприклад S1 і S5 при нульовій напрузі на виводі P3.1 і одиничній на P3.0, призводить до протікання струму з виводу P3.0 до P3.1. Сила струму обмежується тільки внутрішніми елементами мікросхеми. При скануванні рядків від мікроконтролера, виводи порту P3.0 – P3.3 працюють в режимі джерела струму і використання діодів не обов'язкове.

Процедура формування сигналу, що сканує, може бути виконана на підставі операції зсуву, встановлення/скидання біта або запису числа з таблиці.



а *б*

Рисунок 6.10 Отримання даних з клавіатури:
а - типова схема включення клавіатури; *б* - часові діаграми сигналів в схемі

Варіант програми сканування клавіатури на підставі операції зсуву. У регістрі R7 зберігається лічильник рядків матриці, R6 – поточний сигнал сканування.

```
SCAN_SHIFT:
        MOV R7,#4           ; завантаження лічильника сканування – 4 рядки
        MOV R6,#11111110b   ; завантаження початкового байта сканування
LOOP:
        MOV A,R6            ; виклик байта сканування
        MOV P3,A            ; запис байта сканування до порту
        RL A                ; зсув байта сканування
        MOV R6,A            ; збереження байта сканування
        MOV A,P3            ; читання значення порту P3
        ORL A,#00001111b    ; виділення значення стовпців
        CJNE A, #0FFh, KEY  ; якщо натиснута клавіша та перехід
                            ; на підпрограму обробки натиснення
        DJNZ R7,LOOP        ; перевірка закінчення сканування рядків
        AJMP SCAN_SHIFT     ; перехід на початок нового циклу сканування
        ; …. тут може бути текст програми
KEY:                        ; підпрограма обробки натиснення клавіші
                            ; у A інформація про натиснуті клавіші
                            ; у R7 дані про сканований рядок
```

Варіант програми сканування клавіатури з використанням встановлення/скидання біта. У регістрі R4 зберігається лічильник рядків матриці, у R3 – лічильник стовпців матриці.

```
SCAN_BIT:                   ; ініціалізація бітів сканування
        CLR P3.0            ; початкове включення рядка 1
SET_KEYBOARD:               ; формування сигналу, що сканує
        MOV R4,#0           ; скидання лічильника рядків
L1:     JB  P3.3, L2        ; перевірка сканування рядка 4
        SETB  P3.3          ; встановлення раніше скинутого біта
        CLR  P3.0           ; якщо сканувався рядок 4, сканувати 1
        AJMP  READ_         ; перехід на читання стовпців
L2:     INC  R4             ; збільшення лічильника рядків
        JB  P3.0, L3        ; перевірка сканування рядка 1
        SETB  P3.0          ; установка раніше скинутого біта
        CLR  P3.1           ; якщо сканувався рядок 1, сканувати 2
        AJMP  READ_         ; перехід на читання стовпців
L3:     INC  R4             ; збільшення лічильника рядків
        JB  P3.1, L4        ; перевірка сканування рядка 2
```

```
        SETB  P3.1              ; установка раніше скинутого біта
        CLR  P3.2               ; якщо сканувався рядок 2, сканувати 3
        AJMP  READ_             ; перехід на читання стовпців
L4:     INC  R4                 ; збільшення лічильника рядків
        JB  P3.2, L1            ; перевірка сканування рядка 3
        SETB  P3.2              ; установка раніше скинутого біта
        CLR  P3.3               ; якщо сканувався рядок 3, сканувати 4
        AJMP  READ_             ; перехід на читання стовпців
;
        ; …. тут може бути текст програми
;
READ_:                          ; у регістрі R4 - номер рядка матриці
        MOV  A, P3              ; у старшій тетраді акумулятора –
                                ; інформація про натиснуті клавіші
        MOV  R2, A              ; збереження інформації в тимчасовому регістрі
        ADD  A, #16             ; перевірка наявності нулів в старшій тетраді
        JC SET_KEYBOARD         ; клавіші не натиснуті – початок
                                ; нового сканування
; …. тут може бути текст процедури усунення деренчання контактів
        MOV  R3, #255           ; скидання лічильника стовпців
                                ; перший приріст регістра в циклі зсуву дасть
R3=0
        MOV  A, R2              ; відновлення даних
        SWAP A                  ; перенесення інформації про натиснуті клавіші
                                ; у молодшу тетраду
                                ; цикл SHIFT дозволяє визначити номер однієї
SHIFT:                          ; натиснутої клавіші
                                ; (номера першого біта, що дорівнює нулю)
        INC  R3                 ; приріст лічильника стовпців
        RRC  A                  ; зсув біта
        JC  SHIFT               ; повторити, якщо клавіша не натиснута
; у R3 номер натиснутої клавіші 0 – 3, у R4 номер рядка 0 – 3
        MOV  DPTR, #KEY_TABLE   ; передача адреси таблиці кодів клавіш
        MOV  B, #04             ; у B кількість клавіш в рядку
        MOV  A, R4              ; у A номер рядка
        MUL  AB                 ; у A кількість клавіш в рядках, що скановані
        MOV  B, R3              ; у B номер натиснутої клавіші в рядку
        ADD  A, B               ; у A номер клавіші з початку масиву
        MOVC  A,@A+ DPTR        ; у A код клавіші
```

```
        AJMP  SET_KEYBOARD      ; продовження сканування клавіатури
KEY_TABLE:                      ; таблиця кодів клавіш
DB    01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16
```

6.2.4. Підрахунок кількості імпульсів

Часто в керуючих програмах виникає необхідність очікування ряду подій, що подається послідовністю імпульсних сигналів від датчиків. Розглянемо дві типові процедури підрахунку числа імпульсів між двома подіями і підрахунок кількості імпульсів за заданий інтервал часу.

Підрахунок кількості імпульсів між двома подіями. Цю типову процедуру зручно проілюструвати на конкретному прикладі.

Припустимо, що необхідно підрахувати кількість деталей, що зійшли з конвеєра від моменту його включення до моменту вимикання. Факт сходу деталі з конвеєра фіксується фотоелементом, на виході якого формується імпульсний сигнал. Включення конвеєра супроводжується скиданням біта P3.4, проходження деталі – установленням/скиданням біта P3.5 (рис. 6.11).

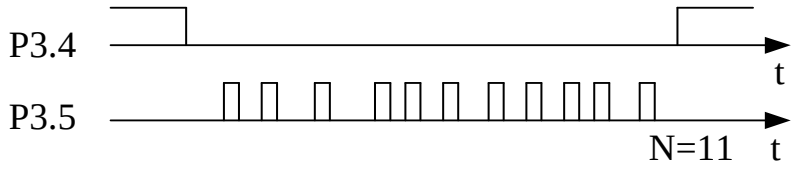


Рисунок 6.11 - Часова діаграма формування імпульсного сигналу

Програма має такий вигляд:

```
        MOV  TMOD, #01010000B   ; завдання лічильнику 1
                                ; режиму 16 бітного лічильника
        MOV  TL1, #0            ; скидання лічильника деталей
        MOV  TH1, #0            ;
WAIT0:  JB  P3.4, WAIT0         ; очікування вмикання конвеєра
        SETB TCON.6             ; пуск лічильника 1
WAIT1:  JNB  P3.4, WAIT1        ; очікування вимикання конвеєра
        CLR  TCON.6             ; зупинка лічильника 1
        MOV  A, TL1             ;
        MOV  B, TH1             ; у регістрах (B)(A) – кількість деталей
```

Підрахунок кількості імпульсів за заданий проміжок часу. При вирішенні задач виміру тривалості імпульсних сигналів, перетворення число-імпульсного коду в двійковий код, а також у ряді інших задач може виник-

нути необхідність підрахунку кількості імпульсів за заданий інтервал часу $t_0 – t_1$ (рис. 6.12).

Ця процедура може бути реалізована чотирма різними способами:

1) програмною реалізацією часового інтервалу і програмним підрахунком кількості імпульсів на вході МК;
2) програмною реалізацією часового інтервалу й апаратним підрахунком кількості імпульсів (внутрішнім лічильником);
3) апаратною реалізацією часового інтервалу (внутрішнім таймером) і програмним підрахунком кількості імпульсів;
4) апаратною реалізацією часового інтервалу (внутрішнім таймером) й апаратним підрахунком кількості імпульсів (внутрішнім лічильником).

Перший спосіб неефективний і значно складніший за інші, другий і третій також не повністю використовують можливості МК, а тому не розглядаються. Розглянемо програму, що реалізує четвертий спосіб і підраховує кількість імпульсів за 10 мс.

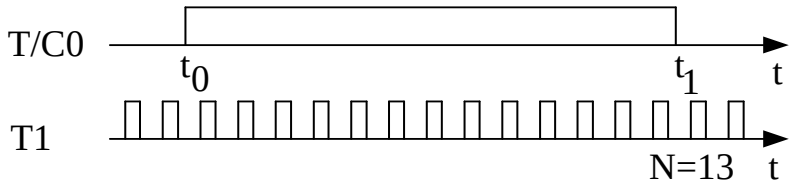


Рисунок 6.12 - Часова діаграма підрахунку кількості імпульсів за заданий інтервал часу

Таймер/лічильник 0 виконує відлік інтервалу часу (інтервал часу $t_0 _ t_1$, рис. 6.11), таймер/лічильник 1 рахує імпульси, що надходять на його лічильний вхід T1. Частота синхронізації 12 МГц.

```
TIME        EQU   ((-10000)+1)            ; константа для інтервалу часу 10мс
        MOV   TMOD, #01010001b    ; встановлення режимів
                                  ; таймерів/лічильників – 16 біт
                                  ; 1-й – лічильник, 0-й – таймер
        CLR   A                   ; скидання акумулятора
        MOV   TH1,  A             ; скидання таймера/лічильника 1
        MOV   TL1,  A             ;
        MOV   TH0,  #(TIME SHR 8) ; завантаження старшого байта
                                  ; інтервалу часу у старший байт таймера - TH0
        MOV   TL0,  #TIME         ; завантаження молодшого байту інтервалу
                                  ; часу у молодший байт таймера - TL0
        ORL   TCON,  #50h         ; одночасний пуск таймера та лічильника
WAIT:
        JBC   TCON.5, EXIT        ; перевірка переповнення таймера з скиданням
```

```
                                ; прапора переповнення
    SJMP   WAIT                 ; продовження циклу,
                                ; якщо інтервал часу не закінчився
EXIT:
    ANL   TCON,  #NOT(50h)      ; зупинка таймера та лічильника
    MOV   B,  TH1               ; у регістрах ( B)(A) – кількість імпульсів
    MOV   A,  TL1               ; за 10 мс
```

6.3. Виведення керуючих сигналів із МК

При використання мікроконтролерів для формування керуючих сигналів необхідно враховувати навантажувальну здатність виводів портів. Типові характеристики виробів фірми INTEL: струм, що втікає, для портів P1, P2, P3 не перевищує 1,6 мА, для порту P0 – 3,2 мА; струм, що витікає, для портів P1, P2, P3 не перевищує 60 мкА, для порту P0 – 800 мкА [17]. Тобто навантажувальна здатність виводів портів мікроконтролера відповідає підключенню одного або двох входів логічних мікросхем TTL серій К155 або 74XXX (XXX – відповідає номеру мікросхеми). Для підвищення кількості входів логічних елементів, що можна безпосередньо підключити до мікроконтролера, потрібно використовувати логічні мікросхеми сучасних серій з малими вхідними струмами: К555, К1531, К1533 або 74L, 74LS, 74ALS, 74HC та інші. Для узгодження рівня вихідного струму мікроконтролера з струмом керування біполярними транзисторами та з іншими пристроями, що споживають великий вхідний струм, необхідно використовувати буферні логічні елементи, такі, як інвертори, інвертори з відкритим колектором тощо.

Якщо порт P0 призначається для обміну даними з зовнішніми пристроями не в режимі формування адреси/даних, необхідно пам’ятати, що він переходить до режиму з відкритим колектором і для отримання сигналу логічної одиниці потребує використання резисторів, що підключаються між виводами порту і джерелом живлення мікроконтролера.

6.3.1, Формування статичних керуючих сигналів

Для керування виконавчим механізмом, що працює за принципом включено/виключено, на відповідній вихідній лінії порту МК необхідно сформувати статичний сигнал 0 чи 1. Ця дія реалізується командами встановлення/скидання необхідного біта. У випадку паралельного керування групою автономних виконавчих механізмів, підключених до вихідного порту, формується не двійковий керуючий вплив, а керуюче слово (КС), кожному розряду якого ставиться у відповідність 1 чи 0 залежно від того, які виконавчі механізми повинні бути задіяні.

Керуючі слова можна виводити в порт як командами пересилання даних, так і командами логічних операцій над змістом порту.

Команда ANL використовується для скидання тих бітів порту, що в операнді (масці) задані нулем.

Команда ORL використовується для установки бітів порту, що в операнді (масці) задані одиницею.

Командою XRL здійснюється інверсія бітів порту, що в операнді (масці) задані одиницею.

Для формування складних послідовностей керуючих слів зручно користуватися табличним способом, при якому всі можливі КС упаковані в таблицю, а прикладна програма МК обчислює адресу необхідного КС, вибирає її з таблиці та передає в порт виводу.

6.3.2. Формування імпульсних сигналів

Генерація періодичного керуючого впливу (меандру). Для генерації меандру з однаковою тривалістю імпульсу і паузи зручно скористатися процедурою видачі імпульсного керуючого впливу і підпрограмою реалізації часової затримки, яка дорівнює половині періоду сигналу, що формується.

```
M_2:    CPL   P1.3              ; інверсія вихідного сигналу
        ACALL DELAY_05_T        ; затримка на півперіод
        SJMP  M_2               ; повернення до початку процедури
```

Нескінченний періодичний сигнал формується на лінії 3 порту 1. На інших лініях порту сигнали залишаються незмінними.

Апаратна реалізація часової затримки в підпрограмі DELAY_05_T за допомогою таймера дозволить звільнити ресурси МК для виконання інших задач.

Формування меандру також можливе при переведенні таймера/лічильника в режим 2. Часова затримка визначається числом, що знаходиться в регістрі TH. Підпрограма обробки переривання в цьому випадку містить команди

```
CPL   P1.3
RETI
```

Формування аперіодичного керуючого сигналу. Послідовність імпульсних сигналів з довільною тривалістю може бути отримана аналогічним чином, тобто шляхом чергування процедур видачі змінюваного значення сигналу (0 чи 1) і формування часових затримок заданих інтервалів. Потрібні часові затримки обчислюються або вибираються з таблиці за номером інтервалу.

6.4. Реалізація функцій часу

6.4.1. Програмне формування часової затримки

Часова затримка малої тривалості. Процедура реалізації часової затримки використовує метод програмних циклів. При цьому в деякий робочий регістр завантажується число, яке потім у кожному проході циклу зменшується на 1. Так продовжується доти, поки вміст робочого регістра не буде дорівнювати нулю, що інтерпретується програмою як момент виходу з циклу.

Час затримки при цьому визначається числом, завантаженим у робочий регістр, і часом виконання команд, що утворюють програмний цикл. Схема алгоритму такої програми показана на рис. 6.13. Підпрограма має символічне ім’я DELAY.

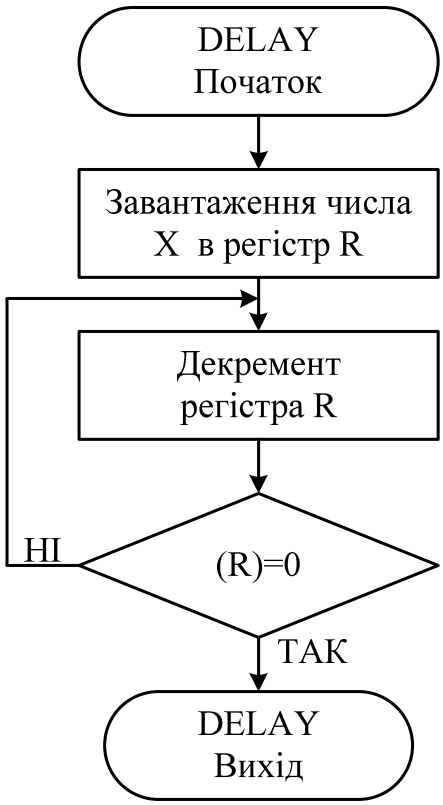


Рисунок 6.13 - Схема алгоритму формування часової затримки малої тривалості

Реалізація підпрограми затримки, яка використовує простий цикл з лічильником, що використовує регістр R7 з початковим значенням лічильника 28, має вигляд

```
DTM EQU  28          ; присвоєння символьному імені DTM значення 28
;          …         ; текст програми
      ACALL  ZATRYMKA_1
```

```
;       …                ; текст програми
ZATRYMKA_1:
        MOV R7, DTM      ; запис до лічильника початкового значення DTM
Z1:
;       …                ; тіло циклу
        DJNZ R7, Z1      ; останній оператор циклу затримки
;       ...              ; однократно виконувані команди
        RET              ; повернення
```

Час затримки, реалізований підпрограмою, що використовує алгоритм простого циклу:

$$t_з = t_{ов} + t_ц * DTM,$$

де $t_{ов}$ – час реалізації однократно виконуваних команд;

$t_ц$ – час реалізації команд, що складають тіло циклу;

DTM – кількість повторень циклу.

Мінімальна величина затримки при *DTM* = 1 визначається за формулою

$$t_{з\,min} = t_{ов} + t_ц.$$

Максимальна величина затримки визначається при $DTM = 2^r-1$, де *r* – розрядність програмного лічильника. У наведеному прикладі використовується восьми-розрядний лічильник – регістр R7, тому максимальне значення *DTM* дорівнює 255 і максимальна величина затримки

$$t_{з\,max} = t_{ов} + t_ц * 255.$$

Розрахунок підпрограми затримки часу виконується із визначенням часу виконання відомих команд підпрограми. Припустимо, що потрібна часова затримка $t_з = 200$ мкс. За умови, що машинний цикл мікроконтролера дорівнює одній мікросекунді, час виконання однократно виконуваних команд такий:

```
        ACALL ZATRYMKA_1     ; 2 цикли – 2 мкс
        MOV R3, DTM          ; 1 цикл – 1 мкс
        RET                  ; 2 цикли  – 2 мкс
```

Час виконання команд циклу:

```
        DJNZ R3, Z1          ; 2 цикли  – 2 мкс
```

Підставивши у формулу $t_з = t_{ов} + t_ц * DTM$ необхідні значення, одержуємо:

$$200 = 5 + 2\ DTM.$$

Визначаємо необхідну кількість виконання циклу:

$$DTM = (200 - 5)/\ 2 = 97{,}5.$$

При *DTM* = 97 довжина затримки 199 мкс, при *DTM* = 98 вона становить 201 мкс.

Для зменшення похибки довжини затримки необхідно додати до однократно виконуваних команд пусту операцію NOP, яка виконується за 1 мкс. Таким чином, з урахуванням часу виконання пустої команди NOP, число *DTM* становитиме: *DTM* = (200 – 6)/ 2 = 97.

Для динамічної зміни величини затримки кількість циклів *DTM* можна зберігати не в константі, а в комірці пам’яті чи в регістрі, що дозволяє використовувати одну підпрограму для формування різних часових затримок. Наприклад, для передачі параметра в підпрограму через комірку пам’яті можна скористатись таким прикладом:

```
DSEG    AT   40H   ; розмістити сегмент за адресою 40H у пам’яті даних
DTM     DS   1     ; зарезервувати один байт з символьним ім’ям DTM
CSEG    AT   0H    ; початок програми з нульової адреси ПЗП
;       …          ; текст програми
;в результаті обчислень, в акумуляторі знаходиться потрібна кількість циклів
        MOV  DTM, A    ; передача числа у комірку пам’яті
        ACALL  ZATRYMKA_1
;       …          ; текст програми
ZATRYMKA_1:
        MOV R7, DTM    ; запис до лічильника початкового значення DTM
;далі текст підпрограми без змін
```

Часова затримка великої тривалості. Програмна реалізація часової затримки великої тривалості можлива методом вкладених циклів. Схема алгоритму програмної реалізації затримки показана на рис. 6.14.

Загальний час затримки, внесений підпрограмою, що реалізує алгоритм із вкладеними циклами, визначається виразом

$$t_з = t_{ов} + (t_{ц1} + t_{ц2}*Y)X,$$

де $t_{ов}$ – час виконання однократно виконуваних команд;

$t_{ц1}$, $t_{ц2}$ – час виконання зовнішнього і внутрішнього циклів;

X, *Y* – початкові значення лічильників зовнішнього і внутрішнього циклів.

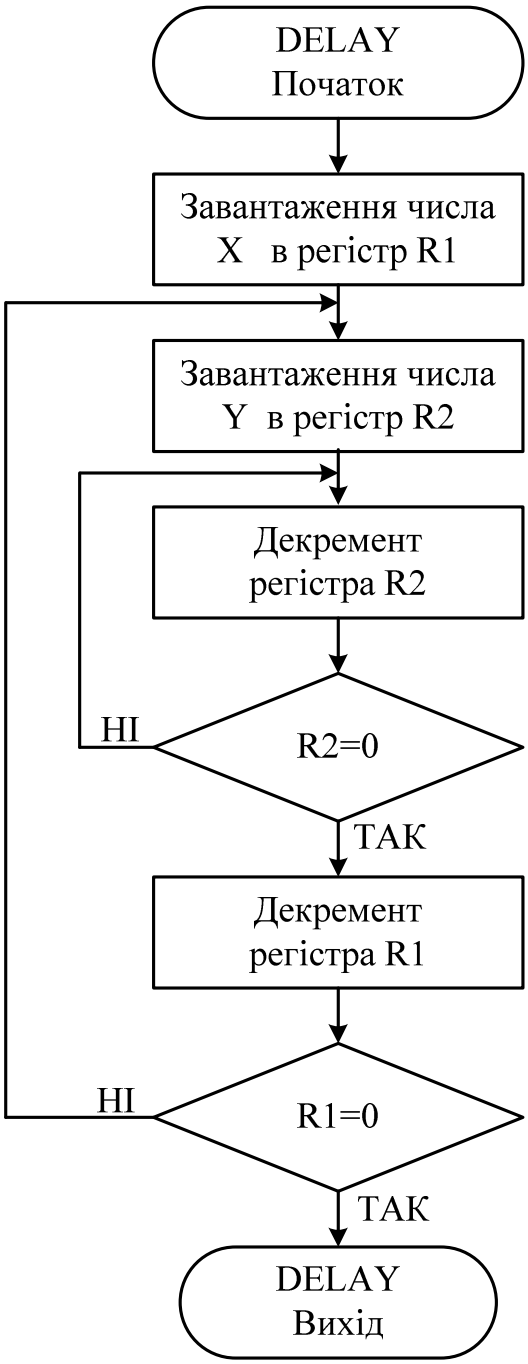


Рисунок 6.14 - Блок-схема підпрограми DELAY часової затримки великої тривалості

6.4.2. Формування часової затримки на основі таймерів

Затримка малої тривалості. Недоліком програмного способу реалізації часової затримки є нераціональне використання ресурсів мікроконтролера: під час формування затримки він практично простоює, оскільки не може вирішувати ніяких задач керування об’єктом. У той же час вбудовані апаратні засоби дозволяють реалізувати часові затримки на фоні роботи основної програми.

У мікроконтролерів сімейства MCS-51 на вхід таймера/лічильника можуть надходити сигнали синхронізації з частотою машинних циклів (режим таймера) чи сигнали від зовнішнього джерела імпульсів (режим лічильника). Обидва ці режими можуть бути використані для формування затримок, але частіше використовується режим таймера, бо режим лічильника вимагає зовнішнього джерела синхронізуючих імпульсів. Якщо використовувати 16-бітний режим таймера, то можна одержати часові інтервали в діапазоні від однієї до 65536 мкс при частоті кварцового резонатора 12 МГц.

Розглянемо організацію часової затримки тривалістю 50 мс на прикладі програми, яка виконує деякі дії, а потім на 50 мс переходить до режиму холостого ходу, що дозволяє зменшити споживання енергії з джерела живлення. Повертання з режиму холостого ходу відбувається по перериванню переповнення таймера/лічильника.

```
ORG 0
        AJMP   START            ; перехід на початкову адресу програми
ORG  0BH                        ; адреса вектора переривання від T/C0
        AJMP   Timer_50         ; перехід на підпрограму обробки переривання
ORG   100H                      ; початкова адреса програми
START:
        SETB   IE.1             ; дозвіл переривання від T/C0
        SETB   IE.7             ; загальний дозвіл переривань
        MOV   TMOD, #01H        ; настроювання T/C0 – таймер, режим 1
NEXT:
; . . .                         ; дії, що виконуються кожні 50 мс
        ACALL   Timer_50        ; виклик підпрограми обслуговування таймера
        ORL   PCON, #01b        ; переведення у режим холостого ходу
                                ; інструкцією ORL, бо біти
                                ; регістра PCON не адресуються.
                                ; у режимі холостого ходу знаходимось 50 мс
        AJMP   NEXT             ; перехід до виконання запланованих дій
Timer_50:                       ; підпрограма обробки переривання
                                ; переповнення таймера T/C0
        CLR   TR0                       ; зупинка T/C0
                                        ; завантаження таймера
        MOV   TL0, #(-50000)            ; молодший байт
        M0V   TH0, #(-50000 SHR 8)      ; старший байт
        SETB   TR0                      ; старт T/C0
        RETI                    ; вихід з підпрограми обробки переривання
```

Для запобігання формальному попередженню про несумісність форма-

тів 8-бітового регістра TL0 і 16-бітового числа (-50000), старша тетрада числа може бути встановлена рівною нулю (-50000 AND 0FFh), що переводить число до 8-бітового формату.

Якщо постійно виконуються деякі дії, а кожні 50 мс потрібно виконувати іншу роботу, то можна скористатись модифікованою програмою

```
ORG   0
      AJMP   Start
ORG   0BH                  ; адреса вектора переривання від T/C0
      AJMP   Timer         ;перехід на підпрограму обробки переривання
ORG   100H                 ; початкова адреса програми
Start:
      MOV   TMOD, #01H     ; настроювання T/C0 – таймер, режим 1
      SETB   IE.1          ; дозвіл переривання від T/C0
      SETB   IE.7          ; загальний дозвіл переривань
Timer_init:                ; ініціалізація таймера
      MOV   TL0, #(-50000 AND 0FFh)   ; молодший байт
      M0V   TH0, #(-50000 SHR 8)      ; старший байт
      SETB   TCON.4                   ; старт T/C0
NEXT:
;     . . .                           ; дії, що виконуються постійно
      AJMP   NEXT                     ; кінець циклу NEXT
Timer:
      CLR   TCON.4                    ; зупинка T/C0
;. . .                                 ; дії, що виконуються кожні 50 мс
                                      ; завантаження таймера
      MOV   TL0, #(-50000 AND 0FFh)   ; молодший байт
      M0V   TH0, #(-50000 SHR 8)      ; старший байт
      SETB   TCON.4                   ; старт T/C0
      RETI                 ; вихід з підпрограми обробки переривання
```

Перед входом до циклу NEXT потрібно ініціювати і запустити таймер, що виконують три рядки програми після мітки Timer_init. Замість них можна скористатись тією частиною програми, яка виконується при подальшій роботі – підпрограмою обслуговування переривання по переповненню таймера. Імітація переривання здійснюється програмним встановленням відповідного прапора – SETB TF0, що приводить до виклику підпрограми і до виконання всіх передбачених дій.

Наведений приклад настроювання таймера на формування інтервалу часу 50 мс дає інтервал 50009 мкс. Для формування точного інтервалу часу потрібно враховувати час виконання команд з моменту переповнення тай-

мера до початку здійснення запланованих дій. Корекція числа, що завантажується до таймера, важлива при формуванні інтервалу часу, який дорівнює одиницям або десяткам мікросекунд, коли похибка становить суттєву частину інтервалу часу. Мінімальний час інтервалу визначається часом виконання команд від старту таймера до початку виконання запланованих дій, його легко знайти, якщо до таймера записати число 0FFFFh і подивитись на отриманий результат.

6.4.3. Вимір часових інтервалів

У задачах керування виникає необхідність вимірювання інтервалів часу між двома подіями. Звичайно події в об'єкті управління фіксуються сигналами від двійкових датчиків. Вважаючи подіями фронт і спад імпульсу, можна визначати часові характеристики імпульсних сигналів: тривалість, період і коефіцієнт заповнення (duty cycle). Крім того, можна визначати швидкість переміщення рухомого органа об'єкта по ділянці заданої довжини.

Найпростішим способом вимірювання тривалості імпульсу є програмний, але він вимагає застосування всіх ресурсів мікроконтролера, бо без цього стає неточним.

Для вимірювання тривалості сигналу може бути використаний таймер. Вимірюваний сигнал можна, наприклад, подавати на вхід INT0, вимірювання тривалості при цьому буде виконуватися в T/C0. Фрагмент програми, що вимірює тривалість "позитивного" імпульсу, буде виглядати так:

```
        MOV  TMOD, #00001001B    ; настроювання T/C0 як таймера
                                 ; у режимі 1 із зовнішнім керуванням
        MOV  TH0, #0             ; скидання таймера
        MOV  TL0, #0             ;
        SETB  TCON.4             ; старт T/C0
WAIT1:
        JNB  P3.2, WAIT1         ; очікування 1
WAIT0:
        JB  P3.2, WAIT0          ; очікування 0
        CLR  TCON.4              ; стоп T/C0
```

Управління повинне бути передане програмі за умови, що вимірюваний сигнал має низький рівень. Переривання від T/C0 і зовнішнє переривання по входу INT0 повинні бути заборонені. Після завершення програми у регістрах таймера T/C0 – TH0, TL0 знаходиться число, пропорційне тривалості "позитивного" імпульсу на вході INT0. Коефіцієнт пропорційності – тривалість машинного циклу. Якщо частота синхронізації мікроконтролера дорівнює 12 МГц, то верхня межа вимірювання дорівнює 65536 мкс з максимальною

похибкою 1 мкс. Цикли WAIT1 і WAIT0 відіграють службову роль – не дають пропустити потрібний сигнал і не впливають на точність вимірів, бо таймер керується апаратно.

При необхідності вимірювання часових інтервалів більшої тривалості можна програмним способом підраховувати кількість переповнень таймера.

6.4.4. Асинхронний обмін даними

Робота сучасних систем управління та обробки інформації неможлива без передачі та отримання даних. Передачу даних між двома мікроконтролерами або між мікроконтролерною системою та персональним комп'ютером можна здійснити за допомогою послідовного інтерфейсу, що вимагає мінімальної кількості з'єднувальних провідників та забезпечує передачу даних на великі відстані. Мікроконтролери сімейства MCS-51 мають змогу передавати і приймати дані за асинхронним інтерфейсом, який за логікою роботи збігається з інтерфейсом RS232, що є одним з індустріальних стандартів.

Характерною рисою послідовного асинхронного інтерфейсу є необхідність забезпечення збігу частот приймача і передавача з похибкою не більше 5%. Виконати цю вимогу можна або попереднім розрахунком параметрів настройки таймера/лічильника 1, який управляє частотою прийому та передачі даних, за формулами розділу 3.2.9.2, або скориставшись даними табл. 6.2.

Таблиця 6.2 - Параметри настройки таймера/лічильника 1 для управління частотою роботи приймача-передавача.

Частота прийому/передачі (BAUD RATE)	Частота кварцового резонатора, МГц	Таймер/лічильник 1			
		SMOD	C/T	Режим (MODE)	Число, що перезавантажується
Режим 0, максимум: 1000000	12	X	X	X	X
Режим 2, максимум: 375000	12	1	X	X	X
Режим 1, 3: 622000	12	1	0	2	0FFH
19200	11,0592	1	0	2	0FDH
9600	11,0592	0	0	2	0FDH
4800	11,0592	0	0	2	0FAH
2400	11,0592	0	0	2	0F4H
1200	11,0592	0	0	2	0F4H

У таблиці наведено параметри настройки для стандартних частот обміну даними при використанні кварцових резонаторів з рекомендованою частотою. Для нових мікроконтролерів можуть використовуватись кварцові резонатори з частотами 14,7456 МГц, 18,4320 МГц і 22,1184 МГц, які дозволяють підвищити швидкість обміну даними.

Передача даних відбувається через вивід TxD (P3.1) мікроконтролера, який повинен бути з'єднаний з виводом RxD віддаленого пристрою і навпаки, вивід TxD віддаленого пристрою повинен з'єднуватись з виводом RxD (P3.0) мікроконтролера. Пряме з'єднання використовується при передачі інформації в межах одного блока. Передача інформації між різними блоками або віддаленими пристроями відбувається з використанням фізичних інтерфейсів RS232, петля струму 20мА та ін. Перетворення сигналів відбувається за допомогою схем перетворення рівня, спеціалізованих конверторів, наприклад MAX232, оптичних ізоляторів сигналів.

Прийом і передача даних потребує постійного контролю за станом приймача-передавача послідовного асинхронного інтерфейсу. Це необхідно для того, щоб знати, чи закінчив він передачу символу та чи одержав він новий символ. Індикація кінця передачі та прийому символу здійснюється апаратним встановленням прапорів TI та RI у регістрі SCON. Якщо дозволено відповідне переривання, то мікроконтролер передає керування на адресу вектора переривання від послідовного інтерфейсу. При обробці переривання потрібно встановити, що його викликало, скинути відповідний біт і виконати заплановані дії. Цей спосіб відволікає МК від виконання програми тільки в момент закінчення прийому чи передачі символу. Інший шлях управління обміну даними передбачає програмний контроль за прапорами TI та RI. Це вимагає всього процесорного часу і унеможливлює дуплексний режим роботи інтерфейсу. Приклад програми, яка обмінюється даними з віддаленим об'єктом по послідовному інтерфейсу з програмним контролем прапорів TI та RI, наведено нижче.

```
BaudRate  EQU  0FDh   ;число, що перезавантажується. 9600 baud 11.0592
МГц
ORG   0                                  ; початок програми з адреси 0
AJMP   Start                             ;
ORG   20h                                ; з адреси 20h
Message1:                                ; записано текст повідомлення
Message1
DB   'S','y','m','b','o','l',' ','A', 00h
Start:
      ACALL   S_INIT                     ; ініціалізація УСАПП
```

```
Read_Char:                              ;
        ACALL   CHAR_IN                 ; читання символу з інтерфейсу
        CJNE    A, #'A', Read_Char      ; перевірка значення символу
        MOV     R0, #Message1           ; якщо отримано символ 'A'
        MOV     R1, #(Message1 SHR 8)   ; передається повідомлення
                                        ; Message1
        ACALL   STR_OUT                 ;
; подальший текст програми
S_INIT:
; підпрограма ініціалізації УСАПП
        CLR     TR1                     ; зупинка таймера 1
        MOV     SCON, #01011010B        ; встановлення режиму 1 УСАПП,
                                        ; прийом дозволено,
                                        ; прапор TI показує, що передавач вільний
        MOV     TMOD, #00100000B        ; таймер 1 у режимі 2
        ANL     PCON, #NOT(SMOD)        ; скидання біта подвійної швидкості
        MOV     TH1, #BaudRate          ; завантаження числа, що задає швидкість
        SETB    TR1                     ; пуск таймера
        RET                             ; повернення до основної програми
CHAR_IN:
; підпрограма читання символу очікує надходження нового символу
; і передає його в акумулятор
        JNB     RI, $           ; очікування приймання символу
        MOV     A, SBUF         ; запис символу в акумулятор
        CLR     RI              ; скидання прапора закінчення приймання
        RET                     ; повернення до основної програми
CHAR_OUT:
; підпрограма передачі символу - передає символ з акумулятора до УСАПП
        JNB     TI, $           ; очікування закінчення передачі
                                ; попереднього символу
        CLR     TI              ; скидання прапора закінчення передачі
        MOV     SBUF, A         ; запис символу для передачі
        RET                     ; повернення до основної програми
STR_OUT:
; підпрограма передачі рядка символів, що знаходиться у пам'яті програм і
; завершується символом 00h – нуль термінований рядок
; адреса рядка передається через регістри R0, R1
        MOV     DPH, R1         ; передача адреси рядка
        MOV     DPL, R0
```

```
STR_OUT_1:
        CLR   A                 ;
        MOVC   A, @A+DPTR ; читання наступного байта
        INC   DPTR              ; перехід до наступного байта
        JZ   STR_OUT_2          ; закінчення, якщо останній символ
        CALL   CHAR_OUT     ; передача символу
        SJMP   STR_OUT_1     ; очікування закінчення передачі
STR_OUT_2:
        RET                     ; повернення до основної програми
```

Програма починається з завдання режимів роботи УСАПП і переходить до читання символів, що передаються по послідовному інтерфейсу. При отриманні символу “A”, віддаленому об’єкту передається повідомлення “Symbol A”, і програма переходить до виконання подальших інструкцій. Використання нульового символу для індикації кінця рядка широко використовується при програмуванні і дозволяє легко знаходити закінчення рядків символів. Від програмного контролю за прапорами стану інтерфейсу легко перейти до роботи з перериваннями.

6.5. Аналогово-цифрове перетворення сигналів

Відсутність в базовому мікроконтролері вбудованих засобів обробки аналогових сигналів привела до появи великої кількості схемних рішень, які використовують зовнішні апаратні засоби. Зосередимось на аналого-цифрових та цифро-аналогових перетворювачах (ЦАП та АЦП). Більшість схем і алгоритмів обробки сигналів подібні, тому розглянемо рішення, що мають найбільш загальні риси.

6.5.1. Інтегруючий АЦП

Простіший спосіб перетворення значення напруги аналогового сигналу в цифрову форму полягає в послідовному перетворенні: напруга - довжина імпульсу - число. Встановлення взаємозв’язку між напругою і довжиною імпульсу легко вирішується засобами аналогової схемотехніки, а вимір часових параметрів імпульсу – одне з найпростіших завдань цифрової. Приклад схемного рішення інтегруючого АЦП наведено на рис. 6.15, часові діаграми роботи - на рис. 6.16.

Вхідна напруга U_2 подається на неінвертувальний інтегратор, на операційному підсилювачі DA1. Інтегруючий конденсатор C1 може розряджатись транзистором VT1, що керується мікроконтролером DD1. Рівність вхідної

напруги U_1 й вихідної напруги інтегратора U_3 визначається компаратором DA2. Вихідний сигнал компаратора надходить на мікроконтролер DD1.

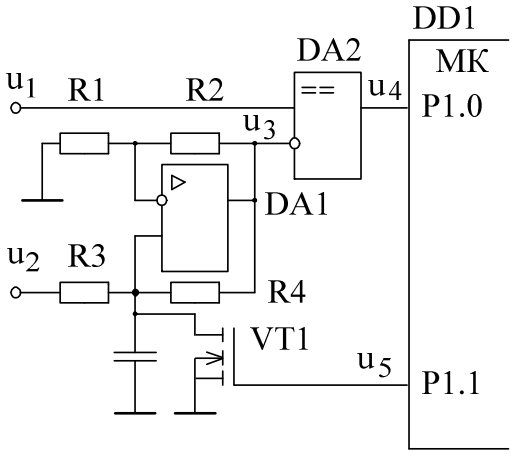


Рисунок 6.15 - Схема АЦП

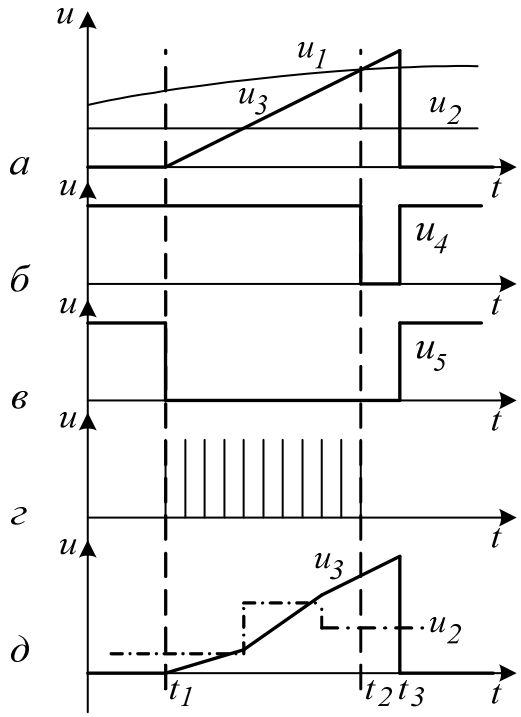


Рисунок 6.16 - Часові діаграми роботи АЦП

Можливі два варіанти роботи цього пристрою:

- напруга u_2 стабільна і не змінюється в часі, вона використовується як опорна напруга (reference voltage);
- як опорна напруга використовується напруга u_1.

У початковому стані схеми конденсатор С1 розряджений. На початку перетворення (момент t_1 на рис. 6.16) транзистор VT1 виключається (рис. 6.16, в) і інтегратор починає інтегрувати опорну напругу, на його виході – лінійно зростаюча напруга u_3 (рис. 6.16, а). Одночасно мікроконтролер починає відлік часу (рис. 6.16, г). В момент часу t_2 напруги u_3 і u_1 зрівнюються, на виході компаратора з'являється сигнал (рис. 6.16, б), у відповідь на який мікроконтролер припиняє відлік часу. Включення транзистора VT1 (момент t_3 на рис. 6.16), що розряджає конденсатор, відбувається після виконання невідкладних дій. Цей метод дозволяє вимірювати миттєве значення напруги, що подається на пристрій: $u_1=2/(RC)*u_2*dt$, за умови, що опір всіх резисторів дорівнює R, а ємність конденсатора – C. Переходячи до параметрів мікропроцесорної системи, маємо

$$u_1=2/(RC)*u_2*N/f,$$

де N – вміст лічильника у момент часу t_2, f – частота рахування лічильника.

Або

$$u_1 = k * N,$$

де $k = 2u_2/(RCf)$.

Вибираючи значення елементів схеми, можна коефіцієнт пропорційності k звести до вигляду, що полегшує обчислення, – наприклад, до степеня числа 2. Програмна реалізація інтегруючого АЦП зводиться до виміру інтервалу часу між двома подіями і подальшим обчисленням добутку коефіцієнта пропорційності k та вмісту лічильника.

У випадку використання напруги u_1, як опорної, принцип роботи схеми не змінюється, але вимірюється вже не миттєве, а середнє значення вхідної напруги. Відповідні діаграми наведено на рис. 6.16, д:

$$u_3 = \frac{2}{RC}\int_{t1}^{t2} u_2(t)dt\,.$$

Переходячи до середніх значень і параметрів мікропроцесорної системи, отримуємо

$$U_2 = u_1 RC\, f\,/(2N).$$

Можливість виміру середніх значень напруг - дуже важлива особливість інтегруючого АЦП, що обумовлює його використання навіть за наявності вбудованих аналого-цифрових перетворювачів. Крім того, інтегруючі АЦП використовуються при необхідності забезпечити високу розрізняльну здатність перетворювача і рівномірний розподіл кроків квантування рівня, у перетворювачах інтервалу часу в амплітуду сигналу та ін. Однак розглянутий метод ставить жорсткі вимоги до стабільності і точності елементів схеми, насамперед до конденсатора і компаратора. Підвищення точності і зниження вимог до елементів можна домогтися, використовуючи АЦП, що працює за методом подвійного інтегрування.

6.5.2. Метод подвійного інтегрування

Метод знайшов широке використання в некритичних до швидкодії пристроях з розрізняльною здатністю до 18 біт, має підвищену завадостійкість, не ставить жорстких умов до елементів схеми. Ідея методу полягає в послідовному інтегруванні напруги, що вимірюється, та опорної напруги тими самими апаратними засобами. Зміна параметрів схеми вносить відхилення в обидва виміри, залишаючи незмінними відносні показники, що забезпечує високу точність. Схема АЦП з подвійним інтегруванням [20] показана на рис. 6.17.

Спочатку на вхід інтегратора подається позитивна напруга $u_{оп}$. При цьому на виході інтегратора через якийсь час установиться негативний рівень, а на виході компаратора буде сформований сигнал 0. Цей стан є почат-

ковим для процесу вимірювання і використовується для підвищення точності визначення моменту початку відліку часу при переході сигналу через нульове значення.

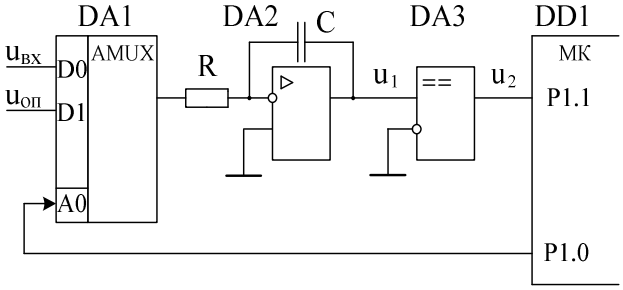


Рисунок 6.17 - Схема АЦП з подвійним інтегруванням

Вхідний сигнал, що вимірюється (для цієї схеми), повинен мати негативну напругу. Процес перетворення складається з двох етапів. Спочатку відбувається інтегрування вхідної напруги $u_{вх}$ протягом визначеного часу *T1* (рис. 6.18, а).

Відлік інтервалу *T1* здійснюється від моменту t_0 переходу напруги на виході інтегратора через нуль. Після закінчення інтервалу *T1* у момент часу t_1 на вхід інтегратора подається опорна напруга $u_{оп}$ (протилежної полярності) і вимірюється час інтегрування *T2*, який пройде до зменшення напруги інтегратора до нульового рівня. Враховуючи, що обидві напруги інтегруються одним інтегратором, справедливе відношення для середніх значень напруг $U_{вх} * T1 = U_{оп} * T2$, тобто $U_{вх} = U_{оп} * T2 / T1$.

Час першого інтегрування *T1* вибирається так, щоб при максимальній вхідній напрузі інтегратор не увійшов у насичення і щоб інтервал *T1* був пропорційним цілій кількості періодів частоти мережної напруги, що зменшує вплив мережних наведень на результати вимірів. З точки зору програмування, діапазон зміни вхідної напруги доцільно обмежити значенням опорної напруги ($U_{вх\ макс} \leq - U_{оп}$), що спрощує обчислення.

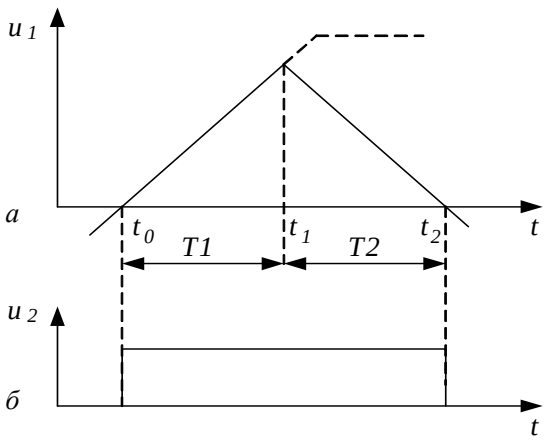


Рисунок 6.18 - Часова діаграма перетворення вхідного сигналу

Отримати значення T2 при відомому T1 можна за допомогою такого фрагменту коду:

```
        MOV  TMOD, #01H          ; настрювання T/C0 – таймер, 16 бітів
        MOV  TH0, #((-T1 + 1) SHR 8); завантаження T/C0
        MOV  TL0, #(-T1 + 1)     ;
        SETB  P1.1               ; настрювання P1.1 на ввід інформації
        SETB  P1.0               ; вибір каналу D1 мультиплексора DA1
                                 ; подача u_оп на вхід інтегратора
WAIT:
        JB P1.1, WAIT            ; очікування появи на виході інтегратора
                                 ; негативного рівня
        CLR  P1.0                ; вибір каналу D0 мультиплексора DA1
                                 ; подача u_вх на вхід інтегратора
WAITT0:
        JNB  P1.1, WAITT0        ; очікування моменту t0
        SETB TR0                 ; старт T/C0
WAITT1:
        JNB  TF0, WAITT1         ; очікування моменту t1
                                 ; – переповнення T/C0
        SETB  P1.0               ; вибір каналу D1 мультиплексора DA1
                                 ; подача u_оп на вхід інтегратора
WAITT2:
        JB  P1.1, WAITT2         ; очікування моменту t2 u1<0, u2=0
        CLR  TR0                 ; стоп T/C0
        CLR  TF0                 ; скидання прапора TF0
        MOV  B, TH0              ; формування результату T2 в
        MOV  A, TL0              ; регістровій парі (B)(A)
```

Програма дозволяє сформувати 16-бітовий код, який еквівалентний вхідному сигналу в діапазоні, наприклад, від 0 до –10 В, тобто забезпечує високу точність перетворення. Максимальний час (T1 + T2) перетворення (при $U_{вх} = U_{вх\ макс}$) складає 2 × 65535 мкс. Виходячи з цього часу, підбираються значення R і C. Якщо така висока точність перетворення не потрібна, то можна використовувати T/C0 у режимі 8-бітового таймера. При цьому максимальний час перетворення зменшується.

6.5.3. АЦП з паралельним інтерфейсом

Використання зовнішніх, по відношенню до мікроконтролера, АЦП у вигляді інтегральних мікросхем дозволяє розширити круг вирішуваних мікроконтролерами завдань. До переваг зовнішніх АЦП слід віднести:

- розширення функціональності МК системи при відсутності вбудованого АЦП;
- кращі, як правило, характеристики, ніж у вбудованих АЦП;
- широкий вибір АЦП з різними функціональними можливостями;
- можливість вибору АЦП, що найкраще відповідає вирішуваній проблемі.
- До недоліків зовнішніх АЦП відносяться:
- необхідність виділення ліній передачі даних для зв'язку з МК;
- потреба вивчення алгоритмів їх функціонування й написання програми обслуговування АЦП для отримання даних і їх передачі;
- додаткові площі на друкованій платі, ускладнення її розробки, підвищення вартості виробу.
- АЦП з паралельним інтерфейсом є найбільш поширеними та використовуваними. Загальними рисами, що описують функціонування АЦП, можуть вважатись:
- наявність сигналу «Початок перетворення», який ініціює процес перетворення аналогового значення вхідної напруги в цифрову форму;
- існування «часу перетворення», протягом якого провадиться перетворення сигналів;
- присутність сигналу «Кінець перетворення», котрий сигналізує про закінчення перетворення сигналів і можливість отримати результат.

Розглянемо взаємодію мікроконтролера з 8-бітовим АЦП ADC0801 фірми National Semiconductor [21]. Схему підключення АЦП до мікроконтролера наведено на рис. 6.19, а часові діаграми роботи – на рис. 6.20.

Мікросхема ADC0801 має керуючі входи вибору кристалу (^CS), читання (^RD) та запису (^WR). Сигнал вибору кристалу дозволяє мікросхемі АЦП приймати та передавати дані. Після встановлення сигналу ^CS, сигнал запису ініціює процес перетворення. Про закінчення перетворення сповіщає нульове значення сигналу ^INTR. Отримання даних відбувається після встановлення сигналу читання, який виводить вихідні лінії мікросхеми з третього стану і скидає сигнал ^INTR. Якщо дані не були прочитані, то сигнал ^INTR зберігає нульове значення. Для початкового скидання сигналу кінця перетворення (^INTR) рекомендується починати перше звернення до АЦП з видачі сигналу читання. Мікросхема ADC0801 може використовувати зовнішнє джерело синхронізації, або вбудований генератор з зовнішньою RC-ланкою. Передбачено можливість коригувати значення опорної напруги, вимірювати диференційні сигнали.

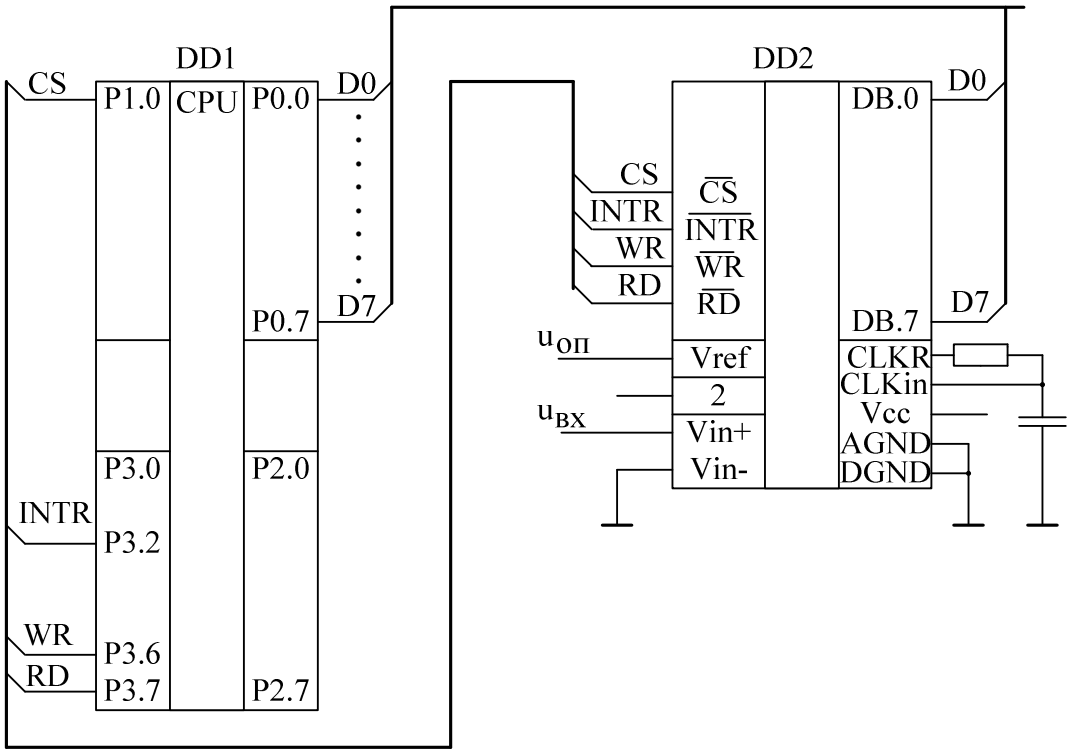
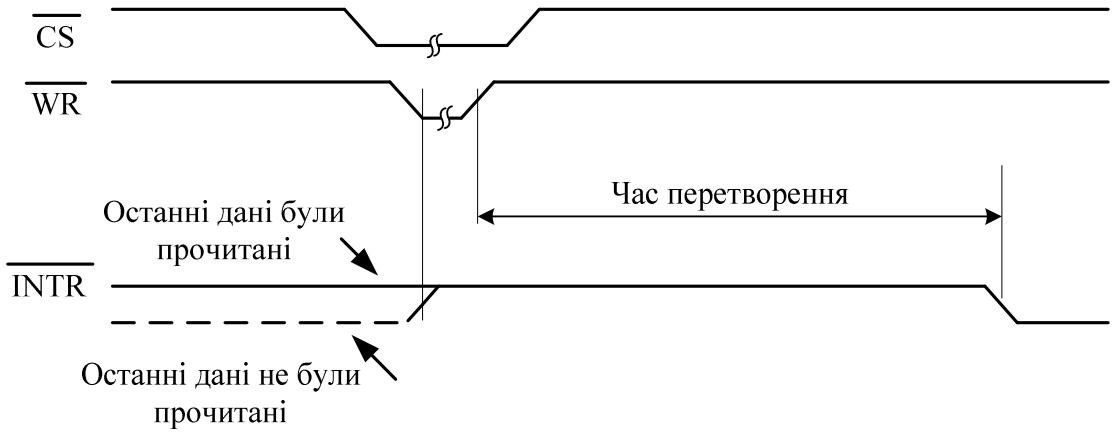
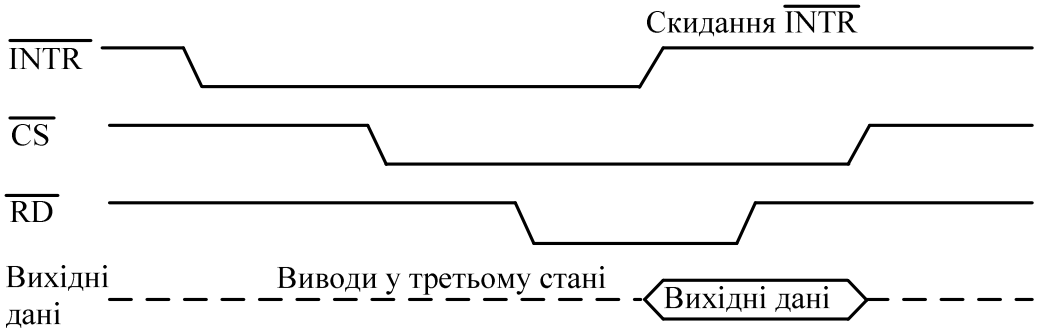


Рисунок 6.19 - Схема підключення АЦП до мікроконтролера



а



б

Рисунок 6.20 - Часові діаграми роботи АЦП
а - початок перетворення; *б* - одержання даних

Приклад програмної реалізації отримання і запису до пам’яті 16 послідовних вимірів, що виконані з максимальною швидкодією АЦП, наведено нижче.

```
ORG 0                ; запис команд починається з нульової адреси
     JMP START       ; перехід до початку програми
ORG 03h              ; адреса обробника зовнішнього переривання INT0
     JMP INT_0       ; перехід до підпрограми обробки переривання
ORG 30H              ; з адреси 30H починається програма
START:
     MOV R0, #20h    ; встановлення адреси записуваних даних
     MOV R1, #0FFh   ; встановлення допоміжної адреси
     MOV R2, #10h    ; ініціалізація лічильника на 16 (записаних даних)
                     ; перше звертання до АЦП -
                     ; скидання сигналу ^INTR
     ANL P1, #0FEh   ; встановлення сигналу ^CS
     MOVX A, @R1     ; формування імпульсного сигналу ^RD
     ORL P1, #1      ; скидання сигналу ^CS
D_16:                ;
     MOV A, #0FFh    ;
     ANL P1, #0FEh   ; встановлення сигналу ^CS
                     ; початок перетворення
     MOVX @R1, A     ; формування імпульсного сигналу ^WR
     SETB IE.0       ; дозвіл переривання INT0
     SETB IE.7       ; дозвіл усіх переривань
WAIT:                ; цикл очікування переривання INT0
     JNZ WAIT        ; цикл виконується, якщо в акумуляторі не нуль
     DJNZ R2, D_16   ; зменшення значення лічильника записаних даних
     NOP             ; подальший текст програми, який виконується
     NOP             ; після запису 16 байт даних
;                    ;
INT_0:               ;
     MOVX A, @R1     ; формування імпульсного сигналу ^RD,
                     ; читання даних
     MOV @R0, A      ; запис прочитаних даних до пам’яті
     INC R0          ; збільшення адреси – наступна комірка пам’яті
     ORL P1, #1      ; скидання сигналу ^CS
     CLR A           ; скидання акумулятора
     RETI            ; повернення до основної програми
END                  ; кінець програми
```

Підключення АЦП до мікроконтролера використовує шину і сигнали керування зовнішньої пам’яті даних. Це дозволяє спростити програму: формування імпульсних сигналів ^WR і ^RD відбувається автоматично при виконанні команд MOVX @R1, A і MOVX A, @R1; при виконанні останньої команди вихідні дані з АЦП, що потрапляють на виводи порту P0, передаються до акумулятора. В прикладі не застосовувалась зовнішня пам’ять даних, що дало можливість використати довільну адресу (допоміжна адреса у R1), за якою записувались та читались дані з АЦП, і підтримувати сигнал ^CS активним на протязі всього циклу перетворення. Одночасне використання АЦП і зовнішньої пам’яті даних приведе до необхідності встановлення адреси АЦП з використанням відповідного дешифратора.

Реалізація циклу очікування готовності даних з контролем за станом акумулятора демонструє, як можна керувати виконанням циклу – нескінченний цикл очікування розривається при скиданні акумулятора, і після обробки переривання програма виходить з циклу.

6.5.4. АЦП з послідовним інтерфейсом

АЦП цього типу використовуються у випадках, коли необхідно зменшити кількість ліній зв’язку між мікроконтролером і периферійними приладами.

Розглянемо взаємодію мікроконтролера з 8-бітовим АЦП ADC0831 фірми National Semiconductor [22]. Схему підключення АЦП до мікроконтролера наведено на рис. 6.21, а часові діаграми роботи – на рис. 6.22. Мікросхема має керуючі входи вибору кристалу (^CS), вихідних даних (DO) та синхронізації (CLK). Передбачено можливість коригувати значення опорної напруги, вимірювати диференційні сигнали. Перетворення аналогового значення вхідної напруги в цифрову форму відбувається одночасно з передачею даних.

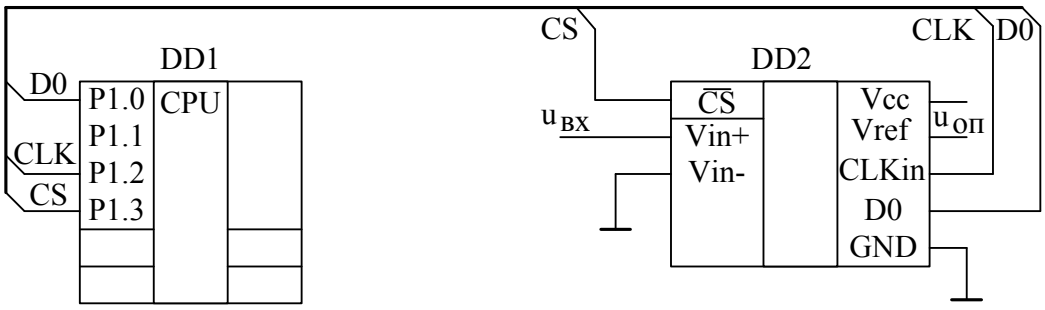


Рисунок 6.21 - Схема підключення АЦП до мікроконтролера

Початок перетворення ініціюється по спаду першого після вибору кристала синхронізуючого імпульсу (рис. 6.22, а). Біти даних видаються АЦП, починаючи зі старшого (MSB - most significant bit) і закінчуючи молодшим (LSB - least significant bit). Встановлення біта даних відбувається за спадом синхронізуючого імпульсу (рис. 6.22, б). Після передачі восьми бітів даних лінія даних встановлюється в стан логічного нуля. Зняття сигналу «вибір кристалу» призводить до встановлення на лінії даних третього стану.

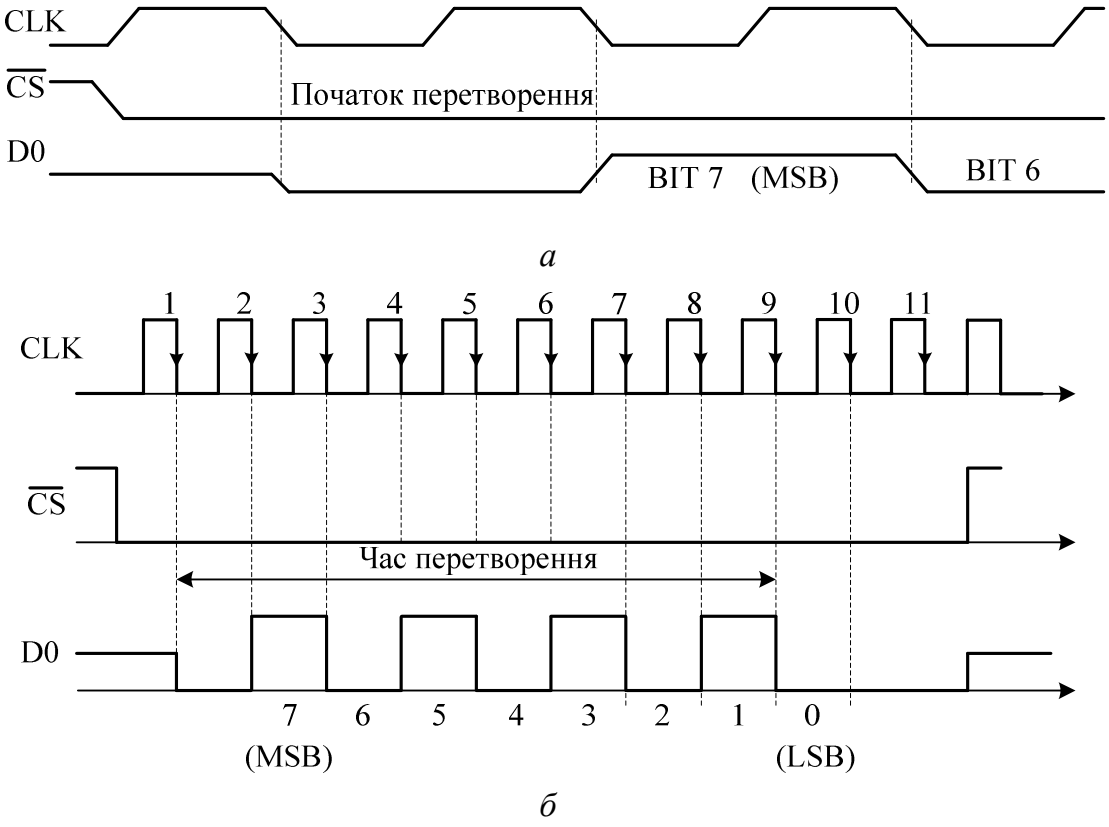


Рисунок 6.22 - Часові діаграми роботи АЦП
а - початок перетворення; *б* - передача даних (число 10101010b)

Приклад програмної реалізації отримання від АЦП байта даних наведено нижче. Отриманий байт поміщується в регістр R1.

```
ORG 0                   ; запис команд починається з нульової адреси
    JMP START           ; перехід до початку програми
ORG 30H
START:                  ; початок програми з адреси 30H
    MOV R1, #0          ; скидання значення байта даних
    ANL P1,#11111011B   ; встановлення початкового стану CLK=0
    ANL P1, #11110111B  ; вибір кристала ^CS=0
    CALL PULSE          ; імпульс синхронізації – початок перетворення
    MOV B, #8           ; ініціалізація лічильника на 8
```

```
R_BIT:                  ; цикл прийому 8 бітів
        CALL PULSE      ; імпульс синхронізації
        MOV A, P1       ; читання значення порту. У P1.0 біт даних
        RRC A           ; передача біта даних у біт переносу
        MOV A, R1       ; завантаження збережених даних
        RLC A           ; об'єднання прийнятого біта зі збереженими даними
        MOV R1,A        ; збереження поточних даних
        DJNZ B, R_BIT   ; повторити прийом біта, якщо їх менше 8
                        ;
        ORL P1, #00001000B      ; скидання вибору кристала ^CS=1
        NOP                     ; текст програми, що виконується після
                                ; прийому байта даних з АЦП
                                ; прийнятий байт міститься в R1
PULSE:                  ; підпрограма формування
                        ; імпульсу синхронізації
        ORL P1,#04      ; встановлення сигналу синхронізації
        NOP             ; формування імпульсу заданої тривалості
        ANL P1,#0FBh    ; скидання сигналу синхронізації
        RET             ; повернення до основної програми
END                     ; кінець програми
```

Програма повторює алгоритм роботи АЦП, що наведено на рис. 6.22

6.6. Управління пристроями індикації

Одними із часто використовуваних периферійних пристроїв, що динамічно розвиваються і підключаються до мікроконтролерних систем, є пристрої індикації. Вони використовуються на всіх етапах розробки й експлуатації МК системи. На етапі налагодження програмного забезпечення на пристрої індикації виводиться інформація про вміст робочих регістрів і комірок пам'яті, на етапі тестування готових виробів - інформація про проходження тестів і знайдених несправностях, на етапі експлуатації пристрої індикації забезпечують інформативний користувальницький інтерфейс, що розширює функціональність виробу й спрощує його використання. Постійний розвиток пристроїв індикації призводить до швидкого морального старіння схемних рішень, які широко використовувалися ще кілька років назад. Поява нових індикаторів і схем їхнього підключення дозволяє створювати нові, економічні й функціональні пристрої.

6.6.1. Статична індикація

Найбільш прості в розробці й реалізації пристрої індикації використовують статичний принцип - на кожний індикатор подається незмінний сигнал керування, що визначає його стан. Прикладом статичної індикації служить керування світлодіодом, підключеним до розряду порту мікроконтролера. Керування ним відбувається аналогічно формуванню статичних керуючих сигналів (розділ 6.2.1). У зв'язку з недостатньою навантажувальною здатністю портів мікроконтролера, безпосереднє підключення світлодіода до виводів порту неприпустиме. Для узгодження рівнів сигналів необхідно використовувати буферний підсилювач, за який часто застосовують логічний елемент НІ з відкритим колектором. Це дозволяє використовувати для живлення світлодіодів не тільки джерело живлення цифрової частини пристрою, але й нестабілізовану напругу понад п'ять вольт. Відносно висока напруга живлення світлодіодів уможливлює їхнє послідовне з'єднання (рис. 6.23), що дозволяє більш ощадливо використовувати енергію джерела живлення, формувати складні та великі мнемонічні знаки. До недоліків подібної схеми підключення можна віднести світіння світлодіодів у перший момент часу після вмикання мікроконтролера. Це пояснюється тим, що настроювання портів на ввід інформації визначається елементом НІ як високий логічний рівень, що приводить до появи на виході елемента сигналу логічного нуля й до протікання струму через світлодіоди. Протікання струму триває до моменту завдання стану ліній порту й може використовуватися для контролю працездатності пристроїв індикації.

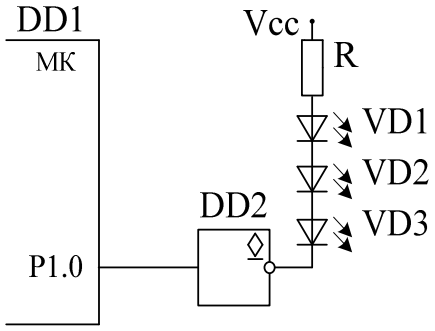
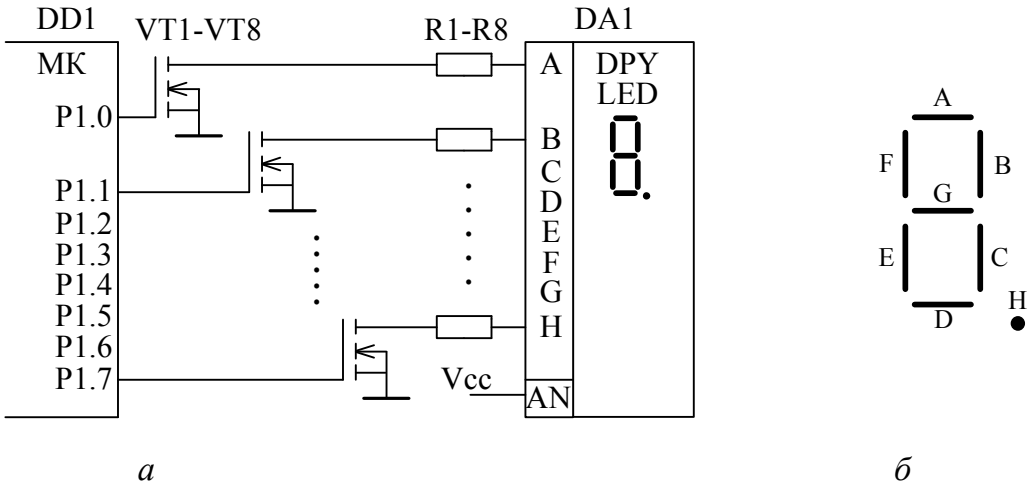


Рисунок 6.23 - Схема підключення світлодіодів до мікроконтролера

Розширення функціональних можливостей пристроїв індикації може бути досягнуте застосуванням семисегментних індикаторів. При використанні мікросхем малого ступеня інтеграції, керування цими індикаторами зводилося до застосування дешифраторів, що забезпечували формування цифр. Використання мікроконтролера дозволяє сформувати значно більше різноманітних знаків. У пристроях зі статичною індикацією знаходять засто-

сування світлодіодні індикатори із загальним анодом, що визначається схемою їхнього включення (рис. 6.24, а). На цій схемі для керування сегментами індикатора використано польові транзистори з логічним рівнем керування. Незважаючи на деяке ускладнення схеми, таке рішення дозволяє значно збільшити струм, що протікає через сегмент індикатора, резистори обмежують його величину.



а *б*

Рисунок 6.24 - Схема підключення семисегментного індикатора до мікроконтролера

Керування семисегментним індикатором має ряд особливостей. При невеликій кількості символів, що виводяться, використовується безпосереднє завдання включених сегментів (рис. 6.24, б) і керування індикатором зводиться до виконання команд вигляду MOV P1, #01001001 (включені сегменти A, G, D) і т.д. При великій кількості символів, використовуваних при виведенні, виникає необхідність у створенні програмного знакогенератора, у якому кожному передбаченому символу присвоюється номер, по якому й провадиться його виведення. Якщо номер символу передається через акумулятор, то для виведення символу з номером 23 необхідно виконати фрагмент програми

```
MOV A, #23
CALL DISPLAY_7
```

де підпрограма DISPLAY_7 забезпечить перетворення номера символу в його сегментне подання й виведе відповідне число в порт.

Найбільш часто семисегментними індикаторами формуються символи, що складаються з одного-двох сегментів: підкреслено (D), тире (G), верхнє тире (A) і їхньої комбінації; цифри 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F; букви A, B, b, C, c, d, E, F, G, H, h, I, i, J, j, L, l, n, O, o, P, r, S, U, u, Б, Г, д (рукописне), П, У, Ч, Ь, Э. З урахуванням однакового накреслення різних букв латинського й кириличного алфавітів, на семисегментному індикаторі

можна виводити текстові повідомлення практично без обмеження використовуваних літер. Крім перерахованих символів можуть створюватися специфічні для даної розробки символи і їхні послідовності.

Приклад програмної реалізації знакогенератора і виведення на індикатор послідовності символів наведено нижче. Номер символу передається до підпрограми через акумулятор.

```
        MOV B, #10          ; ініціалізація лічильника
ANIME:                      ; цикл виведення риски
        MOV A, #3           ; передача в акумулятор номера символу
        CALL DISPLAY_7      ; виведення символу з номером 3 на індикатор
        CALL DELAY          ; затримка між зміною зображень
        MOV A, #4           ; передача в акумулятор номера символу
        CALL DISPLAY_7      ; виведення символу з номером 4 на індикатор
        CALL DELAY          ; затримка між зміною зображень
        MOV A, #5           ; передача в акумулятор номера символу
        CALL DISPLAY_7      ; виведення символу з номером 5 на індикатор
        CALL DELAY          ; затримка між зміною зображень
        DJNZ B, ANIME       ; перевірка кінця циклу
        MOV A, #2           ; передача в акумулятор номера символу
        CALL DISPLAY_7      ; виведення символу з номером 2 на індикатор
        NOP                 ; текст програми, що виконується після
                            ; виведення на індикатор
DISPLAY_7:                  ; підпрограма виведення символу на індикатор
        MOV DPTR, #SIGN     ; передача адреси знакогенератора
        MOVC A, @A+DPTR     ; читання з таблиці байта з номером, що
                            ; лежить в акумуляторі
        MOV P1, A           ; виведення прочитаного байта на індикатор
        RET                 ; повернення до основної програми
DELAY:                      ; підпрограма затримки
;                           ; текст підпрограми
        RET                 ; повернення до основної програми
SIGN:                       ; таблиця знакогенератора
DB   00111111b              ;”0”  – ABCDEF      символ номер 0
DB   00000110b              ;”1”  – BC          символ номер 1
DB   01011011b              ;”2”  – ABDEG       символ номер 2
DB   00001000b              ;”_”  - D           символ номер 3
DB   01000000b              ;”-“  - G           символ номер 4
DB   00000001b              ;” ¯ “ - A          символ номер 5
```

На індикаторі з’являється риска, що піднімається вгору. Ця дія повто-

рюється 10 разів, після чого на індикатор виводиться цифра 2. Анімація руху риски зроблена без застосування циклу, що дає змогу використовувати символи, які мають довільні номери (символи ”_”, ”-“ і ” ¯ “ можуть знаходитись будь де у таблиці). У разі використання таблиці SIGN достатньо інкрементувати номер символу, що зменшить обсяг програми.

6.6.2. Індикація з використанням регістрів зсуву

Основним недоліком, що обмежує застосування багаторозрядних статичних індикаторів, є велика кількість ліній зв’язку між мікроконтролером і пристроєм індикації. Для усунення цього недоліку можна використати послідовну передачу інформації від мікроконтролера до індикаторів. Для цього застосовуються регістри зсуву. Завантаження інформації в регістри потребує тільки дві лінії зв’язку, а число паралельних вихідних ліній обмежується лише часом, що розроблювач може відвести для відновлення інформації. Приклад схеми чотирирозрядного індикатора, з використанням регістрів зсуву, наведено на рис. 6.25.

Передача інформації здійснюється за двома сигнальними лініями – даних та синхронізації. На вхід даних МК побітно передає інформацію, що потрібна для відображення, а на вхід синхронізації – тактові імпульси, що синхронізують регістр зсуву. Такий синхронний послідовний обмін часто використовується при обміні даними між мікроконтролером та периферійними пристроями через малі апаратні і програмні витрати, потрібні для його реалізації.

Як випливає зі схеми (рис. 6.25), спочатку МК повинен передати індикатору сегментне подання символу, який повинен відображатись в крайньому розряді праворуч. За ним передається подання символу, що відображається в другому розряді праворуч і т. д., аж до останнього, десятого.

У схемі (рис. 6.25) використано регістри зсуву типу К555ИР8 (аналог 74xx164). Допустимий струм сегмента становить від 15 до 30 mA, залежно від серії мікросхем. Особливість конструкції регістрів приводить до паразитного світіння сегментів індикаторів при зсуві інформації. Паразитне світіння зменшується при зменшенні розрядності індикатора та зменшенні частоти оновлення інформації. Якщо інформація на індикаторі оновлюється часто і паразитне світіння сегментів неприпустиме, на час оновлення інформації необхідно відключати індикатори від джерела живлення за допомогою транзистора VT1, що керується мікроконтролером. Інший шлях поліпшення характеристик схеми полягає в використанні регістрів зсуву з третім станом вихідних ліній, наприклад, 555ИР25 (аналог 74xx395). На час оновлення інформації мікроконтролер переводить вихідні лінії регістрів до третього стану, що

виключає паразитне світіння, але додає ще одну лінію керування.

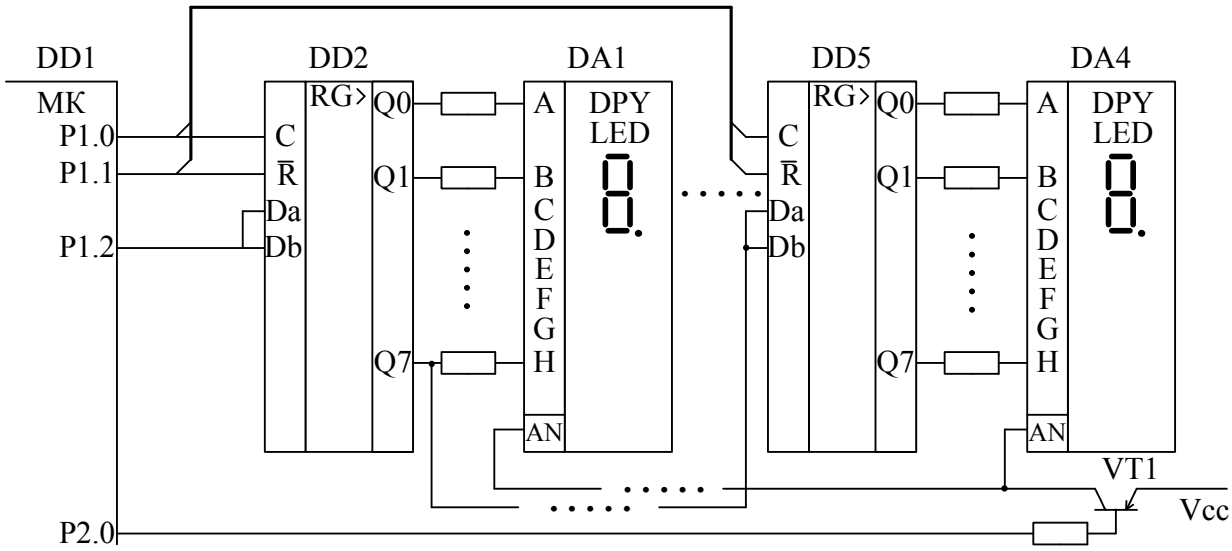


Рисунок 6.25 - Схема чотирирозрядного індикатора з регістрами зсуву

Приклад програми керування індикаторами за схемою (рис. 6.25), що виводить чотири символи, наведено нижче. Номери символів, що виводяться, розташовані за послідовними адресами внутрішньої пам'яті даних, починаючи з адреси, що зберігається у константі BUFFER. Сегментне подання рядка символів зберігається в регістрах R4 – R7. Символ для молодшого розряду індикатора зберігається за молодшою адресою.

```
BUFFER    EQU       20H
;         …                       ;
          CLR   P1.1              ; сигнал скидання регістрів зсуву
          SETB  P1.1              ; регістри зсуву готові до роботи
          MOV   B, #4             ; ініціалізація лічильника символів
          MOV   R0, #4            ; запис адреси сегментного подання рядка
          MOV   R1, #BUFFER       ; запис адреси рядка символів
READ_4:                           ; читання номерів символів,
                                  ; запис сегментного подання
          MOV   A, @R1            ; отримання номеру символу
          CALL  TO_7              ; перекодування номер – сегментне подання
          INC   R0                ; перехід до наступної адреси збереження
          INC   R1                ; перехід до наступного символу у буфері
          DJNZ  B, READ_4         ; перевірка закінчення циклу
          MOV   B, #4*8           ; ініціалізація лічильника бітів
SHIFT_32:                         ;
          CLR   C                 ; скидання біта переносу
```

```
        MOV   R0, #7          ; встановлення адреси четвертого байта
        MOV   A, @R0          ; передача четвертого байта до акумулятора
        RLC   A               ; зсув четвертого байта
        MOV   @R0, A          ; збереження зміненого четвертого байта
        DEC   R0              ; перехід до третього байта
        MOV   A, @R0          ; передача третього байта до акумулятора
        RLC   A               ; зсув третього байта
        MOV   @R0, A          ; збереження зміненого третього байта
        DEC   R0              ; перехід до другого байта
        MOV   A, @R0          ; передача другого байта до акумулятора
        RLC   A               ; зсув другого байта
        MOV   @R0, A          ; збереження зміненого другого байта
        DEC   R0              ; перехід до першого байта
        RLC   A               ; зсув першого байта,
        MOV   @R0, A          ; збереження зміненого першого байта
        MOV   P1.2, C         ; запис біта, що передається
        SETB  P1.0            ; імпульс синхронізації регістрів зсуву
        CLR   P1.0            ; скидання імпульсу синхронізації
                              ; регістрів зсуву
        DJNZ  B, SHIFT_32     ; перевірка закінчення циклу передачі бітів
        NOP                   ; текст програми, що виконується після
                              ; передачі інформації до індикаторів
TO_7:                         ; підпрограма перекодування
                              ; і збереження результатів
        MOV   DPTR, #SIGN     ; запис адреси таблиці знакогенератора
        MOV   A, @A+DPTR      ; отримання сегментного подання символу
        MOV   @R0, A          ; запис сегментного подання до пам’яті
        RET                   ; повернення до основної програми
SIGN:                         ; таблиця знакогенератора
DB    00111111b               ;0 – ABCDEF      символ номер 0
;     …                       ; продовження таблиці
```

Для послідовної передачі даних потрібно по одному біту передати чотирибайтове число. Зсув байта сегментного подання символу здійснюється вліво, бо у таблиці знакогенератора старший біт відповідає сегменту H, а він підключений до старшого біта регістра зсуву, що досягає свого місця останнім. Зсув чотирибайтового числа потрібно починати з символу старшого розряду. Старший біт байта сегментного подання передається до біта переносу C, а потім бере участь у зсуві символів молодших розрядів. Передача починається з символу молодшого регістра, бо він найдальший. Старший біт пере-

дається до біта переносу С, а потім записується до відповідного розряду порту і передається до регістрів зсуву після формування імпульсу синхронізації. Дії повторюються до передачі всіх бітів сегментного подання чотирьох символів.

Необхідність мати на кожен індикатор свій регістр зсуву, значний час перевантаження інформації до індикаторів і проблеми з їх паразитним світінням обумовили використання цього способу індикації в приладах, де відчувається нестача ліній керування.

6.6.3. Динамічна індикація

У більшості випадків виведення інформації здійснюється з використанням індикаторів, які мають однотипні структури. Це можуть бути матричні індикатори, які формують зображення з точкових джерел, або багаторозрядні семисегментні індикатори. Однотипними структурами в першому випадку є рядки або стовпці матриці індикаторів, в другому – знакомісця (індикатор, або група індикаторів, які формують символ).

Широкого розповсюдження набув спосіб динамічної індикації, який передбачає виведення інформації в кожен інтервал часу лише на одну однотипну структуру – рядок, стовпець чи знакомісце (рис. 6.26). Цикл індикації складається з ряду інтервалів, у кожному з яких інформація виводиться на іншу структуру індикатора. Кількість інтервалів у циклі відповідає кількості однотипних структур індикатора, порядок виведення інформації, в загальному випадку, довільний.

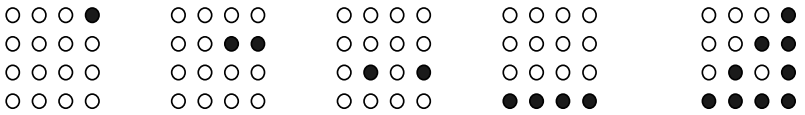


Рисунок 6.26 - Отримання зображення при динамічній індикації. Формування зображення по рядках

В індикаторах, що використовують динамічну індикацію, передбачаються виводи, що керують підключенням однотипних структур, а керуючі виводи всіх однойменних елементів структур з’єднуються. Для виведення числової та деякої символьної інформації використовуються семисегментні багаторозрядні індикатори. Схема керування десятирозрядним семисегментним індикатором наведена на рис. 6.27.

Вибір знакомісця виконується дешифратором DD2 К555ИД10, що дозволяє використовувати до 10 розрядів індикації зі струмом розряду до 80 мА. Керування сегментами провадиться за допомогою транзисторів VT1 –

VT8. Конфігурація схеми потребує використання індикаторів із загальним катодом.

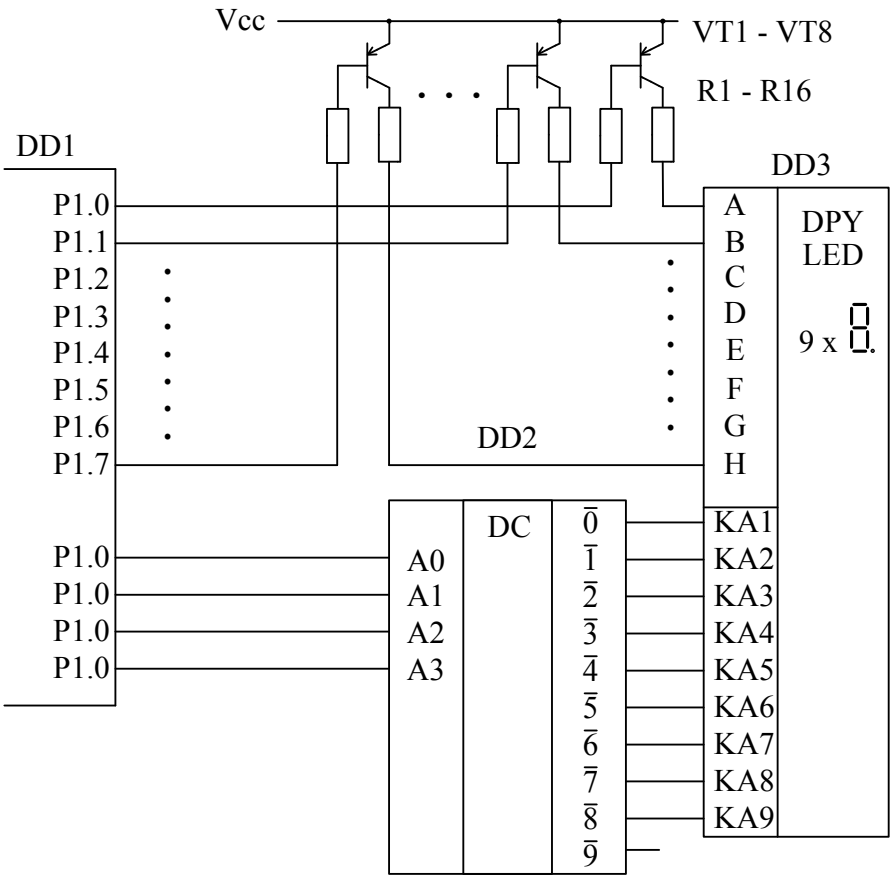


Рисунок 6.27 - Багаторозрядний семисегментний індикатор з динамічною індикацією

Приклад програми, що реалізує виведення рядка символів методом динамічної індикації, наведено нижче. Код символу, що відображається, одночасно подається на аноди всіх розрядів. Відображається ж цей символ в тому розряді, катоди якого знаходяться під нульовим потенціалом, при цьому решта розрядів індикатора виявляються погашеними.

Програма зроблена у вигляді підпрограми обробки переривання таймера 0. Це дає можливість виводити символи через інтервали часу TIME_DELAY, що вимірюються таймером. У прикладі основна програма знаходиться у холостому циклі LOOP. Номери символів, що виводяться, розташовані за послідовними адресами внутрішньої пам’яті даних, починаючи з адреси, що зберігається у константі STR_10. Символ для молодшого розряду індикатора зберігається за молодшою адресою. Поточні лічильники: адреси символу – R1, адреси регістра індикатора – R2.

```
STR_10    EQU       32
ORG 0
```

```
        AJMP   START            ; перехід до початку програми
ORG 0Bh                         ; адреса переривання від таймера T/C0
        CLR   TR0               ; зупинка таймера
        MOV   DPTR, #SIGN       ; запис адреси знакогенератора
        MOV   A, @R1            ; запис номера символу, що виводиться
        MOVC   A, @A+DPTR       ; читання сегментного подання символу
        MOV   P1, A             ; виведення даних до порту
        ANL   P2, #11110000b    ; очистка адреси регістра індикатора
        ORL   P2, R2            ; запис адреси регістра індикатора
        INC   R1                ; перехід до наступного символу
        INC   R2                ; перехід до наступного розряду
        CJNE   R2, #10, END_INT      ; досягнуто останній розряд індикатора?
        MOV   R1, # STR_10      ; ініціалізація адреси символів
        MOV   R2, #0            ; ініціалізація адреси регістра індикатора
END_INT:
        MOV   TH0, #(-TIME_DELAY SHR 8)     ;завантаження таймера
        MOV   TL0, #(-TIME_DELAY)           ;
        SETB   TR0                          ; пуск таймера
        RETI                    ; повернення до основної програми

START:                          ; початок програми
; ініціалізація таймера
; дозвіл переривань
; пуск таймера
        MOV   R1, # STR_10      ; ініціалізація адреси символів
        MOV   R2, #0            ; ініціалізація адреси регістра індикатора
;       …                       ; продовження програми
LOOP:                   ; цикл, у якому очікується переповнення таймера
;       …               ; текст програми, що повторюється циклічно
        AJMP   LOOP             ;
;       …                       ; можливе продовження програми
SIGN:                           ; таблиця знакогенератора
DB    00111111b                 ;0 – ABCDEF       символ номер 0
;       …                       ; продовження таблиці
```

6.6.4. Рідкокристалічні індикатори

Після значного зниження цін і розширення номенклатури виробів рідкокристалічні індикатори (РКІ або LCD – Liquid Crystal Display) стали дуже популярними у розроблювачів мікроконтролерних пристроїв. Вони дозво-

лили ввести в проектовані пристрої функції, раніше доступні лише у виробах дорожчої цінової групи. До позитивних рис РКІ варто віднести можливість формування інтелектуального користувальницького інтерфейсу, що підтримує текстові повідомлення й систему меню; графічне відображення процесів у системі, що дозволяє здійснювати одночасне виведення більшої кількості інформації й робити оперативне керування об'єктом без одержання числових даних.

РКІ діляться на символьні (алфавітно-цифрові) й графічні. Перші можуть відображати визначений набір символів, допускаючи програмування декількох символів користувача. Другі забезпечують виведення довільного зображення користувача на площі дисплея. Як правило, керування РКІ здійснюється по восьмибітовій шині даних з використанням чотирьох-шести ліній керування. При необхідності, шина даних може бути звужена до 4 біт, але це доступно не у всіх моделях РКІ. Функціонування індикаторів обумовлюється розвиненою системою команд.

До причин, які стримують застосування РКІ, варто віднести відсутність уніфікації інтерфейсу, архітектури й системи команд, що більшою мірою стосується графічних РКІ. Освоєння інтерфейсу індикатора базується на документації його виробника або виробника мікросхем контролера РКІ. Наступний матеріал дозволить спростити освоєння деяких широко розповсюджених індикаторів, але не замінить самостійне вивчення оригінальної документації.

Символьний РКІ. Індикатори, що виводять алфавітно-цифрову інформацію, знайшли широке застосування в приладах. Вони вирізняються кількістю символів у рядку й кількістю рядків символів. Випускаються індикатори з такими типовими значеннями співвідношення символ-рядок: 8x1, ..., 16x2, ... , 24x4, ..., 40x4. Більшість із них працює під управлінням контролера HD44780 [23]. Дисплеї з використанням контролера HD44780 або його клонів випускаються більшістю виробників РКІ й у даний момент вважаються промисловим стандартом.

Типова схема підключення РКІ до мікроконтролера наведена на рис. 6.28. Таке підключення передбачає реалізацію програмного інтерфейсу з індикатором, з використанням 8-бітової шини даних і сигналів: синхронізації – E, читання/запису – R/^W (0 – запис даних), вибору регістрів – RS (0 – запис до регістра інструкцій, читання – прапор зайнятості, лічильник адреси; 1 – запис і читання даних).

Після подачі напруги живлення внутрішня схема скидання автоматично здійснює ініціалізацію контролера індикатора. На протязі ініціалізації, що триває близько 10 мс, прапор зайнятості встановлено (BF=1), виконується

завдання початкових значень функцій керування (тут і надалі використовуються імена бітів відповідно до [23]). Наступна інструкція не приймається контролером до скидання прапора зайнятості.

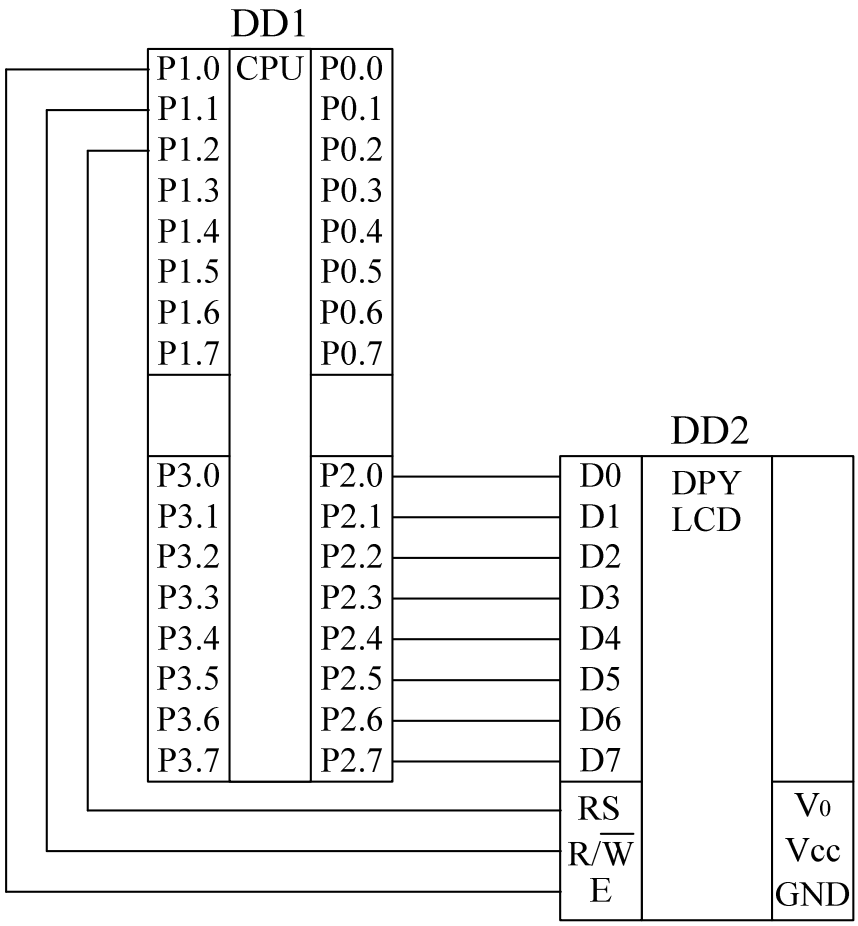


Рисунок 6.28 - Підключення символьного РКІ до мікроконтролера

Після ініціалізації стан індикатора визначається такими функціями:

- **індикатор очищено**;
- **встановлені функції**:
  DL=1 – 8-бітовий інтерфейс;
  N=0 – одна лінія символів;
  F=0 – використовується шрифт 5x8 крапок;
- **керування включенням індикатора**:
  D=0 – індикатор вимкнено;
  C=0 – курсор вимкнено;
  B=0 – миготіння вимкнено;
- **режим індикатора**:
  I/D=1 – інкремент лічильника адреси пам’яті даних;
  S=0 – без зсуву.

Операції передачі даних здійснюються згідно з часовими діаграмами,

що наведені на рис. 6.29. Інтервали часу між сигналами вибираються відповідно до технічних характеристик індикатора. Типове значення ширини імпульсу синхронізації 230 нс з періодом повторення 500 нс. Виконання інструкцій відбувається значно довше – типове значення часу виконання досягає 37 мкс, а виконання інструкції скидання адреси та скасування зсуву – 1,52 мс. Швидкодія мікроконтролера значно перевищує швидкодію індикатора, тому при програмуванні операцій з індикатором потрібно передбачати використання часових затримок, або обов'язковий контроль прапора зайнятості.

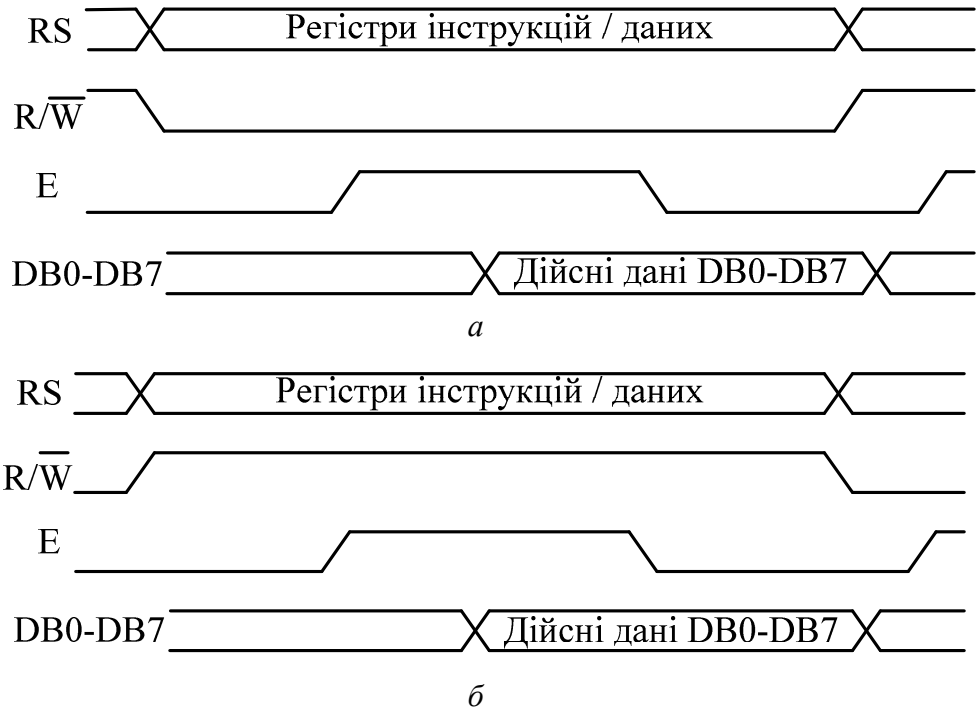


Рисунок 6.29 - Часові діаграми обміну даними з РКІ:
а - операції запису; *б* - операції читання

Приклад програми, що виводить інформацію на символьний індикатор 8x2 (рис. 6.28), наведено нижче.

```
;Адреси, за якими відбувається звернення до регістрів індикатора
LCD_CMD_WR   EQU  0      ; запис команди
LCD_DATA_WR EQU  4      ; запис даних
LCD_BUSY_RD  EQU  2      ; читання лічильника адреси (біта зайнятості)
LCD_DATA_RD  EQU  6      ; читання даних

;Початок тексту програми
ORG 0000H
      JMP START
;Вектор переривань
```

```
;…
        ORG 0100H
START:
;Ініціалізація HD44780
        ANL   P1, #11111110B  ; скидання сигналу синхронізації
        MOV  A, #038H         ; 8-бітовий інтерфейс, 2 лінії символів
        CALL WRCMD            ; запис команди встановлення режиму
        MOV  A, #0CH          ; дісплей ввімкнено, курсор вимкнено
                              ; миготіння вимкнено
        CALL  WRCMD           ; запис команди
        MOV  A,#06H           ; режим інкрементування адреси і переміщення
                              ; курсора вправо при запису символу
        CALL  WRCMD           ; запис команди
        MOV   A, #01H         ; очищення екрану
        CALL WRCMD            ; запис команди
; текст програми, наприклад виведення символів A та B
        MOV  A, #'A'
        CALL WRCHAR
        MOV  A, #'B'
        CALL WRCHAR
        AJMP  $                       ; нескінченний цикл

WRCMD:                                ; підпрограма запису команди
        ANL  P1, #11111001B           ; очищення бітів звернення до регістрів
        ORL  P1, #LCD_CMD_WR          ; виведення адреси запису команди
        MOV P2, A                     ; виведення команди
        ORL P1,#001B                  ; встановлення сигналу синхронізації
        NOP                           ; часова затримка
        ANL P1,#11111110B             ; скидання сигналу синхронізації -
                                      ; запис команди
        JMP WTBUSY                    ; перехід до контролю біту зайнятості

WRCHAR:                               ; підпрограма запису символу
        ANL  P1, #11111001B           ; очищення бітів звернення до регістрів
        ORL  P1, #LCD_DATA_WR         ; виведення адреси запису символу
        MOV P2, A                     ; виведення символу
        ORL  P1, #001B                ; встановлення сигналу синхронізації
        NOP                           ; часова затримка
        ANL  P1,#11111110B            ; скидання сигналу синхронізації -
```

```
                                ; запис символу

WTBUSY:                         ; підпрограма контролю біту зайнятості
        ANL  P1, #11111001B     ; очищення бітів звернення до регістрів
        ORL  P1, #LCD_BUSY_RD   ; виведення адреси читання
                                ; біта зайнятості
        ORL P1,#001B            ; встановлення сигналу синхронізації
        MOV  A, P2              ; читання інструкції (біта зайнятості)
        ANL P1,#11111110B       ; скидання сигналу синхронізації
        JB ACC.7, WTBUSY    ; повторення, якщо біт зайнятості встановлено
        RET                 ; повернення з підпрограми

END
```

Підключення символьного РКІ до системної шини МК. Програмний обмін даними із РКІ, розглянутий раніше, вимагає точного програмного формування часових параметрів сигналів. Це приводить до збільшення об'єму програмного коду й до зменшення швидкодії програми. Для передачі частини функцій по обміну даними апаратним засобам мікроконтролера застосовується підключення РКІ до системної шини мікроконтролера й спілкування з ним, як з елементом пам'яті даних. Схема мікроконтролерної системи, що використовує зовнішню пам'ять програм, пам'ять даних і РКІ, була запропонована фірмою INTEL [24] (рис. 6.30). Достоїнством підключення РКІ до системної шини є апаратне формування строба читання/запису, це ж одночасно є й обмежуючим фактором. Часові параметри системної шини задані виробником мікроконтролера й можуть бути змінені лише зі зміною тактової частоти системи, що, у цьому випадку, обмежується швидкодією індикатора.

Вивід вибору регістра РКІ підключено до адреси А0 системної шини, а вивід вибору операції читання/запис - до А1. Стробування даних здійснюється кожним з сигналів читання або запису, що генеруються мікроконтролером при звертанні до зовнішньої пам'яті даних, але за наявності в цей момент виставленої адреси А15. При такому включенні індикатора всі операції обміну даними зводяться до операцій читання/запис пам'яті даних за визначеними адресами:

- 8000Н - запис команд;
- 8001Н - запис даних;
- 8002Н - читання лічильника адреси (статусу РКІ)
- 8003Н - читання даних
- 8004Н - 0FFFFH не визначені.

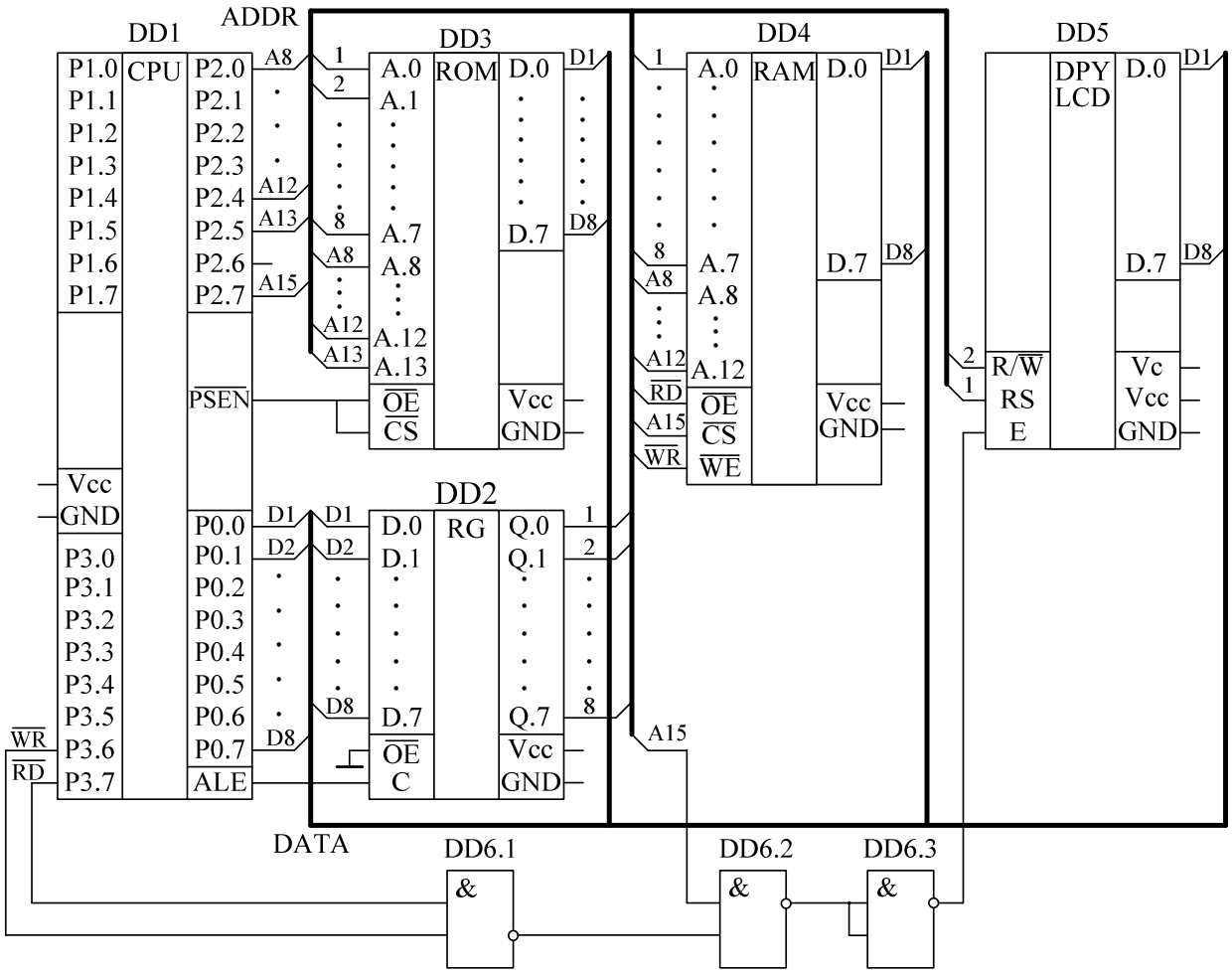


Рисунок 6.30 - Підключення РКІ до мікроконтролера разом з мікросхемами пам’яті

При такому підході запис даних з акумулятора в індикатор реалізується командами:

```
MOV   DPH, #80h
MOV   DPL, #01h        ; адреса запису даних
MOVX  @DPTR, A         ; передача даних
```

Перевірка прапору зайнятості відбувається за допомогою команд:

```
MOV   DPH, #80h
MOV   DPL, #02h        ; адреса читання статусу
MOVX  A, @DPTR         ; читання статусу
JB    ACC.7, BUSY      ; перехід на мітку BUSY,
                       ; якщо біт зайнятості встановлено
```

Спрощена схема підключення індикатора до МК наведена на рис. 6.31. У схемі використано мікросхеми 80С51, 555ЛА3 (74LS00), HD44780. У цьому випадку не використовується зовнішня пам’ять, що дає можливість спростити схему підключення.

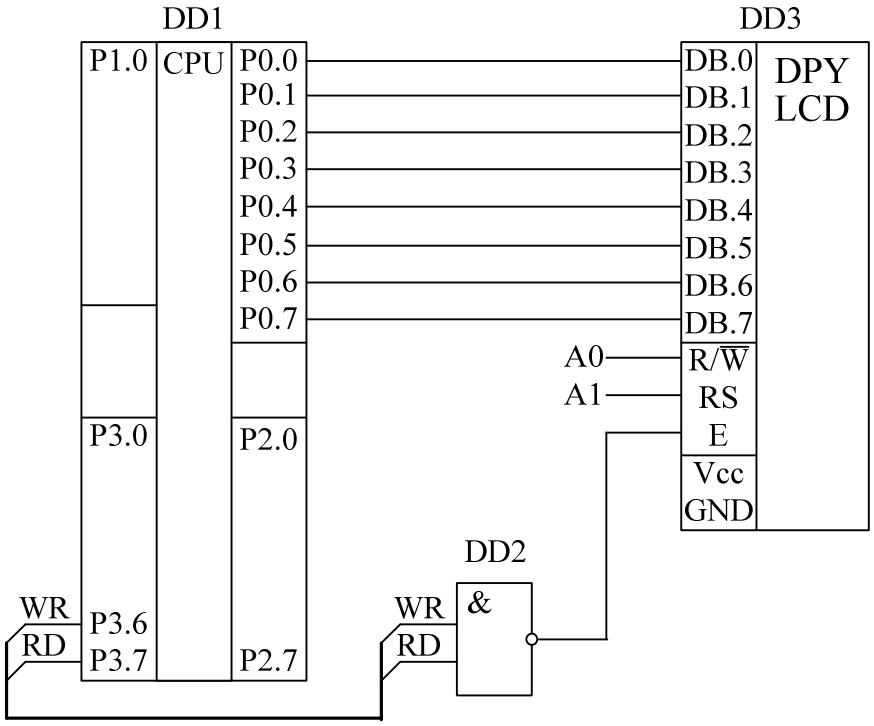


Рисунок 6.31 - Підключення РКІ до мікроконтролера за адресами пам’яті

В адресному просторі індикатора використовується чотири адреси в діапазоні від 0 до 3. У зв’язку з цим, звертання до індикатора може здійснюватися командами MOVX @Ri, A і MOVX A, @Ri.

РКІ з послідовним інтерфейсом. Керування РКІ, здійснюване по паралельному інтерфейсу, вимагає наявності великої кількості вільних виводів мікроконтролера (7 – 11 у розглянутих випадках). Використання для керування РКІ системної шини також має ряд недоліків:

- багато мікроконтролерів не розраховані на застосування зовнішньої пам’яті й у них системна шина відсутня;
- розвиток мікроконтролерів привів до підвищення частоти системної шини до значень, які не підтримуються контролером РКІ;
- необхідне введення схем дешифрації сигналів керування РКІ, що ускладнює схему пристрою.

Виходом, що дозволяє використовувати РКІ з мінімальними вимогами до надаваних апаратних ресурсів, є застосування РКІ з послідовним інтерфейсом передачі даних.

Донедавна РКІ з послідовним інтерфейсом були широко представлені у виробах масового виробництва, наприклад, стільникових телефонах, але практично були відсутні на ринку електронних компонентів і виробів. Потреби ринку привели до розробки й виробництва РКІ, структурна схема яких

наведена на рис. 6.32 [25, 26, 27]. Основою пристрою є перетворювач інтерфейсів. Він виконаний на основі восьмирозрядного мікроконтролера й здійснює перетворення сигналів послідовного інтерфейсу RS232 або I^2C у паралельний інтерфейс РКІ. Вибір вхідного інтерфейсу й швидкість передачі по асинхронному інтерфейсу задаються перемикачами. Передача команд здійснюється двобайтовими посилками, де перший байт є зарезервованим для цієї мети символом.

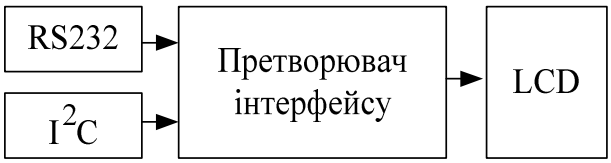


Рисунок 6.32 - Структурна схема РКІ з перетворювачем інтерфейсу

Серед РКІ з послідовним інтерфейсом виділяються графічні індикатори з інтерфейсом I^2C [28] і індикатори, які, крім функції відображення інформації, мають додаткову функцію підключення клавіатури. Зовнішня клавіатура реалізується у вигляді стандартної матриці контактів з 5 рядків і 5 стовпців [29]. Дані про натиснуту клавішу передаються по послідовному інтерфейсу I^2C або RS232, по якому, крім цього, здійснюється керування функціями індикатора й передача графічної інформації.

Для керування РКІ використовуються послідовні посилки, формовані відповідно до вимог обраного інтерфейсу й алгоритму роботи індикатора.

Як приклад розглянемо керування індикаторами фірм Milford Instruments Limited і Scott Edwards Electronics, Inc. [30, 31]. Вони використовують однопровідну асинхронну лінію зв’язку (рис. 6.33), сумісну зі стандартом RS232, яка підтримує рівні сигналів логічних елементів CMOS/TTL серій.

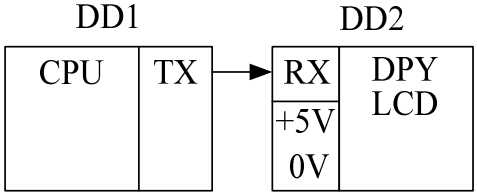


Рисунок 6.33 - Підключення РКІ з послідовним інтерфейсом до мікроконтролера.

Для розрізнення команд і даних, що передаються послідовним інтерфейсом, використовується двобайтовий формат команди. Першим передається число 254, потім – код команди. Робота з індикатором починається з команди очищення індикатора, що має код 1. Приклад програми, що виводить на індикатор MILFORD-2X16-BKP десять послідовних символів, наве-

дено нижче:

```
START:                          ; початок програми
      MOV   SCON, #01000000B    ; встановлюємо режим 1 УСАПП
      MOV   TMOD, #00100000B    ; встановлюємо режим 3 таймера 1
      MOV   TH1, #0FDH          ; частота прийому/передачі 9600 бод при
      MOV   TL1, #0FDH          ; частоті синхронізації 11,059 МГц
      SETB  TR1                 ; пуск таймера
      ACALL  PAUSE1             ; формування паузи, потрібної для
                                ; скидання індикатора
      CLR   TI                  ; скидання прапора «передача закінчена»
                                ; перед початком передачі символу
                                ; передача команди очищення індикатора:
      MOV   SBUF, #254          ; передача символу з номером 254
C1:
      JNB   TI, C1              ; очікування закінчення передачі символу
      CLR   TI                  ; скидання прапора «передача закінчена»
      MOV   SBUF, #01           ; передача символу з номером 1
C2:
      JNB   TI, C2          ; очікування закінчення передачі символу
      ACALL   PAUSE2        ; формування паузи після команди очищення
                            ; індикатора перед передачею даних
      MOV   R1, #'A'        ; запис початкового символу
      MOV   R2, #10             ; лічильник символів, що виводяться
C2_1:
      CLR   TI                  ; скидання прапора «передача закінчена»
      MOV   SBUF, R1            ; передача поточного символу
C3:
      JNB   TI,C3               ; очікування закінчення передачі символу
      INC   R1                  ; перехід до наступного символу
      DJNZ   R2, C2_1           ; контроль закінчення передачі символів
; подальший текст програми
```

Керування РКІ за допомогою регістра зсуву. Зручність використання РКІ на різних етапах виробництва й експлуатації мікроконтролерних систем особливо виразно виявляється на етапі налагодження програмного забезпечення. Робота в режимі реального часу пов'язана з дією на систему недетермінованих впливів, ефект від яких складно або неможливо змоделювати в програмному симуляторі мікроконтролерної системи. Це вносить у процес налагодження елемент непередбачуваності. Програмно-апаратні комплекси,

призначені для вирішення даних проблем, мають вартість, що найчастіше значно перевищує вартість розробки виробу. Використання в процесі налагодження РКІ, на який виводяться значення регістрів і змінні в контрольних точках, допомагає прискорити й спростити процес налагодження.

Серйозною перешкодою на шляху використання РКІ в процесі налагодження й експлуатації МК систем стає необхідна для передачі інформації кількість вільних виводів мікроконтролера. При паралельному способі передачі інформації – не менше 6 виводів, що занадто багато для більшості застосувань. Популярним є використання РКІ з послідовним інтерфейсом, у цьому випадку потрібен тільки один провід. Недоліком РКІ зі стандартним послідовним інтерфейсом є необхідність строгого дотримання вимог протоколу: формування часових інтервалів в асинхронних протоколах і завдання послідовності запису/читання даних – у синхронних. Дотримання вимог протоколу може бути важким або неможливим. У цьому випадку для передачі даних у РКІ може бути використаний регістр зсуву з послідовним введенням і паралельним виведенням даних, що синхронізуються мікроконтролером. Регістр зсуву апаратно перетворює послідовні дані, що надходять від мікроконтролера, у паралельне подання набору даних і сигналу вибору регістра RS, необхідне для роботи РКІ. Сигнал синхронізації даних РКІ – E, формується безпосередньо мікроконтролером (рис. 6.34). Як видно з цього рисунка, принцип передачі даних аналогічний розглянутому в розділі 6.7.2 і вимагає використання трьох виводів мікроконтролера.

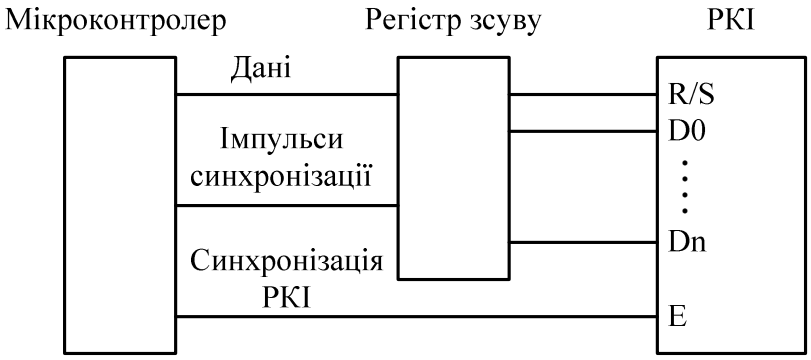


Рисунок 6.34 - Підключення РКІ до мікроконтролера за допомогою регістра зсуву

Керування РКІ на основі контролера HD44780 можливе в режимі восьмирозрядної або чотирирозрядної шини даних. Для спрощення схеми можна використати один напрямок передачі даних – запис у РКІ, що дозволяє задіяти лише два сигнали керування. Таким чином, для реалізації найпростішої схеми керування РКІ необхідно використати десяти- або шестирозрядний регістр зсуву. Найбільше застосування в практиці знаходять чотири- і восьмирозрядні мікросхеми регістрів.

Схема підключення РКІ до мікроконтролера з використанням сигналів даних і синхронізації наведена на рис. 6.35, а часові діаграми, що пояснюють її роботу - на рис. 6.36. У схемі використано мікросхеми DD1 – регістр зсуву К555ИР8 (аналог 74LS164) і DD2 – рідкокристалічний індикатор з контролером HD44780. Керування РКІ ведеться з використанням чотирирозрядної шини даних D5-D7 і сигналу вибору регістра RS. Передача сигналу синхронізації даних E здійснюється разом з даними, а його виділення - за допомогою логічного елемента «монтажне І».

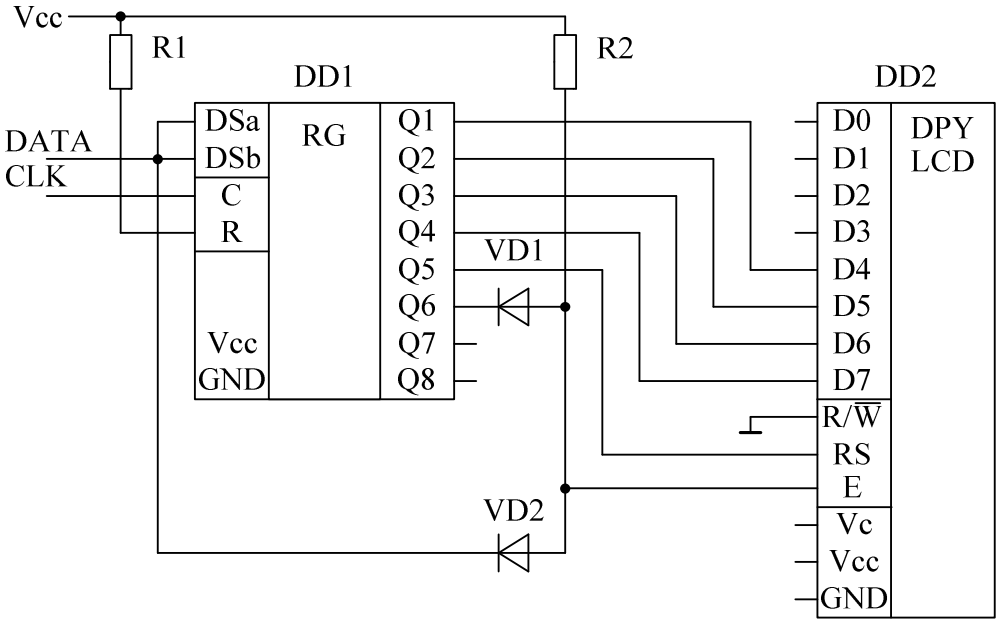


Рисунок 6.35 - Схема підключення РКІ до мікроконтролера

Логічний елемент «монтажне І» реалізований на елементах R2, VD1, VD2. Наявність логічного нуля на виводі Q6 мікросхеми DD1, або на лінії даних DATA, приводить до відкривання відповідного діода й до появи нульового сигналу на вході синхронізації E. Перехід сигналу E у стан логічної одиниці відбудеться лише при одиничних сигналах на виводі Q6 і лінії даних.

Для запобігання запису недійсних даних, що може відбутися при одиничному значенні сигналу синхронізації, перед операцією запису регістр зсуву повинен бути очищений. Для виключення сигналу скидання регістру зсуву, який потребує використання додаткового виводу МК, у регістр записуються шість нульових біт (інтервал t1-t6 на рис. 6.36). Одиничний біт, що підтверджує завершення зсуву керуючого слова й дозволяє формування сигналу синхронізації E, записується в регістр зсуву першим. Відповідно до часових діаграм запису даних у РКІ сигнал вибору регістра RS передує появі даних, тому за одиничним бітом записується біт вибору регістра, а за ним чотири біти даних, старшим бітом уперед (інтервал t7-t12 на рис. 6.36).

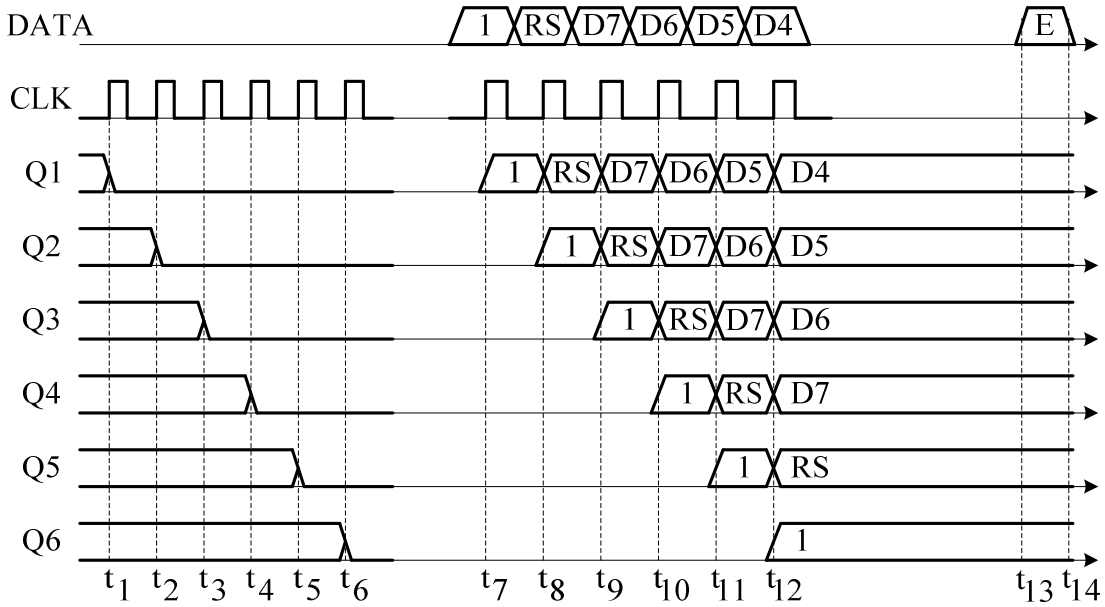


Рисунок 6.36 - Часові діаграми передачі даних до РКІ за допомогою регістра зсуву

Після моменту часу t12 на виводах регістра зсуву з'являється сформоване керуюче слово РКІ. Формування одиничного сигналу на лінії даних DATA у момент часу t13 приводить до появи сигналу синхронізації, по задньому фронту якого здійснюється запис даних у РКІ.

Можливе використання шестирозрядного регістра зсуву, побудованого на основі набору D-тригерів – К555ТМ9 (аналог 74LS174) [32].

Фрагмент програми, що реалізує керування РКІ за допомогою регістра зсуву, наведений нижче. Байт даних знаходиться в акумуляторі, значення біта вибору регістра - у біті з ім'ям RS. Лінії даних і синхронізації підключені до бітів порту з іменами DATA і CLK відповідно.

```
        ACALL   CLR_REG         ; очищення регістра зсуву
        ACALL   SHIFT_DATA      ; виведення старшої тетради даних РКІ
        SETB    DATA            ; формування фронту сигналу E
        CLR     DATA            ; формування зрізу сигналу E
        ACALL   CLR_REG         ; очищення регістра зсуву
        ACALL   SHIFT_DATA      ; виведення молодшої тетради даних РКІ
        SETB    DATA            ; формування фронту сигналу E
        CLR     DATA            ; формування зрізу сигналу E
; подальший код програми
CLR_REG:                        ; підпрограма очищення регістра зсуву
        CLR     DATA            ; скидання лінії даних
        MOV     B, #6           ; завантаження лічильника бітів, що виводяться
CLR_BIT:                        ;
        SETB    CLK             ; формування фронту імпульсу синхронізації
```

```
        CLR   CLK           ; формування зрізу імпульсу синхронізації
        DJNZ   B, CLR_BIT   ; перевірка числа бітів, що не виведені
        RET                 ; повернення до основної програми
SHIFT_DATA:                 ; підпрограма виведення службових бітів
                            ; і бітів даних
        SETB   DATA         ; формування сигналу логічної одиниці
        SETB   CLK          ; формування фронту імпульсу синхронізації
        CLR   CLK           ; формування зрізу імпульсу синхронізації
        MOV   C, RS         ; передача значення біта вибору регістра
        MOV   DATA, C       ; формування сигналу вибору регістра
        SETB   CLK          ; формування фронту імпульсу синхронізації
        CLR   CLK           ; формування зрізу імпульсу синхронізації
        MOV   B, #4         ; завантаження лічильника
SHIFT_BIT:                  ;
        RLC   A             ; передача старшого біта акумулятора до біта C
        MOV   DATA, C       ; виведення біта даних
        SETB   CLK          ; формування фронту імпульсу синхронізації
        CLR   CLK           ; формування зрізу імпульсу синхронізації
        DJNZ   B, SHIFT_BIT ; перевірка числа бітів, що не виведені
        RET                 ; повернення до основної програми
```

Виведення даних відбувається відповідно до вимог протоколу контролера HD44780 при роботі з чотирибітною шиною даних: спочатку старша тетрада, потім молодша тетрада байта даних. Виведення біта даних до регістра зсуву здійснюється через біт переносу за допомогою циклічного зсуву вліво. Після виведення старшої тетради даних її місце в акумуляторі займає молодша, що дає можливість використовувати ту ж підпрограму виведення. Демонстраційний фрагмент програми не передбачає формування часових затримок, що повинні бути присутні при запису інструкцій РКІ.

6.7. Управління перетворювачами електричної енергії

6.7.1. Управління напівпровідниковим перетворювачем постійної напруги

Напівпровідникові перетворювачі постійної напруги здійснюють перетворення постійної напруги одного рівня в постійну напругу іншого рівня. Середнє значення напруги на навантаженні можна регулювати, змінюючи параметри імпульсної напруги (рис. 6.37). Детальніше про принципи перетворення електричної енергії, силові схеми перетворювачів і алгоритми керу-

вання можна дізнатись з літератури та підручників з перетворювальної техніки та силової електроніки, наприклад, з роботи [33]. Залежно від змінної величини розрізняють: широтно-імпульсне регулювання (ШІР) t_I = var, T = const; частотно-імпульсне регулювання (ЧІР) з постійною шириною імпульсу t_I = const, T = var і з постійною шириною паузи $t_П$ = const, T = var.

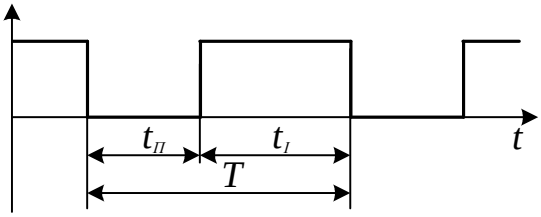


Рисунок 6.37 - Параметри імпульсної напруги

До перетворювача (рис. 6.38) входить джерело живлення, імпульсний ключ (S), фільтр, що виділяє постійну складову напруги, навантаження. Працює перетворювач під керуванням системи управління на основі МК. До системи управління можуть бути підключені зворотні зв’язки за параметрами джерела живлення, струму і напруги навантаження.

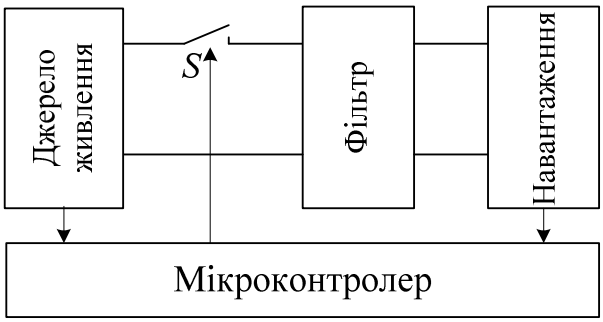


Рисунок 6.38 - Схема підключення перетворювача постійної напруги

В однотактних перетворювачах використовується один керований ключ S. Змінюючи співвідношення між включеним і виключеним станом ключа S можна регулювати середнє значення напруги навантаження $U_Н=U_{ДЖ}*t_I/T$. Схема алгоритму роботи перетворювача з ШІМ наведена на рис. 6.39. На початку роботи перетворювача виконується розрахунок часу включеного стану ключа S t_I. Розрахунок часу імпульсу проводиться за даними зворотних зв’язків і за сигналами керування. Завдання часових інтервалів забезпечується таймером, який завантажується обчисленим числом. До переповнення таймера МК знаходиться у циклі очікування, звідки виходить, як правило, за перериванням. Час знаходження у циклі може використовуватись для вимірювання, попередніх розрахунків, діагностики та індикації. Після закінчення формування імпульсу проводиться обчислення часу паузи і її формування.

Використання обчислень для одержання значень інтервалів часу знижує швидкодію системи. Отримання значення складної функції за невеликий проміжок часу потребує значної продуктивності МК. У цьому випадку доцільно використовувати значення функції і відповідного слова стану ключів, обчислені на етапі створення програми і записані до пам'яті МК. Під час роботи обчислюється поточний номер інтервалу часу і таймер завантажується числом з таблиці, а в порт виводиться відповідне слово стану ключів.

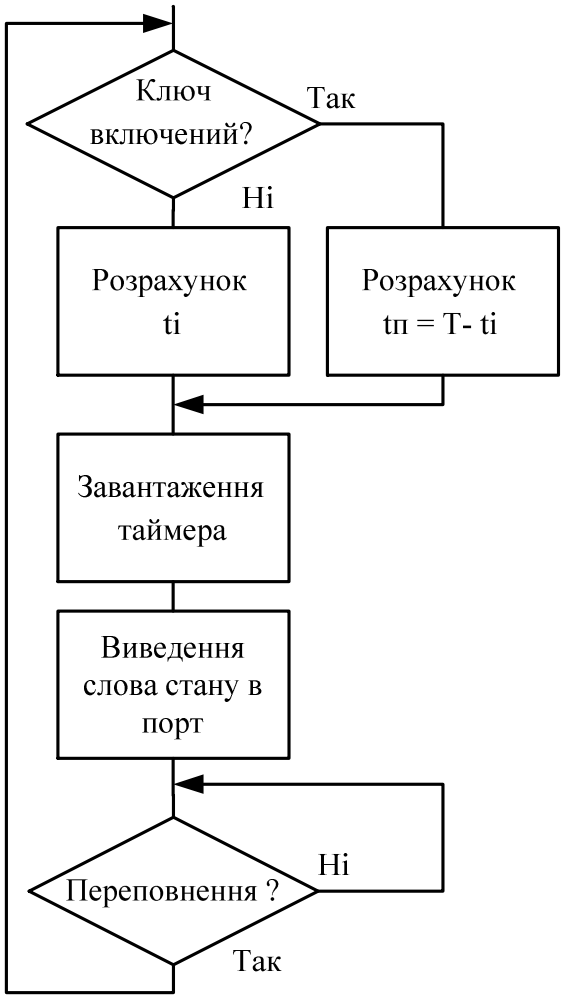


Рисунок 6.39 - Алгоритм роботи перетворювача з ШІМ

У двотактних перетворювачах використовується декілька ключів. Найбільш поширені мостові перетворювачі (рис. 6.40), які використовують чотири ключі (система управління не показана).

Форма напруги, прикладеної до навантаження Н, залежить від алгоритму управління ключами S1 – S4. При включенні ключів попарно (S1 – S4, S2 – S3, S1 – S4 …) одержимо дворівневу криву напруги (рис. 6.41, *а*).

При ввімкненні ключів попарно (S1 – S4, S1 – S3, S2 – S3, S2 - S4, S1 – S4 …) одержимо трирівневу криву напруги (рис. 6.41, *б*).

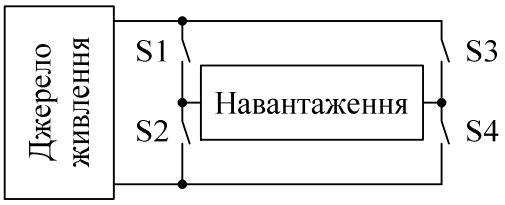
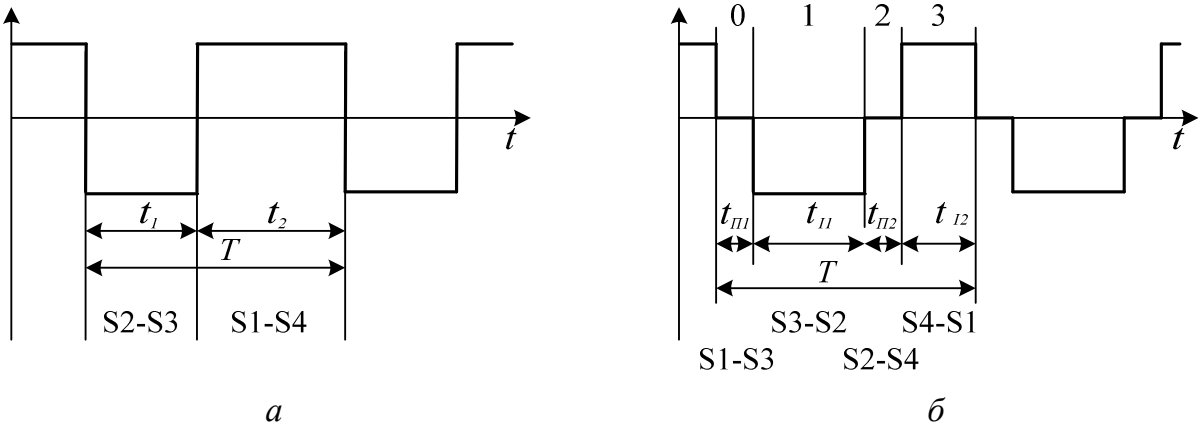


Рисунок 6.40 - Схема мостового перетворювача

У загальному випадку всі часові інтервали можуть бути різними, але звичайно на практиці використовують $t_{п1}= t_{п2}$, а співвідношення часу t_1/t_2 або t_{I1}/t_{I2}, визначає полярність напруги на навантаженні.



а *б*

Рисунок 6.41 - Форма вихідної напруги мостового перетворювача

Врахування особливостей функцій, що формуються перетворювачем, дозволяє істотно скоротити ємність необхідної пам’яті при табличному способі зберігання часових інтервалів і слів стану ключів. Наприклад, при аналізі функції “синус” видно, що півхвилі симетричні відносно точок Т/4 і 3Т/4 відповідно, а для формування негативної півхвилі напруги достатньо інвертувати слово стану ключів для позитивної. Такий підхід дозволяє в 4 рази скоротити необхідну ємність пам’яті, а в програмі потрібно буде контролювати моменти закінчення періоду, півперіоду і чвертьперіодів синусоїди.

Якщо встановлений біт у слові стану ключів відповідає включеному ключу, а порядок бітів збігається з послідовністю ключів S4, S3, S2, S1, то для алгоритму управління, наведеному на рис. 6.41, б, можна використати таку таблицю слів стану ключів, де номер рядка відповідає номеру інтервалу:

DB 0101b

DB 0110b

DB 1010b

DB 1001b

Вільні біти у слові стану ключів можна використати для збереження значення довжини відповідного інтервалу тощо.

6.7.2 Управління однофазним тиристорним випрямлячем

Для живлення нагрівальних елементів невеликої потужності та ряду інших споживачів використовуються однофазні тиристорні випрямлячі (рис. 6.42).

Спосіб імпульсно-фазового управління таким перетворювачем показаний на рис. 6.43. Під час переходу мережевої напруги $u_{\text{мережі}}$ через нуль (моменти часу t_1, t_2, t_3 на рис. 6.43, *а*) формується синхронізуючий імпульс (рис. 6.43, *б*), від якого відраховується кут управління α (рис. 6.43, *г*). Для синхронізації може використовуватися і сигнал, який несе інформацію про полярність напруги $u_{\text{мережі}}$ (рис. 6.43, *в*). У цьому випадку для синхронізації кута α необхідно використовувати передній і задній фронти імпульсу.

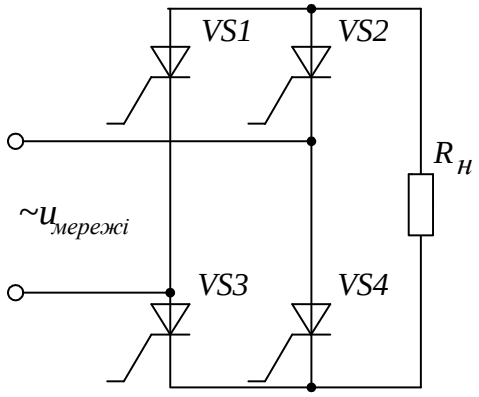


Рисунок 6.42 - Схема включення однофазного тиристорного випрямляча

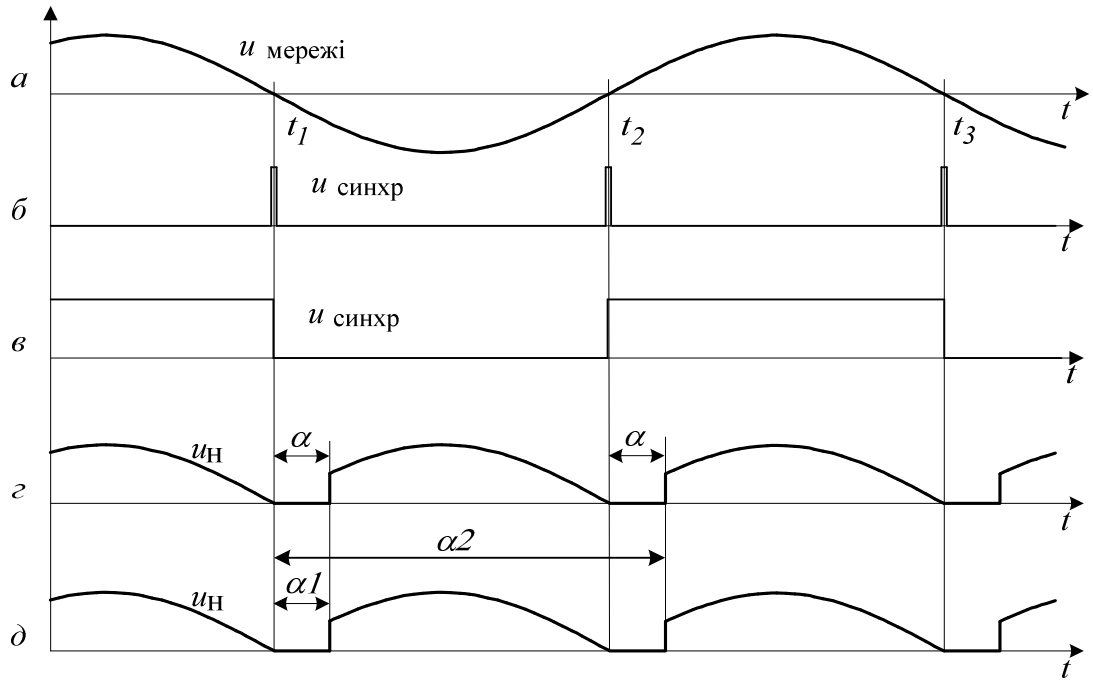


Рисунок 6.43 - Імпульсно-фазове управління перетворювачем

На рис. 6.43, *д* показано варіант одночасного формування двох кутів управління для послідовних півперіодів вхідної напруги з використанням одного синхронізуючого сигналу. Відлік двох кутів управління (α_1 - для першого півперіоду і α_2 - для другого півперіоду) вимагає двох таймерів і застосовується рідко у зв'язку з нестабільністю кута α_2 через коливання частоти напруги мережі.

Алгоритм роботи тиристорного випрямляча з МК-системою імпульсно-фазового управління залежить від моменту формування і виду синхронізуючого сигналу, з якого починається відлік кута управління для кожного півперіоду напруги мережі. Момент обчислення кута управління залежить не тільки від зручності програмування, але і від параметрів системи автоматичного регулювання, реалізованої в перетворювачі, що розробляється. Схему алгоритму роботи системи імпульсно-фазового управління з використанням синхронізуючих імпульсів (рис. 6.43, б) наведено на рис. 6.44.

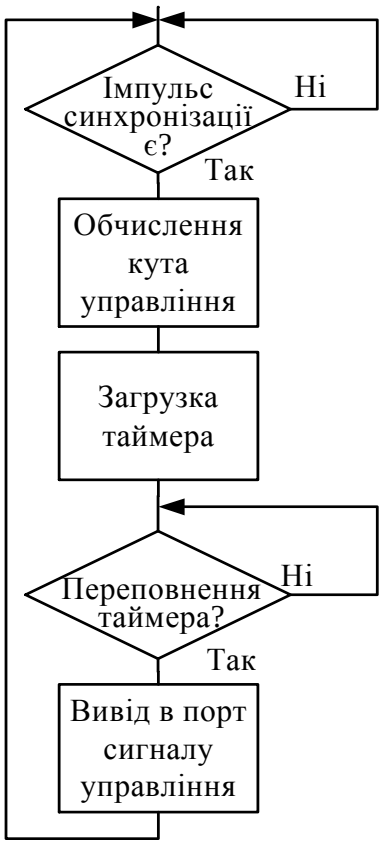


Рисунок 6.44 - Алгоритм роботи тиристорного випрямляча з МК-системою імпульсно-фазового управління

Після детектування імпульсу синхронізації обчислюється кут управління і запускається таймер, завантажений відповідним числом. Очікування переповнення таймера здійснюється під час знаходження МК у циклі очіку-

вання. Після переповнювання таймера в порт виводиться сигнал для формування імпульсу управління тиристорами керованого випрямляча.

Очікування сигналу синхронізації і переповнення таймера може відбуватись контролюванням відповідного біта порту та прапора переповнення (рис. 6.44), або за перериванням. У останньому разі сигнал синхронізації звичайно подається на вхід зовнішнього переривання. Основна програма складається з нескінченного циклу. Обчислення кута управління, завантаження таймера і його пуск відбувається у підпрограмі обробки зовнішнього переривання при детектуванні сигналу синхронізації. Формування сигналу управління тиристорами керованого випрямляча відбувається в підпрограмі обробки переривання переповнення таймера. Після обробки цих переривань програма повертається до нескінченного циклу, де може виконувати виміри, індикацію та ін.

6.7.3. Управління тиристорним ключем змінного струму

Ключ змінного струму, виконаний на симісторах або зустрічно-паралельно включених тиристорах (рис. 6.45) призначений для формування змінної напруги зі змінними параметрами: напругою, частотою. Такі ключі використовуються для пуску двигунів змінного струму, живлення потужних нагрівальних елементів тощо. Вони можуть застосовуватися або як прості однотактні схеми (рис. 6.45, *а*), ябо як двотактні схеми з розширеними функціональними можливостями (рис. 6.45, *б*).

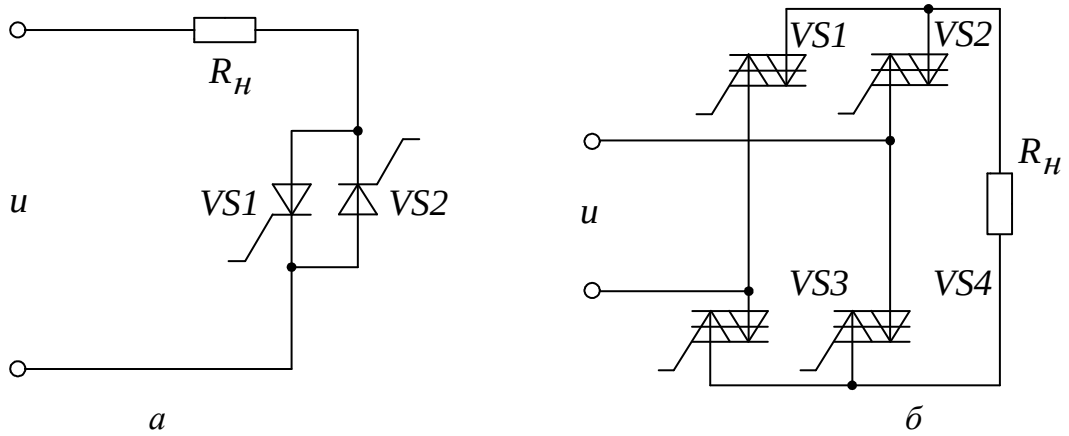


Рисунок 6.45 - Управління тиристорним ключем змінного струму

Форма напруги $u_н$ на навантаженні ключа змінного струму (рис. 6.45, *а*) наведена на рис. 6.46, *а*. Треба мати на увазі, що напруга на навантаженні формується з напруги мережі змінного струму, яку зображено пунктиром. При регулюванні величини напруги кути керування $\alpha 1 - \alpha 4$ обчислюються на підставі сигналів завдання і зворотного зв'язку; у загальному випадку

вони не рівні між собою. Зміною кутів можна регулювати низькочастотну обвідну напруги навантаження ключа змінного струму. Формування синхронізуючих сигналів і принципи отримання кута управління збігаються з аналогічними для керованого випрямляча.

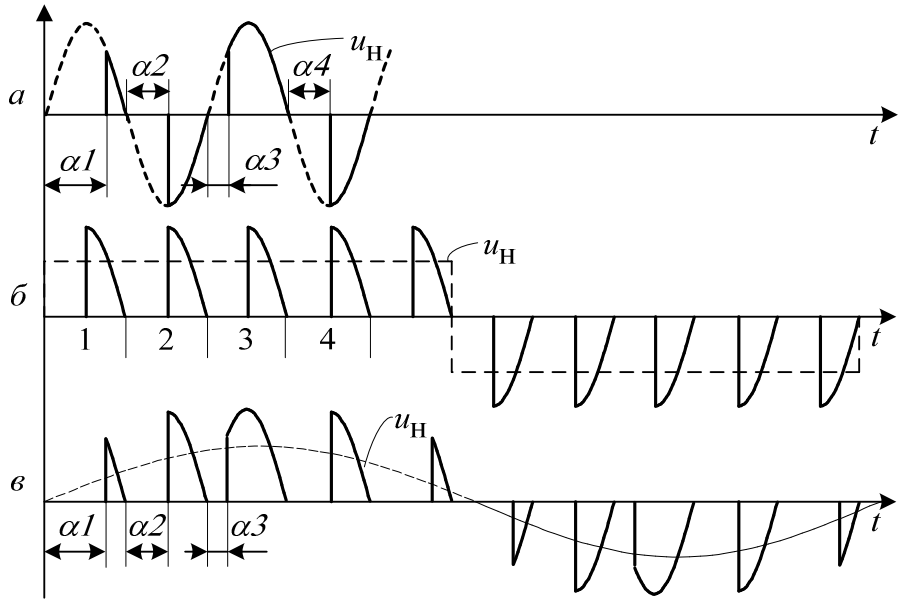


Рисунок 6.46 - Форми напруги на навантаженні тиристорного ключа змінного струму

Використання двотактної схеми ключа змінного струму (рис. 6.45, б) дозволяє регулювати частоту і величину напруги навантаження. Формування кутів керування залежить від вимог до низькочастотної складової змінної напруги на навантаженні (прямокутна обвідна напруги - рис. 6.46, б та синусоїдальна – рис. 6.46, в). На інтервалах 1 і 3 (рис. 6.46, б) працюють ключі VS2, VS3, а на інтервалах 2 і 4 – VS1, VS4, що дозволяє отримувати вихідну напругу постійної полярності із змінної вхідної напруги.

Кут управління залежить від амплітуди та періоду низькочастотної обвідної напруги навантаження і, крім того, від положення поточного півперіоду напруги живлення відносного періоду низькочастотної обвідної напруги навантаження (кути $\alpha 1 - \alpha 3$ на рис. 6.46, в).

6.8. Управління кроковим двигуном

Кроковий двигун (КД) – це синхронна електрична машина, що перетворює електричні імпульси на дискретне переміщення вала. До достоїнств КД слід віднести: однозначний зв’язок переміщення і кількості поданих імпульсів, що дозволяє здійснювати позиціонування без зворотного зв’язку;

повний момент на валу в режимі зупинки за наявності живлення обмоток; можливість роботи на малих швидкостях; висока надійність. До недоліків відноситься можливість втрати контролю положення через відсутність зворотного зв'язку.

Існує два основні типи КД: зі змінним магнітним опором (реактивні КД) і з постійними магнітами (гібридні КД). Двигуни з постійними магнітами діляться на біполярні, до обмоток яких в процесі роботи прикладається напруга змінної полярності, і уніполярні, живлення обмоток яких здійснюється напругою однієї полярності [34]. Схеми і алгоритми управління уніполярними гібридними двигунами і двигунами зі змінним магнітним опором схожі і будуть розглянуті на прикладах.

При подачі напруги живлення на одну з обмоток ротор займає певне положення (положення рівноваги). Для зміни положення ротора необхідно відповідно до алгоритму управління подати напругу на наступну обмотку. Крутний момент на валу пропорційний струму, що протікає через обмотку.

Прикладання зовнішнього моменту до вала двигуна викликає відхилення вала від положення рівноваги. При досягненні зовнішнім моментом величини "утримуючого моменту" ротор перейде в наступне стабільне положення КД. При перевищенні частотою імпульсів, що подаються на обмотки КД, "частоти прийомистості" відбувається пропуск імпульсу і втрата кроку переміщення. Пропуск імпульсу відбувається і при подачі імпульсів управління з частотою, вищою за максимальну робочу частоту КД. Дані процеси приводять до втрати інформації про положення ротора.

Типова залежність частоти управління від моменту на валу двигуна наведена на рис. 6.47. Крива, яка з'єднує точки, що відповідають максимальному пусковому моменту $M_{max\ п}$ і частоті прийомистості $f_{пр}$, показує, при якому максимальному моменті опору, для даної частоти управління, КД здатний рушити (зона старту). Типове значення частоти прийомистості – більше 200 кроків за секунду. Утримуючий момент перевищує максимальний пусковий момент. Зовнішня крива показує, при якому максимальному моменті опору, на даній частоті управління, КД продовжує обертання без пропуску кроків (зона розгону). Значення максимальної частоти досягає 10000 кроків за секунду.

На рис. 6.48, а показана схема управління уніполярним крововим двигуном, у якого дві пари обмоток розміщені на окремих магнітних системах; таку ж схему має чотирифазний реактивний КД. Для управління таким двигуном використовується декілька основних способів, які відрізняються максимальним моментом на валу і величиною кроку.

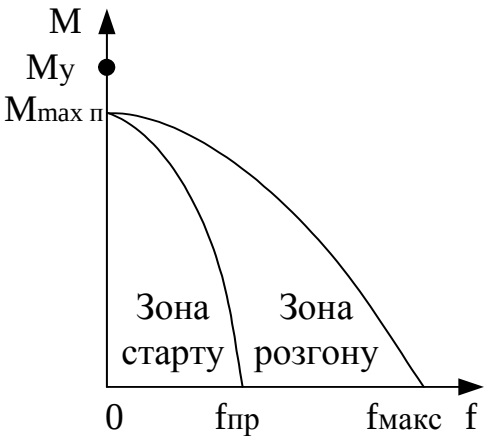


Рисунок 6.47 - Типова залежність частоти управління від моменту на валу двигуна

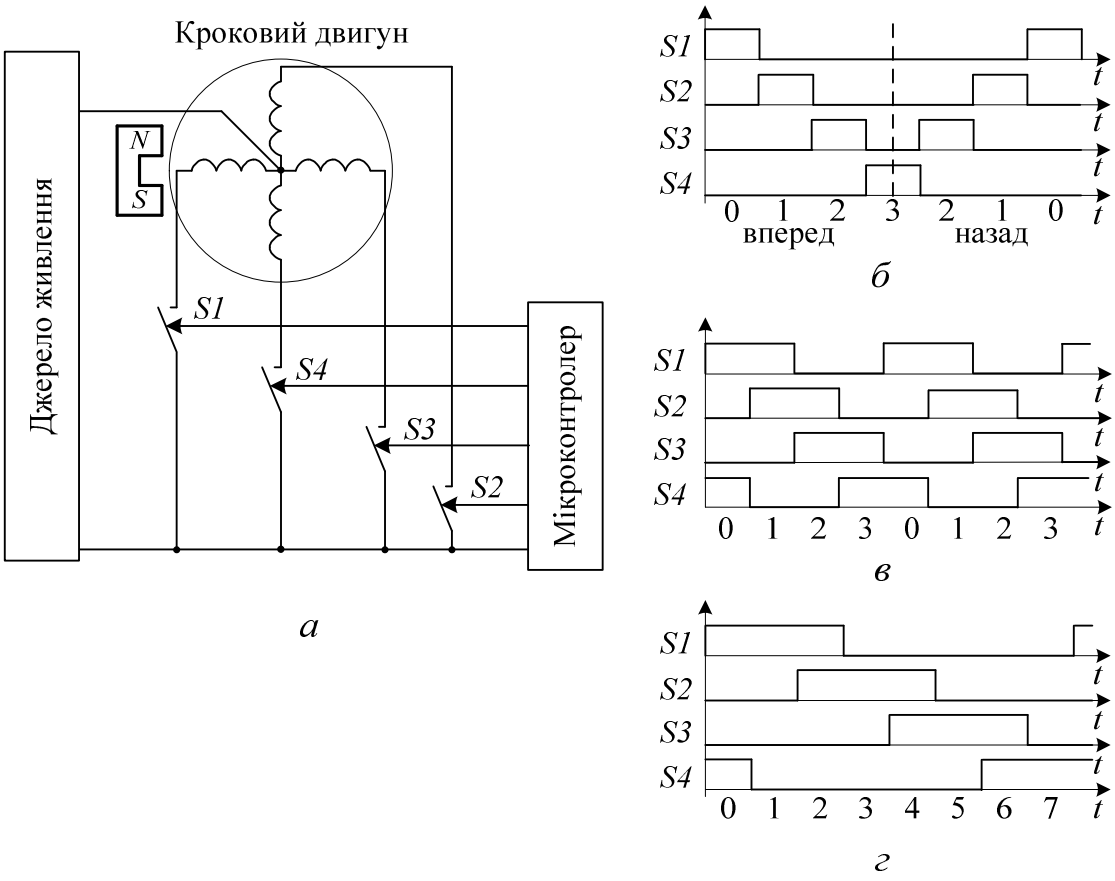


Рисунок 6.48 - Управління уніполярним кроковим двигуном:
а - варіант схеми управління уніполярним кроковим двигуном;. *б* - кроковий спосіб управління; *в* - кроковий спосіб управління з включенням двох фаз; *г* - півкроковий спосіб управління

Перший спосіб характеризується поперемінною комутацією фаз, при цьому в кожен момент часу включена тільки одна фаза (рис. 6.48, б). Точки рівноваги ротора для кожного кроку збігаються з “природними” точками рів-

новаги ротора, які для реактивного двигуна відповідають мінімальному магнітному опору, а для гібридного – мінімальній відстані між полюсами статора і ротора. Момент на валу і крок ротора відповідають паспортним значенням.

Для підвищення моменту на валу двигуна використовується другий спосіб управління, при якому напруга живлення одночасно подана на дві обмотки (рис. 6.48, в). Положення рівноваги ротора займає середнє положення між “природними” точками рівноваги ротора. Момент на валу перевищує паспортне значення, а крок ротора відповідає йому.

Третій спосіб, при якому двигун робить крок вполовину основного кроку, називається півкроковим режимом (рис. 6.48, г). Кожен другий крок напруга живлення подається на одну фазу, а в решті випадків – на дві, в результаті кутове переміщення ротора складає половину кроку для першого способу управління, момент може досягати паспортного значення.

Ускладнення алгоритмів керування КД, пов’язане з використанням ШІМ для формування струму фази двигуна і застосування для живлення КД джерела напруги з більшим, ніж паспортне, значенням, дозволяє розширити функціональні можливості КД [34]:

– поліпшити динамічні характеристики двигуна;
– зменшити споживання енергії;
– здійснювати регулювання розміру кроку, який залежить від струмів, що протікають у включених фазах.

Зміна напрямку руху ротора КД відбувається при зміні напряму руху магнітного поля або, що те ж саме, напрямку зсуву імпульсів управління (рис. 6.48, б).

Процес формування імпульсів управління кроковим двигуном має багато спільного з розглянутим у розділі 6.1.3 процесом формуванням сигналу сканування клавіатури. При формуванні імпульсів управління використовуються операції зсуву числа, встановлення/скидання біта на підставі контролю поточного керуючого слова або читання слова стану ключів з таблиці (розділ 6.8.1) за номером інтервалу (кроку).

Алгоритм формування сигналів управління чотирифазним кроковим двигуном відповідно до рис. 6.48 *г* із зберіганням слів стану ключів у таблиці, наведено на рис. 6.49.

В алгоритмі передбачено одержання даних з таблиці і їх корекція – виділення слова стану (бітів управління), що дозволяє зберігати в таблиці не тільки інформацію про стан ключів, але й допоміжну інформацію. Найчастіше операція виділення бітів управління провадиться шляхом накладання маски за допомогою логічної операції «AND» або «OR».

Безпосереднє виведення в порт слова стану може змінити інформацію на його виводах, що не відносяться до керування КД. Для її збереження необхідно прочитати зміст регістра-фіксатора, очистити місце, що займають біти управління, і записати їх нове значення, після чого інформація записується до порту.

Рисунок 6.49 - Алгоритм формування сигналів управління відповідно до рис. 6.48, *г*

7. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ ПРОГРАМ

7.1. Надійність програмного забезпечення

«...солідна частка підготовчого етапу роботи на ЕОМ іде на усунення помилок, зроблених при складанні програми. За допомогою здорового глузду й підпрограм для налагодження більшість помилок вдається знайти й виправити досить швидко. Однак деякі з них настільки невловимі, що не піддаються виявленню дивно довгий час» [35]. Ця згадка про боротьбу з помилками в програмах для обчислювальних машин, що пролунала в 1952 році, уперше визнає існування проблеми надійності ПО, трудомісткості тестування й налагодження, можливість існування невиявлених помилок.

Розширюючи визначення програмного аспекту проектування МКС, наведене в другому розділі, можна сформулювати основні етапи розробки програмного забезпечення:

1. Визначення мети розробки;
2. Вироблення вимог до програмного забезпечення (специфікацій ПО);
3. Розробка архітектури ПО (високорівневе проектування);
4. Детальне проектування програмних модулів;
5. Кодування;
6. Тестування й налагодження;
7. Супровід системи.

Помилки можуть бути допущені на будь-якому етапі розробки програми, тому навіть робота програми відповідно до вимог, не гарантує відсутності в ній помилок. Найбільш загальне визначення помилки ПЗ дав Г. Майерс: «У програмному забезпеченні є помилка, якщо воно не виконує того, що користувачеві розумно від нього очікувати. Відмова програмного забезпечення – це прояв помилки в ньому» [36].

Характеристиками якості ПЗ є [37]:

1. Відсутність дефектів у специфікації, проекті й реалізації системи.
2. Здатність системи виконувати необхідні функції у визначених умовах, середній інтервал між відмовами.
3. Ступінь безпомилковості виконання вимог специфікації.
4. Здатність продовжувати роботу при введенні неприпустимих даних.
5. Ступінь використання системних ресурсів, у тому числі швидкодія програми й необхідний їй обсяг пам’яті.
6. Можливість виконання блокового й системного тестування.
7. Легкість розуміння системи як на рівні загальної організації, так і на рівні окремих операторів.
8. Можливість внесення змін з метою реалізації додаткової функціональності системи.

Перші чотири характеристики якості указують на відсутність у програмному забезпеченні помилок, інші обумовлюються якістю й стилем проектування ПЗ, у тому числі програмування.

Для підвищення якості ПЗ доцільно явно визначати його пріоритетні характеристики: швидкодію; вимоги до обчислювальних ресурсів процесора та до обсягу використовуваної пам’яті; легкість модифікації або введення додаткових функцій; спрощення тестування програмних модулів і т.д. Частиною контролю якості розроблюваної системи є її детальне тестування на всіх етапах розробки, перевірка відповідності ПЗ базовим принципам програмування [37].

7.2. Тестування програмного забезпечення

Тестування - це процес виконання програми з метою виявлення помилок [38]. На етапі тестування програми виявляються помилки ПЗ, на етапі налагодження - провадиться пошук і усунення їхніх причин. Як відомо, при створенні типового програмного проекту близько 50% загального часу й понад 50% загальної вартості витрачається на перевірку (тестування) розроблюваної програми або системи, а частка власне програмування становить 10-20%.

Для наочного доказу даного твердження нижче наводиться просте завдання, вирішивши яке читачі одержать можливість оцінити свій рівень підготовки.

Необхідно перевірити програму, що обчислює корінь квадратного

рівняння. Одержуючи три числа (A, У та С), вона знаходить два значення X, що задовольняють умові $AX^2+BX+C=0$. Спробуйте розробити систему тестів для перевірки цієї програми. Розглянемо тести, що запропоновані в роботі [39]:

1. A=0, B=0, C=0. Рівняння зводиться до вигляду 0=0 і не може бути розв'язане відносно X.
2. A=0, B=0, C=1. Це відповідає рівнянню 1=0, що не має рішення. Тест на помилкові вхідні дані.
3. A=0, B=2, C=11. Рівняння 2X+11=0 не є квадратним. Може бути виявлена спроба ділення на нуль.
4. A=6, B=1, C=2. Нормальні вхідні дані. Чи обчислений заздалегідь результат для цього тесту?
5. A=3, B=7, C=0. Нормальні вхідні дані. Один з корнів дорівнює нулю.
6. A=3, B=2, C=5. Випадок комплексних корнів.
7. A=7, B=0, C=0. Добування квадратного кореня з нуля.
8. Корисні тести, що перевіряють границі діапазонів арифметичних значень і точність програми.

Наведені приклади тестів показують, що їхня розробка є творчим процесом, що вимагає знання предметної області й бажання знайти помилку.

Всі способи тестування можуть бути розділені на дві категорії: тестування методом чорного ящика й тестування методом білого (прозорого) ящика. У першому випадку тестер не має інформації про внутрішню структуру й принципи дії об'єкта, що перевіряється, у другому внутрішня реалізація об'єкта тестеру відома.

Використання методу чорного ящика або тестування з керуванням по входу-виходу передбачає з'ясування обставин, за яких поведінка програми не відповідає її специфікації (виявлення помилки). Тестові дані використовуються відповідно до специфікації. Для вичерпного вхідного тестування на вхід програми повинні бути подані ВСІ можливі комбінації вхідних даних. Для підвищення ефективності тестів рекомендується використовувати тестові дані, які відповідають граничним умовам, перебувають у безпосередній близькості від них у припустимій області й незначно виходять за припустимі значення. Наприклад, якщо програма розрахована на обробку вхідного цілочисельного значення, що лежить у діапазоні від 12 до 1012, то доцільно подати на її вхід тестові значення 11, 12 і 1012, 1013. Окремо перевіряється точка середини діапазону, що у ряді випадків відповідає нульовому значенню вимірюваного сигналу (датчики змінного сигналу з однополярним поданням результатів вимірів). Для програм, що мають припустиме вхідне

значення «0», для нього складається окремий тест. Якщо подання нуля неоднозначне, то перевіряються всі варіанти. Крім перевірки граничних значень вхідних даних, необхідно скласти тест, що перевіряє функціонування програми на границях вихідних параметрів. Наприклад, для програми формування імпульсів керування ключами керованого випрямляча з діапазоном кутів керування від 0 до 180 електричних градусів необхідно скласти тест, що приводить до появи кутів керування -1, 0 і 180, 181 ел. град. За наявності усередині чорного ящика елементів пам'яті реакція на вхідні дані залежить не тільки від поточних, але й від попередніх значень, що робить побудову вичерпного тесту неможливим з міркувань витрат часу й коштів. Для досягнення максимального результату тестування можливо скоротити набір вхідних даних при використанні знань про внутрішню структуру програми (наприклад, якщо число 2 дорівнює 2, то справедливим буде й те, що число 45 дорівнює 45, при цьому розрядність обчислень дозволяє обробляти всі вхідні дані без переповнення).

Метод білого ящика, або стратегія тестування, керованого логікою програми, дозволяє досліджувати внутрішню структуру програми. Перевіряючи свій код, програміст використовує саме цей метод. Тестові дані утворюються шляхом аналізу логіки роботи програми. Для виявлення всіх помилок необхідно провести вичерпне тестування маршрутів. Програма перевірена повністю, якщо за допомогою тестів удалося виконати всі можливі маршрути її графа передач керування.

Для програми, що містить три умовних оператори й цикл із числом повторень не більше 20 разів, граф передач керування наведено на рис. 7.1.

Число неповторюваних маршрутів із точки А в точку Б дорівнює 5, але, припускаючи, що умови передачі взаємно незалежні, для 20 циклів одержимо $5^{20}+5^{19}+\ldots=10^{14}$ неповторюваних маршрутів, які досить складно протестувати за розумний інтервал часу (при витраті на тестування одного маршруту $1*10^{-6}$ с тестування триватиме 317 років). Крім того, дана стратегія не дозволяє знайти помилки, пов'язані з оброблюваними даними. Наприклад, при аналізі збіжності повинен виконуватися оператор IF (||A|-|B||<Epsilon)..., котрий у результаті помилки був записаний у вигляді IF ((A-B)<Epsilon)..., що приводить до появи помилки, яка залежить від значень змінних А і В, але може не визначатися при тестуванні маршрутів. Перспективною є стратегія покриття рішень, відповідно до якої повинне бути написане таке число тестів, при якому кожний напрямок умовного переходу буде реалізований хоча б один раз. При цьому звичайно виконуються й всі оператори програми.

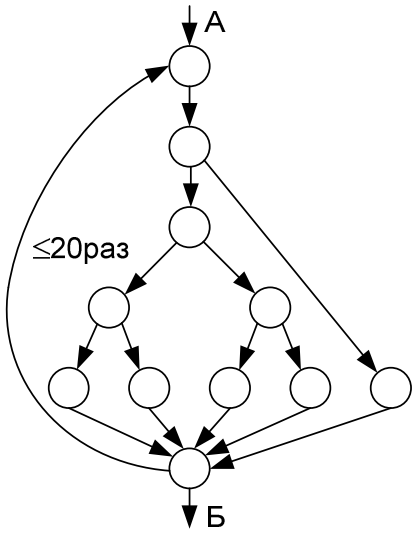


Рисунок 7.1 - Граф передач керування

Хоча тестування з керуванням по входу-виходу краще за тестування внутрішньої будови програми, проте ні те ні інше не можуть бути використані самостійно у зв’язку з їх малою реалізованістю. Найбільш доцільно використовувати для тестування програм їхнє сполучення. Проектування тестів може бути розбите на кілька етапів:

1. Керуючись зовнішніми специфікаціями модуля, підготовлюються тести для кожної ситуації і кожної можливості, для кожної межі областей припустимих значень всіх вхідних даних, областей зміни даних, для всіх неприпустимих умов.
2. По тексту програми перевіряється, що всі умовні переходи будуть виконані в кожному напрямку.
3. Аналізуючи текст програми, переконуються, що тести охоплюють доволі багато можливих шляхів. Наприклад, для кожного циклу повинен бути тест, що відповідає шляху без виконання тіла циклу, з однократним його виконанням і максимальним числом повторень.

Створення тестів і приклади їхнього застосування розглянуто в роботі [38]. Тестування базується на ряді принципів, які підвищують його ефективність:

1. Необхідною частиною тесту повинні бути значення передбачуваних вихідних даних або результатів. Помилкові, але правдоподібні результати можуть бути визнані правильними.
2. Варто уникати тестування програми її автором. Програмістові психологічно складно перешикуватися на пошук помилок у власній програмі, показувати оточуючим їхню наявність. Реалізація даного принципу не завжди можлива у зв’язку з дефіцитом кваліфікованих

програмістів. Виправлення знайдених помилок ефективніше робить автор програми.

3. Програмуюча організація не повинна самостійно здійснювати остаточне тестування розроблених нею програм.
4. Необхідно вивчати результати застосування кожного тесту. Увага, спрямована на пошук певних помилок, може не торкатися другорядних на перший погляд результатів.
5. Необхідно документувати й зберігати тести, що дозволяє повторити тест із даними, які призвели до помилки, наприклад, після її виправлення.
6. Необхідно мати тести для некоректних вхідних даних.
7. Необхідно перевіряти не тільки те, чи робить програма те, для чого вона призначена, але й чи не робить вона те, що не повинна робити. Наприклад, чи немає режимів, у яких одночасно включаються всі ключі напівпровідникового перетворювача, якщо по алгоритму вони повинні працювати по черзі.
8. Імовірність виявлення помилок у частині програми пропорційна числу помилок, уже виявлених у цій частині. Помилки можуть групуватися в модулях, розроблюваних програмістами низької кваліфікації або слабко пророблених ідеологічно.

З розвитком програмування стала превалювати думка, що програми повинні легко читатися і їхнє тестування можна здійснити без застосування ЕОМ. Етап ручного тестування, ефективний для знаходження різнопланових помилок, дозволяє виявити групи помилок. Інспекція вихідного тексту програми здійснюється групою осіб, включаючи автора програми. Метою роботи є знаходження, але не виправлення помилок. Члени групи знайомляться з лістингом програми і з її специфікацією, програміст розповідає про логіку роботи програми й відповідає на виниклі запитання. Програма аналізується для виявлення типових помилок програмування. Виявлені помилки надалі усуваються автором. У критичних випадках, у процесі інспекції тексту, можливе застосування методу мозкового штурму, спрямованого на вирішення проблем, що виникли при виявленні помилок.

Пошук типових помилок спрощується застосуванням конкретних питань, що залежать від виду помилки.

Помилки звертання до даних:

1. Чи існують звертання до змінних, значення яких не привласнені або не ініційовані?
2. Значення індексів при звертанні до масиву перебувають у заданих межах?

Помилки опису даних:

1. Чи всі змінні описані явно?
2. Чи правильно ініційовані масиви й рядки?
3. Чи є змінні з подібними іменами (наприклад, Volt, Volts, volts)? Такі імена - потенційне джерело помилки.

Помилки обчислень:

1. Чи існують обчислення, що використовують дані різної довжини?
2. Чи не менша довжина результату, ніж довжина виразу, що обчислюється?
3. Чи можливе переповнення або втрата проміжного результату при обчисленні виразу?
4. Чи може значення змінної виходити за межі відведеного їй діапазону?

Помилки при порівнянні:

1. Чи порівнюються змінні різних типів?
2. Чи коректні оператори порівняння?
3. Чи кожен логічний вираз сформульовано так, як передбачалося?

Помилки при передачі керування:

1. Чи буде кожен цикл завершений?
2. Чи буде підпрограма завершена?
3. Чи існують рішення, які приймаються за умовчанням? Чи є це припущення правильним?

Помилки інтерфейсу:

1. Чи дорівнює число вхідних параметрів числу аргументів?
2. Чи збігаються одиниці виміру параметрів і аргументів?
3. Чи правильно задане число й порядок проходження аргументів для вбудованих функцій?
4. Чи не змінює підпрограма аргументи, що є тільки вхідними?

Помилки вводу-виводу:

1. Чи відповідає розмір буфера розміру запису?
2. Чи виявляються помилки вводу-виводу?

Інші види контролю:

1. Чи здійснюється контроль правильності вхідних даних?
2. Якщо програма відтрансльована успішно, але компілятор видає попередження, уважно перевірте кожне.

Процес тестування програми й програмних модулів залежить від створення ефективної системи тестів і способу, за допомогою якого з модулів одержують робочу програму. Процес одержання працездатної програми із програмних модулів може здійснюватися двома шляхами: покроковим й

монолітним тестуванням. При монолітному тестуванні провадиться окреме тестування кожного модуля, а потім вони збираються в програму. При покроковому тестуванні модулі, які підлягають перевірці, підключаються до модулів, що раніше тестувались. У цей час кращим вважається покрокове тестування, що дозволяє на ранніх стадіях розробки програми виявляти помилки не тільки в програмних модулях, але й в інтерфейсах між ними. Результати цього тестування більш досконалі, оскільки частини програми, що раніше тестувались, проходять додаткове тестування при кожному модулі, що підключається знову.

Покрокове тестування можна проводити, починаючи з головного модуля програми (спадне тестування) або з нижніх, чи термінальних модулів (висхідне тестування). Простота створення тестових умов і оцінки отриманих результатів, властиві висхідному тестуванню, дозволяє застосовувати його в більшості проектів.

7.3. Процес налагодження

Налагодження програми - це етап розробки комп'ютерної програми, протягом якого виявляються, локалізуються й усуваються явні помилки у програмі [40]. Процес налагодження починається при виявленні помилки. У процесі налагодження визначається природа й місцезнаходження підозрюваної помилки в програмі, помилка фіксується або виправляється. Налагодження вимагає найбільших інтелектуальних витрат програміста, оскільки потенційно помилка може виникнути в будь-якому операторі програми.

Найбільш загальним методом налагодження є трасування - налагоджувальне виконання програми, при якому на екран або на принтер виводяться аргументи й результати виконання кожної команди [40]. До недоліків трасування варто віднести необхідність аналізу великого числа даних, але воно застосовується для окремих ділянок програми або невеликих програм. За допомогою трасування не можна налагоджувати системи реального часу, оскільки часові параметри виконання програми змінюються. Трасування з використанням точок переривання або контрольних точок (breakpoint), дозволяють виконувати програму до настання певної події: виконання певного оператора, зміни значення певної змінної і т. ін. Припинення програми в точці переривання дозволяє проконтролювати поточний стан програми. Використання апаратних засобів налагодження дозволяє трасувати програму в режимі реального часу, оскільки контроль значень регістрів процесора й за-

даних областей пам’яті в точках переривання здійснюється незалежно від виконання програми.

Багаторазове використання трасування без аналізу результатів виконання програми не прискорює процес налагодження програми.

Для локалізації помилки, поряд з методами індукції й дедукції, докладно розглянутими в літературі [36, 38], може використовуватися метод зворотного простежування логіки виконання програми. Налагодження починається в точці програми, у якій був виявлений невірний результат. Для цієї точки за допомогою логіки визначаються значення змінних, які дають такий результат. Подумки виконуючи програму у зворотному порядку й логічно визначаючи значення вхідних змінних модуля за значеннями його вихідних змінних, досить швидко локалізується місце, де логіка роботи програми порушується.

Для уточнення помилки можна використовувати тести, що конкретизують її прояв і визначають, з чим вона пов’язана: із зовнішніми впливами на систему, з викликом сервісних програмних модулів, переповненням у результаті обчислень, переповненням стеку тощо.

Для пошуку помилки звичайно застосовують таку послідовність дій:

1. Стабілізація помилки. Якщо помилка виявляється не завжди, то її складно діагностувати. Необхідно знайти тест або групу тестів, що постійно призводять до помилки.
2. Визначення помилки:
   а) збір даних, що призводять до помилки;
   б) аналіз зібраних даних і формулювання гіпотези, що пояснює помилку;
   в) визначення способу, що підтверджує або спростовує гіпотезу на основі тестування або аналізу тексту програми;
   г) підтвердження або спростування гіпотези.
3. Виправлення помилки.
4. Тестування виправлення.

Діагностика помилки полегшується, якщо вона з’являється регулярно, однак тести, при яких помилка не виявилася, також подають інформацію для аналізу. Непередбачена поява помилки на тих самих тестах, як правило, пов’язана з наявністю неініційованої змінної.

7.3.1. Виправлення помилки

Знайшовши одну помилку, перевірте, чи немає поруч із нею підозрілих фрагментів.

Зміни тексту програми, покликані виправити помилку, можуть міс-

тити нові помилки. У найгіршому разі нові помилки маскують старі і їхня кількість збільшується. За один раз у текст програми можна вносити тільки одну зміну. Після внесення коректив необхідне повторне тестування програми. Якщо передбачуване виправлення помилки усуває не всі її прояви, то помилка не знайдена.

Експериментальна зміна вихідного тексту програми відповідно до ідеї «якщо це змінити (дописати, усунути), то може допоможе», ускладнює налагодження програми, оскільки міняє логіку її роботи, збільшує ймовірність нових помилок.

Перед виправленням помилки збережіть вихідний текст програми з коментарем про поточне поводження програми. До цього тексту можна буде повернутися у випадку невдалих змін, особливо якщо вони торкаються великих фрагментів програми.

7.3.2. Причини помилок

Низький професійний рівень. Однією з найбільш неприємних причин помилок є неповне розуміння програмістом написаної ним програми. Початкуючі програмісти, не володіючи повною мірою використовуваною мовою програмування або знанням архітектури процесора, іноді застосовують програмування методом спроб і помилок. При цьому помилки неминучі й поряд з виправленням помилок потрібно вчитися не допускати їх, удосконалювати вміння програмувати.

Помилки через неуважність. Замість 20h записано 20; не #3, а 3; замість 2 записано 3 тощо. Причиною появи подібних помилок часто є необладнане робоче місце, відволікання уваги на рішення сторонніх проблем, а останнім часом зловживання технологією «скопіювати» - «вставити», при якій провадиться корекція не всіх значень, котрі не збігаються у фрагментах, що копіюються.

Ігнорування документації. Використання фрагментів коду й підпрограм, написаних іншими програмістами, якщо це допускається ліцензійною угодою, дозволяє значно прискорити роботу над проектом. Однак ознайомлення з документацією на використовуваний програмний продукт часто закінчується в момент знаходження першого опису вхідних-вихідних даних, хоча надалі цей опис може уточнюватися й деталізуватися, можуть вказуватися помічені помилки й обмеження.

Стиль програмування. Щоб знайти помилку, програміст повинен читати вихідний текст своєї програми. Якщо програма перенасичена фрагментами, які важко зрозуміти з погляду логіки виконання, то з великою часткою ймовірності саме вони містять помилки.

7.3.3. Інструменти налагодження програмного забезпечення

Одним з найпростіших інструментів налагодження ПЗ є використовуваний компілятор або асемблер. Розробка програм для мікроконтролерів ведеться з використанням крос-компілятора (*cross compiler*) — компілятора, який виробляє код, що виконується, для платформи, відмінної від тієї, на якій виконується сам крос-компілятор (те ж можна сказати й про крос-асемблер). Вони використовуються для одержання коду для платформи, екземплярів якої немає в наявності, або у випадках, коли компіляція на цільовій платформі неможлива або недоцільна. Це, наприклад, стосується мобільних систем або мікроконтролерів з мінімальним обсягом пам'яті [41].

У результаті роботи компілятор видає повідомлення про помилки й попередження. Попередження вказують на місце появи можливої помилки, на некоректне, на думку розроблювачів компілятора, використання конструкцій мови, на невикористовувані змінні й фрагменти коду. Всі попередження необхідно аналізувати як можливу вказівку на помилку.

Компілятор перевіряє синтаксичні помилки в тексті програми. Знайдена помилка може викликати появу декількох повідомлень. Необхідно усунути помилку й повторити компілювання програми. При локалізації помилки в тексті програми компілятор іноді вказує на рядок вищий або нижчий від помилкового, що може бути особливістю його діагностичного модуля.

Більшість сучасних інтегрованих середовищ розробки (IDE - Integrated Development Environment) включає спеціальні засоби налагодження розроблюваних програм. Відлагоджувач (debugger) дозволяє покрокове виконання програми, інспектування й зміну значень змінних.

Керування налагодженням програми здійснюється командами, такими, як «Виконати програму до рядка, у якому зараз перебуває курсор», «Виконати один рядок програми», «Виконати один рядок програми, але якщо усередині цього рядка є виклик функції, то перейти усередину цієї функції». Виконання команд керування закріплюється за спеціальними клавішами.

Запуск налагодження програми здійснюється командою «Почати налагодження» або однією з перерахованих команд. Виконання програми припиняється на її першому рядку.

Команда «Виконати програму до рядка, у якому зараз перебуває курсор» ефективно застосовується при налагодженні програм, що включають тривалі цикли.

Більшість відлагоджувачів містять у собі можливості установки контрольних точок. При досягненні контрольної точки програма зупиниться, з'явиться можливість подивитися (змінити) значення змінних або продовжити покрокове виконання програми.

Значну допомогу при написанні програм можуть подавати інструменти, що виконують профілювання - збір характеристик роботи програми. До них належить профайлер, що дозволяє оцінити час виконання ділянок програми й кількість звернень до різних ділянок програми. Також виділяють аналіз покриття (Code Coverage) - процес виявлення невикористовуваних частин коду за допомогою, наприклад, багаторазового запуску програми. Відповідно до даних профілювання знаходять вузькі місця по швидкодії, приймаються рішення про оптимізацію фрагментів програми, зіставляються часи виконання різних частин програми.

7.4. Особливості тестування й налагодження програмного забезпечення систем, що вбудовуються

У системах реального часу, пов'язаних з роботою зовнішніх пристроїв, необхідно перевіряти функціонування комплексів програм при оперативній динамічній взаємодії всіх їхніх компонентів у процесі обробки різних вихідних даних. Потрібна велика кількість первинних даних і еталонних значень, важливий й час надходження цієї інформації. Для налагодження ПЗ систем реального часу застосовується динамічне тестування (комплексне динамічне налагодження), у ході якого перевіряється виконання програм і обробка вхідних даних з урахуванням часу їхнього надходження, тривалості обробки, пріоритетності, динаміки використання пам'яті й взаємодії з іншими програмами.

Ручна підготовка такої кількості тестів нерентабельна, тому для підготовки тестів і еталонних прикладів часто використовують програмні й апаратні моделі й імітатори. Генерація тестової інформації, функціонування програмного забезпечення й процес обробки результатів налагодження повинні бути синхронізовані. Відповідно ускладнюється перевірка правильності реалізації кожного тестового значення. Оцінки якості програм стають інтегральними й динамічними. При відхиленні результатів виконання програми від тих, що передбачалися, для локалізації помилки необхідно зафіксувати час виявлення відхилення й передісторію надходження тестових даних, після чого переходити до детального тестування. У цих випадках практично незастосовна ручна підготовка тестів і ручна перевірка їх по програмі. Однак лишається високою ефективність інспекцій вихідного тексту програми й ручного тестування.

При динамічному налагодженні систем, що вбудовуються, необхідні автоматичні імітатори стохастичних наборів тестових даних, що зміню-

ються по заданих динамічних законах. Підготовлена тестова інформація повинна містити інформацію про час її одержання комплексом, що налагоджується. Для обробки результатів виконання програм і порівняння їх з еталонними значеннями широко застосовуються автоматичні засоби обробки. У випадку малих витрат часу на генерацію тестів і обробку результатів, можуть використовуватися програмні моделі й логічні аналізатори.

Якщо час на генерацію тестової інформації настільки великий, що не забезпечується необхідний темп її надходження, то інформація, що імітується, переноситься на допоміжні носії, з яких вона може бути легко й швидко передана на комплекс, що налагоджується. При такому способі утруднена імітація зворотних зв'язків і реакції об'єктів зовнішнього середовища на керуючі сигнали.

У деяких випадках можливе застосування старт-стопного способу налагодження, при якому реальний час ураховується тільки при роботі налагоджуваної програми, а на час генерації тестів програма зупиняється. Цей спосіб використовується для перевірки функціонування й швидкодії програмного забезпечення й мало підходить для налагодження реальної апаратури.

Внаслідок складності налагодження систем реального часу, засоби забезпечення автоматизації динамічного тестування можуть бути складнішими, ніж комплекс програм, що тестуються. Рентабельність засобів автоматизації тестування залежить від функціонального призначення створюваного комплексу програм. Так, при відпрацьовуванні програм для керування в авіації й у космосі без таких засобів неможливо одержати програми необхідного рівня якості.

7.5. Інструментальні засоби розробки й налагодження ПО для мікроконтролерів

До числа основних інструментальних засобів налагодження належать:

- внутрісхемні емулятори;
- програмні симулятори;
- оцінні плати;
- монітори налагодження;
- емулятори ПЗП.

7.5.1. Внутрісхемні емулятори

Внутрісхемний емулятор - це програмно-апаратний засіб, здатний заміщати собою процесор, що емулюється, в реальній схемі. Внутрісхемний

емулятор є найбільш потужним й універсальним засобом налагоджування. Він робить процес функціонування контролера, що налагоджується, прозорим, тобто легко контрольованим, довільно керованим і таким, що підтримує модифікацію в процесі виконання програми.

Для розширення функціональності й зниження ціни внутрісхемні емулятори можуть використовувати обчислювальні потужності зовнішньої обчислювальної машини. Автономні внутрісхемні емулятори мають індивідуальні обчислювальні ресурси, засоби вводу-виводу, не вимагають для своєї нормальної роботи яких-небудь зовнішніх обчислювальних засобів.

Звичайно стикування внутрісхемного емулятора з системою, що налагоджується, провадиться за допомогою кабелю зі спеціальним перехідним модулем. Перехідний модуль вставляється замість мікроконтролера в систему, що налагоджується. Якщо мікроконтролер неможливо видалити з системи, то використання емулятора можливе, тільки якщо цей мікроконтролер має режим, при якому всі його виводи перебувають у третьому стані. У цьому випадку для підключення емулятора використовують адаптер, що підключається безпосередньо до виводів мікроконтролера.

Як правило, емулятор містить такі функціональні блоки:

- налагоджувач;
- вузол емуляції мікроконтролера (програмний або програмно-апаратний);
- пам’ять емулятора;
- логічний аналізатор;
- профайлер (аналізатор ефективності програмного коду);
- інтегроване середовище розробки.

Налагоджувач є інтерфейсом між розроблювачем і налагоджувальним засобом. Він дозволяє:

- завантажувати до пам’яті системи програму , що налагоджується;
- виводити на монітор вміст всіх регістрів і комірок пам’яті і, при необхідності, модифікувати їх;
- управляти процесом емуляції;
- вести символьне налагодження. Багато компіляторів можуть генерувати налагоджувальну інформацію, завдяки чому налагоджувач підставляє замість адрес - імена символьних змінних, масивів і структур, що використовувалися у вихідному тексті програми. При цьому користувач оперує прийнятними для людини символьними іменами, а не значеннями їхніх адрес;

- контролювати й аналізувати не тільки дизасембльований текст, але й вихідний текст програми, на якому за допомогою виділення рядка показується поточна виконувана команда.

Такий налагоджувач дозволяє користувачеві одночасно контролювати хід виконання програми й бачити відповідність між вихідним текстом, образом програми в машинних кодах і станом всіх ресурсів мікроконтролера, що емулюється.

Наявність пам'яті емулятора дає можливість використовувати її в процесі налагодження замість ПЗП системи, що налагоджується. При необхідності внесення змін в програму досить завантажити нову або модифіковану програму до пам'яті емулятора, замість перепрограмування ПЗП (функція емулювання ПЗП). Пам'ять являє собою ОЗП зі схемою керування, розташований на платі емулятора.

Логічний аналізатор працює синхронно із процесором і фіксує потік виконуваних інструкцій і стани обраних зовнішніх сигналів, що дозволяє провести аналіз роботи системи в часовій області. У ряді випадків можлива синхронна фіксація стану внутрішніх ресурсів мікроконтролера, наприклад, регістрів.

Інтегроване середовище розробки, що входить до складу поставки внутрісхемного емулятора, являє собою сукупність програмних засобів, що підтримує всі етапи розробки програмного забезпечення від написання вихідного тексту програми до її компіляції й налагодження. Інтегроване середовище забезпечує взаємодію з програматором відповідного мікроконтролера й програмним симулятором. Наявність інструментів взаємодії з програматором і симулятором є відмітною рисою програмного забезпечення, орієнтованого на роботу з програмами для мікроконтролерів, однак самі програматор і симулятор, які можуть бути самостійними комерційними продуктами, часто входять у комплект поставки у вигляді оцінних версій.

7.5.2. Програмні симулятори

Симулятор - це програмний засіб, здатний імітувати роботу мікроконтролера і його пам'яті. Як правило, симулятор містить у своєму составі:

- налагоджувач;
- модель центрального процесорного пристрою й пам'яті.
- моделі вбудованих периферійних пристроїв, таких, як таймери, порти, АЦП, системи переривань.

Симулятор повинен вміти завантажувати файли програм у всіх популярних форматах, максимально повно відображати інформацію про стан ре-

сурсів мікроконтролера, що симулюється, а також надавати можливості по симуляції виконання завантаженої програми в різних режимах. У процесі налагодження модель “виконує” програму, і на екрані комп’ютера відображається поточний стан моделі.

Завантаживши програму в симулятор, користувач має можливість запускати її в покроковому або безперервному режимах, задавати умовні й безумовні контрольні точки, контролювати й вільно модифікувати вміст комірок пам’яті й регістрів мікроконтролера, що симулюється. За допомогою симулятора можна швидко перевірити логіку виконання програми, правильність виконання арифметичних операцій.

Деякі симулятори можуть містити ряд додаткових програмних засобів, наприклад: інтерфейс зовнішнього середовища; вбудоване інтегроване середовище розробки.

У реальній системі мікроконтролер звичайно займається зчитуванням інформації з підключених зовнішніх пристроїв, обробкою цієї інформації й видачею керуючих впливів на виконавчі пристрої. Щоб у симуляторі, що не має інтерфейсу зовнішнього середовища, змоделювати роботу датчика, потрібно вручну змінювати поточний стан моделі периферійного пристрою, до якого в реальній системі підключений датчик. Якщо, наприклад, при прийманні байта через послідовний порт встановлюється прапор, а сам байт попадає в певний регістр, то обидві ці дії потрібно робити в такому симуляторі вручну. Наявність же інтерфейсу зовнішнього середовища дозволяє користувачеві створювати й гнучко використовувати модель зовнішнього середовища мікроконтролера, що функціонує і взаємодіє з програмою, яка відлагоджується, по заданому алгоритму.

Очевидною особливістю програмних симуляторів є те, що виконання програм, завантажених у симулятор, відбувається в масштабі часу, відмінному від реального. Однак низька ціна, можливість проведення налагодження навіть в умовах відсутності макета пристрою, що налагоджується, роблять програмні симулятори досить ефективним засобом налагоджування. Окремо необхідно підкреслити, що існує цілий клас помилок, які можуть бути виявлені тільки за допомогою симулятора.

7.5.3. Налагоджувальні монітори

Налагоджувальний монітор - це спеціальна програма, яка завантажується до пам’яті системи, що налагоджується. Вона змушує процесор корис-

тувача виконувати, крім прикладного завдання, ще й функції відлагоджування:

- завантаження прикладних кодів користувача у вільну від монітора пам’ять;
- установку контрольних точок;
- запуск і зупинку завантаженої програми в реальному часі;
- прохід програми користувача по кроках;
- перегляд, редагування вмісту пам’яті й керуючих регістрів.

Програма монітора обов’язково повинна працювати із зовнішнім комп’ютером або пасивним терміналом, на яких і відбувається візуалізація й керування процесом налагодження. Перевагою використання налагоджувального монітора є дуже малі витрати при збереженні можливості провадити налагодження в реальному часі. Головним недоліком є відволікання ресурсів мікроконтролера на налагоджувальні процедури: монітор займає деякий об’єм пам’яті, переривання, послідовний канал передачі даних. Обсяг ресурсів, що відволікаються, залежить від мистецтва розроблювача монітора. Останнім часом з’явилися вироби, які практично не займають апаратних ресурсів процесора.

7.5.4. Оцінні плати

Оцінні плати (Evaluation Boards) є своєрідними конструкторами для макетування прикладних систем. Останнім часом, при випуску нової моделі кристала мікроконтролера, фірма-виробник обов’язково випускає й відповідну оцінну плату. Звичайно це друкована плата із встановленим на ній мікроконтролером і всіма необхідними йому зовнішніми компонентами. На цій платі також встановлюють схеми зв’язку із зовнішнім комп’ютером. Як правило, там же є вільне поле для монтажу прикладних схем користувача й рекомендованих фірмою додаткових пристроїв, таких, як ПЗП, ОЗП, РКІ-дисплей, клавіатура, АЦП тощо.

Для більшої зручності оцінні плати комплектуються ще й найпростішими засобами налагоджування на базі монітора налагоджування. Однак тут виявилися два різних підходи: один використовується для мікроконтролерів, що мають зовнішню шину, а другий - для мікроконтролерів, що не мають зовнішньої шини.

У першому випадку налагоджувальний монітор поставляється фірмою у вигляді мікросхеми ПЗП, що вставляється в спеціальну розетку на оцінній

платі. Плата також має ОЗП для програм користувача й канал зв’язку із зовнішнім комп’ютером або терміналом.

У другому випадку оцінна плата має вбудовані схеми програмування внутрішнього ПЗП мікроконтролера, які управляються від зовнішнього комп’ютера. У цьому випадку програма монітора просто заноситься в ПЗП мікроконтролера разом із прикладними кодами користувача. Прикладна програма при цьому повинна бути спеціально підготовлена: у потрібні її місця вставляють виклики підпрограм монітора налагоджування. Потім здійснюється виконання програми. Щоб внести в програму виправлення, користувачеві треба стерти ПЗП й зробити повторний запис. Готову прикладну програму одержують із налагодженої шляхом видалення всіх викликів функцій монітора і самого монітора відлагоджування.

7.5.5. Емулятори ПЗП

Емулятор ПЗП - програмно-апаратний засіб, що дозволяє заміщати ПЗП на платі, що налагоджується, підставляючи замість нього ОЗП, у який може бути завантажена програма з комп’ютера через один зі стандартних каналів зв’язку. Цей пристрій дозволяє користувачеві уникнути багаторазових циклів перепрограмування ПЗП. Емулятор ПЗП є сенс застосовувати тільки для мікроконтролерів, які можуть звертатися до зовнішньої пам’яті програм; він за складністю й вартістю порівнянний з оцінними платами.

Ранні емулятори ПЗП дозволяли тільки завантажувати програму, запускати її й зупиняти, використовуючи загальне скидання. Потім з’явилися ускладнені моделі з апаратною генерацією сигналів синхронізації при досягненні певної адреси. Пам’ять у таких виробах доступна для перегляду й модифікації.

7.6. Програмні засоби, що включають засоби налагодження ПО

При розгляді програмних засобів, що включають засоби налагодження програм для мікроконтролерів сімейства MCS-51, зупинимося на програмних симуляторах. Програмні симулятори входять до складу всіх комерційних інтегрованих середовищ розробки [42] і поставляються окремо як у вигляді вільно розповсюджуваного програмного забезпечення [43], так і у вигляді комерційних продуктів [44].

Як приклад розглянемо засіб розробки Pinnacle 52, що розповсюджується на умовах *shareware* і має демонстраційну версію, доступну на сайті

розроблювача. Інтегроване середовище розробки Pinnacle включає вбудований асемблер і ґрунтується на концепції проекту – сукупності файлів, що містять всю інформацію про розроблювальний пристрій і дії по проектуванню й налагодженню програми. До такої інформації належить тип мікроконтролера, його тактова частота, вихідний текст програми (програмних модулів), написаний мовою асемблера, бінарне подання програми, таблиця адрес іменованих змінних і міток, список ресурсів мікроконтролера, що спостерігалися в останньому сеансі налагодження, відкриті інтерфейси периферійних пристроїв і т.д. Файли з вихідними текстами програми (модулів) можуть мати імена, що, як правило, пояснюють функції даного модуля і не збігаються з назвою проекту. Загальноприйнятим є тільки розширення ім'я файла *.asm. Рекомендуються всі файли проекту зберігати в одній директорії.

Робота в інтегрованому середовищі починається зі створення нового проекту, для цього слід вибрати в меню опцію Projekt New. У вікні Projekt Filename «…» вибирається директорія, у якій будуть зберігатися файли проекту й вказується ім'я проекту (файл із розширенням .pmk, наприклад, serial.pmk). Інтегроване середовище розробки має у своєму составі текстовий редактор, що використовується для написання тексту програми. Текст програми може бути набраний у новому файлі, що створюється через меню File – New і що повинен бути збережений з розширенням .asm, наприклад, main.asm. Можна використовувати існуючий файл (File - Open). Для включення нового або існуючого файлу в проект, необхідно відредагувати проект. Для цього необхідно вибрати Projekt - Edit - +Include. Вибрати необхідний файл із розширенням .asm і підтвердити свій вибір. У вікні Component Files показуються імена всіх файлів, які підключені до проекту.

Після написання програми виконується побудова всього проекту (Projekt – Build Projekt). При необхідності може бути відкомпільований і налагоджений файл, що перебуває в активному вікні текстового редактора (опція Projekt - Compile&Link); створення проекту при цьому не потрібне. Перевага проекту в тім, що він може містити кілька програмних модулів і дозволяє організувати розробку системи, а не тільки компіляцію й трасування програми.

Знайдені помилки документуються у вікні Output із вказівкою імені файла, номера рядка, у якому знайдена помилка, і її короткий опис. При подвійному клацанні мишею на рядку з описом помилки покажчик середовища розробки переходить до тексту програми на рядок з помилкою. При відсут-

ності помилок, у вікні Output вказується розмір бінарного коду програми.

Виконання програми (пункт меню Execute) може здійснюватися покроково, із заходом у процедури, із пропуском виконання кроку. Можливе виконання програми з використанням контрольних точок.

Для полегшення налагодження програми передбачена можливість інспектування й зміни всіх внутрішніх ресурсів мікроконтролера. Відкрити вікно, у якому відображається стан ресурсу можна через меню View; доступні окремі вікна: для регістрів – Registrs; для портів – Ports; для таймерів – Timers і т.д. Особливу увагу варто приділити вікну Code Memory (Disassembly), у якому власне й провадиться налагодження програми – вказується поточна виконувана інструкція, її адреса, мітки, коментарі, встановлюються контрольні точки.

Редагуючи властивості проекту Projekt – Projekt Options, можна задати тип мікроконтролера із сімейства MCS-51, його частоту, визначити параметри ряду моделей зовнішніх пристроїв, які підтримуються симулятором: клавіатури, пристроїв з інтерфейсом I^2C тощо.

Для зручності налагодження додатків, що використовують послідовний канал передачі даних, до складу симулятора введений емулятор термінала, підключений до UART мікроконтролера. Виведення інформації на термінал здійснюється автоматично, а введення - подвійним клацанням мишею на вікні термінала.

Широкі можливості має симулятор Topview Simulator індійської компанії Frontline electronics, оцінна версія якого доступна на сайті виробника [45]. Цільове призначення симулятора – використання в освітніх установах. Він орієнтований на симуляцію мікроконтролерів сімейства MCS-51, що виробляються фірмою Atmel, підтримує зовнішню програму асемблера, багато моделей периферійних пристроїв, включаючи різні типи індикаторів.

СПИСОК ЛІТЕРАТУРИ

1. Толковый словарь по вычислительным системам / Под ред. В. Иллингоута и др.: Пер. с англ. А.К. Белоцкого и др.; Под ред. Е.К. Масловского. – М.-: Машиностроение, 1990.- 560 с.
2. http://en.wikipedia.org/wiki/Harvard_Mark_I
3. http://en.wikipedia.org/wiki/Harvard_architecture
4. http://en.wikipedia.org/wiki/ENIAC
5. http://en.wikipedia.org/wiki/EDVAC
6. http://en.wikipedia.org/wiki/Von_Neumann_architecture
7. http://en.wikipedia.org/wiki/John_von_Neumann
8. *First Draft of a Report on the EDVAC* by John von Neumann – Contract No.670-ORD-4926. Between the United States Army Ordnance Department and the University of Pennsylvania. Moore School of Electrical Engineering, University of Pennsylvania, June 30, 1945. http://www.virtualtravelog.net/entries/ 2003-08-TheFirstDraft.pdf
9. http://en.wikipedia.org/wiki/Princeton_architecture
10. ГОСТ 19.201-78 ЕСПД. Техническое задание. Требования к содержанию и оформлению
11. Юров В.И. Assembler: Учеб. для вузов. 2-е изд. – СПб.: Питер, 2007.- 640 с.
12. http://www.gnu.org/licenses/licenses.html#GPL
13. Перспективы развития вычислительной техники: В 11 т.: Справ. Пособ. / Под ред. Ю.М. Смирнова. Т.6: Специализированные ЭВМ / Ю.М. Смирнов, Г.Н. Воробьев. – М.: Высш. шк., 1989. - 144 с.
14. ГОСТ 19.102-77 ЕСПД. Стадии разработки
15. Intel. “MCS(R) 51 Microcontroller Family User’s Manual” Document order nomber: 272383-002 February 1994 ftp://download.intel.com/design/MCS51/MANUALS/27238302.pdf
16. Intel. “80C51FA/83C51FA event-control CHMOS single-chip 8-bit microcontroller” Document order nomber: 270501-008 June 2004 ftp://download.intel.com/design/MCS51/datashts/27050108.pdf
17. Intel. “80C31BH/80C51BH/87C51 MCS-51 CHMOS single-chip 8-bit microcontroller” Document order nomber: 270419-008 June 2004 ftp://download.intel.com/design/MCS51/datashts/27041908.pdf

18. John Wharton “An introduction to the INTEL MCS-51(R) single-chip microcomputer family” INTEL Application note AP-69 1980 ftp://download.intel.com/design/MCS51/applnots/01502a01.pdf
19. http://plit.de/asem-51/
20. Проектирование цифровых устройств на однокристальных микроконтроллерах/ В.В. Сташин, А.В. Урусов, О.Ф. Мологонцева. – М.: Энергоатомиздат, 1990. – 224 с.
21. http://www.national.com/ds.cgi/DC/ADC0801.pdf
22. http://www.national.com/ds.cgi/DC/ADC0831.pdf
23. http://www.renesas.com/media/products/lcd/discontinued_products/d_e780u.pdf
24. Rick Schue Interfacing the Densitron LCD to the 8051 AB-39 INTEL 1996 ftp://download.intel.com/design/MCS51/applnots/27052903.pdf
25. http://www.elementinc.com/project.html
26. Parallax Serial LCD http://www.parallax.com/dl/docs/prod/audiovis/SerialLCD-v2.0.pdf,
27. MT serial LCD controller http://www.mcu.hk/Doc/sLCD.pdf
28. Integrated circuits data sheet. PCF8531 34 128 pixel matrix driver. Product specification. http://www.standardics.nxp.com/products/pcf/pdf/pcf8531.pdf
29. GLK 12232-25 User Manual http://www.matrix-orbital.com
30. http://www.milinst.com/Audio_Visual/lcd-text.htm#scott
31. http://www.seetron.com/slcds.htm
32. Myke Predko’s 2-Wire LCD Interface using the PIC16C84 http://www.rentron.com/Myke1.htm
33. Зиновьев Г.С. Основы силовой электроники: Учеб. пособие. – Изд. 2-е, испр. и доп. – Новосибирск: Изд-во НГТУ, 2003.-664 с.
34. Кенио Т. Шаговые двигатели и их микропроцессорные системы управления: Пер. с англ.- М.: Энергоатомиздат, 1987. – 200 с.
35. Brooker R.A., Gill S., Wheeler D.J. The adventures of a blinder, Mathematical tables and other aids of computation, 6 (38), 112-113 (1952)
36. Майерс Г. Надежность программного обеспечения. - М.: Мир, 1980.-356 с.
37. Макконнелл С. Совершенный код. - Спб.: Питер, 2006.-896 с.
38. Майерс Г. Искусство тестирования программ. - М.: Финансы и статистика, 1982.-176 с.

39. Gruenberger F. Program testing and validation. DATAMATION, 14(7), 39-47, 1968.
40. slovari.yandex.ru
41. http://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81-%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80
42. http://www.keil.com
43. http://home.arcor.de/jensaltmann/jsim-e.htm
44. https://www.vaultbbs.com/pinnacle
45. http://www.frontline-electronics.com
46. 80C51/87C51/80C52/87C52 Product specification. http://www.nxp.com/acrobat/datasheets/8XC51_8XC52_6.pdf
47. AT89S51 http://www.atmel.com/dyn/resources/prod_documents/doc2487.pdf

Додаток А
Електричні параметри мікроконтролерів сімейства MCS-51

Таблиця А.1 - Електричні параметри мікроконтролерів (для номінальної напруги живлення)

Назва параметра	Марка мікроконтролера (фірма-виготовлювач)		
	80C31BH 80C51BH (Intel) [17]	80C51 87C51 (Philips) [46]	AT89S51 (Atmel) [47]
Температура навколишнього середовища	Від –40 °С до +125 °С	Від –40 °С до +85 °С	Від –55 °С до +125 °С
Номінальна напруга живлення, Vcc	5В ± 10%	Від 2,7В до 5,5В	Від 4,0В до 5,5В
Напруга живлення, Vcc	Від 2,2В до 5,5В	Від 2,7В до 5,5В	Від 2,0В до 5,5В
Вхідна напруга низького рівня, V_{IL}	Менше 0,2*Vcc – 0,25В	Менше 0,7В при Vcc<4В, (0,2*Vcc+0,9В) при Vcc>4В	Менше 0,2*Vcc – 0,1В
Вхідна напруга високого рівня, V_{IH}	Більше 0,2*Vcc+1В	Більше 0,7Vcc	Більше 0,2*Vcc+0,9В
Вихідна напруга низького рівня (порт 1, 2, 3), V_{OL}	0,45В при I_{OL}=1,6 мА	0,4В при I_{OL}=1,6 мА	0,45В при I_{OL}=1,6 мА
Вихідна напруга низького рівня (порт 0), V_{OL1}	0,45В при I_{OL}=3,2 мА	0,4В при I_{OL}=3,2 мА	0,45В при I_{OL}=3,2 мА
Вихідна напруга високого рівня (порт 1, 2, 3) V_{OH}	2,4В при I_{OH}=–60мкА	Vcc–0.7В при Vcc=4,5В I_{OH}=–30мкА	2,4В при Vcc=5В I_{OH}=–60мкА
Вихідна напруга високого рівня (порт 0) V_{OH1}	2,4В при I_{OH}=–800мкА	Vcc–0.7В при Vcc=2,7В I_{OH}=–3,2мА	2,4В при Vcc=5В I_{OH}=–800мкА
Максимальний втікаючий ток виводу, I_{OL}	10мА	15мА	10мА
Максимальний втікаючий ток 8-бітного порту 0, I_{OL}	26мА	26мА	26мА
Максимальний втікаючий ток 8-бітного порту 1, 2 або 3, I_{OL}	15мА	26мА	15мА
Максимальний втікаючий ток всіх портів, I_{OL}	71мА	71мА	71мА
Частота синхронізації	Від 3,5МГц до 12 або 16МГц	Від 0МГц до 16 або 33МГц	Від 0МГц до 33МГц

Додаток Б
Мікроконтролери сімейства MCS-51

Таблиця Б.1 – Функціональні можливості мікроконтролерів виробництва NXP (Philips)

Позначення										
Корпус		DIP40, PLCC44, QFP44	PLCC44	TSSOP28, PLCC28	TSSOP28, PLCC28	TSSOP16	SO8	TSSOP20	DIP40, PLCC40, TQFP40
Максимальна частота, МГц		33	18	12	12	12	IRC[4]	20	33
Переривання Кількість (зовнішні) /пріоритетів		6(2)/2	17(3)/4	15(3)	15(3)	13(3)	9(1)	12(3)/4	7(2)/4
Компаратор					2			2	
АЦП Бітів/Каналів			10/8	10/8	2x10 /4	8/4		8/4	
Спеціальні можливості		2.7-5.5V	-		Два АЦП			АЦП/ ЦАП	3V
Послідовні інтерфейси		UART	UART, I²C	UART, I²C, SPI	UART, I²C, SPI	UART, I²C	UART	UART, I²C	SPI
Лінії вводу/виводу		32	42	26	26	14	6	18	32
Таймер/ лічильник	WD[3]	-	+	+	+	+	+	+	+
	CCU[2]	-	-	+	+	+	-	-	+
	PWM[1]	-	2	1	1	1	-	1	1
	кількість	3	4	4	4	4	4	2	4
Пам'ять	RAM, байт	128/ 256	512	768	768	256	128	128	1K
	EEPROM, байт		8	512	512				
	FLASH, Кбайт	4/8		8	8	2	1	ROM 8	64
80C51/87C51/80C52/87C52									
P89LPC952									
P89LPC938									
P89LPC935									
P89LPC917									
P89LPC908									
87LPC779									
P89LV51RD2									

[1] – широтно-імпульсний модулятор; [2] - режим захват/порівняння; [3] – сторожовий таймер;

[4] – вбудований RC-генератор.

Таблиця Б.2 – Функціональні можливості мікроконтролерів виробництва фірми Atmel

Позначення	Пам'ять			Таймер/лічильник	Лінії вводу/виводу	Послідовні інтерфейси	Спеціальні можливості	АЦП, Бітів/Каналів	Максимальна частота, МГц	Корпус
	FLASH, Кбайт	EEPROM, Кбайт	RAM, байт							
AT80251G2D			1024	3	32	UART, SPI, TWI	2.7-5.5 В		24	PDIP 40 LQFP 44 Plastic J-Lead 44
AT80C51RD2			1280		32	UART, SPI	2.7-5.5 В		60	PDIP 40 LQFP 44 LQFP 64
AT89C51ID2	64	2	2048	3	32	UART, SPI	2.7-5.5 В		60	LQFP 44 Plastic J-Lead 44
AT89C51CC01	32	2	1280	2		UART, CAN	CAN		40	CBGA 64 LQFP 44 Plastic J-Lead 44
AT89C5122	32					USB	Smart Card Reader		16	LQFP 64 Plastic J-Lead 28
AT89C51SND1C	64	4	2.3K			UART, SPI, TWI, USB	Mp3 2.7-3.3 В	8/2		CBGA 81 LQFP 80
AT89C51SND2C	64					UART, SPI, TWI, USB	Mp3 2.7-3.3 В	ЦАП 18/2		CBGA 100
AT89C5132	64		2304		44	UART, SPI, TWI, USB	4 Function Endpoints			LQFP 80

Додаток В
Система команд мікроконтролерів сімейства MCS-51

Таблиця В.1 – Команди, що впливають на прапори

Команди	Прапори		
	Переносу **C**	Переповнення **OV**	Додаткового переносу **AC**
ADD	+	+	+
ADDC	+	+	+
SUBB	+	+	+
MUL	0	+	
DIV	0	+	
DA	+		
RRC	+		
RLC	+		
SETB C	1		
CLR C	0		
CPL C	+		
ANL C, bit	+		
ANL C, /bit	+		
ORL C, bit	+		
ORL C, /bit	+		
MOV C, bit	+		
CJNE	+		

Примітка: операції з регістром PSW або з його бітами, що прямо адресуються, також впливають на стан прапорів.

Таблиця В.2 – Умовні позначення, використані при опису команд

Умовні позначення	Опис умовних позначень
Rn	Регістри R7-R0 в поточному банку регістрів
direct	8-бітна адреса внутрішніх даних (адреса комірки вбудованого ОЗП або адреса РСФ)
@Ri	8-бітна адреса комірки вбудованого ОЗП, що непрямо адресується за допомогою регістра R0 або R1
#data	8-бітна константа
#data 16	16-бітна константа
addr 16	16-бітна адреса. Перехід у межах 64 Кбайт адресного простору.
addr 11	11-бітна адреса. Перехід у межах 2 Кбайт адресного простору починаючи з початку поточної команди
rel	Число в діапазоні -128 to +127
bit	Адреса біта, що адресується прямо у вбудованому ОЗП або РСФ

Таблиця В.3 - Команди арифметичних операцій

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
ADD	A, Rn	Додавання регістра до акумулятора	1	1	(A)←(A)+(Rn)
ADD	A,direct	Додавання байта, що адресується прямо, до акумулятора	2	1	(A)←(A)+(direct)
ADD	A, @Ri	Додавання байта, що адресується непрямо, до акумулятора	1	1	(A)←(A)+((Ri))
ADD	A, #data	Додавання константи до акумулятора	2	1	(A)←(A)+ #data
ADDC	A, Rn	Додавання регістра до акумулятора з урахуванням переносу	1	1	(A)←(A)+(Rn)+(C)
ADDC	A,direct	Додавання байта, що адресується прямо, до акумулятора з урахуванням переносу	2	1	(A)←(A)+(direct)+(C)
ADDC	A, @Ri	Додавання байта, що адресується непрямо, до акумулятора з урахуванням переносу	1	1	(A)←(A)+((Ri))+(C)
ADDC	A, #data	Додавання константи до акумулятора з урахуванням переносу	2	1	(A)←(A)+#data+(C)
SUBB	A, Rn	Віднімання з акумулятора регістра і позики	1	1	(A)←(A)-(Rn)-(C)
SUBB	A,direct	Віднімання з акумулятора байта, що адресується прямо, та позики	2	1	(A)←(A)-(direct)-(C)

Таблиця В.3 - Команди арифметичних операцій (продовження)

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
SUBB	A,@Ri	Віднімання з акумулятора байта, що адресується непрямо, та позики	1	1	(A)←(A)-((Ri))-(C)
SUBB	A, #data	Віднімання з акумулятора константи та позики	2	1	(A)←(A)- # data -(C)
INC	A	Збільшення акумулятора	1	1	(A)←(A)+1
INC	Rn	Збільшення регістра	1	1	(Rn)←(Rn)+1
INC	direct	Збільшення байта, що адресується прямо	2	1	(direct)←(direct)+1
INC	@Ri	Збільшення байта, що адресується непрямо	1	1	((Ri))←((Ri))+1
DEC	A	Зменшення акумулятора	1	1	(A)←(A) -1
DEC	Rn	Зменшення регістра	1	1	(Rn)←(Rn) -1
DEC	direct	Зменшення байта, що адресується прямо	2	1	(direct)←(direct) -1
DEC	@Ri	Зменшення байта, що адресується непрямо	1	1	((Ri))←((Ri)) -1
INC	DPTR	Збільшення покажчика даних	1	2	(DPTR)←(DPTR)+1
MUL	AB	Множення акумулятора на регістр B	1	4	(B)(A)←(A)*(B)
DIV	AB	Ділення акумулятора на регістр B	1	4	(A).(B)←(A)/(B)
DA	A	Десяткова корекція акумулятора	1	1	Якщо (A_{0-3}) >9 або ((AC)=1), то (A_{0-3})←(A_{0-3})+6; потім, якщо (A_{4-7}) >9 або ((C)=1), то (A_{4-7})← (A_{4-7})+6

Таблиця В.4 - Команди логічних операцій

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
ANL	A,Rn	Логічне І акумулятора та регістра	1	1	(A)←(A) AND (Rn)
ANL	A, direct	Логічне І акумулятора та байта, що адресується прямо	2	1	(A)←(A) AND (direct)
ANL	A, @Ri	Логічне І акумулятора та байта, що адресується непрямо	1	1	(A)←(A) AND ((Ri))
ANL	A, #data	Логічне І акумулятора та константи	2	1	(A)←(A) AND #data
ANL	direct, A	Логічне І байта, що адресується прямо, та акумулятора	2	1	(direct)←(direct) AND (A)
ANL	direct,#data	Логічне І байта, що адресується прямо, та константи	3	2	(direct)←(direct) AND #data
ORL	A, Rn	Логічне АБО акумулятора та регістра	1	1	(A)←(A) OR (Rn)
ORL	A, direct	Логічне АБО акумулятора та байта, що адресується прямо	2	1	(A)←(A) OR (ad)
ORL	A, @Ri	Логічне АБО акумулятора та байта, що адресується непрямо	1	1	(A)←(A) OR ((Ri))
ORL	A, #data	Логічне АБО акумулятора та константи	2	1	(A)←(A) OR #data
ORL	direct, A	Логічне АБО байта, що адресується прямо, та акумулятора	2	1	(direct)←(direct) OR (A)
ORL	direct, #data	Логічне АБО байта, що адресується прямо, та константи	3	2	(direct)←(direct) OR #data
XRL	A,Rn	Виключне АБО акумулятора та регістра	1	1	(A)←(A) XOR (Rn)

Таблиця В.4 - Команди логічних операцій (продовження)

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
XRL	A, direct	Виключне АБО акумулятора та байта, що адресується прямо	2	1	(A)←(A) XOR (direct)
XRL	A, @Ri	Виключне АБО акумулятора та байта, що адресується непрямо	1	1	(A)←(A) XOR ((Ri))
XRL	A, #data	Виключне АБО акумулятора та константи	2	1	(A)←(A) XOR #data
XRL	direct, A	Виключне АБО байта, що адресується прямо, та акумулятора	2	1	(direct)←(direct) XOR (A)
XRL	direct, #data	Виключне АБО акумулятора та константи	3	2	(direct)←(direct) XOR #data
CLR	A	Скидання акумулятора	2	1	(A)←0
CPL	A	Інвертування акумулятора	2	1	(A)←NOT (A)
RL	A	Зсув акумулятора ліворуч циклічно	3	1	$(A_{n+1})\leftarrow(A_n)$, n=0÷6, $(A_0\leftarrow(A_7)$
RLC	A	Зсув акумулятора ліворуч циклічно через біт переносу	1	1	$(A_{n+1})\leftarrow(A_n)$, n=0÷6, $(A_0\leftarrow(C)$, $(C)\leftarrow(A_7)$
RR	A	Зсув акумулятора праворуч циклічно	1	1	$(A_n)\leftarrow(A_{n+1})$, n=0÷6, $(A_7\leftarrow(A_0)$
RRC	A	Зсув акумулятора праворуч циклічно через біт переносу	1	1	$(A_n)\leftarrow(A_{n+1})$, n=0÷6, $(A_7)\leftarrow(C)$, $(C)\leftarrow(A_0)$
SWAP	A	Обмін місцями тетрад в акумуляторі	1	1	$(A_{0-3})\leftrightarrow(A_{4-7})$

Таблиця В.5 - Команди передачі даних

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
MOV	A, Rn	Пересилання регістра в акумулятор	1	1	(A)←(Rn)
MOV	A, direct	Пересилання в акумулятор байта, що адресується прямо	2	1	(A)←(direct)
MOV	A,@Ri	Пересилання в акумулятор байта, що адресується непрямо	1	1	(A)←((Ri))
MOV	A,#data	Завантаження в А константи	2	1	(A)←#data
MOV	Rn, A	Пересилання в регістр із А	1	1	(Rn) ←(A)
MOV	Rn, direct	Пересилання в регістр байта, що адресується прямо	2	1	(Rn)←(direct)
MOV	Rn, #data	Завантаження в регістр константи	2	1	(Rn) ←(#data)
MOV	direct, Rn	Пересилання за прямою адресою регістра	2	2	(direct) ← (Rn)
MOV	direct, A	Пересилання за прямою адресою А	2	1	(direct) ← (A)
MOV	direct,direct	Пересилання байта, що адресується прямо, за прямою адресою	3	2	(direct)←(direct)
MOV	direct,@Ri	Пересилання, що адресується непрямо, за прямою адресою	2	2	(direct) ←((Ri))
MOV	direct, #d	Пересилання за прямою адресою незмінної	3	2	(direct) ←#d

Таблиця В.5 - Команди передачі даних (продовження)

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
MOV	@Ri, A	Пересилання в байт, що адресується непрямо, з А	1	1	((Ri)))←(A)
MOV	@Ri, direct	Пересилання в байт, що адресується непрямо, байта, що адресується прямо	2	2	((Ri)))← (direct)
MOV	@Ri, #data	Пересилання в РПД константи	2	1	((Ri)))← #data
MOV	DPTR,#data16	Завантаження покажчика даних	3	2	(DPTR)← #data16
MOVC	A, @A + + DPTR	Пересилання в акумулятор байта із пам’яті програм	1	2	(A)←((A) + + (DPTR))
MOVC	A,@A+PC	Пересилання в акумулятор байта із пам’яті програм	1	2	(PC) ← (PC)=1 (A)←((A)+ (PC))
MOVX	A,@Ri	Пересилання в акумулятор байта із зовнішньої пам’яті даних	1	2	(A) ← ((Ri))
MOVX	A, @DPTR	Пересилання в акумулятор байта із зовнішньої пам’яті даних	1	2	(A)←((DPTR))
MOVX	@Ri, A	Пересилання в зовнішню пам’ять даних із акумулятора	1	2	((Ri)) ←(A)
MOVX	@DPTR, A	Пересилання в зовнішню пам’ять даних із акумулятора	1	2	((DPTR))←(A)

Таблиця В.5 - Команди передачі даних (продовження)

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
PUSH	direct	Завантаження в стек байта, що адресується прямо	2	2	(SP)←(SP) + 1 ((SP))←(direct)
POP	direct	Вивантаження із стека до байта, що адресується прямо	2	2	(direct)←(SP) (SP)←(SP) - 1
XCH	A, Rn	Обмін акумулятора з регістром	1	1	(A)↔(Rn)
XCH	A, direct	Обмін акумулятора з байтом, що адресується прямо	2	1	(A)↔ (direct)
XCH	A, @Ri	Обмін акумулятора з байтом байта, що адресується непрямо	1	1	(A)↔((Ri))
XCHD	A,@Ri	Обмін молодшої тетради акумулятора з молодшою тетрадою байта, що адресується непрямо	1	1	$(A_{0-3})\leftrightarrow((Ri)_{0-3})$

Таблиця В.6 - Команди операцій з бітами

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
CLR	C	Скидання біта переносу	1	1	(C)←0
CLR	bit	Скидання біта	2	1	(bit) ←0
SETB	C	Встановлення біта переносу	1	1	(C)←1
SETB	bit	Встановлення біта	2	1	(bit) ←1
CPL	C	Інверсія біта переносу	1	1	(C)←NOT(C)
CPL	bit	Інверсія біта	2	1	(bit) ← NOT(bit)
ANL	C, bit	Логічне І біта переносу і біта	2	2	(C)←(C) AND (bit)
ANL	C, /bit	Логічне І біта переносу і інверсії біта	2	2	(C)←(C) AND NOT(bit)
ORL	C, bit	Логічне АБО біта переносу і біта	2	2	(C)←(C) OR (bit)
ORL	C, /bit	Логічне АБО біта переносу і інверсії біта	2	2	(C)←(C) OR NOT(bit)
MOV	C, bit	Пересилання біта в біт переносу	2	1	(C)←(bit)
MOV	bit, C	Пересилання біта переносу в біт	2	2	(bit) ←(C)

Таблиця В.7 - Команди передачі управління

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
LCALL	addr16	Довгий виклик підпрограми	2	2	(PC) ← (PC) + 3, (SP) ← (SP) + 1, ((SP)) ← (PC_{0-7}), (SP) ← (SP) + 1, ((SP)) ← (PC_{8-15}), (PC) ← addr16
ACALL	addr11	Абсолютний виклик підпрограми	3	2	(PC) ← (PC) + 2, (SP) ← (SP) + 1, ((SP)) ← (PC_{0-7}), (SP) ← (SP) + 1, ((SP)) ← (PC_{8-15}), (PC_{0-10}) ← addr11
RET		Повернення з підпрограми	1	2	(PC_{8-15}) ← ((SP)), (SP) ← (SP) - 1, (PC_{0-7}) ← ((SP)), (SP) ← (SP) - 1
RETI		Повернення з підпрограми обробки переривання	1	2	(PC_{8-15}) ← ((SP)), (SP) ← (SP) - 1, (PC_{0-7}) ← ((SP)), (SP) ← (SP) - 1
LJMP	addr 16	Довгий перехід в повному обсязі пам`яті програм	3	2	(PC) ← addr 16
AJMP	addr 11	Абсолютний перехід усередині сторінки в 2 кбайти	2	2	(PC)←(PC) + 2 (PC_{0-10})←ad 11
SJMP	rel	Короткий відносний перехід усередині сторінки в 256 байтів	2	2	(PC)←(PC) + 2 (PC)←(PC) + re1

Таблиця В.7 - Команди передачі управління (продовження)

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
JMP	@(A+ DPTR)	Непрямий відносний перехід	1	2	(PC)←(A) + (DPTR)
JZ	rel	Перехід, якщо акумулятор дорівнює нулю	2	2	(PC)←(PC) + 2; якщо (A) = 0, то (PC)←(PC)+ re1
JNZ	rel	Перехід, якщо акумулятор не дорівнює нулю	2	2	(PC)←(PC) + 2; якщо (A) ≠ 0, то (PC)←(PC)+re1
JC	rel	Перехід, якщо біт переносу встановлено	2	2	(PC)←(PC) + 2; якщо (C) = 1, то (PC)←(PC) + re1
JNC	rel	Перехід, якщо біт переносу скинуто	2	2	(PC)←(PC) + 2; якщо (C) = 0, то (PC)←(PC) + re1
JB	bit, rel	Перехід, якщо біт дорівнює одиниці	3	2	(PC)←(PC) + 3; якщо (bit) = 1, то (PC)←(PC) + re1
JNB	bit, rel	Перехід, якщо біт дорівнює нулю	3	2	(PC)←(PC) + 3; якщо (bit) = 0, то (PC)←(PC) + re1
JBC	bit, rel	Перехід, якщо біт встановлений, з наступним скиданням біта	3	2	(PC)←(PC) + 3; якщо (bit) = 1, то (bit)←0 та (PC)←(PC) + re1
CJNE	A, direct, rel	Перехід, якщо акумулятор не дорівнює байту, що прямо адресується	3	2	(PC)←(PC) + 3; якщо A ≠ (direct), то (PC)←(PC) + re1; якщо A < (direct), то (C) ← 1, інакше (C) ← 0

Таблиця В.7 - Команди передачі управління (продовження)

Команда	Операнди	Назва команди	Довжина, байтів	Час виконання, циклів	Операція
CJNE	A, #data, rel	Перехід, якщо акумулятор не дорівнює константі	3	2	(PC)←(PC) + 3; якщо A ≠ #data, то (PC)←(PC) + re1; якщо A < #data, то (C) ← 1, інакше (C) ← 0
CJNE	Rn, #data, rel	Перехід, якщо регістр не дорівнює константі	3	2	(PC)←(PC) + 3; якщо Rn ≠ #data, то (PC)←(PC) + re1; якщо Rn < #data, то (C) ← 1, інакше (C) ← 0
CJNE	@Ri, #data, rel	Перехід, якщо непрямо адресований байт не дорівнює константі	3	2	(PC)←(PC) + 3; якщо ((Ri)) ≠ #data, то (PC)←(PC) + re1; якщо ((Ri)) < #data, то (C) ← 1, інакше (C) ← 0
DJNZ	Rn, rel	Декремент регістра та перехід, якщо не нуль	2	2	(PC)←(PC) + 2; (Rn)←(Rn) – 1; якщо (Rn) ≠ 0, то (PC)←(PC) + re1
DJNZ	direct, rel	Декремент байта, що адресується прямо, та перехід, якщо не нуль	3	2	(PC)←(PC)+2, (ad)←(ad) -1, якщо (ad)≠0,то (PC)←(PC) + re1
NOP		Без операції	1	1	(PC)←(PC) +1

ЗМІСТ

Навчальне видання

Сокол Євген Іванович

Домнін Ігор Феліксович

Рисований Олександр Миколайович

Замаруєв Володимир Васильович

Єресько Олександр В'ячеславович

СПЕЦІАЛІЗОВАНІ МІКРОКОНТРОЛЕРНІ СИСТЕМИ: ТЕОРІЯ І ПРАКТИКА

Підручник

Редактори: *О.С. Саминіна, Л.Л. Яковлева*

План 2005 р., поз. 58

Підписано до друку 18.06.2007 р. Формат 60х84 $^{1}/_{16}$. Папір офісний. Riso-друк. Гарнітура Таймс. Ум. друк. арк. 14,6. Обл.-вид. арк. 16,2. Наклад 300 прим. Зам. № 327. Ціна договірна.

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 116 від 10.07.2000 р.
61002, Харків, вул. Фрунзе, 21.

Друкарня НТУ «ХПІ». 61002, Харків, вул. Фрунзе, 21.

СПЕЦІАЛІЗОВАНІ МІКРОКОНТРОЛЕРНІ СИСТЕМИ Теорія і практика

У підручнику розглянуто факти з історії виникнення сучасних обчислювальних приладів, наведено принципи організації мікроконтролерних систем, особливості проектування пристроїв та їх використання, розглянуто систему команд і засоби програмування мікроконтролерів сімейства MCS-51, організацію взаємодії мікроконтролерів з об'єктами управління, дано рекомендації щодо тестування та налагодження програм. Матеріал підручника ілюстровано великою кількістю прикладів програм, що написані мовою асемблера.

Призначено для студентів, які вивчають навчальні дисципліни «Спеціалізовані мікроконтролерні системи» та «Мікроконтролерні системи», а також може бути корисним усім, хто займається розробкою та експлуатацією сучасної електронної техніки.

ISBN 978-966-593-551-3

9 789665 935513

Друкарня НТУ «ХПІ»