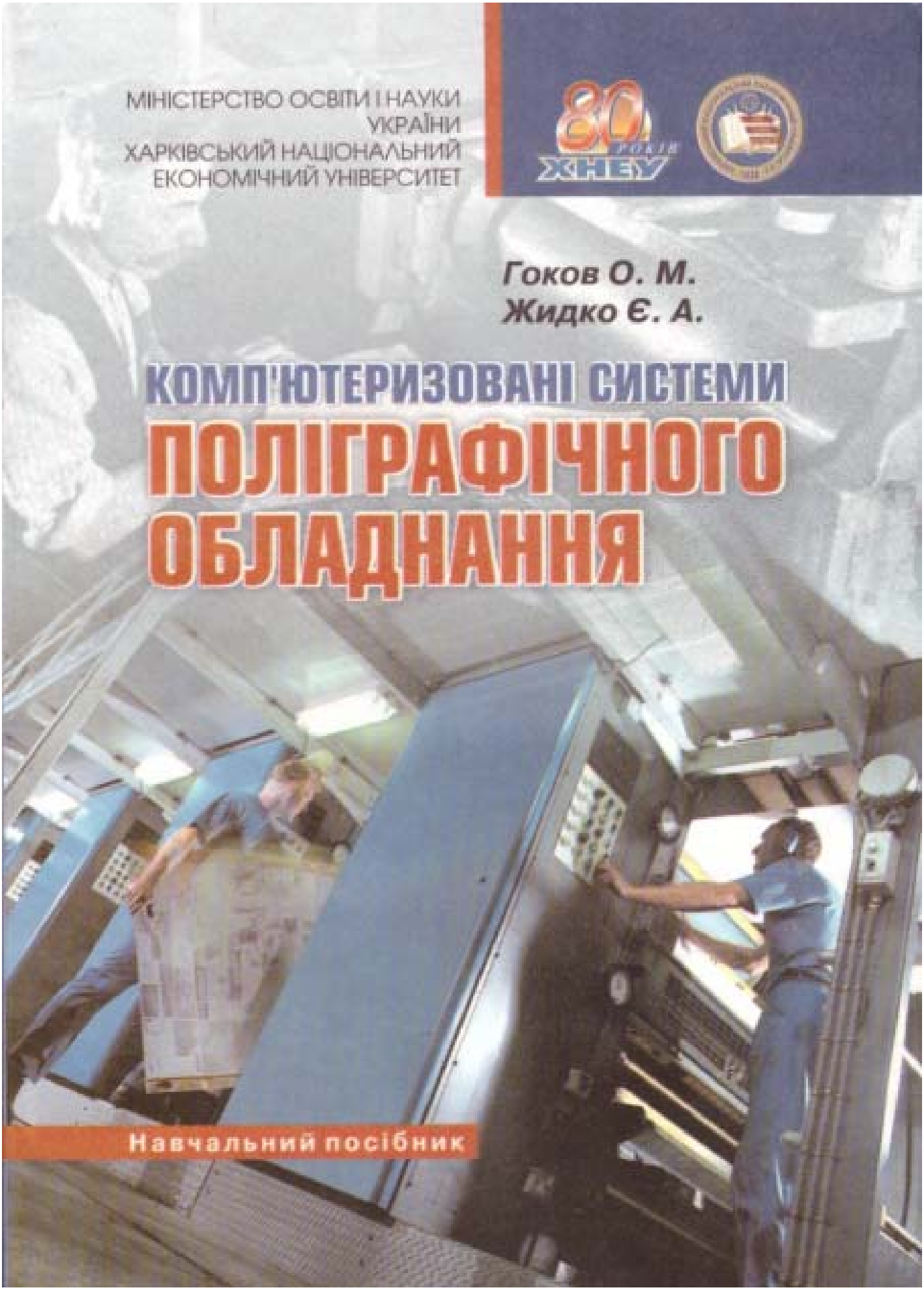
МІНІСТЕРСТВО ОСВІТИ І НАУКИ
УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Гоков О. М.
Жидко Є. А.

КОМП'ЮТЕРИЗОВАНІ СИСТЕМИ ПОЛІГРАФІЧНОГО ОБЛАДНАННЯ



Навчальний посібник

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Гоков О. М. Жидко Є. А.

КОМП'ЮТЕРИЗОВАНІ СИСТЕМИ ПОЛІГРАФІЧНОГО ОБЛАДНАННЯ

Навчальний посібник

.

Харків, Вид. ХНЕУ, 2009

УДК 004.9:655(075.8)
ББК 37.8я73
Г59

Рецензенти: докт. техн. наук, професор кафедри експериментальної фізики Харківського національного університету імені В. Н. Каразіна *Пойда В. П.*; канд. фіз.-мат. наук, професор, зав. кафедри космічної радіофізики Харківського національного університету імені В. Н. Каразіна *Тирное О. Ф.*; канд. фіз.-мат. наук, ст. науковий співробітник Інституту сцинтиляційних матеріалів НАН України *Катрунов К. О.*

Рекомендовано до видання рішенням вченої ради Харківського національного економічного університету.
Протокол № 9 від 24.02.2009 р.

Гоков О. М.

Г59 Комп'ютеризовані системи поліграфічного обладнання : навчальний посібник / О. М. Гоков, Є. А. Жидко. – Харків : Вид. ХНЕУ, 2009. – 244 с. (Укр. мов.)

Викладено питання, пов'язані з технічним і програмним забезпеченням сучасних комп'ютеризованих систем поліграфічного обладнання. Розглянуто загальні принципи побудови і функціонування комп'ютеризованих систем та вузлів обладнання, обміну інформацією в сучасних системах управління поліграфічним обладнанням і роботи основних функціональних вузлів мікропроцесорів; архітектуру, типові функції та особливості виконання контролерів AVR, алгоритми роботи його типових вузлів і програмне забезпечення; принципи алгоритмічного та програмного забезпечення мехатронних систем; основи написання і виконання програм.

Рекомендовано для студентів, що навчаються за профілем "Видавничо-поліграфічна справа", а також для тих, хто вивчає однойменну дисципліну за іншими профілями підготовки.

ISNB 978-966-676-352-8

УДК 004.9:655(075.8)
ББК 37.8я73

Вступ

Добре відомо, що без залучення інформаційних ресурсів не можуть бути вирішені ніякі завдання творення і не дивно, що останнім часом стрімкими темпами розвиваються нові технології в областях, безпосередньо пов'язаних з інформацією. Значні зміни відбулися і в підходах до рішення проблем виробництва друкарської продукції. Постійно зростаючі вимоги ринку до якості, вартості і доступності друкарських засобів інформації, привели до того, що сьогодні в розвитку і створенні нового високопродуктивного обладнання і технологій друку стали визначальними автоматизація, комп'ютерні технології, інновації, тісно пов'язані з тим, що називають «високими технологіями».

Аналіз літератури про сучасне поліграфічне обладнання дозволив сформувати базові відмітні ознаки, які були покладені в основу концепції даного навчального посібника:

1. Процеси випуску поліграфічної продукції і обладнання, що бере участь у цьому, в даний час нерозривно пов'язані з системним підходом. Обладнання, застосовуване для виробництва друкарських засобів інформації, є цілеорієнтованою системою і являє собою сукупність елементів і процесів, що знаходяться у відносинах і взаємозв'язках між собою, створюючих єдине ціле. Такі системи, створюючі певну керовану цілісність, обов'язково мають в своєму складі схему управління, що складається з частини системи і підлягає управлінню, і частині, що управляє і яка це управління виробляє.

Для сучасного етапу розвитку поліграфічного обладнання є характерним те, що в ньому використовуються високоточні системи автоматики, керовані від комп'ютерів і спеціалізованих обчислювальних систем. У поліграфічному обладнанні, як в системі, можна виділити два канали – силовий і інформаційний. Важливу роль в обладнанні відіграють виконавчі механізми, що входять до складу поліграфічних машин і призначені, як правило, для здійснення механічних рухів.

2. Сучаснє поліграфічне обладнання в більшій своїй частині є мехатронним. Поняття мехатроніка (слово складається з двох частин – механіка і електроніка) включає цілий науково-технічний напрям, що об’єднує в собі сучасні комп'ютерні технології і технічні пристрої. Поліграфічне обладнання, як мехатронна система, обов'язково включає в свій склад електромеханічні перетворювачі з електронною комутацією,

різного роду датчики (сенсори), силові напівпровідникові перетворювачі, мікроконтролери і персональні комп'ютери.

3. Процеси і обладнання, які застосовуються сьогодні для виробництва друкарських засобів інформації, використовують різноманітні інноваційні технології і компоненти з різних областей науки і техніки: електроніки, комп'ютерної техніки, фізики, хімії, теорії управління. Завдяки використанню різних інноваційних технологій у області електроніки, стало можливим злиття таких, раніше відособлених в поліграфії процесів, – додрукового, друку, післядрукової обробки, – в один технологічний потік виробництва друкарської продукції.

Процес випуску поліграфічної продукції в даний час нерозривно пов'язаний з обладнанням, у функціональних вузлах якого вбудовані комп'ютеризовані пристрої, здатні швидко виконати величезну кількість обчислень в секунду і формувати точно розраховані в певному напрямі кроки (мікропроцесори, мікроконтролери). Цифровий інформаційний потік став складовою частиною виробництва друкарської продукції.

Комп'ютеризація обладнання змінила істотним чином способи управління обладнанням і елементи інтерфейсу інформаційного каналу, що бере участь в процесі управління, в першу чергу, пристроїв «збору і обробки» інформації про стан обладнання і навколишнього середовища. Використання дешевих мікроконтролерів привело до розвитку апаратно-програмних засобів, які здійснюють з недоступною для людини швидкістю збір, обробку, аналіз і відображення інформації.

4. Розуміння роботи сучасного комп'ютеризованого обладнання, що бере участь у випуску поліграфічної продукції, скрутно без використання для цього понять, звичних для інформатики, без розуміння на новому рівні терміну «комп'ютеризована система».

Для комп'ютеризованих систем є характерним використання виробів, у яких існує так званий семантичний зв'язок між апаратними (електронними) засобами і їх програмним забезпеченням. Такі програмно-апаратні засоби визначаються поняттям архітектури, що характеризує їх різну організацію. Архітектура комп'ютеризованого обладнання являє собою безліч взаємозв'язаних компонентів, що включають апаратне забезпечення (hardware), програмне забезпечення (software), алгоритмічне забезпечення (brainware) і спеціальне фірмове забезпечення (firmware), які підтримують його злагоджене функціонування у формі єдиної системи, що дозволяє вести ефективну обробку різної інформації.

Знання особливостей архітектурних рішень, застосованих в сучасному комп'ютеризованому обладнанні, дає можливість ефективно розпоряджатися апаратними і програмними ресурсами, що надаються, здійснювати їх оптимальний вибір і, тим самим, підвищувати ефективність роботи обладнання. Фахівці поліграфічної галузі повинні бути в рівній мірі готовими до використання існуючих нових технологій, що постійно з'являються.

В даний час фахівцю треба представляти архітектуру обладнання, роботу окремих його «цифрових» вузлів з програмованим мікроконтролером управління, розуміти програму його роботи, розбиратися в інструкціях і операціях, які він виконує, і, в цілому, розбиратися з тим, як взаємодіють вузли комп'ютеризованої системи на різних рівнях.

Відомості про аспекти розвитку комп'ютеризованого обладнання публікуються в літературі, розрахованій на фахівців. Але для окремої людини важко постійно знаходитися в курсі новацій і грамотно виробити позицію у змінах технологій у поліграфії. У зв'язку з цим є необхідність створення орієнтованого на магістрів у області поліграфії і, разом з тим, максимально доступного користувачам різної кваліфікації, навчального посібника. Такий посібник повинен дозволяти читачу одержати початкову інформацію про основи побудови комп'ютеризованого обладнання, основні принципи його роботи, структурну побудову різних вузлів комп'ютеризованих систем, про створення доступних для вивчення, простих і зрозумілих текстів програм, які дають можливість реалізувати різні алгоритми функціонування окремих елементів промислового об'єкта.

Метою пропонованого навчального посібника є вивчення архітектури типового комп'ютеризованого обладнання друку і післядрукової обробки продукції, основ функціонування окремих елементів мехатронної системи, розгляд питань програмного забезпечення типового мікроконтролера, що бере участь в управлінні вузлом обладнання.

Основою навчального посібника став курс лекцій з навчальної дисципліни «Компьютерізовані системи поліграфічного обладнання», який читається в Харківському національному економічному університеті студентам магістерської підготовки, які навчаються за фахом 0927 «Видавничо-поліграфічна справа».

Структура навчального посібника включає питання, що вивчаються у перших двох модулях «Основи комп'ютеризованого поліграфічного обладнання» і «Програмне забезпечення мікропроцесорної техніки, вжи-

ваної в поліграфічному обладнанні», які можна об'єднати під загальною назвою «Архітектура і організація управління цифровими процесами». Питання, що вивчаються в третьому і четвертому модулях «Реалізація функцій збору, обробки і перетворювання інформації у комп'ютеризованому поліграфічному обладнанні» і «Компоненти управлінських систем та інтерфейсу комп'ютеризованого поліграфічного обладнання», які можна об'єднати під загальною назвою «Підсистеми управління обладнанням і технологічними процесами», тут не розглядаються.

У посібнику розглянуті п'ять тем: 1. Загальні відомості про сучасні комп’ютеризовані системи випуску друкарської продукції. 2. Управління обладнанням друку в комп'ютеризованих системах. 3. Архітектура цифрового пристрою керування й виконання мікроконтролера AVR. 4. Початкові відомості про програмне забезпечення мікроконтролерів. 5. Програмне забезпечення взаємодії людини з комп'ютеризованим обладнанням. Кожна з тем крім теоретичної частини, має практикум, підкріплений практичними пристроями, виконаними на основі мікроконтролера.

Теоретичний виклад матеріалу супроводжується лабораторними роботами, які слугують базою для розвитку понять, введених в посібнику, несуть не тільки навчальне навантаження, але і виявляються зручними для вивчення роботи обладнання. У них невід'ємною частиною представлений матеріал, який містить проекти всіх модулів і повні тексти програмного забезпечення, описаного в посібнику.

У практичній частині використані моделі, які адекватні найбільш типовим вузлам реального поліграфічного обладнання. У них використані алгоритми, які дозволяють перетворити формули на працюючі програми і здійснити модельний експеримент. Відповідно до наших уявлень процес розробки програм, як наукоємкого виробу, виконаний таким, що проходить всі етапи від строгого теоретичного обгрунтування до практично готового програмного продукту. При цьому користувачу рекомендується самому досліджувати модельований пристрій згідно опису. Практичну частину посібника автори визнали доцільною винести за межі книги.

Для посібника характерне те, що навчальний матеріал в ньому логічно і за змістом тісно пов'язаний з навчальним матеріалом, який студенти вивчають в навчальній дисципліні «Технічне забезпечення видавничих систем». Це у певній міри визначило принцип відбору матеріалу і ступінь детальності освітлення.

При роботі над змістом посібника був зроблений наголос на фундаментальні знання, які, як відомо, є основною достоінством університетської освіти. Разом з тим, враховуючи той факт, що сучасна комьютерна техніка і електроніка є галузями знань, що надзвичайно бурхливо розвиваються, автори, при відборі матеріалу, що викладається в курсі, прагнули і до того, щоб він відповідав сучасним вимогам і практичним завданням, що вирішуються в даний час.

При написанні посібника автори ставили за мету чітко, строго і логічно викласти матеріал навчальної дисципліни відповідно до сучасних стандартів вищої освіти в Україні.

У результаті вивчення дисципліни студент повинен набути таких компетенцій:

1. Знати і вміти кваліфіковано характеризувати основні поняття комп'ютеризованих систем поліграфічного обладнання, засобів людино-машинного інтерфейсу.

2. Добре розуміти і вміти кваліфіковано пояснювати призначення, основні функції і схемо-технічну побудову типових пристроїв з мікроконтролерами, призначеними для управління поліграфічним обладнанням.

3. Оцінювати режими роботи і параметри типових пристроїв комп'ютеризованих систем поліграфічного обладнання, моделювати їх, виконувати наладку в програмному середовищі CodeVisionAVR, AVR-Studio, ProteusVSM, виміряти їх за допомогою контрольно-вимірювальної апаратури.

4. Користуватися керуваннями з експлуатації обладнання, довідниками і науково-технічною літературою, документацією, матеріалами, отриманими в мережі INTERNET і самостійно освоювати нові питання теорії і практики комп'ютеризованих систем поліграфічного обладнання.

5. Оцінювати логіку засобів забезпечення безпечної роботи, розуміти дії операторів під час надходження сигналів про виникнення несправностей, візуальної і звукової сигналізації.

6. Оцінювати вплив умов експлуатації на параметри і характеристики комп'ютеризованих систем.

Як відзначалося в останній час в розвитку і створенні нового високопродуктивного обладнання і технологій друку стали визначальними автоматизація, комп'ютерні технології, інновації, тісно пов'язані з «високими технологіями». Тобто сучасній електроніці притаманний стрімкий, прискорений характер розвитку і змін. Відповідно учбова література за-

раз швидко застаріває, особливо в навчально-методичному аспекті. Тому існує постійна потреба у нових учбових посібниках з цих розділів. Вона обумовлена також задачами, що постали зараз перед системою освіти України у зв'язку з проведенням її реформування на основі компетентного підходу для її інтеграції у світову освіту. Можна сподіватися, що даний навчальний посібник буде сприяти розв'язку цих задач і буде корисним для студентів у їх навчанні.

Модуль 1. Основи комп'ютеризованого поліграфічного обладнання

Тема 1. Загальні відомості про сучасні комп’ютеризовані системи випуску друкарської продукції

1.1. Основні тенденції в поліграфії

Сьогодні в розвитку і створенні нових високопродуктивних засобів і технологій друку, визначальними стали інновації, засновані на зроблених протягом останніх років відкриттях і винаходах у області фізики, хімії, інженерних наук, в технології друку. У сучасному поліграфічному обладнанні використовують компоненти з різних областей науки і техніки, які одержані на основі так званих «високих технологій». Відзначимо основні тенденції.

Перше. Важливою тенденцією останнім часом є використання в поліграфії новітніх компонентів і схемних рішень з електроніки, мікроелектроніки з її унікальними можливостями зберігати і переробляти інформацію, і різних комбінацій з областей фотоніки і електроніки. Необхідність впровадження електронно-фотонної і мікроелектронної техніки в нові поліграфічні машини обумовлена як загальною тенденцією розвитку суспільства – постійним посилюванням вимог до енергозбереження, так і першорядним для виробника завданням – залишитися в лідируючому положенні на ринках продажів обладнання.

Ефективне виробництво обладнання це, як відомо, швидке і вигідне виготовлення виробів необхідної якості і вартості. «Електронно-фотонна» техніка, з даної точки зору, дозволяє буквально революціонізувати обладнання. Вона, до того ж, дозволяє підвищити рівень друкарської продукції, що випускається, оскільки тільки з її допомогою можна реалізувати з високою якістю багато ключових у виробничому циклі функцій виготовлення друкарської продукції. Крім операцій, безпосередньо пов'язаних з процесом друку, електронна техніка також може бути успішно застосована в допоміжних операціях, де вона реалізує облік сировини і матеріалів, запобігає порушенню нормальних умов роботи друкарської машини. Електронна техніка успішно вирішує також задачу створення

обладнання, здатного виконувати свої функції при найменших розмірах виробу, найменшій масі і граничній дешевизні. Через те, що обладнання стає меншим, легшим, менше споживає енергії, менше коштує і надійніше в роботі, ніж обладнання без електроніки, виробники охоче упроваджують електронно-фотонну техніку в нові поліграфічні машини.

Завдяки досягненням мікроелектроніки відкриваються можливості створення принципово нового поліграфічного обладнання за рахунок використання засобів, призначених для вирішення завдань збору, обробки, зберігання, пошуку, видачі і передачі інформації. Електроніка дозволяє здійснити обробку і проміжне зберігання інформації в місцях, наближених до її отримання, організувати спільну роботу різних функціональних вузлів поліграфічного обладнання, «зв'язати» їх в єдину систему в якій блоки, що виконують різні функції, «обмінюються» інформацією.

Друге. Останнім часом на процеси друку, на конструктивне виконання обладнання істотний вплив надають інформаційні технології. У поліграфію широко упроваджуються інноваційні досягнення з таких областей, як комп'ютерні системи, мережеві технології. Завдяки цьому в даний час фактично відбулося злиття традиційно відособлених в поліграфії процесів (додрукового, друку, післядрукової обробки) в один інформаційний і технологічний потік виробництва друкарської продукції. Комп'ютерна техніка діє в межах загального виробничого процесу, оснащеного обладнанням різних виробників.

Третє. Хоча поліграфічне обладнання, в загальному випадку, уявляє собою людино-машинну систему, проте, більшість технологічних операцій зараз виконуються практично без участі людини в автоматичному режимі. На зміну старому приходить автоматизоване обладнання, що володіє широкими технологічними можливостями, високою продуктивністю, що працює без безпосередньої участі людини або при його мінімальній участі. Автоматизація дозволяє за рахунок технічних засобів і систем управління звільнити людину частково або повністю від безпосередньої участі в процесах отримання, перетворення, передачі і використання енергії, матеріалів або інформації. Автоматизація дозволяє не тільки розвантажити людину від важкої і одноманітної праці, але і забезпечує роботу обладнання з такою продуктивністю, яку людина не може здійснити. У багатьох випадках автоматизація дозволяє забезпечити збір і обробку інформації про процес друку, управління технологічними процесами з показниками, недоступними людині за його фізіологічними мо-

жливостями. Відомо, що виробничі швидкості у сучасних багатобарвних машин мають типові значення 15 000 відтиснень в годину (від/год) для листових машин і 60 000 від/год для рулонних машин. Зрозуміло, що таке виробництво продукції можливо, якщо воно автоматизоване в переділах всього технологічного процесу.

В даний час без участі людини повинні виконуватися підготовка друкарської машини до початку виробничого процесу виготовлення тиражу, її настройка. Такі операції як, наприклад, перенастроювання формату, зміна друкарської форми, коректування приведення і зміна валиків повинні бути автоматизованими. Зараз потрібно виконувати автоматизоване розвантаження і подачу пластин, використовувати автоматизацію зміни друкарських форм (наприклад, на основі касетної системи, коли в касетах зберігаються друкарські форми декількох послідовно виконуваних замовлень), автоматичний змив офсетних циліндрів і барвистих апаратів.

Цифровий інтерфейс додрукового процесу дозволяє в автоматичному режимі, зміною кута повороту дукторного циліндра в барвистому резервуарі, встановлювати подачу відповідної товщини шару фарби для конкретної друкарської форми. Сучасні автоматизовані системи управління дають можливість повністю представляти замовлення в цифровому вигляді на додруковій стадії. Установка подачі фарби за даними про друкарське зображення на додрукової стадії виконується із застосуванням так званого додрукового інтерфейсу (Prepress Interface).

У автоматизованих системах є також можливість здійснювати розрахунок і установку подачі фарби за допомогою засобів автоматики з застосуванням оптичної системи, що визначає середню процентну площу поверхні в межах барвистої зони (на основі характеристик віддзеркалення від друкуючих елементів зображення і від вільних від фарби ділянок поверхні). У алгоритмі, що враховує конструкцію барвистого апарату, характеристичну криву друкарського процесу, вплив особливостей фарби і паперу, визначається установка барвистих зон і об'єм фарби, що подається дукторним циліндром. На основі цих даних регулювання подачі фарби може виконуватися заздалегідь у всіх друкарських секціях машини до того, як будуть закріплені форми і почнеться процес подачі паперу в машину. За даними про потребу у фарбі і її розподілу на друкарській формі може здійснюватися регулювання подачі фарби для окремих дру-

карських секцій, площ поверхонь в різних колірних зонах основних фарб (чорної, блакитної, пурпурної і жовтої) чотирьохбарвистого процесу.

Сказане про автоматизацію відноситься також до питань подачі зволожуючого розчину в друкарські секції при офсетному друці. На основі потреби у фарбі, залежній від характеру замовлення, можна шляхом розрахунків визначити необхідну кількість зволожуючого розчину і за цими даними заздалегідь встановлювати подачу зволожуючого апарату. Фактичне нанесення фарби на друкарську форму залежить від численних початкових даних, серед яких гідродинамічні процеси перенесення фарби з барвистого ящика на систему валиків, розщеплювання і проходження потоку фарби в проміжку між окремими контактними зонами, а також фазове положення осьового розкату. Важливо врахувати вплив зволожуючого розчину при утворенні емульсії «фарба-вода». Характер перенесення фарби від барвистих накатних валиків на друкарську форму, так само як і з друкарської форми на офсетне полотно, а потім з нього на папір, повинен враховуватися при установці натиску в друкарському апараті. Теоретичні розрахунки дозволяють приблизно задати його установку, потім вона коректується з урахуванням результату процесу друку. Можливо впровадження систем, що застосовують спеціальні алгоритми, що навчаються. Вони дозволяють за відхиленнями між попередньою і фактичною установкою подачі фарби точно настроювати систему без участі людини, використовувати одержані результати в подальших регулюваннях. Такі навчані адаптивні системи сприяють успішній попередній установці подачі фарби, базуючись на прийнятих стандартних величинах.

У даний час при великому тиражуванні продукції для введення фарби для окремих друкарських секцій є системи, що здійснюють автоматичне регулювання стану наповнення ящика для фарби. Звичайно за допомогою насоса фарба закачується по трубопроводу з резервуара в ящик для фарби. Контроль наповнення здійснює датчик. У листових машинах також можуть застосовуватися спеціальні автоматичні системи подачі фарби. Аналогічно автоматичні системи здійснюють подачу в машину зволожуючого розчину. Вони стежать за тим, щоб друкарський апарат завжди мав його необхідну кількість, а також за складом рідини, її необхідною концентрацією. Підтримка постійної температури зволожуючого розчину також передбачена в подібних автоматизованих системах.

У ряді технологій подача фарби на відтиснення здійснюється через систему барвистих валиків, що мають інерційність. Тому на початку друку стабільний стан процесу друку досягається після певного числа оборотів формового циліндра, коли стабілізується рух паперу. Щоб зменшити вихід макулатури, використовуються спеціальні рішення з прискорення цього процесу, зокрема за допомогою алгоритмів управління потоком фарби. Точне регулювання машини при проводці паперу відбувається при нижчій швидкості, ніж швидкість друку тиражу. При збільшенні швидкості машини до виробничої змінюються умови подачі фарби. Для високопродуктивних концепцій побудови систем управління на базі комп'ютера все це враховується в алгоритмі регулювання. У процесі одночасно задіяні також пристрої подачі зволожуючого розчину, повітродувне і пневматичне і, можливо, також сушильні агрегати.

Сучасні технології дистанційного регулювання і позиціонування дозволяють автоматично виконувати операції, пов'язані з переходом з одного формату паперу на інший. Для цього системою автоматичного управління, за допомогою широкого набору різних пристроїв, проводиться перебудова самонаклада і вивідного пристрою (елементів виводу і рівняння листів на накладному столі), юстируються направляючі ролики, передні і бічні упори. Точно також на нові формат і щільність паперу повинні бути настроєні системи подачі повітря і вакууму в присосах, що забезпечують подачу паперу.

В даний час автоматизовані технологічні процеси, коли треба обмежувати моменти, що виникають в робочих органах, для запобігання поломці робочої машини або механізму при раптовому стопорінню виконавського органу.

Відзначимо, що в даний час практично повністю автоматизовані і управляються комп'ютером машини безконтактного (NIP –Non- Impact-Priting) друку. У всіх випадках автоматизовані системи без участі людини підтримують з необхідною точністю необхідні режими роботи, забезпечують необхідне прискорення або уповільнення виконавських органів, створюють і здійснюють управління положенням листів, здійснюють їх позиціонування.

Технічний прогрес дозволив оснастити поліграфічні машини і комплекси обладнання й іншими важливими компонентами засобів автоматизації, управління і контролю. Сучасні машини ідеально стикуються з пристроями Computer-to-Plate (що виготовляють як металеві, так і полі-

ефірні друкарські форми); системами цифрового управління виробництвом з електронною передачею інформації про зображення на друкарську машину; програмним забезпеченням для автоматичної початкової настройки друку; автоматичним контролем друкованої продукції і автокоректуванням режимів роботи машини в процесі друкування. У сучасних поліграфічних машинах реалізовані всі переваги цифрової технології Direct Imaging, що дозволяє обходитися без виготовлення форм і істотно прискорює процес друку. Друкарські форми експонуються в самій машині.

Помітимо, що в недалекому майбутньому можна чекати створення повністю автоматизованих цехів, в яких будуть об'єднані і автоматизовані не тільки обладнання друку, але і засоби, що здійснює завантаження і розвантаження напівфабрикатів і готової продукції, обладнання транспортно-складської системи.

Четверте. Важливою тенденцією останнім часом є підвищення експлуатаційної готовності поліграфічного обладнання. Високі показники надійності використовуваних елементів обладнання ще не гарантують високу експлуатаційну готовність до роботи всіх механізмів поліграфічної машини. Відмови можуть виникати рідко, але вимагати великого часу для їх виявлення і ліквідації, або, як говорять, великого часу відновлення працездатності. В цьому випадку обладнання буде значний час простоювати, що викличе, очевидно, серйозні економічні збитки.

Таким чином, для поліграфічного виробництва дуже важлива експлуатаційна готовність, яка визначається пристосованістю обладнання до виявлення несправностей і відмов, здатністю як можна до швидшого усунення цих дефектів шляхом ремонтів і технічного обслуговування.

Одержати великий коефіцієнт готовності дозволяє забезпечення пристроями контролю за роботою окремих вузлів обладнання друку. При виникненні несправності або відмови пристрої такого типа повинні забезпечувати звукову або світлову сигналізацію, а в аварійних ситуаціях – і спрацьовування захисту.

Крім того, підвищити коефіцієнт готовності можна за рахунок використання діагностуючих пристроїв, які, на основі порівняння фактичних показників з бажаними, здатні оцінити працездатність вузла і видати інформацію про місце і характер несправності.

1.2. Системна побудова сучасного поліграфічного обладнання і його компонування

При створенні поліграфічного обладнання в даний час широко використовується функціонально-модульний (блоково-модульний) метод компоновки (доцільного розміщення і взаємодії). Суть функціонально-модульного методу компоновки полягає в тому, що поліграфічне обладнання ділиться на окремі конструктивно і функціонально закінчені одиниці – модулі. При цьому звичайно прагнуть зробити так, щоб модулі мали однакові розміри або, в крайньому випадку, два розміри були однаковими, а третій – кратний якому-небудь значенню. Це дозволяє зручно, без втрати об'єму, компонувати з модулів, як з паралелепіпедів, конструктивні утворення вищого рівня. Конструктивна закінченість модуля скорочує тривалість циклу виготовлення всього виробу, оскільки дозволяє здійснювати паралельно збірку різних модулів. На рис. 1.1 показана друкарська машина, побудована за модульним принципом.

Рис. 1.1. **Модульний принцип побудови друкарської машини**

Використовуючи модульність можна виконати компактними «друкарські вежі» машини. Це дозволяє зменшувати потрібне для установки місце і робить швидшим і простішим доступ до друкарських секцій при експлуатації і технічному обслуговуванні. Її конфігурації різні залежно від конкретних завдань клієнтів. Крім того, одна і та ж конфігурація машини може працювати для отримання різноманітної друкарської продукції.

Важливим є те, що модуль становить пристрій, що виконує цілком певну закінчену функцію. У всіх різновидах друкарських машин обов'язково наявність наступних основних функціональних пристроїв: 1) папероживильна система, що подає папір в друкарський апарат (наклад, самонаклад, рулонне розмотування); 2) барвний апарат, що наносить при

кожному циклі машини фарбу на друкарську форму; 3) друкарський апарат, що здійснює притиск паперу до друкарської форми, внаслідок чого і виходить відтиснення; 4) паперовивідний пристрій, що приймає відтиснення з друкарського апарату і виводить його на приймання (виклад), іноді з попередньою обробкою – лакуванням, сушкою, розрізанням, фальцюванням і ін. Такі функціонально закінчені пристрої можуть бути незалежно один від одного відрегульовані і перевірені. Функціональна закінченість пристрою дозволяє створити взаємозамінну конструкцію, при якій будь-який модуль обладнання може бути замінений на іншій, однотипний без підгонок і регулювань.

Найбільш досконалою є конструкція, в якій модульний принцип реалізується на всіх рівнях конструктивної ієрархії, коли модулі вищого рівня складаються з модулів нижчого рівня ієрархії. У цьому випадку система випуску друкарської продукції, складається із стандартизованних конструктивно і функціонально завершених одиниць технологічного обладнання. Кожна така одиниця має можливість функціонувати автономно і вбудовуватися в ієрархію вищого рівня. На рис. 1.2. зображені модулі, що здійснюють проходження друкарського полотна.

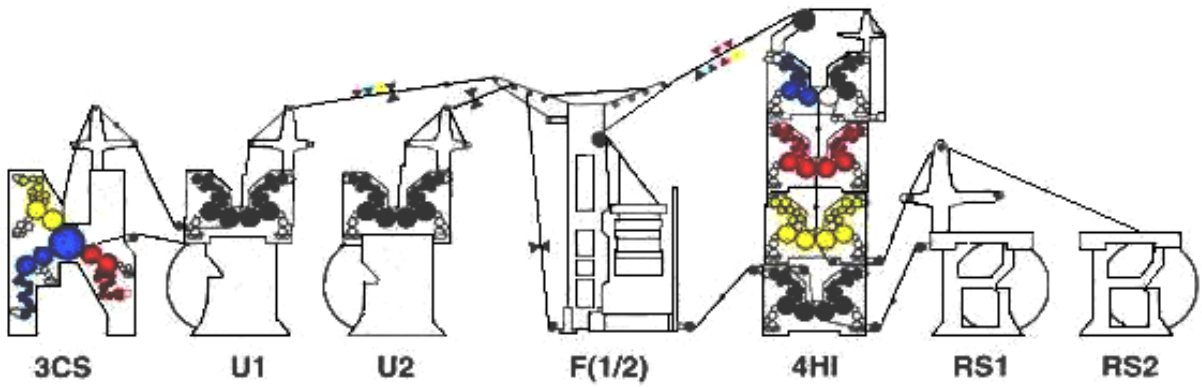


Рис. 1.2. **Багатомодульна конструкція що забезпечує проходження друкарського полотна (тут: 3CS – трифарбовий модуль із центральним друкарським циліндром; U1, U2 – двофарбові модулі з вбудованою рулонною зарядкою; F(1/2) – фальцапарат; 4HI – башта з чотирьох двофарбових модулів; RS1, RS2 – дві окремі рулонні зарядки або пристрої автоматичної зміни рулонів)**

Відзначимо, що типовими модулями обладнання друку можуть бути: 1) енергетичний модуль, що забезпечує енергією всю друкарську ма-

шину; 2) модуль виконання операцій, пов'язаних з переходом з одного формату паперу на інший, і регулювання форматів (перебудови самонакладa, елементів виведення і рівняння листів, юстирування направляючих роликів, передніх і бічних упорів, виключення можливості попадання в друкарську секцію двох і більш листів); 3) модуль захватного органу, у функції якого входить захоплення листа паперу, утримання захопленого листа в процесі переміщення і його звільнення; 4) модуль транспортуючої системи. Він служить для доставки листа від стапеля через самонаклад до першої секції друкарської машини, проводки листа від однієї друкарської секції до іншої без коливань листа і отмарування свіжонанесеної фарби. Модуль транспортуючої системи необхідний для забезпечення необхідного прискорення або уповільнення виконавських органів, регулювання натягнення полотна, управління положенням листа, здійснення його позиціонування, плавності руху відтиснення без прослизання і підкидання; 5) модуль позиціонування форм і приведення (для забезпечення відповідності зображення, що одержується на відтисненні, зображенню оригіналу); 6) модуль регулювання (за даними з сектора додрукової підготовки, сканера друкарських форм або пульта управління оператора) кількості фарб, що подаються, в зону друку і поєднання їх на відтисненні; 7) модуль регулювання тиску між циліндрами залежно від структури поверхні задрукованого матеріалу і його товщини (зміни відстані між поверхнями в зоні контакту на необхідну величину); 8) модуль регулювання системи дозаправки і безперервної подачі зволожуючого розчину відповідно до швидкості друку для підтримки балансу фарби і зволожуючого розчину; 9) модуль управління режимами сушки фарби; 10) модулі фінішного обладнання: лакування, ламінування, холодного тиснення, висікання, видалення облоя; 11) модуль зміни форм із збереженням точності приведення.

Одиницею модульного обладнання звичайно є стандартна конструкція, реалізована у вигляді стійки. Стійка звичайно містить уніфіковану несучу конструкцію. Частина несучої конструкції, яка забезпечує їй міцність, стійкість, називають каркасом. Звичайно стійка має каркас, виконаний з фасонних профілів (куточків, швелерів і т. п.) і стінок з листового металу. Прикладом стійки може служити електрошафа, усередині якої розміщені лінії введення електроенергії, ввідні вимикачі, запобіжні пристрої та ін. Для захисту обслуговуючого персоналу від дотику до струмоведучих частин електрообладнання, що знаходяться в шафі, він звичай-

но забезпечений блокуючим пристроєм, що вімикає напругу при відкритті дверцят електрошафи.

Наступний системний рівень такої ієрархії називають субсистемою (підсистемою) або блоком (коміркою). Дуже небагатьом вузлам поліграфічного обладнання можна дозволити працювати самим по собі; більшість з них потребує управління того або іншого вигляду, щоб забезпечити їх безпечне і економічне функціонування. Тому в обладнанні завжди є підсистеми управління. Як субсистема можуть виступати вузли поточного контролю, які відображають оператору стани процесу друку і звертають увагу на ненормальні умови або помилки, що вимагають його особливої уваги. Підсистемою може бути центр управління електродвигуном. У блоці може знаходитися електрообладнання, пристрої автоматики або електронні пристрої. Якщо абстрагуватися від частковостей, то блок має механічні направляючі для вбудовування в стійку і звичайно забезпечений електричними з'єднувальними роз'ємами, по яких можна, подавати на нього електричну енергію, передавати електричні сигнали в блок і виводити їх з нього. При установці блоку в обладнання відбувається з'єднання обох половин електричного роз'єму, і блок підключається відповідно до схеми до обладнання.

У сучасному комп'ютеризованому обладнанні як ієрархічна одиниця ще нижчого рівня може бути електронна плата, яка містить з'єднані за допомогою провідників інтегральні схеми і інші компоненти, застосовувані в мікроелектроніці, оптоелектроніці і фотоніці. Плата працює не сама по собі, а спираючись на так звані стандартні інтерфейси. Поняття інтерфейсу, в першому наближенні, включає лінії електричного зв'язку й інші пристрої, про які ми поговоримо дещо пізніше.

Модулі, блоки і електронна плата звичайно є легкознімними. Заміна модуля на іншій повинна проводитися повністю, без яких-небудь підрегулювань чи ж з використанням нескладних регулювань, які може виконати персонал, що експлуатує обладнання. Наприклад, серед багатобарвних машин з числом секцій від 4 до 7 можна вибрати оптимальну конфігурацію друкарської машини з 30 модифікацій. Критеріями вибору може служити кількість секцій, короткий, середній або довгий виклад, наявність ІЧ- і/або УФ-сушки, секції лакування, пристрої перевороту листів, потужність приводу і т. п.

Основна перевага функціонально-модульних систем полягає в тому, що обладнання може бути побудоване системно, із стандартизова-

них блоків. Це дозволяє виготовляти різні модифікації обладнання залежно від вимог замовника.

Модульний принцип дозволяє створювати «гнучкі» технології друку. Модулі легко з'єднуються, утворюючи складні технічні системи. Вони легко замінюються з метою формування обладнання з іншими компонентами, технічними характеристиками, а також у разі потреби модернізації або ремонту технічної системи. Функціональна закінченість модулів дозволяє проводити незалежну модернізацію окремих модулів, покращувати характеристики обладнання в процесі експлуатації (цей процес часто називають апгрейдом). Помітимо, що сьогодні виробники обладнання надають великі можливості, щоб реалізувати потенціал апгрейда.

Перевага функціонально-модульних систем полягає і в тому, що скорочується час відшукання несправності, оскільки для модуля завжди є можливість створити необхідну кількість контрольних точок оцінки працездатності і елементів діагностики. Якщо обладнання розділене на модулі, то необхідно знайти тільки модуль, в якому знаходиться елемент, що відмовив. Інформацію про відмову модуля можна вивести на пульт оператора. Крім того, такий процес розбиття (дроблення) обладнання є вельми корисним як при його проектуванні, так і при осмисленні його принципу дії і вирішуваних ним задач. Можна на підставі перших вражень прийти в замішання від розмірів і складності крупних поліграфічних машин. Проте більшість обладнання можна спростити, розглядаючи його таким, що складається з безлічі простих виробів. При поясненні принципу дії системи будь-якої складності спочатку, в цьому випадку, пояснюється функціонування системи у загальних рисах, потім пояснюються основні субсистеми і їх складові частини до тих пір, поки не приходимо до найдрібніших подробиць. Важливість функціонально-модульного принципу обумовлена і тим, що є необхідність розділити при проектуванні сучасного обладнання роботу між декількома конструкторами, електронщиками, програмістами. Мінімум неузгодженості досягається в тому випадку, якщо кожному з них доручається проектування своєї, добре описаної субсистеми типу «чорного ящика». Відомо, що принцип «чорного ящика» дозволяє розбити систему на дрібні блоки, які легко піддаються дослідженню. При цьому можна вказати докладний перелік того, які функції виконує цей «чорний ящик», навіть не розглядаючи питання про те, що знаходиться усередині нього і, ігноруючи деталі, які є неістотними на даному системному рівні. На кожному наступному схемному рівні можна

ввести додаткові деталі, при якому з'явиться нова інформація, необхідна для повного розуміння роботи блоку. Таким чином, здійснюється рух вперед при концентрації уваги на питаннях, що мають відношення до поточної стадії розгляду блоку. Для сучасного обладнання розбиття на субсистеми звичайно виконується на стадії написання технічного завдання на проектування. На цьому етапі проектування ухвалюється також рішення про апаратуру, яка включатиметься до складу кожної субсистеми.

Функціонально-модульний принцип побудови обладнання дозволяє сильно спростити позначення не тільки вхідних в обладнання одиниць, але і всіх сигналів. Звичайно кожен сигнал позначається фізичним положенням у відповідній стійці, положенням плати в стійці і вказівкою, з яким контактом на цій платні цей сигнал зв'язаний.

Характеризуючи виробничу діяльність, структуру сучасних комп'ютеризованих систем поліграфічного виробництва, її внутрішню організацію, важливо розуміти наступне. При заданій технології виробництва друкарської продукції ми завжди маємо справу з «системою». Терміном «система» ми користуємося для того, щоб підкреслити, що вона складається з декількох структурованих і взаємозв'язаних частин (елементів), в якій зв'язки між елементами обумовлені конкретним чином операції енергією, речовиною, інформацією (тобто відносинами перетворення).

Відомо, що основний вид поліграфічного обладнання, призначений для друкування тиражів різних видань – це друкарська машина. Вона складається з системи різних вузлів і ланок, розміщених відповідно до конструкції і призначення. Основна відмітна ознака друкарських машин – це технологічний процес, що виконується друкарською машиною. Відповідно до цього критерію розрізняють друкарські машини для високого, плоского, глибокого і трафаретного друку. Оскільки основні способи друкування мають безліч різновидів і модифікацій, то і друкарські машини одного способу друкування мають різні конструкції і відмінності, іноді дуже істотні.

З системного принципу побудови виходить, що в обладнанні завжди можливо виявити структуру – перелік використовуваних вузлів, механізмів і їх взаємозв'язок. Зокрема, в комп'ютеризованій системі можна завжди виділити сукупність елементів і зв'язків між ними, які здійснюють виконання наступних функцій: 1) перетворення якого-небудь виду енергії в механічну роботу, необхідну для здійснення виробничих процесів вико-

навчими механізмами або робочими органами машини; 2) управління – збору, обробки інформації про стан обладнання і вироблення рішень про дію на нього, коли не забезпечується задана відповідність цільовому процесу перетворення.

Таким чином, в комп'ютеризованій системі завжди є енергетичні (силова) і інформаційно-управляча частини. У комп'ютеризованому обладнанні можна виділити енергоживлення, «потоки» енергії, канали передачі інформації про стан окремих елементів і сигнали управління. За допомогою силового «потоку» енергія підводиться, транспортується в потрібну точку приводу, де і перетворюється. За допомогою інформаційних каналів здійснюється управління потоком енергії, а також збір інформації про стан і режими функціонування системи в цілому. Ці частини і потоки системи істотним чином відрізняються з погляду призначення, вирішуваних ними задач і критеріїв якості функціонування.

Необхідна для друкарського обладнання потужність звичайно визначається навантаженням на вихідному валу. Ця величина, з урахуванням втрат потужності у всіх ланках кінематичного ланцюга, визначає потужність необхідну для джерела руху всієї системи.

Для перетворення потоків енергій в механічну роботу в комп'ютеризованій системі використовуються механізми, робочі машини, приводи. Механізми – це виконані на основі конструктивних рішень системи деталей (ланок), що служать для перетворення руху одного або декількох тіл в необхідні рухи (поступальні, обертальні, коливальні) інших тіл. Простим прикладом механізму може служити гвинтовий механізм, що містить гвинт і гайку, і що служить для перетворення обертального руху одного елементу в поступальну ходу іншого. Іншим, добре відомим механізмом, є зубчатий механізм, що складається із зубчатих коліс і забезпечує передачу руху між валами із зміною швидкості. Всякий механізм характеризується конструктивними особливостями і подробицями виконання кінематичної схеми.

Група механізмів, зв'язана в загальний кінематичний ланцюг і виконуюча механічні рухи, з метою перетворення енергії, матеріалів або інформації, називається робочою машиною. Відомо, що до складу поліграфічного обладнання може входити декілька машин, призначених для здійснення необхідних силових дій, для виконання механічних рухів якихнебудь елементів щодо інших. Робоча машина характеризується загаль-

ною компоновкою (відносним розташуванням вузлів у виробі) і компоновкою вузлів (відносним розташуванням деталей у вузлі).

Для того, щоб реалізувати доцільні переміщення, повороти робочого органу, силові дії, привести в рух машини і механізми, різні робочі органи, здійснити перетворення якого-небудь виду енергії в механічну роботу, служить привід – пристрій, що складається з двигуна, передавальних механізмів і системи управління. Приводи виконують «силові» функції. Проте реалізовують такі функції приводи лише за наявності перетворювачів команд, що задаються, на рух виконавчих механізмів технологічного обладнання. Тому в приводі має місце тісний взаємозв'язок електромеханічної частини з енергетичним каналом живлення і каналом управління. У загальному випадку, привід, як керований енергосиловий пристрій, складається не тільки з джерела руху (двигуна), механізму передачі, що пов'язує двигун з переміщуваним елементом або виконавським органом обладнання, але і пристроїв управління вихідними змінними, які часто називають координатами.

В сучасному поліграфічному обладнанні як джерела енергії використовують електричну енергію, енергію стислого газу і енергію потоку робочої рідини.

Енергетичною основою багатьох технологічних і виробничих процесів сучасної поліграфії є електропривід, який становить електромеханічну систему, і який здійснює перетворення електричної енергії в механічну роботу. Приводи такого типу, у всій сукупності перетворень і розподілу перетворюваної енергії, в даний час складають основну компоненту багатьох поліграфічних систем. Широке розповсюдження електроприводу для перетворення енергії пояснюється зручністю передачі і акумуляції електроенергії, високим ККД електромеханічних пристроїв. Процеси перетворення електричної енергії в механічну роботу легко піддаються автоматизації і оптимізації. Електроприводи використовуються в обладнанні легкої і середньої вантажопідйомності. За допомогою таких приводів можна добитися великих ступенів рухливості (3 – 6).

Електроприводи є приймачами енергії і для забезпечення своєї роботи вимагають підведення від електричної мережі струму (звичайно трифазного). Електричний привід включає електричну частину (електродвигун і електричний пристрій управління), а також механічну частину для передачі руху робочим органам. Такий привід звичайно складається з джерела механічного руху – електродвигуна, з перетворювального вуз-

ла (змінює параметри електричної енергії, наприклад, величину напруги або його частоту), передавальних (здійснює зв'язок двигуна з механізмом) пристроїв, управлінських (виробляє необхідні закони управління потоками енергії від перетворювального пристрою до двигуна). Останнім часом електричні приводи успішно удосконалюються у зв'язку з використанням в пристроях управління нових силових електронних напівпровідникових приладів. За рахунок використання електронних пристроїв управлінню вдається добитися точності позиціонування робочого органу не менше ± 0,05 мм.

Пневматичні приводи використовуються в обладнанні легкої вантажопідйомності з числом ступенів рухливості не більше 2 – 3. Застосування пневмоприводів обумовлені відносною простотою конструкцій для поступальних і обертальних рухів механізмів обладнання. Проте, пневматичним приводам властиві такі недоліки, як низький КПД, недостатня «жорсткість» через стисливості повітря або газу, низька швидкодія.

Гідравлічний привід характеризується використанням енергії рухомої робочої рідини для отримання механічної енергії. Джерелом енергії рухомої рідини є насосна станція, що працює за рахунок споживання електричної енергії. За допомогою гідроприводів можна забезпечити число ступенів рухливості 3 – 4 і точність позиціонування ±0,5 мм.

Для здійснення процесів друку до складу поліграфічних машин входять різні підсистеми, які поєднують в собі названі вище приводи, механізми, вузли. Наприклад, на самонакладі може бути застосований вакуумний або фрикційний транспортер. Для безконтактної проводки листа можуть використовуватися підсистеми, що використовують «повітряну подушку». У підсистеми проводки звичайно входять пристрої перевороту листів, які характеризуються різними конструктивними особливостями. Приймально-вивідні пристрої машин включають вакуумні барабани для гальмування листів, пристрій, протидіючий скручуванню листів. Крім того, до складу поліграфічної машини можуть входити системи наддуву повітря з регулюванням по зонах для рівномірного укладання продукції, додаткові секції лакування з можливістю нанесення дисперсійного і/або УФ-лаку. Розпилювачі противідморюючого порошку, ІЧ- або УФ-сушки з системою обдува листів гарячим і холодним повітрям, автоматична підтримка рідино-барвистого балансу, автоматичне регулювання температури фарби на валиках і формі.

Для управління рухом виконавських органів робочих машин і механізмів, необхідне регулювання ряду змінних, які називають координатами, наприклад, швидкості, прискорення, положення виконавського органу робочої машини і т. д. З цієї причини для здійснення виробничих процесів виконавчими механізмами або робочими органами машини в приводах технологічного обладнання друку завжди реалізується функція управління. Мета управління полягає в тому, щоб переміщати елементи керованих пристроїв по наперед заданому, доцільному закону. Регулювання координат вимагають багато робочих машин і механізмів. Крім того, при роботі самого приводу необхідно забезпечити певні допустимі режими роботи його елементів. Так, наприклад, при пуску, реверсі або гальмуванні двигунів часто потрібно обмежувати їх струми до допустимих рівнів. Для управління використовують пристрої управління, які шляхом збору і обробки інформації про стан обладнання і зовнішнього середовища виробляють рішення про дію і формують потрібні для цієї мети команди (сигнали). Якщо управління цілісною, взаємозв'язаною, впорядкованою і взаємодіючою сукупністю механізмів, яка забезпечує виконання різних рухів, здійснюється комп'ютером, то в літературі такий клас технічних систем називають мехатронними системами. Слово «мехатроника» складається з двох частин – механіка і електроніка. Спочатку це слово позначало використання комп'ютера для управління механічною системою. Сьогодні цей термін позначає цілий науково-технічний напрям, що спирається на сучасні комп'ютерні технології (теорія) і нові можливості у використанні технічних засобів (практика). Мехатронні системи звичайно включають в свій склад різні приводи, електромеханічні перетворювачі з електронною комутацією (актуатори), різного вигляду датчики (сенсори), силові напівпровідникові перетворювачі, мікроконтролери, персональні комп'ютери.

Головна особливість таких систем полягає в тому, що для виконання у виробничому процесі рухових і силових функцій їх можна програмувати. Залежно від можливостей перепрограмованого пристрою програмного управління і від його здібностей за рішенням завдань управління рухом робочого органу мехатронні системи можуть бути простими, адаптивними або інтелектуальними. У простому випадку в мехатронній системі реалізується режим інформаційного управління, при якому комп'ютер призначений для збору, накопичення і первинної обробки інформації. Адаптивні системи можуть сприймати інформацію про зовнішнє середо-

вище за допомогою сенсорних або тактільних датчиків і в них закладена система управління, яка «вибирає» необхідну підпрограму управління залежно від інформації про зовнішнє середовище. Системи такого типу наділені здатністю «орієнтуватися» в навколишньому оточенні і «пристосовуватися» (адаптуватися) до неї. Інтелектуальні системи наділені «штучним інтелектом» і мають в своєму розпорядженні здатність самонавчання. Інформаційні можливості таких систем значно вищі: вони здатні розпізнавати предмети в просторі, виробляти «плани» рішення поставлених перед ними завдань і контролювати виконання останніх. Це самонавчальні, самоналагоджувальні системи з гнучкими процедурами ухвалення рішень про управління. Їх іноді оцінюють як «думаючі», «розумні» системи, здатні вирішувати завдання управління в умовах невизначеності інформації про властивості середовища функціонування.

1.3. Інформаційні потоки в комп'ютеризованому поліграфічному обладнанні

Для функціонування сучасного поліграфічного обладнання повинні існувати єдині принципи інформаційної взаємодії всіх елементів системи. Для нас, в першу чергу, з погляду інформаційної сумісності, важливі цифрові засоби і, відповідно, пристрої введення – виведення цифрової інформації.

1.3.1. Введення – виведення бінарної цифрової інформації

По суті, сучасне поліграфічне обладнання – це система обробки інформації (числовий і нечисловий) і передачі по різних каналах дискретних сигналів, які можуть мати два наперед визначені стани. Кінець кінцем вся інформація представляється у вигляді цифрових електричних сигналів. Оскільки діючі в обладнанні сигнали звичайно представляються напругами і струмами, то велика частина завдань розв'язується на електричній основі, із залученням цифрової логіки. Тому дуже важливо добре знати і розуміти поняття, які використовуються для характеристики сигналів, і оперувати засобами представлення інформації.

Найбільш простою є двійкова система, оскільки в ній цифра (число) може приймати тільки два значення: 1 і 0. Один розряд двійкової цифри називають терміном «біт». Терміни «бінарний» і «двійковий» є синоніма-

ми. Одна з основних переваг двійкової системи полягає в тому, що деякі величини є вже за своєю природою двійковими і можуть бути легко представлені двома станами. Наприклад, стан контактів може бути представлений – «зімкнено-розімкнено». Функціонування апаратури може бути просто описано в термінах «включено» – «вимкнено», «в межах норми» – «поза нормою», «низький рівень» – «высокий рівень» і т. п.

Якщо в обладнанні використовується двійкова система, то в ній всі вхідні і вихідні сигнали представляються двома логічними рівнями: логічним нулем («0») і логічною одиницею («1»). У позитивній логіці використовуваний логічний «0» означає низький рівень сигналу, а логічна «1» – високий рівень.

Основою логічних сигналів, є електричні сигнали. З їх допомогою імітують логічні рівні (представляють двійкові числа). Звичайно для фізичного (електронного) представлення біта служать постійні напруги і прямокутні імпульси певної величини. У позитивній логіці логічній одиниці відповідають статичний позитивний потенціал, позитивний імпульс напруги або струму. Відсутність позитивного потенціалу, напруги або струму відповідають логічному нулю. Була прийнята угода про те, що в будь-якій комірці цифрового пристрою, призначеному для «цифрової взаємодії з об'єктом», електричні сигнали, будь вони представлені у вигляді постійних (незмінних, статичних) напруг або імпульсів, повинні бути узгоджені (фізично) за рівнем потенціалів з напругами логічних елементів сімейства транзисторно-транзисторної логіки (ТТЛ). Якщо сигнали, що поступають на вхід цифрового елементу, відрізняються від стандартних, то для приведення їх до потрібних використовують перетворюючі схеми (адаптери): резістивні дільники напруги, узгоджувачі рівнів, компаратори, трігери Шмітта. Якщо значення вхідних і вихідних напруг входять в стандартизовані діапазони, то будь-які цифрові пристрої надійно розпізнають ці напруги як 0 або 1.

Простота двійкової системи, а також, те, що біт являє собою мінімальну «порцію» кількості інформації, дає можливість, використовуючи цифрові електронні пристрої, що оперують бінарною інформацією, вирішувати самі різні по своїй фізичній суті задачі. Це дозволяє також створювати складні логічні або обчислювальні системи, що відносно просто взаємодіють з різними вузлами обладнання за допомогою дискретних сигналів, які можуть мати два і більше наперед визначених станів. Вико-

ристання двійкової системи дозволяє відносно просто організувати процес передачі, введення – виведення даних.

Простим прикладом, що демонструє введення в цифровий пристрій двійкової інформації (одного біта даних), є схема (рис. 1.3) бінарного датчика з електромеханічним ключем (кнопкою, клавішею). У такій схемі бінарна інформація про контрольований об'єкт задається станами контактів (замкнуто – розімкнено). Датчики, що використовують ключі у вигляді електрично провідної контактної пари і механізму її замикання – розмикання, широко використовуються в сучасних інтерфейсних схемах введення дворівневого сигналу управління. Ви їх зустрінете в обладнанні у вигляді тумблерів (перекидних перемикачів), натискних одиночних кнопок, службових клавіш, різного роду клавіатур, елементів галетних перемикачів, сенсорних перемикачів, DIP-перемикачів і інших комутаційних виробів.

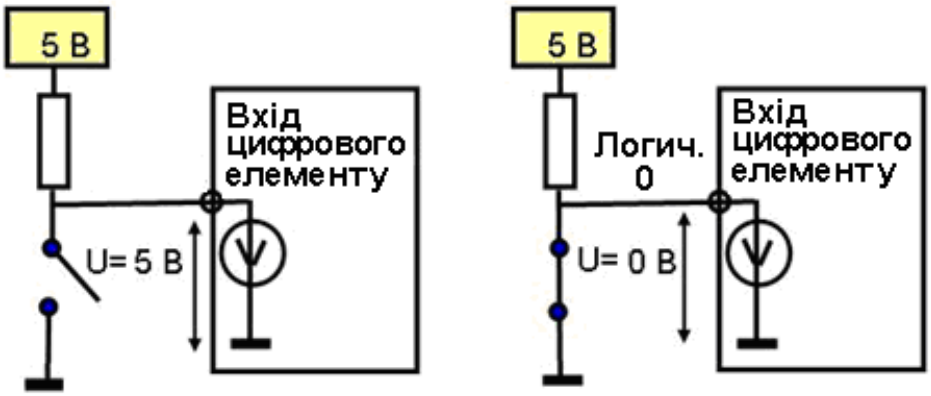


Рис. 1.3. **Введення бінарної інформації за допомогою електромеханічного ключа**

Важливим прикладом застосування ключів в схемах управління обладнанням є так звані «кінцевики» – контактні з'єднувачі, що спрацьовують при досягненні робочим приладом певного положення. Кінцеві (шляхові, кінцеві) вимикачі застосовують у випадках, коли потрібні дані про положення або рух. Принцип їх дії: коли рухомий за попередньо визначеною траєкторією прилад досягає необхідного положення (наприклад, закриваються дверці обладнання або лінійка фотоприймача, що переміщається, в сканері досягає «упору»), «кінцевик», використовуючи з'єднаний з робочим приладом шток або важіль, замикає або розмикає контакти кінцевого електромеханічного вимикача. В обладнанні можуть використовуватися і безконтактні оптоелектронні і магнітні вимикачі, які

виконують ту ж функцію ключа, але не вступають у фізичний контакт. Вони не чутливі до механічних дій і можуть працювати при великих швидкостях переміщень.

У схемі бінарного датчика використовується резистор, підключений до дроту, що має потенціал 5 В, який за допомогою ключа підключається або відключається від схемної «землі». При розімкненому стані ключа позитивний потенціал джерела живлення прикладається через резистор до входу цифрового елементу і забезпечує на ньому напругу +5 В, яка відповідає рівню логічної одиниці (у позитивній логіці). При замиканні ключа і комутації нижнього кінця резистора на «землю» струм «відводиться» від входу цифрового елемента, вся напруга джерела живлення прикладається до резистора, внаслідок чого на вході цифрового елементу напруга близька до нуля, що відповідає низькому логічному рівню («0»). Таким чином, завдяки даній схемі здійснюється «прочитування» інформації про стан ключа і за допомогою електричного потенціалу на вхід цифрового елементу поступає сигнал, відповідний або логічному нулю, або одиниці. У сучасних мікроконтролерах для надходження інформації про стан ключа використовують внутрішній резистор мікроконтролера, який дуже просто «підключається» до входу програмним способом.

Кажучи про методи сполучення вузлів обладнання з електронними блоками, про інтерфейсні функції ключа, що перетворює просторове положення контактів в необхідні логічні рівні напруги, особливо слід зазначити те, що всі контакти ключів, перемикачів, особливо зношені і забруднені, «деренчать» при установці в нове положення. Це призводить до того, що при установці електричного з'єднання, як би якісно не був виготовлений контакт, електричний ланцюг кілька разів розмикається, нерідко до 4 – 6 разів. Коливання контактів ключа, якщо не вживати необхідних заходів, цифровий елемент сприйматиме як окреме перекидання контактів. У результаті брязкіт може викликати хаотичне перемикання цифрового елементу, що реагує на зміну рівня сигналу. Особливо схильні до підвищеного зносу, обумовленого тертям в електричних контактах, кінцеві електромеханічні вимикачі. Тому електричні контакти в них мають відносно обмежений термін служби, близько 10 – 50 млн операцій.

Стандартні електромеханічні кнопки, клавіші мембранних клавіатур, перемикачі відносно швидко зношуються, слабо захищені від дії навколишнього середовища і не завжди відповідають вимогам дизайну кінцевого продукту. Тому на зміну електромеханічним виробам все частіше

приходять сенсорні технології, які дозволяють створювати клавіші, матриці перемикачів, панелі управління, і які позбавлені традиційних проблем минулого: нестабільності в роботі, низької надійності. Сенсори широко використовуються в пультах управління сучасним поліграфічним обладнанням.

Існує декілька безконтактних сенсорних технологій: резістивні плівки, місткістна, акустична, інфрачервона.

Резістивна технологія грунтується на зміні електричного опору контакту у момент дотику до нього. Контакт уявляє собою багатошарову структуру, що складається з двох провідних поверхонь, розділених спеціальним ізолюючим складом. При торканні будь-яким відповідним для цього предметом зовнішнього прозорого шару, його контактна сторона торкається контактної сторони іншого, внутрішнього провідника, створюючи електричне з'єднання в точці дотику. Такі сенсорні «кнопки», витримують десятки мільйонів торкань. У сенсорах, місткостей, торкання визначається завдяки зміні електричної місткості. Величина місткості між двома об'єктами, як вам відомо, обернено пропорційна відстані між ними і прямо пропорційна їх геометричним розмірам. При торканні сенсора ця місткість змінюється і завдяки зміні вихідного сигналу, реєструється факт натиснення «клавіші».

Для виведення повідомлень і необхідних даних використовують пристрої, звані загальним терміном дисплей. Дисплеї, що візуально представляють інформацію, є важливим елементом діалогу людини з комп'ютеризованою системою. Завдяки дисплеям з'явилася можливість виведення оброблюваної інформації таким чином, що її можна «бачити» на пристрої відображення інформації. Це дуже важливо, оскільки вважається, що більш 70% інформації людина сприймає за допомогою зору.

Існує багато типів дисплеїв різного призначення і інформація, що відображається ними, різноманітна. По вигляду інформації, що відображається, і за способом управління простішим є одиничний індикатор, який складається з одного випромінюючого елементу і призначений для представлення інформації у вигляді крапки, що світиться або не світиться. Прикладом, що демонструє виведення двійкової інформації (одного біта даних) з цифрового пристрою, показана на рис. 1.4 схема, яка логічні сигнали візуально відображає свіченням одиничного напівпровідникового індикатора (світловипромінюючого діода, світлодіода).

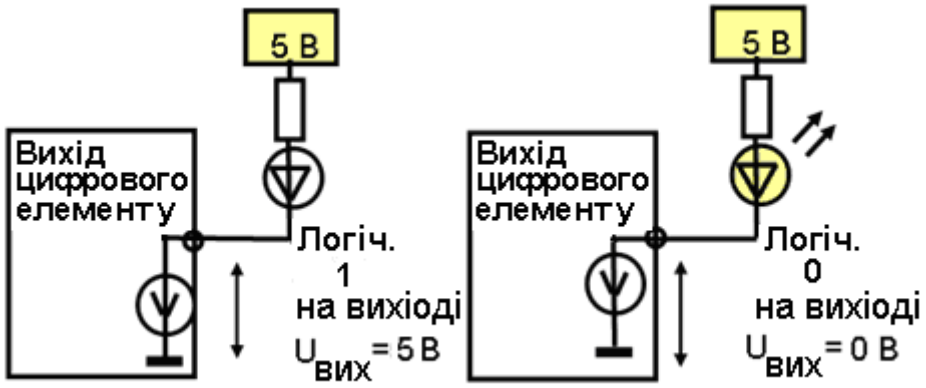


Рис. 1.4. **Виведення бінарної інформації за допомогою світлодіода**

Напівпровідникові одиничні індикатори широко застосовують для візуальної індикації стану окремих вузлів апаратури і обладнання в цілому. За допомогою світлодіодів контролюють поведінку окремого вузла обладнання в якійсь крапці і повідомляють оператора про те, чи знаходиться крапка, що перевіряється, в стані логічної одиниці або стані логічного нуля. Світлодіоди також використовуються для підсвітки написів і кнопок в пультах управління, створення шкал і табло, як випромінювачі в оптронах. Світлодіоди, як елементи схем візуальної індикації стану обладнання, прийшли на зміну лампам розжарювання і неоновим лампам. Основною причиною такої заміни є здатність світловипромінюючого діода працювати при малих струмах і напругах, сумісних з амплітудами логічних рівнів напруг мікросхемної електронної техніки. Крім того, напівпровідникові одиничні індикатори порівняно з лампами накалювання практично не виділяють тепло, надійніші і довговічніші.

Схема на рис. 1.4 є основною для включення світлодіода при роботі на виході цифрового бінарного пристрою. У ній світлодіод підключений анодом до джерела живлення з величиною е.р.с. E = 5 В за допомогою струмообмежуючого резистора R, яке за величиною опору більше диференціального (до джерела напруги з малим внутрішнім диференціальним опором). Катод діода підключений до цифрового пристрою, який на своєму виході залежно від логічного рівня вихідного сигналу матиме потенціал або 0 В, або 5 В. Прі рівні напруги, відповідному логічній одиниці, на виході цифрового елементу, тобто при $U^1_{ВИХ}$ = 5 В, різниця потенціалів між анодом і катодом діода буде рівна 0, прямий струм протікати не буде і світлодіод не світитиметься. При нульовому рівні напруги, відповідному логічному 0 на виході цифрового елемента, тобто при $U^0_{ВИХ}$ = 0 В, різни-

ця потенціалів між анодом і катодом діода буде рівна прямій напрузі на діоді $U_{ПР}$, протікатиме прямий струм і світлодіод світитиметься. При періодичній зміні сигналу з «0» на «1» світлодіодом можна «мигати». При різних кольорах свічення (червоного, зеленого, жовтого) опори струмообмежуючих резисторів R у зв'язку з відмінністю $U_{ПР}$ для кожного кольору, будуть різними. Можна підібрати величини опорів резисторів, щоб при нормальному режимі роботи світився випромінюючий діод одного кольору, а при відхиленні напруг від норми – іншого кольору.

Для представлення двійкової інформації в оптичних системах може використовуватися присутність або відсутність оптичного випромінювання. При цьому присутність пучка світла відповідатиме 1, а відсутність – 0. Випромінюючий діод, підключений до виходу цифрової схеми, може бути використаний як випромінювач в оптопарі (оптроні), в якій зв'язок входу і виходу здійснюється за допомогою оптичних сигналів. В цьому випадку бінарну інформацію можна передавати силовим пристроям великої потужності, одержуючи при цьому практично ідеальну гальванічну «розв'язку», високу швидкодію в поєднанні з повною сумісністю з цифровими схемами.

1.3.2. Введення – виведення цифрової інформації у вигляді групи біт (байт)

Бінарна система числення ідеально підходить для цифрового обладнання, оперуючого двозначними сигналами, що управляють, типу «вкл/викл». Важливою її перевагою є те, що двійкова система числення дає можливість «навчити» цифрові пристрої «працювати» не тільки з мінімальною порцією інформації – бітом, але і «зі словами», з числами. Сучасні цифрові пристрої оперують з групами даних, організованих у вигляді «слів» (n-розрядних груп біт). Довжина n «слова» може змінюватися від 4 до 32 і більше біт. Вісім біт утворюють 1 байт. Іноді використовуються півбайти, рівні 4 бітам.

Використання байтів дозволяє оперувати з символами, оскільки один символ відносно просто представити у вигляді байта. Одним байтом, в простому випадку, можна представити цифрові аналоги таких змінних, як тиск, температура, швидкість потоку та ін. Використання даних, організованих у вигляді «слів», дає можливість проводити обмін ін-

формацією з «периферійними» пристроями, які складають невід'ємну частину будь-якої комп'ютеризованої системи.

У загальному випадку цифрова інформація у вигляді байта може бути розподілена або в просторі, або у часі. Якщо представити цифрові дані у вигляді одночасно існуючих інформаційних сигналів на окремих дротах, то одержимо інформацію, розподілену в просторі. Прикладом представлення розподіленої в просторі інформації може служити використовувана для досягнення високої швидкості передачі даних «паралельна» шина, яка становить загальну для всіх підключених до неї пристроїв сукупність ліній, по якій здійснюється паралельна передача інформації. У шині стільки інформаційних ліній, скільки розрядів має машинне слово, тому вона служить для обміну інформаційними «словами». Кожна з 8 біт (будь то логічний нуль або одиниця) посилається по окремій лінії. Логічний нуль на шині представляється напругою близькою до нуля, а логічна одиниця, або потенціалом 5 В, або імпульсом з амплітудою близької до 5 вольтів. Оскільки поведінка групи ліній уподібнена цифровим логічним словам, то самі лінії упорядковуються відповідно до двійкових вагів. Якщо шина має n ліній, то по ній у принципі можна передати I_B біт інформації $I_B = 2^n$. По восьми дротяній лінії можна передати 2^8 = 256 біт інформації, наприклад, числа від нуля до 255. При 16 лініях можна здійснити передачу 2^{16} = 65536 біт.

Інформацію, розподілену в просторі, у вигляді одного байта можна одержати на виході цифрового вузла, призначеного для передачі 256 чисел в електронному вигляді на інші цифрові вузли (рис. 1.5).

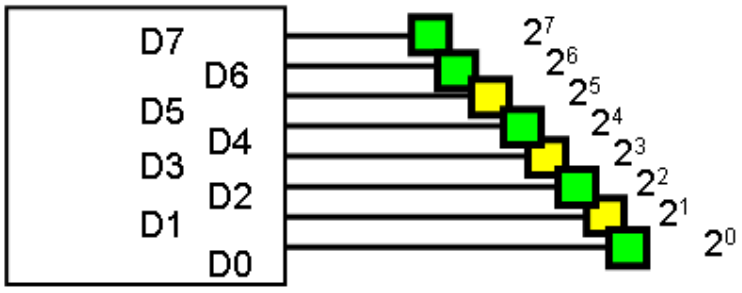


Рис. 1.5. **Представлення одного байта інформації (11010101)** на **виході цифрового вузла, призначеного для передачі в електронному вигляді 256 чисел**

І в цьому випадку стан групи ліній уподібнений логічним словам або байтам. Передаване число представляється в двійковій системі числення. Кожний з виходів, позначений як прийнято в цифровій електроніці D0, D1, D2, D3, D4, D5, D6, D7, відображає один розряд двійкового числа. Він може знаходитися в одному з двох станів: логічного нуля, коли напруга на виході відсутня; логічної одиниці, коли на виході присутня напруга за величиною близька до напруги живлення.

Відповідно до загальноприйнятих угод цифровий пристрій, підключений до шини даних, і що використовує розподіл інформації в просторі, називають портом введення – виведення. На порт введення по шині поступає інформація від деякого зовнішнього джерела. Порт виведення використовується для передачі по шині інформації зовнішньому пристрою.

Розділення сигналів в просторі дозволяє значно підвищити швидкодію системи, але викликає істотні апаратурні витрати. При послідовній передачі байта число ліній мінімальне, тому при використанні режиму розділення часу можна скоротити апаратурні витрати і такий підхід економічніший. Проте, оскільки для передачі «слова», що складається з n біт, потрібні n тактів роботи, то максимально можлива швидкість передачі інформації складатиме $1/n$ швидкості, що досягається при паралельному підході.

У комп'ютеризованій системі, у принципі, для передачі бінарної інформації можна використовувати будь-яку комбінацію розділення в часі і просторі. Такий спосіб роботи називається послідовно – паралельним. Він використовується для того, щоб забезпечити необхідну швидкість передачі інформації при мінімальних апаратурних витратах.

1.3.3. Короткі відомості про способи уявлення і перетворення цифрової інформації

Відомо, що домінуюче положення в електроніці зайняла володіюча рядом достоїнств двійкова система числення. При використанні системи числення з підставою b для запису числа x знадобиться $\log_b x$ розрядів. А оскільки розмір підстави b зменшується і зменшується число використовуваних цифр, то збільшується кількість розрядів, потрібних для представлення чисел.

Всі позиційні системи числення побудовані на підставі наступних правил: 1) будь-яке число записується у вигляді послідовності певних символів a_n, a_{n-1}, a_{n-2}, a_{n-3},…, a_1, a_0, де індекс символу позначає номер позиції в записі числа (номер розряду числа); 2) кожному символу в записі числа відповідає певна вага $\beta_n = p^n$ (p – підстава системи числення, n – номер розряду, рівний індексу при позначенні цифр розряду), значення якої визначається місцезнаходженням його в записі числа. Зміна положення (місця) символу змінює його вагу, що рівноцінно відповідній зміні індексу символу; 3) кількісне значення числа визначається наступним рядом: $N = \sum_{n=-m}^{i} a_n \beta_n$, де β_n – набір з β символів (у десятковій системі числення – $\beta = 10$, в двійковій – $\beta = 2$, в шістнадцятиричній – $\beta = 16$); $n-$ номер розряду числа, причому цифри, що мають позитивні індекси, відповідають цілій, а негативні індекси – дробової частини числа; a – зображення символу (цифри) в даній системі числення. У двійковій системі числення підстава системи числення $\beta = 2$. Таким чином, для запису цифр розрядів потрібен набір всього лише з двох символів, які використовуються 0 і 1. У десятковій системі числення, в якій підстава $\beta = 10$, використовуються десять арабських цифр від 0 до 9. Величини, тобто ваги, які визначені місцем, які займають відповідні символи в зображенні, називають вагами розрядів. Вибираючи різні підстави і використовуючи дані правила представлення чисел, можна «закодувати» всі використовувані на практиці числа в будь-яку систему числення.

Позиційне представлення цифр дозволяє відносно просто перевести число, представлене в двійковій або шестнадцятирічної системі числення, в десяткову.

У комп'ютеризованих системах важливу роль відіграє також шістнадцятирічна система, оскільки вона дає простий спосіб представлення двійкових чисел і дозволяє дуже просто перетворювати числа в двійкову систему і навпаки.

Типовою операцією для комп'ютеризованих систем є перетворення чисел з двійкової в шістнадцятирічну і з шістнадцятирічної системи в двійкову.

У комп'ютеризованих системах для поєднання простоти двійкової системи зі зручностями десяткової застосовують пристрої, в яких цифри

десяткової системи кодуються групами двійкових цифр. Кожну таку групу розглядають як одне ціле. Один розряд десяткового числа в цьому випадку може бути цифрою, яка знаходиться в діапазоні від 0 до 9. Існує декілька таких способів кодування десяткових цифр. У літературі їх називають двійково-десятковими, 8421 або двійково-кодованими десятковими числами BCD (Binary – Coded Decimal). Назва 8421 відображає значення вагових множників, що приписуються відповідним бітам в кодуючій групі. Тому іноді таку схему кодування називають зваженим кодом. Двійково-десяткова система BCD не така ефективна як двійкова система. У двійковій системі дванадцятьма бітами можна представити число в діапазоні 0-4095, а в BCD – тільки в діапазоні 0-999. Проте за допомогою BCD набагато простіше і зручніше з погляду сприйняття здійснювати зв'язок з пристроями відображення числової інформації. Кодування BCD використовують також в десятипозиційних перемикачах, які відповідно до позиції перемикачів, видають чотирьохбітовий код десяткової цифри.

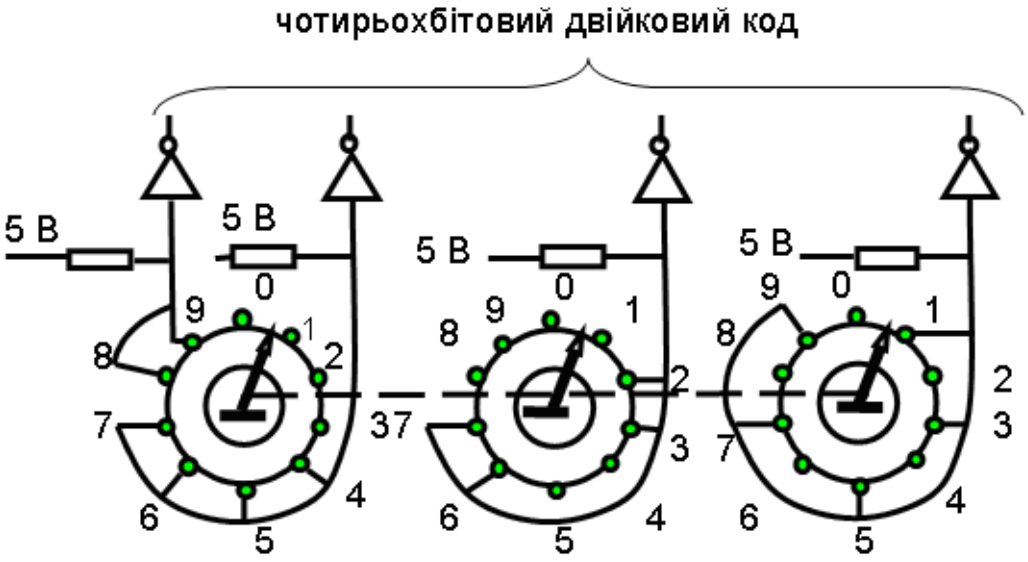


Рис. 1.6. **Пристрій десятипозиційного перемикача, що дає, відповідно до позиції замикаючого на «землю» ключа, на виході чотирьохрозрядний BCD код, відповідній встановленій десятковій цифрі**

Відомо, що окрім чисел в цифровій техніці використовуються букви алфавіту і інші символи, які можуть вводиться з клавіатури або виводиться на друкуючий пристрій. Для опису таких символів часто використовується термін буквено-цифрові символи. Для кодування десяткових цифр, букв алфавіту, спеціальних символів, таких як розділові знаки $,

математичні позначення +,- =, а також символів операцій (як «кінець передачі», «повернення каретки», «дзвінок», «кінець носія» та ін.), що управляють, в даний час використовують код «аски2», який можна отримати на основі універсального стандарту ASCII (American Standard Code for Information Interchange – американський стандартний код для обміну інформацією). ASCII – це 7-бітовий код, що охоплює повний набір буквено-цифрових символів верхнього і нижнього регістрів плюс знаки пунктуації і 32 символи управління, що в цілому дає 128 різних комбінацій.

Семибітову версію цього коду можна знайти в довідниках. Часто цей код доповнюють восьмим бітом для контролю з непарності. При такій схемі цифра 5 кодується наступною бінарною комбінацією 10110101, де старший розряд доповнює число одиниць до непарності.

При передачі інформації використовують поняття «файл», яке позначає послідовність байтів, розміщених на якому-небудь носії, яку можна копіювати, переміщати або змінювати за допомогою програмного забезпечення. Оскільки для представлення байта інформації потрібно тільки два символи шістнадцятирічного коду, то часто використовують шістнадцятирічний файл. При цьому часто виключають розділові пропуски і одержують представлення файлу, відоме як суцільний гекс (straight hex).

Контрольні запитання

1. Стисло охарактеризуйте сучасні технології дистанційного регулювання й позиціювання.

2. Назвіть і охарактеризуйте типові системи управління на мікроконтролерах і БІС програмованої логіки.

3. Охарактеризуйте модульний принцип побудови друкарської машини.

4. Назвіть і охарактеризуйте сучасні технології дистанційного регулювання й позиціювання.

5. Стисло охарактеризуйте типові функції контролерів: управління вимірювальним ланцюгом, управління аналого-цифровим і цифро-аналоговим перетворенням.

6. Поясніть поняття: обчислювальні і тестові функції контролерів.

7. Перерахуйте і стисло охарактеризуйте види мікропроцесорів і різновиду мікроконтролерів.

8. Типові функції високоавтоматизованої системи управління процесом друку і забезпечувальних її додаткових модулів.

9. Назвіть і стисло охарактеризуйте способи представлення і перетворення цифрової інформації.

10. Назвіть способи представлення інформації в мікроконтролерах.

11. Що таке центральний процесор (процесорне ядро) і цифрові шини (внутрішні магістралі) мікроконтролера.

12. Введення – виведення бінарної цифрової інформації.

13. Поняття шини даних.

14. Особливості введення бінарної інформації за допомогою електромеханічного ключа.

15. Стисло охарактеризуйте типові мехатронні системи.

16. Енергетичні (силова) інформаційно-управляюча частини комп'ютеризованої системи.

17. Назвіть і поясніть основні тенденції в сучасній поліграфії.

18. Особливості системної побудови сучасного поліграфічного обладнання і його компонування.

19. Пояснить: модульний принцип побудови друкарської машини.

20. Типовими модулями обладнання друку можуть бути:?

21. Що є одиницею модульного обладнання.

22. Назвіть і стисло охарактеризуйте основні переваги функціонально-модульних систем?

23. Пояснить поняття: робоча машина.

24. Що таке мехатронні системи?

25. Назвіть і стисло охарактеризуйте головну особливість мехатронних систем.

26. Особливості введення – виведення бінарної цифрової інформації.

27. Схема введення бінарної інформації за допомогою електромеханічного ключа

28. Схема виведення бінарної інформації за допомогою світлодіода.

29. Введення – виведення цифрової інформації у вигляді групи біт.

30. Назвіть і поясніть основні способи уявлення і перетворення цифрової інформації

Тема 2. Управління обладнанням друку в комп'ютеризованих системах

Для поліграфічного виробництва є характерними не тільки постійне вдосконалення енергетичної, електромеханічної, електронної елементної бази промислового обладнання. У сучасних умовах, що характеризуються високою інтенсивністю технологічних процесів і зростаючими вимогами до якості продукції, яка випускається, складовою і дуже важливою частиною виробництва друкарської продукції стало вдосконалення управління, як засобами виробництва друкарської продукції, технологіями друку, так і інформаційним потоком.

2.1. Загальні відомості про організацію процесу управління обладнанням виробництва друкарської продукції

2.1.1. Поняття «управління обладнанням»

Управління, в найпершому наближенні, є система, яка забезпечує досягнення певної мети і зростання ступеня організованості. Воно також являє собою сукупність дій, що виконуються на основі одержаної інформації з метою підтримки або поліпшення функціонування обладнання. Відзначимо, що система управління повинна складатися, щонайменше, з двох підсистем: керованого об'єкта і підсистеми, що управляє ним (автоматичного пристрою управління). Керований об'єкт при цьому може бути не один об'єкт, а система, що складається з сукупності взаємодіючих об'єктів. Управління полягає в тому, щоб, впливаючи на об'єкт, змінити протікаючі в ньому процеси так, щоб була досягнута мета управління. Об'єкти управління і підсистеми, що ними управляються, при цьому можуть знаходитися під впливом неконтрольованих обурюючих дій. Основне завдання управління технічним об'єктом або технологічним процесом полягає в тому, щоб знайти і реалізувати в даних умовах алгоритм управління (розпорядження, що визначає зміст і послідовність операцій, які переводять початкові дані в шуканий результат), при якому виконуються всі вимоги, що пред'являються до об'єкта або процесу.

Якщо в системі управління, яка утворюється керованим об'єктом з приєднаним до нього пристроєм, що управляє, процеси управління вико-

нуються без участі людини (автоматично), то система називається системою автоматичного управління (САУ). При цьому передбачається, що пристрій автоматичного управління повинен бути здатний автоматично вирішувати задачі управління, які за швидкістю і об'ємом обчислень не доступні для людського мозку. Щоб можна було уявити собі процеси управління і набір сполучених певним чином основних вузлів, що входять в САУ, на рис. 2.1 наведений приклад узагальненої функціональної схеми системи автоматичного управління.

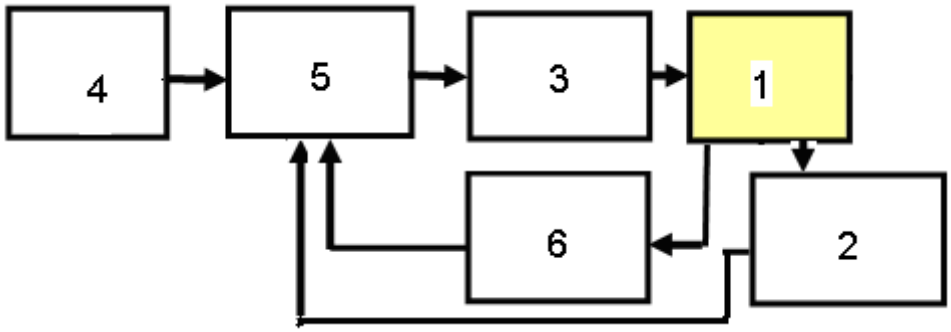


Рис. 2.1. **Узагальнена схема САУ: 1– об'єкт управління; 2 – датчики регульованих змінних (пристрій збору інформації про хід і результати управління); 3 – виконавчий пристрій; 4 – задаючий (еталонне бажане значення поведінки об'єкта) пристрій (джерело інформації про завдання управління); 5 – пристрій порівняння (обробки інформації і вироблення сигналу управління); 6 – спостерігач стану (пристрій отримання інформації про модель об'єкта)**

Найважливішими елементами, які входять до складу САУ і необхідні для її функціонування, є наступні.

1. Об'єкт. Цей елемент схеми становить якийсь виріб з певним принципом дії. Воно виконане відповідно до технічного рішення, яке забезпечує здійснення певних фізичних перетворень. Виріб, у свою чергу, реалізований у вигляді якогось конструктивного рішення, яке виходячи із закладеного в нього принципу дії, визначає конкретні способи з'єднання технічних елементів виробу між собою в просторі. Об'єкт управління характеризується якісними властивостями, які ми називатимемо ознаками і кількісними властивостями – параметрами (змінними, координатами). При функціонуванні об'єкт управління піддається різним силовим і інформаційним діям. Якщо говорити про поліграфічне обладнання, то як об'-

єкт може бути, наприклад, функціональний вузол друкарської машини, що здійснює подачу фарб в зону друку і поєднанню їх на відтисненні. Типовими прикладами вихідних регульованих змінних, що характеризуються числовими значеннями і які часто називають координатами, є швидкість, прискорення, положення робочого приладу машини, момент на вихідному валу двигуна та ін.

2. Датчики регульованих змінних, за допомогою яких одержують інформацію про поточний стан об'єкта управління. Датчики є «очима» системи управління, які дозволяють «бачити» що відбувається в системі управління. Треба мати на увазі, що при проведенні вимірювань, які реалізовуються на основі сучасних датчиків, деякі вимірювання можуть бути недоступні. В цьому випадку важливі частини інформації повинні бути одержані шляхом непрямих вимірювань. З цієї причини в системах управління можуть зустрічатися так звані віртуальні датчики. Якщо частина регульованих змінних або змінних про стан об'єкта не може бути зміряна, то використовується також так званий спостерігач стану (модель об'єкта управління), який відтворює непіддатливі вимірюванню змінні стану. Наприклад, при виготовленні багатобарвної продукції, що відповідає вимогам високої якості, здійснюються або вимірювання оптичної щільності, або колориметричні вимірювання. Тому як вимірювальні датчики можуть бути використані пристрої денситометричного або спектрального вимірювання кольору. Крім того, як відомо, в поліграфічному обладнанні широко застосовуються датчики відносного переміщення, кутової швидкості, тиску, температури.

3. Виконавчі механізми. Такий елемент системи управління необхідний для того, щоб перевести процес з поточного стану в бажаний.

Якщо, наприклад, говорити про регулювання кількості фарб, що подаються, в зону друку, то, як вам відомо, кількість фарби регулюється положенням гвинтів, що змінюють величину щілин в барвистих зонах друкарських секцій, щоб при необхідності збільшувати або зменшувати подачу. Для цілей управління використовуються виконавчі механізми, які переміщають гвинти в задану точку простору і встановлюють їх там із заданою точністю. Переміщення з однієї точки простору в іншу називається позиціонуванням. Тому виконавські органи пристроїв подачі фарби здійснюють необхідне позиціонування гвинтів. Відомо також, що в поліграфічному обладнанні як виконавчі механізми (приводів для клапанів, заслінок і ін.) можуть використовуватися пневматичні, гідравлічні і елек-

тричні виконавчі пристрої, що підрозділяються, у свою чергу, на електромагнітні і електромотори. Всі ці механізми можуть істотно відрізнятися своїми механічними, інерційними властивостями і мати різні види функціональних залежностей між вхідною дією і відгуком, що одержується на виході. Слід мати на увазі, що в деяких САУ виконавчий механізм, як такий, відсутній і дія на об'єкт здійснюється зміною стану будь-якої величини (струму, напруги) без допомоги механічних засобів.

4. В САУ задатчики регульованих змінних (задаючі пристрої) для об'єкта управління, як деякого фізичного виробу, формують (задають) необхідні рівні регульованих змінних (уставки). Задаючий пристрій формує набір тих функціональних залежностей, які ми хочемо одержати від об'єкта при управлінні, і перетворює їх в бажані сигнали управління. На його виході мають місце сигнали управління, які визначають процеси, направлені на досягнення мети управління. Відзначимо, що задатчик може формувати або сигнал управління, незмінний в часі, або змінний в часі за якомось законом. У останньому випадку стан об'єкта у будь-який момент часу характеризують вже миттєві значення команд. Задатчик може формувати також «каталог сигналів управління», на кожній «сторінці» якого приведена одна функціональна залежність, відповідна в цій ситуації бажаному сигналу управління. Наприклад, якщо в процесі друку застосована сушка фарби (за допомогою ІЧ, або УФ випромінювання, або гарячим повітрям), то в САУ, призначеної для підтримки необхідного режиму висихання фарби, задаючий пристрій формуватиме бажаний закон (функцію) зміни в часі температури на поверхні листа. Набір таких бажаних функціональних залежностей, визначуваних типом фарби, товщиною запечатуваного матеріалу, необхідністю подальшого лакування друкарської продукції і складатиме «каталог сигналів управління», що формуються задаючим пристроєм. У зв'язку з великою різноманітністю об'єктів управління різними можуть бути і характеристики, що задаються, які ми хочемо одержати від об'єкта при управлінні: напруга, число оборотів, кутове положення і т. д.

5. Пристрій порівняння. З його допомогою визначається відхилення поточного (фактичного) значення регульованої змінної від заданої. Даний елемент САУ шляхом порівняння бажаних сигналів задаючого пристрою і фактичних сигналів датчиків, що поступають від об'єкта, виробляє сигнал помилки. Через це порівнюючий пристрій іноді називають датчиком помилки, відхилення або розузгодження. Слід мати на увазі, що в

добре спроектованій системі помилка повинна бути мала. Разом з тим на об'єкт управління повинні поступати достатньо могутні дії. Потужності ж сигналу помилки абсолютно недостатньо для роботи виконавчого пристрою. Крім того, сигнали, що поступають від датчиків, перекручені перешкодами і зовнішніми діями, які порушують режим роботи, що може призвести до формування невірних сигналів управління. В цьому випадку сигнал помилки підсилюють і спеціальним чином обробляють.

Відомо, що для того, щоб мати можливість набути хорошого досвіду в системі управління (точність, відсутність перерегулювання при відробіці збурень й ін.) система управління має формувати на своєму виході пропорційно-інтегрально-диференціальний (ПІД) закон регулювання.

Приблизний опис об'єкта або так звана математична модель (оператор) об'єкта управління звичайно будується на основі відомих законів фізики. Усередині блоку порівняння створюється спеціальний обчислювач, який розраховує необхідний сигнал потрібного для виконавчого пристрою рівня. З цієї причини в сучасних системах управління пристрій порівняння є фактично пристроєм обробки інформації і вироблення сигналу управління. Саме цей пристрій формує сигнал дії на даний об'єкт управління, щоб він слідував настільки близько, наскільки можливо, деякій бажаній поведінці. Алгоритми, за якими працює пристрій обробки інформації і вироблення сигналу управління, є «мозком» системи управління. Якщо датчики забезпечують «очі», виконавчі механізми – «м'язи», то алгоритми роботи забезпечують «спритність» об'єкта.

Процеси управління, незалежно від їх внутрішньої суті, завжди вимагають отримання, передачі, переробки і використання інформації. Тому у всякої системи управління однією з основних її функцій є отримання інформації і формування, відповідно до неї, дій на об'єкт управління. З цього виходить, що вбудована в засоби випуску друкарської продукції САУ, є системою інформаційно-управлінською. У ній пристрій автоматичного управління повинен, шляхом збору і обробки інформації про стан об'єкта і зовнішнього середовища, виробляти («обчислювати») рішення про дію на об'єкт і, в автоматичному режимі, формувати команди на їх виконання, забезпечуючи при цьому цільову (бажану) відповідність процесів перетворення енергії, речовини, інформації, підтримку працездатності і безаварійності об'єкта. При створенні САУ важливим є формування трьох інформаційних підсистем: а) для збору і переробки інформації; б) для представлення мети управління у зручній формі; в) для ухвален-

ня рішення про те, що треба зробити для того, щоб забезпечити досягнення мети управління об'єктом.

У САУ завжди можна розрізнити дві тісно переплетені структури – інформаційну (функціонально-алгоритмічну) і апаратну (технічну). Тому сучасна САУ є «симбіоз» (від гр. symbiosis – співжиття, що приносить взаємну користь) апаратних (технічних) засобів і програм (програмного забезпечення), які тісно взаємозв'язані і практично даремні один без одного. Саме єдність апаратної і програмної складових в сучасних засобах випуску друкарської продукції покращує якісні характеристики системи, підвищує точність її функціонування, її надійність і відмовостійкість, додає цим системам такі принципово нові властивості як комерційна життєздатність, гнучкість, адаптивність, перебудову структури, самоконтроль і ін. Програмна частина, в порівнянні з апаратною, є головною. Те, які функції управління реалізуються в поліграфічному обладнанні, визначається, в першу чергу, варіантом алгоритму, покладеним в основу функціонування системи управління і, відповідно, програмним забезпеченням. Програми, в порівнянні з апаратною підтримкою управління, відрізняються великою гнучкістю і простотою в реалізації. Для зміни або вдосконалення технології виконання операції управління потрібно лише змінити програму. До цього слід додати, що створені спеціальні програмні пакети дозволяють різко скоротити витрати часу на створення і корекцію програмного забезпечення. Крім того, програмних ресурсів сучасних пристроїв управління достатньо навіть для підтримки їх програмування на мовах високого рівня. За допомогою програмних засобів легше врахувати ту обставину, що реальні об'єкти управління, як правило, знаходяться під впливом неконтрольованих обурюючих дій. Програмне забезпечення, навіть в складних ситуаціях, відповідно мети управління і обмежень, що накладаються на систему, дозволяє формувати бажані дії управління. Виходячи з сказаного, у тому числі і з економічної доцільності (показника ціна/функціональні можливості), при створенні обладнання намагаються, в співвідношенні об'ємів програмного забезпечення і апаратури збільшити програмну частку і зменшити кількість використовуваної апаратури. Проте, для реалізації САУ дуже важливі і використовувані апаратурні засоби. Вони дозволяють, з одного боку, одержувати інформацію, необхідну для процесу управління (за допомогою датчиків, що визначають режими роботи окремих вузлів обладнання або технологічного процесу), передавати і обробляти її, а, з іншого боку, здійснювати дію на виконавчі

механізми, що впливають на керований об'єкт або протікаючий процес. Апаратурні засоби САУ в даний час виконують у вигляді електронних пристроїв.

Характеризуючи сучасні комп'ютеризовані системи управління поліграфічним обладнанням, слід зазначити, що в даний час в них реалізується концепція функціонально-топологічних і програмних модулів.

Термін функціонально-топологічний перш за все має на увазі, що кожен модуль управління виконує в поліграфічному обладнанні закінчену типову функцію управління. При цьому апаратна (електронна) частина модуля автоматичного управління забезпечує технічне рішення функції управління, збір і обробку інформації, узгодження сигналів. Ця частина реалізована у вигляді спаяного на печатній платі електронного пристрою (блока), зібраного з типових компонентів сучасної електроніки. У загальному випадку модуль характеризується конструктивною і схемотехнічною завершеністю, володіє строго фіксованими функціональними характеристиками. Принцип модульності також припускає, що з'єднання модулів здійснюється за уніфікованими правилами. Функціональна, електрична і конструктивна сумісність модулів забезпечується уніфікацією інтерфейсу (системи зв'язків, сигналів і алгоритмів обміну). Сукупність уніфікованих правил, що реалізовуються в модулі для організації передачі даних, іноді називають логічним інтерфейсом (протоколом обміну).

Програмний модуль забезпечує реалізацію процедур (алгоритмів) управління і виконання заданих технічних вимог для кожного функціонально-топологічного модуля. Програмна частина модуля формально описує алгоритми дій вузлів друкарської машини і формує команди, що забезпечують реалізацію заданого алгоритму обробки інформації. Кожна команда програми містить інформацію про те, що потрібно робити на даному етапі виконання операції. Команди виконуються в покроковому режимі в строго заданій послідовності і, в цілому, утворюють програму.

Функціонально-модульна структура програмного забезпечення робить його достатньо гнучким в частині модернізації, а також використання в конкретних завданнях забезпечення роботи обладнання. Програмне забезпечення при цьому фігурує як обов'язкова складова частина обладнання, роль якого в процесі управління постійно зростає.

При створенні програмного модуля всі технологічні дані, потрібні для здійснення процесу друку, як, наприклад, значення швидкості обертання циліндрів, крутильного моменту, зусилля, ступеня переміщення

паперу, кількість подачі фарби, напрям переміщення робочих органів, послідовність операцій і т. д., задаються у вигляді чисел. Сукупність всіх чисел утворює цифрову програму, необхідну для виконання якої-небудь операції друку. При цьому програма заздалегідь відповідним чином розраховується і поміщається на носій програм. Програма записується на програмоносії у вигляді окремих блоків. Кожен блок містить всю інформацію про параметри процесу на даному етапі технологічного циклу, наприклад, про кутову швидкість валу, про напрям і швидкість переміщення по координатних осях фотодатчика, швидкості лінійного переміщення штока виконавчого механізму, режимі сушки і т. д.

Програма (сукупність команд) пристрою управління, вбудованого у функціонально-топологічному модулі, записується у вигляді двійкових чисел, тобто у вигляді так званого машинного коду. Цей код, будучи машинно-орієнтованою мовою програмування, складається з великого числа нулів і одиниць. Він важкий для сприйняття. Зручнішим є мнемокод і використання мови Асемблера. У цій мові замість кодових комбінацій використовується мнемонічна форма запису операцій. Таким мнемонічним записом (у вигляді поєднання букв, узятих від відповідних англійських слів) представляється вид виконуваної операції. Мова Асемблера спрощує запис команд, забезпечує кращий огляд програми, полегшує пошук в ній помилок, забезпечує простоту внесення виправлень в записану на цій мові програму. Ще більші можливості надає використання мови програмування вищого рівня типу СІ. Складені на цих мовах програми перед виконанням транслюються (переводяться) за допомогою спеціальних програм в систему машинних кодів, «зрозумілих» мікроконтролеру і у такому вигляді поміщаються в пам'ять програм мікроконтролера.

2.1.2. Еволюція систем управління обладнаннями

Для розуміння логіки розвитку систем управління треба мати на увазі, що важливу роль в історії САУ зіграло десятиліття з 70 – 80-х років. Завдяки успіхам мікроелектронній технології і вдосконаленню технології виготовлення інтегральних схем (ІС) з'явилася можливість розміщувати на одному кристалі мільйони електронних елементів, що реалізовують функціональні вузли цифрових пристроїв (трігери, лічильники, суматори та ін.). При цьому виготовлені на одному напівпровідниковому кристалі схеми цифрових пристроїв характеризувалися високою надійні-

стю, малими габаритними розмірами і масою, низьким енергоспоживанням і вартістю. У електроніці відбувся якісний стрибок, оскільки з'явилася можливість виготовляти основні вузли електронної обчислювальної машини (ЕОМ) в габаритах однієї мікросхеми (на одному кристалі). Однокристальна МІКРО-ЕОМ дозволила об'єднати в своєму складі основні пристрої, необхідні для того, щоб набудувати її на виконання будь-яких функцій. До того ж, досить було цю «голу» ЕОМ «одягнути» в зовнішні пристрої, додати до неї джерело живлення і вона вже була закінченим конструктивним вузлом, що працює не гірше за «велику» ЕОМ. Оскільки «здібності» ЕОМ (у вигляді програм) зберігаються в пам'яті, то технологія flash-пам'яті забезпечила різке зниження вартості мікросхеми з перезаписуваною пам'яттю і створила можливість відносно просто програмувати такі схеми. Поява «великих» ІС з можливостями програмування призвела до того, що подібні ІС, в сенсі реалізації заданих функцій, стали універсальним засобом автоматики. Одна велика універсальна ІС стала застосовуватися як пристрій збору, обробки інформації і вироблення сигналів управління. Така мікросхема, що виконує основні функції управління, мала можливість гнучкої і оперативної перебудови, як алгоритму роботи, так і структури мікросхеми з метою пристосування її до змінних умов роботи. Ця ІС з великою кількістю функціональних вузлів дозволила на практиці здійснити ідею автоматичного управління вузлами обладнання за допомогою фактично однієї програмно-керованої універсальної мікросхеми. У технічній літературі таку інтегральну схему, використовувану у складі САУ, назвали мікроконтролером (МК).

Мікроконтролер, орієнтований на використання для управління технічними об'єктами і технологічними процесами, має багато загального з ЕОМ (комп'ютером). МК – це пристрій, який, також як і ЕОМ, здатний виконувати наперед встановлені операції над вхідними даними, щоб утворювати нові вихідні дані. У МК є «серце» ЕОМ – центральний процесор (процесорне ядро). Він здатний інтерпретувати і виконувати всі команди, що зберігаються в пам'яті, включаючи складну логіку управління, засновану на виконанні арифметичних і логічних операцій. Мікроконтролер має пристрій (пам'ять), що запам'ятовує, зберігає послідовності команд, які виконує центральний процесор. МК є легко перебудовуваним. Один і той же мікроконтролер під управлінням різних програм може виконувати різні завдання управління, що забезпечує його високу універсальність. Мікроконтролер, що володіє універсальними функціями обчис-

лень і управління, подібно до ЕОМ, здатний вирішувати складні задачі. Разом з тим мікроконтролери, використовуваним для управління обладнанням, мають істотні відмінності від комп'ютерів, які вирішують різні прикладні завдання. До мікроконтролерів пред'являються абсолютно інші вимоги, чим до великих ЕОМ.

Звичайний комп'ютер орієнтований на автоматизацію розумової діяльності і рішення складних обчислювальних задач. Він, як правило, одержує інформацію у вигляді даних з клавіатури, а одержані результати виводить на дисплей або принтер у формі зручної для сприйняття людиною. Зовсім інакше йде справа з управлінським МК. На його вхід поступає інформація про стан вузлів, тобто сигнали від великої кількості різноманітних датчиків і вимірювальних приладів. Багато з таких сигналів є сигналами логічного управління, дискретними однобітовими сигналами типу «включено-виключено». Інші сигнали є напруги з датчиків у вигляді деяких чисел, що характеризують температуру, тиск, значення витрати, або аналогових сигналів у вигляді струмів. Щоб можна було «одержати» ці сигнали для обробки в МК потрібні не клавіатура, а спеціальні «з'єднувальні» вироби і блоки, що поставляють інформацію. Крім того, при обробці одержаної інформації не завжди потрібно виконувати складні арифметичні операції.

Мікроконтролер, який управляє обладнанням, повинен бути здатним нормально працювати в «поганих» умовах навколишнього середовища. Він повинен працювати без втручання людини при великих коливаннях температури, за наявності пилу і «грязі», поява яких обумовлена виробничими умовами. МК повинен бути здатним нормально працювати при «брудних» джерелах живлення і адекватно реагувати на будь-які пропажі живлення. При цьому деякі вихідні сигнали повинні повертатися до тих значень, які вони мали до пропажі живлення, інакше подальша робота механізму може ініціювати поломку машини.

Якщо при використанні звичайної ЕОМ людина, що сидить біля комп'ютера, має можливість контролювати функціонування комп'ютерної програми і втручатися в її роботу, то МК сам повинен «спостерігати» за роботою обладнання і «виявляти» несправності в своїй роботі, оскільки функціонування обладнання звичайно відбувається без втручання оператора. Мікроконтролер повинен «уміти» виявляти несправності в керованому обладнанні або «підказувати» оператору, де в обладнанні виник дефект. Пристрій управління повинен реагувати на всі виникаючі події

негайно після їх появи, тобто повинен «уміти» працювати в реальному масштабі часу. Всі програми, що управляють, не повинні містити помилок, і бажано, щоб мова програмування МК була зрозуміла обслуговуючому персоналу, який навіть не має спеціальної підготовки.

Вже з 70-х років минулого сторіччя системи управління на основі МК швидко набули масового поширення. Для того, щоб збільшувати можливості системи управління розробники мікроконтролерів спочатку йшли шляхом збільшення розрядності (числа бітів даних оброблюваних процесором) – переходили від 8- до 16- або 32-розрядним ІС. Збільшення розрядності оброблюваного слова навіть при незмінній частоті тактування істотно збільшує продуктивність (вона вимірюється в мільйонах команд за секунду – MIPS). При цьому кожного разу новий рівень продуктивності процесорного ядра супроводжувався вдосконаленням структури периферійних пристроїв. Для створення інформаційно-управллінських систем того періоду, вбудованих у тому числі і в засоби випуску друкарської продукції, використовувалися спеціалізовані електронні обчислювальні пристрої, іменовані МІНІ-ЕОМ, які за схемоюобслуговування окремих вузлів і крайових пристроїв можна назвати інформаційно-управлінськими системами централізованого управління.

У другій половині 90-х років ситуація в системах автоматичного управління на основі мікроконтролерів істотно помінялася. Спочатку швидке здешевлення мікроконтролерів привело до доцільності заміни ними ряду малонадійних і дорогих компонентів. Наприклад, конструктори стали використовувати МК для того, щоб, замінювати механічні кнопкові перемикачі на сенсорні перемикачі. Відповідні МК виконували необхідні для цього операції. Потім знову основою для створення сучасних систем автоматичного управління багатоцільового призначення стали сучасні 8-розрядні МК. Відомо, що для алгоритмів, не пов'язаних з великим об'ємом обчислювальних операцій і переважанням логічних операцій, збільшення розрядності оброблюваного слова практично не позначається на продуктивності. Ця обставина є головною причиною того, що 8-розрядні МК можуть з успіхом реалізувати багато завдань управління і не вимагають процесорів з вищою розрядністю оброблюваного слова. Це обумовило збільшену потребу в 8-розрядних МК і, як наслідок, постійне розширення номенклатури 8-розрядних МК і появу на ринку великої кількості їх моделей. Якісно нові можливості при виготовленні 8-розрядних мікроконтроллрних систем управління дістали тоді, коли в мікроконтро-

лер були інтегровані цифрові вимірювальні засоби, стандартні пристрої введення і виведення цифрової інформації. Це дозволило застосовувати спільно з 8-розрядним МК стандартні цифрові інтерфейси і промислові функціональні блоки, сумісні між собою за інформаційними, метрологічними, енергетичними і конструктивними характеристиками.

Нарешті, змінилися принципи, що становлять ідеологію побудови системи управління. Були прийняті до керівництва доводи, чому небажано передавати всі сигнали в одну точку. Змінився баланс між централізацією і децентралізацією управління із-за очевидних міркувань, пов'язаних з обмеженнями за часом обчислень, ремонтопридатності, надійності, складності і вартості.

Слідством використання 8-розрядних МК з'явилося те, що в сучасних системах управління обробка і проміжне зберігання інформації стали виконуватися в місцях, максимально наближених до її отримання. МК стали інтегрованими – embedded controllers – (вбудовуваними) безпосередньо в об'єкт управління. «Локальний» мікроконтролер придбав здатність «працювати» тільки з своїм об'єктом управління і обробляти тільки «свою» інформацію. Впровадження МК безпосередньо в об'єкт управління, в системи управління окремими (локальними) технологічними процесами, призвело до створення розподіленої мікроконтролерної системи, яка володіє рядом привабливих можливостей. Система такого типу добре «вписується» в модульний принцип компоновки, є більш ремонтопридатною. Це відкрило нові можливості не тільки в питаннях розосередження апаратури і процесів управління по підсистемах, але і в децентралізації функцій обробки інформації, в розподілі управління між автономними пристроями управління. Такий підхід до управління дозволяє одночасно здійснювати операції обробки інформації в декількох місцях обладнання, істотно зменшити потоки інформації, зробити раціональнішим обмін інформацією між функціональними пристроями і вищими ступенями ієрархічної системи управління. У таких системах, що працюють у реальному часі, об'єднуються процедури вимірювання і обробки інформації. При цьому вимірювальні і обчислювальні процедури виконуються одночасно і нерозривно. З'явилася можливість попереднього опрацювання локальних даних і пересилки ведучому МК тільки результатів, і то лише у разі потреби.

Достоїнством розосередженої системи є розподіл функціональних завдань з управління, що призводить до більшої структуризації програм-

ного забезпечення, а значить, до зниження кількості помилок при його модифікації, до зменшення часу появи відгуку обладнання на якийсь вхідний сигнал. Завдяки «паралельній» обробці інформації, з'являється можливість використання «локальних» програм. Відносна простота програмування і контроль стану обладнання обчислювальними засобами зменшує вартість системи в цілому.

Використання розподілених систем управління обладнанням дає можливість робити деякі кроки у напрямі інформаційних технологій, пов'язаних з «штучним інтелектом». Відзначимо, наприклад, що в даний час вже продається безліч датчиків з вбудованим мікроконтролером. Такі датчики прийнято називати «розумними – smart». У «розумному» датчику може безпосередньо проводитися обробка вимірювань. Наприклад, працюючи під управлінням мікроконтролера «розумний» датчик здатний вирішувати задачі виділення корисного сигналу з шумів, перетворювати масштаб, враховувати вплив на свідчення температури навколишнього середовища, одержувати лінійну залежність від нелінійного датчика та ін. На додаток до сигналів тривоги, що свідчать про перевищення заданого значення сигналу, «розумний» датчик може самостійно протоколювати вимірювані дані і потім передавати протокол на пульт оператора.

2.2. Характеристика архітектури (структури) розподіленої комп'ютеризованої системи і основних функцій пристроїв управління

Для характеристики організації і структури сучасної комп'ютеризованої системи автоматичного управління, що уявляє собою взаємозв'язану безліч програмно-апаратних засобів, буде недостатнім говорити лише про те, що стосується складу і взаємозв'язків вхідних в неї елементів. У такій системі є можливість, за рахунок програмної реалізації, змінювати склад елементів, взаємодію різних елементів один з одним. Крім того, за допомогою послідовності сигналів управління в такій системі можна змінювати функції, що виконуються апаратурою. З цієї причини вводится поняття «архітектура», яке включає весь комплекс програмних і апаратних засобів, за допомогою яких виконується завдання користувачів з управління. Поняття «архітектура» дозволяє визначити склад, призначення, логічну організацію і порядок взаємодії функціональних засобів, об'єднаних для вирішення завдання певного вигляду. Знання архіте-

ктурних рішень дозволяє оцінювати особливості застосування управлінської системи, дає можливість ефективно розпоряджатися всіма ресурсами, що надаються. Архітектура об'єднує апаратні і програмні засоби і, в зв'язку з цим, дозволяє чітко виділити те, що при створенні системи повинно бути реалізовано користувачем програмним шляхом або апаратними засобами. Архітектура також відображає функціональні можливості електронних засобів, використовувані для представлення даних, опису алгоритмів і процесів обчислень. «Архітектура» як би характеризує систему в цілому, показує, як система представляється користувачу.

Сучасну комп'ютеризовану систему, що інформаційно-управляє поліграфічним обладнанням, вбудовану в засоби випуску друкарської продукції, можна умовно розділити на два рівні (рис. 2.2). На першому (нижньому) рівні, який часто називають локальним, периферійним, розташовані пристрої управління (мікроконтролери), які здійснюють взаємодію з периферійними об'єктами (що забезпечують процес друку) і виконують збір інформації про стан керованого об'єктата безпосереднє управління окремими механізмами друкарської машини.

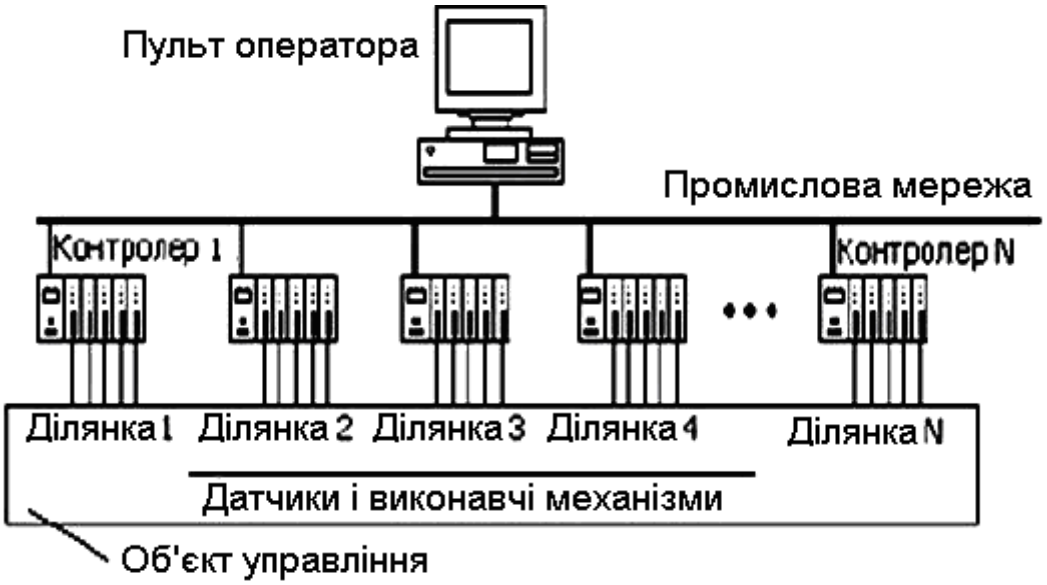


Рис. 2.2. **Розподілена система, що інформаційно управляє, представлена на двох рівнях управління**

Приклад. МК, що здійснюють взаємодію з периферійними об'єктами, можуть вирішувати завдання підтримки необхідної температури на окремих вузлах друкарських апаратів. Так, для роботи в режимі друку

без зволоження барвистий апарат машини оснащується трьома охолоджуваними водою розкатними циліндрами. Приєднання охолоджуючої системи до розкатних циліндрів, що обертаються, виконане за рахунок клапанів-втулок і розподільних блоків. Для підтримки температури барвистого апарату до машини можна приєднати обладнання, що забезпечує охолоджування або нагрів циркулюючої води і оптимальне регулювання температури води, яка поступає від друкарських апаратів. Для забезпечення незалежного регулювання температури води призначено пристрій управління. Він містить контур термостатирування, який підтримує однакову температуру на друкарських апаратах. При цьому температура барвистого апарату вимірюється на накатному валику за допомогою інфрачервоного датчика.

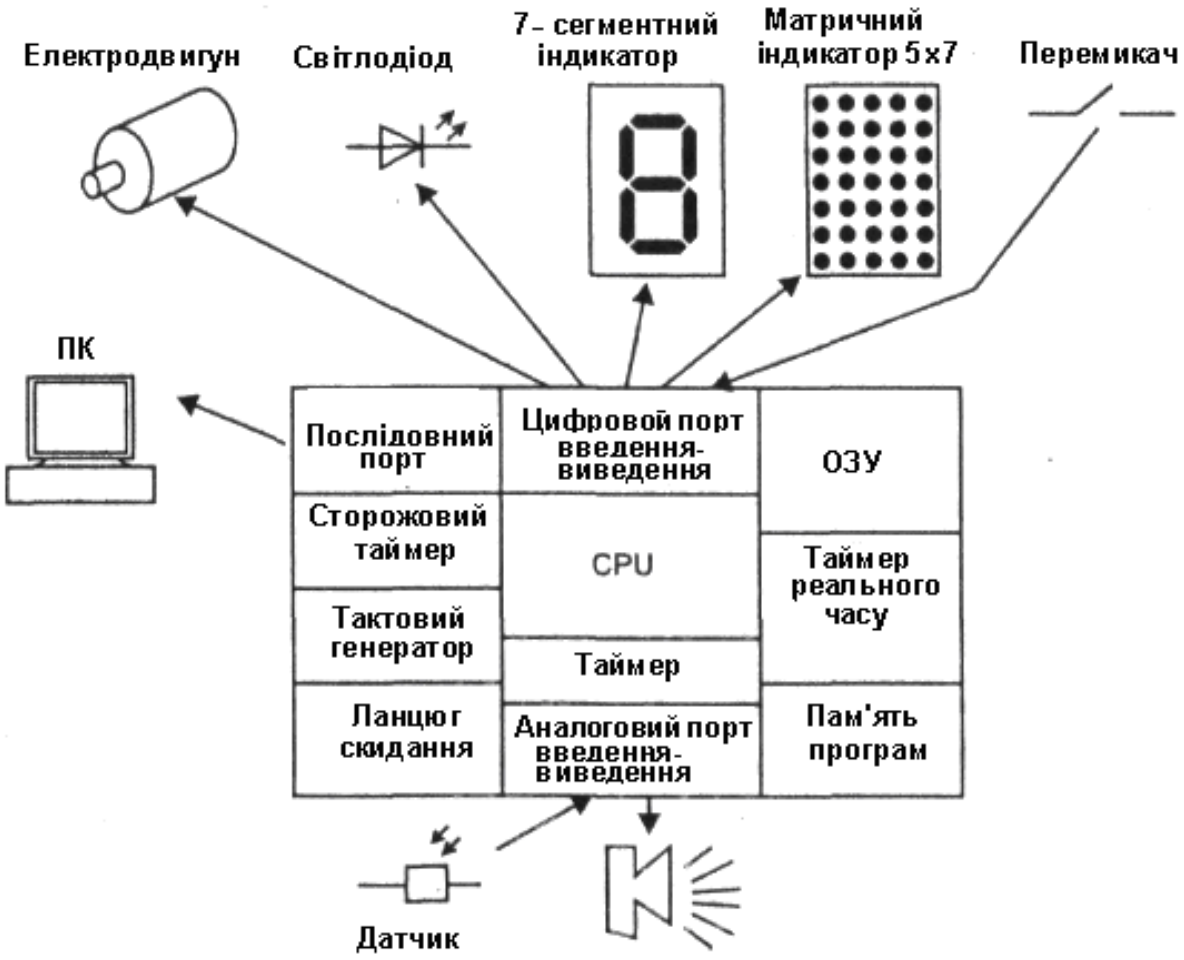


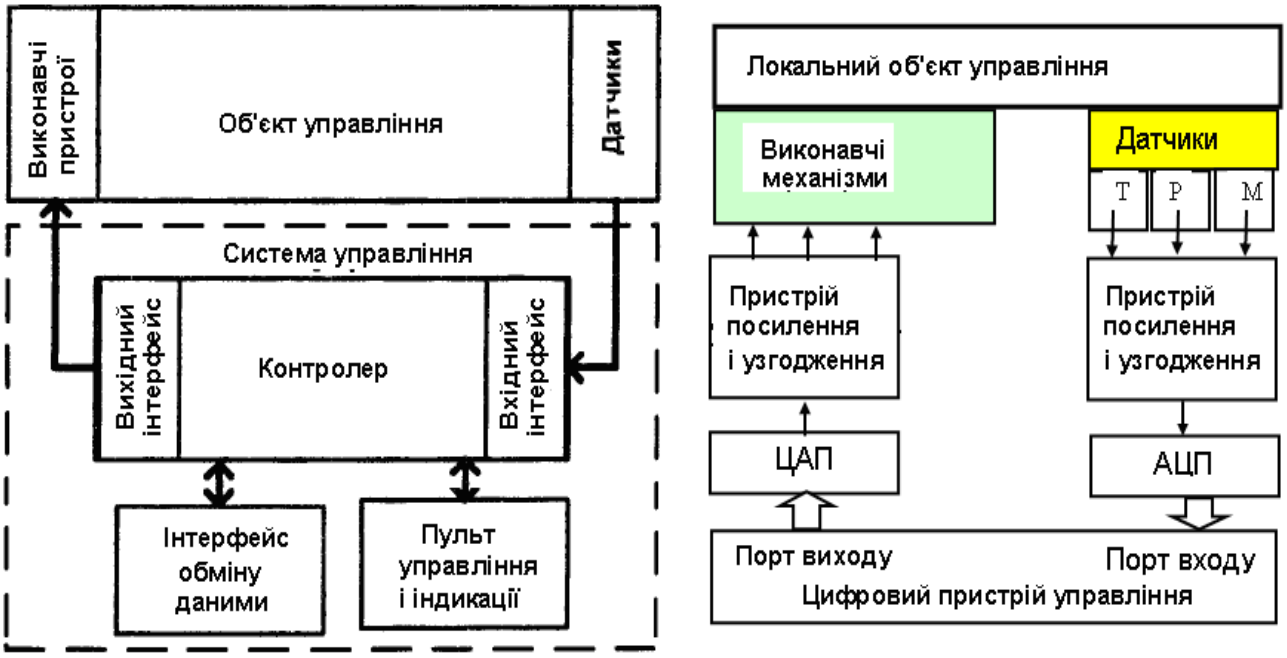
Рис. 2.3. **Узагальнена схема взаємодії пристрою управління об'єктами нижнього рівня з периферійними пристроями**

МК, розташовані на першому рівні, звичайно вирішують наступні задачі (рис. 2.3): 1) виконують збір сигналів від датчиків, встановлених на

об'єкті управління, а також з пультів ручного введення даних (для цього у них передбачені аналогові і цифрові порти введення інформації); 2) здійснюють попередню обробку сигналів (фільтрацію і масштабування); 3) реалізують алгоритми управления 4) формують сигнали управління на виконавчі механізми об'єкта управління (на двигуни, засоби відображення інформації, формувачі звукових сигналів і ін.). Для досягнення мети управління, передбаченої в пп.2, 3, 4, в МК є спеціальний обчислювальний пристрій (CPU), що виконує арифметичні і логічні операції, а також пристрої синхронізації сигналів управління; 5) виконують виведення інформації у формі, зручної для сприйняття людиною (візуальної і звукової) або виконавчим механізмом (для цього в мікроконтролерах передбачені аналогові і цифрові порти виведення інформації); 6) здійснюють взаємодію, як між собою, так і з пристроями управління вищого рівня, виконуючи передачу і прийом інформації. Для цього в мікроконтролерах передбачені різного роду таймери і різні спеціальні пристрої.

Для пристрою управління об'єктами нижнього рівня дуже важливими є властивості, що характеризують «уміння взаємодіяти» з багатьма зовнішніми (по відношенню до МК) пристроями. Вбудовуваний в обладнання МК, повинен легко сполучатися з датчиками, оперуючими сигналами малої амплітуди, для визначення режимів роботи, стану окремих частин об’єкта або ходу технологічного процесу. Локальний МК повинен уміти «взаємодіяти» з виконавчими механізмами, що впливають на протікаючий процес, які вимагають досить могутніх енергетичних дій.

Типову структуру системи управління нижнього рівня в найзагальнішому вигляді можна представити таким чином (рис. 2.4 а). Елементи такої системи управління виконують наступні функції: 1) вхідний інтерфейс – прийом і перетворення сигналів від датчиків у формат, зручний для подальшої обробки; 2) вихідний інтерфейс – перетворення вихідних даних системи в сигнали управління виконавчими пристроями; 3) контролер (основний елемент системи управління) реалізує алгоритми управління і обробки даних відповідно до поставлених завдань, потоку даних вхідного інтерфейсу, управлінських команд пульта управління та інтерфейсу обміну з іншими засобами управління; 4) пульт управління і індикації, що містять засоби управління і відображення даних про параметри і режими роботи обладнання, дозволяє людині «спілкуватися» з системою управління.



а) б)

Рис. 2.4. **Структурна схема системи управління нижнього рівня, представлена в найзагальнішому вигляді (а) та узагальнена схема вбудованої в локальний об'єкт системи, що інформаційно-управляє (embedded controllers) (б). Т, Р, М – датчики температури, тиску, моменту**

Щоб датчики і виконавчі механізми, що працюють з аналоговими сигналами різної амплітуди і форми уявлення, якомога простіше сполучалися з МК, оперуючими цифровими сигналами, щоб самі МК добре «вписувалися» у функціональну схему сучасної аналого-цифрової системи управління об'єктом (рис. 2.4 б), було потрібно, щоб в МК нижнього рівня були компоненти перетворення аналогових даних в цифрові і навпаки. У сучасного мікроконтролера підсистема введення інформації оснащена пристроями аналогового введення. Такі пристрої фактично становлять сумісну з МК систему збирання аналогових даних. Вона призначена для введення в МК аналогових сигналів з датчиків фізичних величин і перетворення цих сигналів в двійковий код з метою подальшої обробки одержаних даних. Така система звичайно містить: аналоговий мультиплексор (комутатор, перемикач), АЦП, логіку дешифрації адреси для вибору каналу, схеми синхронізації і буферні регістри на виході для зберігання перетворених цифрових даних. Багатоканальний аналоговий

комутатор служить для підключення одного з джерел аналогових сигналів. Вибір джерела здійснюється за допомогою запису номера каналу комутатора. Відмітною особливістю такої системи збору аналогових даних є те, що вона поміщена (інтегрована) на кристал великої інтегральної схеми мікроконтролера. Для взаємодії з виконавчими механізмами, що працюють з аналоговими сигналами, сучасний МК оснастили вбудованим ЦАП, який підключений до портів введення-виведення.

У загальному випадку система автоматичного управління з цифровим пристроєм (УЦП), що управляє, становить замкнений контур управління (рис. 2.5.).

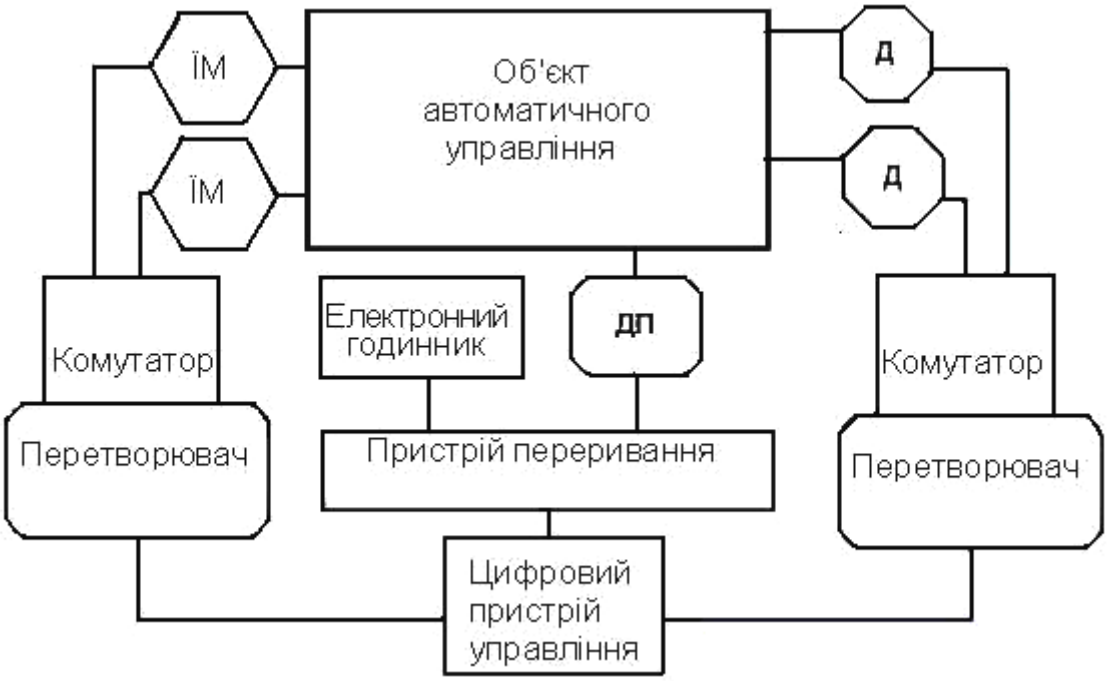


Рис. 2.5. **Структурна схем інформаційно-управлінської системи, що являє замкнутий контур**

При цьому система управління будується таким чином, що окремі параметри об’єкта управління регулюються відповідними автоматичними регуляторами, а ЦПУ обробляє вимірювальну інформацію, розраховує і встановлює оптимальні настройки регуляторів. На вхід ЦПУ від датчиків Д (термопар, витратомірів і т. д.) подається вимірювальна інформація про поточні значення параметрів. Згідно з алгоритмом управління ЦПУ визначає величини дій, які необхідно прикласти до виконавчих механізмів (ІМ) для зміни регульованих параметрів, що впливають на регулятори, з тим, щоб процес управління протікав оптимально. Вимірювальні да-

тчики виробляють безперервний сигнал (напруга, струм, кут повороту і т. д.), а оскільки УЦУ працює з цифровими сигналами, то двічі відбувається перетворення з безперервної форми в дискретну і навпаки. Для зменшення складу обладнання в системі управління перетворювачі інформації можуть бути виконані багатоканальними. За допомогою комутатора перетворювач по черзі підключається до кожного датчика і здійснюється перетворення безперервної величини в дискретну. Дія управління, що поступила, зберігається до вироблення наступної дії управління ЦПУ. Щоб забезпечити функціонування системи управління в певні інтервали часу і засинхронізувати процеси, що відбуваються, потрібен «годинник».

Між МК і периферійними пристроями крім даних передаються сигнали управління, які названі наказами контролера або словами управління (командними). Загальна система сполучення МК з каналами введення – виведення і периферійними пристроями одержала назву інтерфейсу. Для здійснення взаємодії з периферійними пристроями нижнього рівня МК можуть виконувати наступні основні типові функції: 1) вироблення сигналів управління; 2) мультиплексування сигналів і управління вимірювальним ланцюгом; 3) управління аналого-цифровим і цифро-аналоговим перетворенням; 4) управління засобами спілкування з оператором; 5) управління зовнішньою пам'яттю, реєстрація оперативної інформації і виведення результатів і службових сигналів для документування та архівації; 6) підтримка взаємодії з іншими модульними контролерами; 7) обслуговування, шляхом самостійного аналізу значень сигналів з деяких датчиків, підсистеми безпеки.

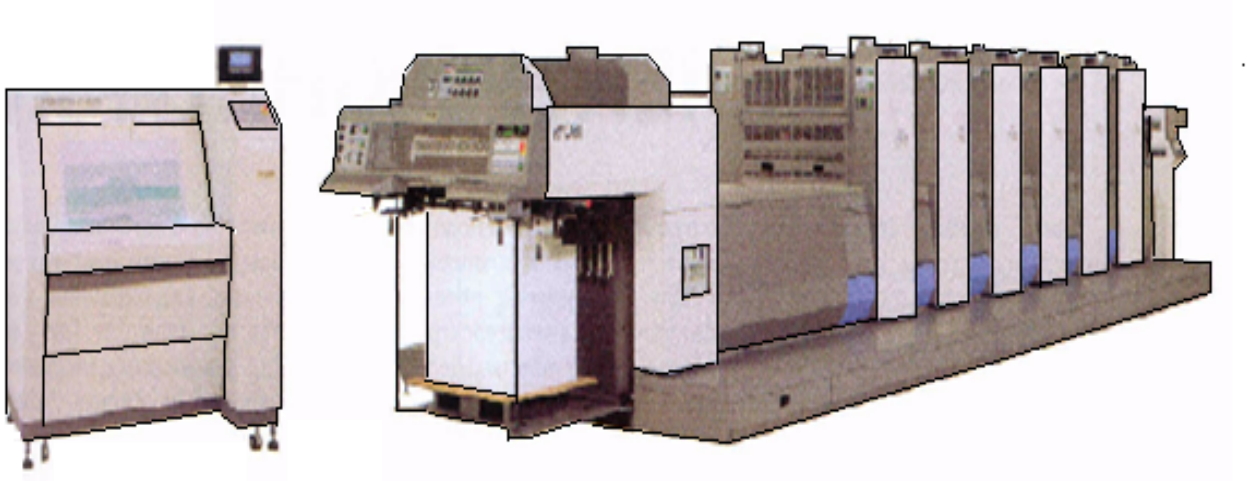


Рис. 2.6. **Пульт управління друкарською машиною**

На другому рівні у сучасного комп'ютеризованого обладнання розташована виконана на основі комп'ютера робоча станція, яка має систе-

му управління обладнанням друкарської машини і растровий процесор. Звичайно її називають комп'ютером з пультом управління (рис. 2.6). Центральний (дистанційний) пристрій управління через комунікаційну лінію пов'язаний з виконавчими механізмами і датчиками друкарської машини. Він служить для вироблення і передачі команд управління, а також для отримання інформації про поточний стан машини.

Комп'ютер другого рівня управління є компонентом інформаційної і виробничої систем. Растровий процесор дозволяє проводити електронний підбір і персоналізацію кольорів, багатоколірне растрування і реалізовувати інші можливості цифрового друку. Крім того, комп'ютер друкарської машини з пультом оператора надає оператору різну лаконічну інформацію про параметри і режими роботи окремих частин обладнання і дозволяє вводити для локальних МК початкові дані. За допомогою пульта управління оператор одержує різноманітну інформацію про поточний стан виконання завдання. Для простоти орієнтації друкаря прості друкарські машини оснащуються дисплеєм з світлодіодами LED, які відображають робочий стан основних функціональних вузлів, блоків і елементів системи управління і захисту. Складні друкарські машини оснащені діагностичним дисплеєм, який, крім поточної інформації про стан вузлів, елементів управління і захисту, відображає також причини дефектів і іншу інформацію, необхідну для ремонтників. При цьому звичайно застосовуються сенсорні дисплеї, що працюють в декількох режимах. До пульта управління можна підключати електронні і оптичні вимірювальні системи. За допомогою мережевої передачі даних на комп'ютер виводяться відомості не тільки з друку, а й про етапи додрукових процесів. Так, наприклад, можуть бути представлені як відомості про план виробництва, завантаження машини і терміни виконання завдань, так і стан обробки замовлення на даний момент. За допомогою комп'ютера з пультом управління в автоматизованому режимі ведуться підготовчі і налаштувальні роботи, необхідні для початку подачі паперу в машину. У даний час з пульта відповідно до специфіки тиражу можна виконати практично всі регулювання, що дозволяє істотно скоротити час на підготовку машини до друку. Відзначимо, що застосування комп'ютера верхнього рівня з можливостями дистанційного керування найважливішими функціями друкарської машини стало звичайним рішенням. Звернемо увагу на те, що за допомогою пульта оператора не тільки формуються запити до контролерів нижнього рівня, але і від них одержують оперативну інформа-

цію про хід технологічного процесу. На екрані монітора пульта управління відображають весь хід технологічного процесу в зручному для оператора вигляді. За допомогою пульта здійснюють довготривале зберігання динамічної інформації (ведення архіву) про хід процесу, проводять корекцію необхідних параметрів алгоритмів управління і уставок регуляторів в контролерах нижнього рівня. Наявність у складі засобів верхнього рівня управління дисплея, клавіатури і створення відповідного програмного забезпечення стосовно конкретного технологічного об’єкта дозволяє організувати діалоговий режим спілкування з оператора з системою управління.

Друкарські машини можуть бути укомплектовані системами дистанційного керування барвистими апаратами і регістрами приведення. Ця комп'ютерна система полегшує обслуговування та істотно скорочує час, необхідний для підготовки машини і виконання нового або повторного замовлення. Окремі регулятори управляються з самостійного пульта, де розміщені й інші елементи управління, включаючи монітор. Задані величини окремих технологічних параметрів і режимів можна зберігати на дискеті.

Для полегшення роботи оператора центральні пульти управління останнім часом оснащують сенсорними екранами. Це дозволяє друкарю легким дотиком руки виконувати всі операції з управління друкарською машиною: пуск, регулювання швидкості, зупинку машини, приведення формових циліндрів і ін.

Важливим застосуванням комп'ютера верхнього рівня в АСУ є його робота в режимі порадника оператору. Він при цьому працює в розімкненому контурі, тобто його виходи не пов'язані з органами управління технологічними процесами, а необхідна інформація виводиться лише на пристрій відображення. Одночасно система з закладеної в неї математичної моделі обчислює «поради оператору», необхідні для наближення режиму функціонування обладнання до оптимального. Ці значення виводяться на засоби відображення.

Комп'ютерами верхнього рівня в реальному масштабі часу можуть розв'язуватися наступні завдання: 1) збір поточної інформації від контролерів нижнього рівня або інших приладів і пристроїв, зв'язаних безпосередньо або через мережу з пультом оператора; 2) обчислювальна і логічна обробка вимірювальної інформації; 3) архівація і зберігання поточної інформації та її подальша обробка; 4) представлення поточної та

історичної інформації на дисплеї у вигляді мнемосхем, гістограм, анімаційних зображень, таблиць, графіків, трендів; 5) виділення аварійних і передаварійних ситуацій з автоматичною генерацією сигналів тривоги; 6) введення і передача команд і повідомлень оператора в контролери і інші пристрої системи; 7) реєстрація всіх дій оператора (ручний запуск процесу, аварійний останов, зміна настроювальних параметрів системи і т. д.); 8) реєстрація всіх помилок і подій усередині системи управління (апаратні сигнали тривоги, помилки роботи мережі і т. д.); 9) захист від несанкціонованого доступу і надання різних прав користувачам під час роботи з системою; 10) формування звітів і протоколів довільної форми в задані моменти часу, уявлення і запис аварійних ситуацій в моменти їх виникнення; 11) рішення прикладних програм користувача та їх взаємозв'язок з поточною вимірюваною інформацією і управлінськими рішеннями; 12) забезпечення інформаційних зв'язків з серверами кольорового друку та іншими робочими станціями через різні мережеві структури.

Під контролем звичайно розуміється встановлення відповідності між станом (властивістю) об’єкта контролю і заданою нормою, що визначає якісно різні області його стану. В результаті контролю видається думка про стан об’єкта контролю, про те до якої з нормованих областей, що якісно розрізняються, відноситься даний стан об’єкта контролю. Часто для відновлення нормальної роботи об’єкта необхідно виявити елементи, що послужили причиною його неправильного функціонування. Такий напрям контролю роботи технічних пристроїв називається технічною діагностикою. Діагностична інформація про відмови і порушення в роботі окремих модулів формується в мікроконтролерах, що забезпечують автоматичний збір і первинну обробку інформації. У підсистемі автоматичного збору і обробки інформації формуються діагностичні дані про стан: а) виконавчих пристроїв; б) датчиків аналогових і дискретних сигналів; в) ліній зв'язку. Всі комп'ютеризовані засоби самодіагностики визначають конкретну адресу несправного модуля. Діагностичні повідомлення поступають і фіксуютсяся на пульті оператора. Для діагностики тестовою апаратурою виробляються імітаційні сигнали. В результаті діагностування видається умовний номер несправного блоку і номера можливих несправних вузлів в даному блоці. У розвиненіших системах діагностики видається також код несправності (наприклад, обрив – 01, коротке замикання – 02 і т. д.). Наявність спеціальних кодів стану поліграфічного обладнання істотно полегшує процедури проведення ремонту.

2.3. Інтерфейс – найважливіша частина комп'ютеризованих систем керування поліграфічним обладнанням

У комп'ютеризованих системах як переносник інформації між її джерелом і приймачем виступають електричні сигнали. Ці сигнали повинні бути узгодженими. Щоб уникнути непотрібних непорозумінь при обміні інформацією придумали правила взаємодії групи компонентів системи, а також правила побудови технічних засобів, що забезпечують реалізацію цієї взаємодії.

Сукупність таких правил, методів і засобів сполучення одного елементу системи з іншим, які забезпечують електричну, часову, за формою представлення інформації та інші види сумісності, назвали загальним терміном «інтерфейс» (від англ. Interface – сполучати, погоджувати). Інтерфейсом, у вузькому сенсі, називають пристрій сполучення. Це спрощене визначення. Кожному пристрою сполучення, який входить до складу комп'ютеризованої системи, властиві інтерфейсні функції. Інтерфейсні функції – це сукупність типових операцій, що виконуються при обміні даними в системі. Кожна інтерфейсна функція в пристрої сполучення здійснюється апаратно або програмно, дозволяючи йому приймати, передавати повідомлення або виконувати певну їх обробку.

У строгішому тлумаченні під інтерфейсом розуміють сукупність електричних, механічних, апаратних і програмних засобів, що дозволяють сполучати елементи системи і периферійні пристрої між собою, об'єднувати модулі в систему. Інтерфейс, як пристрій, що управляє потоком і форматами даних, включає також протокол, що описує процедуру взаємодії елементів при обміні даними.

Безліч інтерфейсів, залежно від їх призначення, можна розділити на три типи: 1) під комп'ютерним інтерфейсом мають на увазі такі інтерфейси, які вирішують задачу з'єднання центрального процесора ЕОМ (контролера) з іншими її функціональними блоками, а також підключення периферійних пристроїв, у тому числі і засобів сполучення з об'єктом управління; 2) до системно-модульних відносять інтерфейси, які вирішують завдання уніфікації сполучення модулів (функціональних блоків), призначених для роботи в системі (модулі, виконані з урахуванням подібного інтерфейсу, як правило, не розраховані на використання як автономні прилади, що працюють окремо, поза системою); 3) системно-приладові інтерфейси об'єднують в систему модулі, які можуть працюва-

ти автономно і для яких характерні великі функціональні можливості. Логіка їх функціонування складна і конструктивні вимоги до інтерфейсів цього типу, як правило, торкаються лише роз'єму.

Поява інтерфейсу радикально змінила принципи побудови складних систем збірання і обробки інформації, систем управління і послужило основою для створення складних високоавтоматизованих систем управління. Інтерфейс визначає технологію передачі даних (сигналів): немодульовану або модульовану. Дані в немодульованій (base band) системі передачі передаються у вигляді цифрових сигналів, що становлять дискретні електричні або світлові імпульси. Щоб уникнути загасання і спотворення сигналів, в немодульованих системах використовують повторювачі, які підсилюють сигнал. Модульовані або широкосмугові (broad band) системи передають дані у вигляді модульованого аналогового сигналу, що використовує деяку відносно невелику смугу частот. У цьому випадку по одному каналу, якщо смуга пропускання достатня, можуть одночасно передаватися декілька сигналів. У модульованих для відновлення сигналу використовують підсилювачі.

Апаратний інтерфейс визначає систему зв'язку, так звану магістраль, фізичне середовище передачі даних (інформаційну магістраль) або, іншими словами, засоби (систему ліній електричного зв'язку) зв'язку між підключеними до каналу пристроями. Конструктивно інтерфейс складається з набору провідників, роз'ємів і плати з електронними засобами (з інтегральними мікросхемами), що здійснюють управління. Плати, за допомогою яких проводиться обмін інформацією, називаються інтерфейсними картами. Функціонально лінії, створюючі магістраль, групуються в шину. До інтерфейсу апаратно відносять шини і порти – логічні схеми тієї або іншої розрядності, за допомогою яких підключаються периферійні пристрої.

Найбільш простим каналом передачі даних є двопровідні лінії зв'язку – два відрізки дроту, по яких передаються різного роду електричні сигнали. Якщо два паралельні дроти скручені разом, то середовище передачі даних називається скрученою парою (twisted pair). Двопровідна лінія, скручена пара можуть бути симетричними і несиметричними. У несиметричному ланцюзі передбачається, що один провідник сполучений з точкою нульового потенціалу або, іншими словами, має потенціал «землі». «Земля» (точніше схемна «земля») це загальна точка відліку в схемі, щодо якої вимірюються всі напруги і потенціали. У симетричному ланцю-

зі жоден провідник не використовується як «земля». Сигнали представляють різницю потенціалів (напруг) на двох дротах. Оскільки велика частина перешкод і паразитних дій є несиметричні сигнали, то симетричний ланцюг значно знижує перешкоди. Скручена пара зменшує небажані сигнали від сусідніх каналів (що є результатом їх електричної і магнітної взаємодії), фон змінного струму.

Широко поширеним засобом зв'язку між підключеними до каналу пристроями є коаксіальний кабель (coaxial cable) або екранована двопровідна лінія. Коаксіальний кабель складається з двох провідників. У центрі кабелю проходить суцільний мідний (алюмінієвий) провідник або багатожильний провід. Він поміщений в ізолюючий шар. Цей шар по периметру покриває другий провідник – металеве обплетення або металева фольга. Обплетення оберігає дріт від електромагнітних перешкод, тому її часто називають «екраном». Зовнішній шар такого кабелю утворює жорстка пластмасова оболонка, що забезпечує захист і електричну ізоляцію. Коаксіальні кабелі бувають тонкими (thinnet) –діаметром близько 0,5 см і товстими (thicknet), діаметром близько 1 см. Для підключення коаксіальних кабелів, для зрощення двох відрізків коаксіального кабелю використовуються BNC з'єднувачі (коннектори).

Кажучи про екранування відзначимо, що вита пара може бути неекранованою (UTP) і екранованою (STP). Кабель екранованої витої пари (STP) має мідне обплетення, яке забезпечує надійніший захист від перешкод, чим неекранована вита пара. Крім того, пари дротів STP обмотані фольгою. В результаті екранована вита пара добре захищає передавані дані від зовнішніх перешкод, і по ній дані можуть передаватися з вищою швидкістю.

У оптоволоконних кабелях цифрові дані розповсюджуються по оптичних волокнах у вигляді модульованих світлових імпульсів. Це відносно захищений спосіб передачі даних, оскільки в ньому не використовуються електричні сигнали. До оптоволоконного кабелю неможливо підключитися, щоб «перехопити» дані.

Оптичне волокно – тонкий, добре передавальний уздовж осі оптичне випромінювання діелектричний циліндр, званий «жилою» (core). Жила покрита шаром діелектрика, який називається оболонкою, з іншим, ніж у жили коефіцієнтом переломлення. Оптоволоконні кабелі призначені для передачі великих об'ємів даних на високих швидкостях (100 Мбит/с).

Як фізичне середовище передачі даних, засоби зв'язку між підключеними до каналу пристроями широко використовуються «шини» (Bus). Під шиною мається на увазі використовувана для виконання певної функції фізична група ліній передачі сигналів, що володіє функціональною спільністю. По шині до функціональних вузлів інформація може передаватися паралельно, наприклад, по восьми лініях, причому по кожній лінії передається один двійковий розряд інформації. Фізично шини реалізуються у вигляді паралельних провідних смужок друкарської плати або у вигляді «зв'язаних в джгут» дротів. Для передачі сигналів між пристроями часто використовують шлейфи («плоскі кабелі» з роз'ємами на кінцях).

Оскільки шини це частина інтерфейсу, то, з цієї точки зору, інтерфейс характеризується розмірністю «одиниці обміну» – числом сигнальних дротів, які відведені на одночасну (паралельну) передачу даних. Крім того, за допомогою інтерфейсу визначають, як здійснюється передача даних – одночасно по всіх лініях (паралельно) або по одній лінії послідовно порція інформації за порцією (біт за бітом). Інтерфейс визначає спосіб передачі сигналів: синхронний (з постійною тимчасовою прив'язкою у циклі збору інформації до синхронізуючих імпульсів) або асинхронний (без тимчасової прив'язки до певного тимчасового інтервалу). Інтерфейс визначає також наявність або відсутність органів, що забезпечують автономне управління об'єктом, можливість дистанційного вимірювання режимів його роботи.

Характеризуючи фізичні середовища передачі даних, треба пам'ятати про наступне. У моменти перемикання цифрових логічних схем з логічного стану «0» в «1», як вам відомо, має місце різке короткочасне зростання струму, споживаного від вторинного джерела електроживлення. Енергія, що відбирається від джерела живлення в ці моменти часу, витрачається на заряд паразитних місткостей. Подальший розряд паразитних місткостей супроводжується появою короткочасних імпульсних струмів по земляних шинах. Із-за кінцевої індуктивності провідників (шин) живлення і землі імпульсні струми викликають (пригадайте властивості індуктивності) появу імпульсних напруг як позитивної, так і негативної полярності, які прикладені між провідниками живлення і «землі». Амплітуда таких імпульсів, якщо не вживати спеціальним заходам, може досягати 2-х вольт і більше. Щоб уникнути збоїв в «передачі інформації» із-за

такого роду перешкод при з'єднанні всіх елементів і пристроїв САУ в єдиний комплекс дотримують наступні рекомендації.

1. Провідники живлення і «землі» виконують так, щоб вони володіли мінімальною індуктивністю. Для цього вони виконуються у вигляді «могутніх» провідників, наприклад» у вигляді мідних планок великого перетину.

2. Роз'єми, що служать для з'єднання магістральних ліній зв'язку, виконують так, щоб неоднорідність, що вноситься ними в лінію зв'язку була незначною, щоб імпульси, що «біжать» по магістралі, «не відчували» роз'єму.

3. У пристроях, виконаних у вигляді конструктивно закінчених блоків, реалізують, принаймні, два типи «землі» – корпусна і схемна. Корпусна «земля» служить для цілей безпеки і вона в обов'язковому порядку підключається до «заземлення» – мідному або алюмінієвому провіднику (з малим опором), прокладеному в приміщенні. За допомогою схемної «землі» відлічуються рівні потенціалів сигналів щодо корпусу. Вона виведена у вигляді окремої клеми, ізольованої від корпусу. Схемна «земля» не повинна бути з’єднана з корпусною «землею» усередині блоку. Недотримання цієї вимоги може приводити до серйозних збоїв у роботі обладнання.

4. Для однонаправленої передачі електричних сигналів доцільно використовувати лінію зв'язку, так званою «диференціальною парою». Обидва провідники в цьому випадку не з’єднані з «землею». З цієї причини перешкода, що наводиться ззовні, діє на обидва дроти приблизно однаково. Якщо використовується пристрій, який реагує на різницю сигналів (пригадайте диференціальний підсилювач), то лінія типу «диференціальна пара» характеризується підвищеною перешкодозахисною функцією що до синфазным перешкод.

Поняття інтерфейсу функціонально включає адаптер, драйвер і контролер.

Адаптер – ця збірна назва пристроїв, використовуваних для здійснення взаємодії із зовнішніми джерелами і приймачами даних. Він виконує функцію електричного узгодження пристроїв і, можливо, їх логічного сполучення (перекладу бітів інформації у форму, зрозумілу пристрою). Завдання адаптера полягає і в тому, щоб «переводити» біт інформації у відповідні електричні або оптичні імпульси, потік послідовних даних в потік паралельних даних, і навпаки. Адаптер повинен здійснювати «гальва-

нічну розв'язку» (електричне ізолювання) пристроїв (вона може бути реалізована на основі оптронів). У простому випадку адаптер виконує лише функцію електричного перетворення сигналів (наприклад, сигналів з амплітудою 9 В в сигнали з амплітудою 4 В. Як адаптер, що перетворює високі рівні напруги в низькі, можуть застосовуватися дільники напруги. Адаптером може служити підсилювач, аналоговий компаратор (трігер Шмідта). Останній може представляти простий перетворювач аналогового сигналу на вході в сигнал двійкової логіки на виході.

За допомогою адаптерів мікроконтролери сполучаються з пристроями, орієнтованими на послідовне введення – виведення інформації. У цьому випадку адаптери містять сдвигові регістри, а також спеціальні перетворювачі, що забезпечують узгодження рівнів (за напругою і потужністю). Ви можете зустріти адаптери зовнішнього інтерфейсу, адаптери послідовного інтерфейсу асинхронної передачі даних (ACIA – Asynchronous Communication Interface Adapter). Адаптери сполучення можуть програмуватися.

До складу інтерфейсів входять драйвери. Вони бувають апаратними і програмними.

Апаратними драйверами називають спеціальні схеми управління, які слугують для здійснення специфічних функцій електричного узгодження пристроїв, оптимального управління силовими приладами. Вам відомо, що як керовані прилади в обладнанні можуть застосовуватися біполярні транзистори, могутні польові прилади (МОП – транзистори) і прилади трігерного типу (тиристори, симисторы). Вимоги, що пред'являються до їх оптимального управління, різні. Драйвер біполярного транзистора повинен управляти струмом бази при включенні і забезпечувати розсмоктування неосновних носіїв у базі на етапі замикання. МОП транзистор управляється напругою, проте при цьому на початку інтервалу включення і замикання драйвер повинен забезпечувати великі імпульсні струми заряду і розряду місткостей приладу. Прилади типу трігера, вимагають для своєї роботи формування короткого імпульсу струму на початку інтервалу часу включення. Для підключення до шин даних використовують пристрої, що виконують функції підсилювача потужності. Їх називають шинними-формувачами (Bus driver).

Програмним драйвером (software driver) називають програму, що виконує операції введення-виведення. З її допомогою з метою передачі

до певного зовнішнього пристрою послідовність команд перетворюється до необхідного формату даних.

Контролером називають спеціальний пристрій управління, призначений для сполучення одного або декількох пристроїв ввода – вивода з магістраллю при обміні інформацією. Призначення такого пристрою управління – контролера – організація взаємодії пристроїв в системі. Команди контролера «указують» на адресу пристрою, на те, який пристрій повинен передавати дані, а який – приймати. Контролер встановлює метод доступу до магістралі, тривалість кожного біта, відповідає за кодування даних, за синхронізацію бітів. Контролер повинен гарантувати, щоб передана «одиниця» була сприйнята як одиниця, а не як нуль.

Програмовані контролери, крім звичайних функцій управління, виконують також певні логічні і математичні операції, необхідні для аналізу прийнятих даних, для їх обробки, для ухвалення рішень, що визначають «поведінку» як окремих пристроїв, так і системи в цілому. Подібні контролери є, як правило, адаптивні (пристрої, що пристосовуються), які будують з використанням великих інтегральних схем.

2.4. Взаємодія функціональних вузлів обладнання у реальному часі

Технологічні процеси друку – це сукупність способів і засобів проведення конкретних операцій для отримання друкарської продукції, які здійснюються механізмами і рухомими за певними траєкторіями робочими приладами. Характеризуючи ці процеси з погляду організації процесу управління, відзначимо наступне. Щоб технологічний процес став керованим в обладнанні для виконання технологічного процесу повинно бути забезпечено «узгодження» в часі і в просторі всіх взаємодій робочих органів. Всі технологічні процеси в поліграфії, всі послідовності виконання певних законів руху робочими органами (операцій) повинні бути строго синхронізовані між собою в часі і суміщені в просторі за положенням робочого приладу. Це обумовлює необхідність того, що в системі управління технологією друку повинні бути «запрограмовані» в заданій послідовності всі види переміщень і інших дій, необхідних для виготовлення друкарської продукції.

За характером рухи робочих приладів обладнання можна розділити на дві групи. До першої групи відноситься обладнання, в якому рухи ро-

бочих органів незмінно повторюються. До другої групи – обладнання з різноманітним поєднанням рухів робочих органів. Робота такого обладнання носить гнучкий характер, оскільки поєднання вступаючих у роботу робочих органів і їх стан може залежати від різних чинників.

Часто в поліграфічному обладнанні, особливо при середньо- і великосерійному виробництві, при реалізації різних технологічних процесів має місце повторюваність процесів або їх періодичність. Говорять, що при роботі друкарського обладнання існують цикли – сукупності взаємозв'язаних рухів, елементарних операцій, процесів, робіт, що виконуються у певній послідовності, створюючих певну систему (закінчене коло) і що забезпечують повне виконання обладнанням робочих функцій. Такі елементарні операції називаються етапами циклу або тактами роботи системи. Протягом етапу циклу в обладнанні не відбувається змін в стані дій, що управляють. Етапи циклу характеризуються інформацією про цикл і такими параметрами, як швидкість, ступінь переміщення робочого приладу і т. д.

Відомо, що в друкарській машині наявні робочі і кінематичні цикли. Під час робочого циклу виконується деяка сукупність операцій, після закінчення якої їх послідовність повторюється (наприклад, при русі циліндрів, штоків, паперу, а також енергетичних і силових потоків). На листовій друкарській машині після виконання одного робочого циклу виконується друк листа. Час, після закінчення якого всі виконавчі і допоміжні механізми займуть своє початкове положення, називають кінематичним циклом. Кінематичний цикл в технологічному обладнанні друку звичайно формується одним або декількома оборотами приводного валу основного виконавчого механізму. Початок відліку кінематичного циклу звичайно визначається початковим положенням одного з виконавчих механізмів, що виконує основну, найбільш енергоємну операцію в конкретному технологічному процесі.

Цикли можуть бути також послідовними і суміщеними. У послідовному циклі механізми машини працюють у режимі одиночних ходів (під час руху одного механізму всі інші «простоюють») і всі рухи робочих органів виконуються по черзі за спеціальними командами управління. При цьому послідовність роботи виконавчих механізмів точно синхронізована як в часі, так і з погляду рухів окремих виконавчих механізмів у просторі. Суміщеному циклу властиво одночасний рух декількох механізмів. У цьому випадку машина працює в режимі безперервних ходів.

Програма роботи обладнання у вигляді етапів циклу, режими його роботи, тобто перехід від виконаного етапу циклу до наступного, залежать від положення робочого органу. Це положення контролюється датчиками. В якості останніх широко застосовуються різні «путні» перемикачі, у тому числі і безконтактні, мікроперемикачі. Необхідна послідовність переміщень робочих ланок, їх взаємозв'язок і узгодженість дій (рухів) виконавчих механізмів технологічного комплексу друку визначається циклограмою – певною програмою узгодженості рухів виконавчих механізмів в часі і за положенням у просторі робочих органів. Функціонування машин-автоматів різного призначення у вигляді сукупності операцій робочого процесу на циклограмі виходить узгодженим через те, що воно «прив'язане» до певного часу.

Циклограма створюється на етапі розробки поліграфічного обладнання. Технологи при цьому уточнюють цілі і завдання, які мають вирішувати обладнання, детально планують, детально і, за можливістю, математично чітко уточнюють всі логічні зв'язки між причинами і наслідками. На циклограмі знаходить віддзеркалення те, як і в якій послідовності виконується сукупність багатьох операцій технологічного процесу, тобто послідовність і взаємозв'язок рухів робочих органів механізмів, а також моменти початку і тривалість дій, що повторюються в кожному циклі роботи. Для побудови циклограми функціонування поліграфічного обладнання необхідно мати кінематичні схеми механізмів в дійсних розмірах для всіх циклових механізмів, функціональні залежності переміщення ланок від кута повороту та ін. На циклограмі повинні також знайти віддзеркалення всі команди, що управляють, моменти початку їх подач.

Циклограми можуть зображатися двома способами: у полярних координатах (кругові циклограми) і в прямокутних координатах (лінійні циклограми). Кругові циклограми будуються у вигляді концентричних кругів, число яких відповідає числу рухів виконавчих механізмів. Складові циклу виражаються у кутках повороту головного валу. Такий спосіб зображення зручний, коли виконавчий механізм здійснює обертальний рух і цикл є суміщеним. При зображенні циклограми концентричними колами зовнішнє коло розмічають в межах від 0 до 360^0. Вона відображає один оберт розподільного валу, тобто того валу, що є ведучим для ряду механізмів і що здійснює один оберт за цикл. Кільцеві частини кола, ув'язнені між окремими колами, відповідають робочим органам, а виділені на них сегменти – фазовим кутам.

Циклограми в прямокутних координатах досконаліші, оскільки на них можна суміщати різнохарактерні рухи типу приводу, указувати послідовність команд, періоди включення або блокування органів управління. На циклограмі уздовж осі часу, відображаються, строго «прив'язані» до певного часу початку (звичайно до початку циклу) дії, що зводять технологічне завдання до узгодженого в часі і просторі руху. На ній також відображається стан органів управління у функції часу. Використовуючи горизонтальні рядки, кількість яких рівна числу командних (кнопок, тумблерів і т. д.), виконавчих (електродвигунів, електромагнітів і т. д.) і контролюючих (датчиків тиску, температури, вимикачів, перемикачів і т. д.) елементів, на циклограму наносяться сигнали, що поступають при роботі механізму з початку циклу від вказаних елементів у порядку їх появи. Наявність сигналу відображається на циклограмі в логічному сенсі (високий рівень за наявності сигналу і нульовий рівень в його відсутність).

На лінійній циклограмі руху, послідовність команд відображаються приблизно у такому вигляді: «якщо до часу t_1 натиснута кнопка А і спрацювало реле В, то за час Дt відбувається кутове переміщення валу на один оберт (на кут 2 р), а шток К переміщається на величину 2 см». Послідовно аналізуючи події і стани, тобто те, що відбулося і чому, на циклограму наносяться всі необхідні дії і сигнали управління, які при цьому повинні з'явитися. Циклові системи управління є найбільш простими, дешевими і надійними. Проте вони володіють малими технологічними можливостями. Оскільки цикли повторюються один за одним, незалежно від змісту виконуваного завдання, то циклічні автоматичні системи управління, досягають заданої мети на основі «жорсткої» координації всіх операцій і синхронізації всіх тимчасових співвідношень. Циклові системи управління не завжди годяться при управлінні складними технологічними процесами, коли потрібно враховувати «готовність» певних частин обладнання. З цієї причини в даний час при організації управління, разом з циклічними, використовують також ациклічні автоматичні системи, які досягають заданої мети відповідно до інформаційних процесів, що відбуваються в обладнанні. При такому способі організації дій пристрій, що управляє, сприймає зовнішні події (так звані переривання), «прив'язує» «запити на виконання управління» до тимчасової сітки, обробляє їх, і формує необхідну дію. Пристрій, що при цьому управляє, може, реагуючи на зовнішні події, тимчасово відкласти виконання поточ-

ного завдання і виконати іншу, важливішу для даного моменту часу програму, після чого знов повернутися до перерваного завдання.

В обладнанні з ациклічним управлінням може виконуватися декілька функцій управління з розділенням їх в часі. Використання «квазіпаралельних» процесів управління у реальному часі» дозволяє оптимізувати режими роботи керованих об'єктів. Поняття «управління у реальному часі» означає, що пристрій, що управляє обладнанням, здатний одержати інформацію про стан керованого об’єкта, виконати необхідні розрахунки і сформувати дії управління протягом інтервалу часу, після закінчення якого ці дії викличуть бажану зміну поведінки об’єкта.

При «управлінні у реальному часі» кожна дія повинна бути виконано до певного часу, тому функції управління реалізуються як процеси на тимчасовій сітці з елементарним інтервалом часу. Для ефективного управління у реальному часі необхідно сформувати тимчасову сітку, визначивши при цьому величину елементарного інтервалу часу. В цьому випадку кожен імпульс позначає початок нового інтервалу часу. Завдання формування тимчасової сітки із заданим елементарним інтервалом часу, відліку рівних інтервалів заданої тривалості, формування сигналів управління об'єктами, повтору алгоритму після закінчення певного тимчасового інтервалу, вирішує спеціальний пристрій, контролер управління. Рішення подібних задач називають формуванням міток реального часу. Оскільки процес формування сигналів управління використовує інтервали системного часу, то тимчасова сітка є безперервною і рівномірною.

У електронному пристрої управління для вирішення завдання формування тимчасової сітки передбачений спеціальний апаратний модуль – таймер. Таймер – це пристрій відліку часу. По суті це свого роду годинник, з якого можна «прочитувати час», який можна використовувати як «будильник», що в певний час видає «сигнал – дзвінок».

Таймер може виступати як секундомір при вимірюванні інтервалів часу між якимись подіями. Він також формує «системний час» у вигляді періодичної послідовності прямокутних імпульсів, створює сигнали з програмованою затримкою, а також імпульсні сигнали з програмованою частотою і програмованим коефіцієнтом заповнення. Часто таймер використовується для затримки процесів включення або виключення. У таймері першого типу перехід з 0 в 1 здійснюється із затримкою на наперед заданий час. Таймер, що здійснює затримку виключення, забезпечує мит-

тєвий перехід з 0 в 1, але перехід з 1 в 0 із затримкою. Затримка звичайно використовується для усунення програмним шляхом ефекту брязкоту контактів.

Таймери будь-якого типу мають декілька параметрів, які повинні бути наперед встановлені користувачем. Найважливішим параметром є базова одиниця часу (у цих одиницях вимірюються всі тимчасові інтервали). Іншим параметром (preset), що задається, є величина затримки. Типові таймери можуть відлічувати інтервали часу до 32767 базових одиниць, що відповідає 16 двійковим розрядам. Істотною особливістю багатьох програм, що забезпечують обладнання, є підрахунок чого-небудь. Наприклад, можна рахувати число надрукованих листів, можна зафіксувати скільки разів відбудеться подія, що полягає в переміщенні зубів шестерні валу мимо датчика. Для цієї мети використовуються лічильники, що підраховують кількість імпульсів і формуючі двійковий еквівалент кількості імпульсів. Завдання, що вирішуються таймерами і лічильниками певною мірою аналогічні, тому звичайно створюють один цифровий пристрій, який називають таймер-лічильник.

Контрольні запитання

1. Охарактеризуйте основні тенденції розвитку мікропроцесорних систем управління.
2. Що таке дообробка локальних даних і багатопроцесорні системи.
3. Назвіть основні апаратні і програмні реалізації функцій управління.
4. Охарактеризуйте поняття: квазіпаралельні процеси управління.
5. У чому полягають особливості організації і взаємодії квазіпаралельних процесів управління?
6. Наведіть відомості про програмне забезпечення, як про процедуру управління апаратурою.
7. Поняття технічної діагностики роботи технічних пристроїв.
8. Концепції сучасних систем управління і тенденції розвитку мікропроцесорних систем управління.
9. Структурна схема інформаційно-керувальної системи, що представляє собою замкнутий контур.
10. Охарактеризуйте наступні поняття: системно-модульні інтерфейси.

11. Стисло охарактеризуйте наступні поняття: системно-приладові інтерфейси.

12. Стисло охарактеризуйте наступні поняття: апаратні й програмні драйвери.

13. Охарактеризуйте основні програмувальні контролери.

14. Поняття системи автоматичного управління (САУ).

15. Концепції сучасних систем управління і тенденції розвитку мікропроцесорних систем управління.

16. Концепція функціонально-топологічних і програмних модулів.

17. Поняття циклів функціонування поліграфічного обладнання.

18. Особливості побудови циклограм функціонування поліграфічного обладнання.

19. Поясніть поняття: функціонально-модульна структура програмного забезпечення.

20. Поясніть поняття розподілених систем управління обладнанням, їх особливості і переваги.

21. Наведіть і поясніть узагальнену схему взаємодії пристрою управління об'єктами нижнього рівня з периферійними пристроями.

22. Наведіть і поясніть узагальнену схему вбудованої в локальний об'єкт системи, що інформаційно-управляє.

23 Наведіть і поясніть структурну схему інформаційно-управлінської системи, що представляє замкнутий контур.

24. Пояснить призначення (і назву) комп'ютера на другому рівні у сучасного комп'ютеризованого обладнання.

25. Які завдання можуть розв'язуватися комп'ютерами верхнього рівня в реальному масштабі часу?

26. Охарактеризуйте поняття «інтерфейс». На які типи можна розділити залежно від їх призначення?

27. Яки рекомендації дотримують при з'єднанні всіх елементів і пристроїв САУ в єдиний комплекс щоб уникнути збоїв в «передачі інформації» із-за перешкод?

28. Стисло охарактеризуйте наступне поняття: адаптер. Для виконання яких функцій він предназначений?

Тема 3. Архітектура цифрового пристрою керування й виконання мікроконтролера AVR

3.1. Початкові відомості про виконання пристроїв управління

Поліграфічне обладнання не може працювати саме по собі, а потребує управління. Тому в сучасному обладнанні для забезпечення його безпечного і економічного функціонування, для автоматизації процесів управління технологічними процесами широко використовують універсальні управляючі обчислювальні машини, які виконують попередньо встановлені операції над вхідними даними, щоб утворити необхідні вихідні дані. Необхідність виконання складних функцій управління призвела до створення мікроконтролерів (МК) – програмно-керованих пристроїв, виконаних у вигляді єдиної мікросхеми на одному кристалі. Мікроконтролери призначені для вирішення завдань управління, але вони також виконують обробку аналогових і цифрових сигналів, вирішують різні завдання логічного аналізу інформації. Виконані на одному напівпровідниковому кристалі, що по суті є системою, що управляє, мікроконтролерні пристрої управління включають всі основні апаратні засоби обробки даних (арифметико-логічні блоки, регістри, зв'язуючі їх інформаційні магістралі, магістралі передачі всіх сигналів, схеми управління). За рахунок «програмування» своєї роботи, логічній і функціональній гнучкості МК відносно просто «пристосовуються» до рішення конкретних задач в конкретних умовах.

У МК, які об'єднують сукупності апаратурних і програмних засобів, доводиться оперувати не тільки апаратурними засобами, функціональною структурою і особливостями фізичної реалізації вузлів системи, але і поведінкою системи при організації обчислювального процесу в ній, інформаційними потоками даних і управлінської інформації. При цьому якнайкращі характеристики системи управління у конкретних умовах можна одержати шляхом компромісу між апаратурними і програмними засобами. Збалансовані вимоги до апаратурних і програмних засобів дають можливість оптимізувати обчислювальний і керований процес.

Мікроконтролер є складним програмно-керованим цифровим процесором з високим ступенем інтеграції електронних елементів. Набуваючи МК, споживач сприймає його не на рівні окремих транзисторів, а як

щось цілісне що має зовнішні споживчі властивості, завдяки використанню яких можна вирішити завдання своєї конкретної системи управління. Розробляючи програмне забезпечення для МК, користувач не повинен розбиратися в його елементах схемотехніки так детально, як розробник мікроконтролера. Вирішити проблему вибору мікроконтролера можна на основі уявлення його своєрідною моделлю, заснованою на понятті «архітектура», за допомогою якої споживач сприймає МК як щось цілісне, таке, що має зовнішні споживчі властивості, закладені в його внутрішній організації. Ця модель характеризує внутрішню організацію обчислювально-керувальної системи, описує методологію об'єднання сукупності апаратних, програмних засобів в МК з позицій властивостей, що надаються в розпорядження розробників систем управління і програмістів-користувачів. Знання архітектури і особливостей різних архітектурних рішень виконання МК дає можливість користувачам ефективно розпоряджатися всіма наданими їм ресурсами, дозволяє раціонально використовувати ресурси системи і проектувати ефективні програми, підвищуючи тим самим ефективність обробки даних.

Поняття «архітектура», з погляду організації обчислювальної системи, семантичного зв'язку між можливостями апаратних засобів МК і їх програмного забезпечення, можна розкрити з декількох сторін. З одного боку, архітектура МК, яка характеризує його логічну організацію, може бути представлена як безліч взаємозв'язаних компонентів, які включають, на перший погляд, елементи різної природи і що підтримують його злагоджене функціонування у формі єдиного архітектурного ансамблю, що дозволяє вести ефективну обробку даних: програмне забезпечення (software), апаратне забезпечення (hardware), алгоритмічне забезпечення (brainware), спеціальне забезпечення (firmware). З іншого боку, архітектура може бути визначена як абстрактне багаторівневе представлення фізичної системи з погляду програміста із закріпленням функцій за кожним рівнем і встановленням інтерфейсу між різними рівнями. Нарешті, архітектура МК – функціональні можливості апаратурних електронних засобів, використовуваних для представлення даних, режимів роботи, машинних операцій, опису алгоритмів і процесів обчислень. З цієї точки зору, архітектура характеризує такі параметри, як швидкодія, об'єм пам'яті, набір периферійних пристроїв, система використовуваних команд, формати даних, швидкість виконання операцій і ін. Одним із напрямів розвитку комп'ютеризованих засобів, використовуваних для цілей авто-

матизації управління, є вдосконалення архітектури, що дозволяє на новому витку складності алгоритмів управління забезпечити адекватну продуктивність обчислювальних засобів. В даний час в комп'ютеризованому обладнанні пристрої управління виконуються, головним чином, за RISC – архітектурою. Характерними особливостями архітектури RISC є: 1) обмежена кількість команд. Суть цієї особливості полягає у виділенні найбільш споживаних операцій і створенні засобів їх швидкої реалізації; 2) Всі команди при архітектурі RISC реалізуються схематичним шляхом без інтерпретації у формі мікрокоманд. Процес декодування команд є простим. Як правило, кожна команда незалежно від її типу виконується за один машинний цикл; 3) всі команди повинні мати однакову довжину і використовувати мінімум адресних форматів, що різко спрощує логіку управління процесором, у тому числі і при програмуванні на мовах високого рівня; 4) використання регістрової структури. У поєднанні з швидкодіючою арифметикою RISC-операції типу «регістр – регістр» стають могутнім засобом підвищення продуктивності процесора; 5) RISC-контролери працюють за Гарвардською моделлю організації пам'яті, що має на увазі розділення пам'яті для програм і даних. Це дозволяє використовувати конвеєрну обробку. Завдяки цьому досягається можливість виконання команд протягом одного тактового циклу; 6) простота архітектури RISC-процесора забезпечує його компактність, практичну відсутність проблем з охолоджуванням кристала.

Світова промисловість випускає величезну номенклатуру мікроконтролерів, як спеціалізованих, так і універсальних, таких, що не мають конкретної спеціалізації. Сучасні МК утворюють цілі сімейства, вони настільки різноманітні і функціонально насичені, що виникає проблема вибору. Ви можете зустріти вироби, що випускаються такими відомими компаніями, як Motorola, Texas Instruments, NEC, Mitsubishi, Hitachi, Toshiba, Philips, Siemens. На нашому ринку добре влаштувалися недорогі МК PIC від компанії Microchip. Популярною продукцією є мікроконтролери AVR що випускаються американською компанією Atmel Corporation. Використавши комбінацію енергозберігаючої технології виготовлення комплементарних МОП-транзисторів, системно-програмованої пам'яті типу Flash – EPROM і прогресивної RISC-архітектури, компанія Atmel розробила дуже ефективний спосіб рішення всіх задач «внутрішньокристального» управління і добилася чудових успіхів відносно ціни і якості. Серед тих, що існують в світі МК, сімейство AVR, мабуть, має найбільш

високі темпи приросту виробництва. В даний час випущено декілька десятків типів 8-розрядних мікроконтролерів, які підрозділяються на декілька сімейств. Споживач має можливість зробити оптимальною вибір залежно від типу вирішуваної задачі. Вичерпну інформацію про них можна знайти в довідниках за мікросхемами AVR або в Інтернет.

Всі моделі МК AVR можна об'єднати в декілька груп: 1. Застаріле сімейство «Classic AVR» (AT90Sxxx) в даний час практично вже не використовується. 2. Сімейство МК «Tiny AVR» (ATtinyxxx) – це низькошвидкісні МК мінімальної конфігурації і, переважно, невеликих габаритів. Вони з'явилися останніми, мають невелику вартість, невелику кількість виводів, здатні працювати від джерела зниженої напруги і при цьому володіють такими функціонально важливими периферійними вузлами, як, наприклад, АЦП. Вони призначені для простих недорогих і мініатюрних електронних пристроїв управління. 3. МК «Mega AVR» (ATmegaxxx) є модель, що має розвинену архітектуру і орієнтовану на високопродуктивну роботу зі складними завданнями, що вимагають великих ресурсів пам'яті. Вони мають найбільші об'єми пам'яті і якнайповніший набір периферійних вузлів. Компанія Atmel випускає ще декілька видів спеціалізованих мікроконтролерів, які відносяться до серії AVR, наприклад, AVR для Smart Cards (AT90SCC).

МК будуються на основі модульної організації, при якій всі МК сімейства містять в собі базовий функціональний блок (процесорне ядро), який однаковий для всіх МК сімейств. Всі МК AVR вдало втілюють сучасні тенденції архітектури RISC-МК, будуються за єдиним принципом. Вони мають єдину систему команд, яка для різних моделей мікроконтролерів може відрізнятися наявністю або відсутністю декількох не принципових команд. Тому доцільно почати вивчення мікроконтролерів AVR з «загального» погляду на їх будову і функціонування.

3.2. Архітектура типового мікроконтролера AVR

3.2.1. Загальні відомості

Мікроконтролери AVR становять апаратно-програмну систему обробки інформації, яка реалізована у вигляді однієї великої ІС. Тому для характеристики архітектури МК, перш за все, необхідно представляти логічну структуру МК, тобто конфігурацію функціональних вузлів, що

становлять МК, і зв'язків між ними. МК в межах одного напівпровідникового кристала об'єднує основні функціональні блоки (модулі) системи управління (рис. 3.1): центральний процесор, пристрій що постійно запам'ятовує (ПЗП), пристрій (ОЗП) що оперативно запам'ятовує, периферійні пристрої для введення і виведення інформації (периферійні вузли або, коротко, периферію). Всі окремі модулі МК (процесор, ПЗП, ОЗП, периферійні вузли) є автономними, виконуються геометрично незалежними з під'єднуванням до внутрішніх магістралей адреси і даних. При цьому використовується принцип, який характеризується відсутністю на виводах ІС МК ліній магістралей адреси, управління і даних.

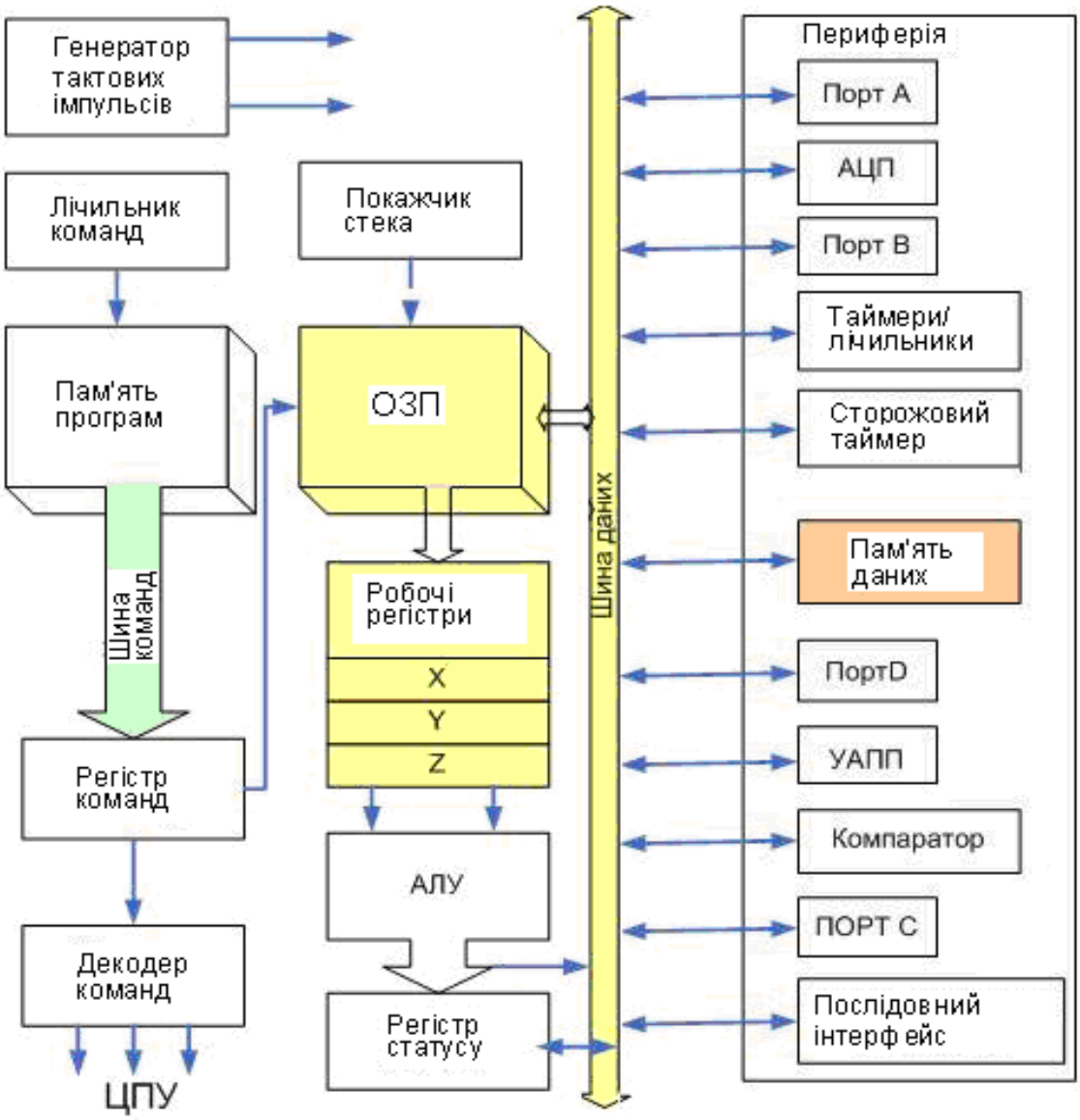


Рис. 3.1. **Схема, що відображає логічну організацію МК і його основні апаратні засоби**

Щоб одержати детальніші уявлення про архітектуру МК треба дати відповіді, принаймні, на такні питання: 1. Скільки застосовується регістрів для реалізації архітектурних вимог і як вони використовуються для обчислень і зберігання проміжних даних? 2. Як переміщаються команди і дані по інформаційних магістралях МК між регістрами, АЛУ і елементами пам'яті в процесі виконання кожної команди? 3. Яка система команд визначає набір можливих дій над операндами? 4. Які використовуються способи адресації операндів в просторі пам'яті?

3.2.2. Логічна організація процесора

Центральне місце в структурі МК, як великої інтегральної схеми з програмованою логікою роботи, займає процесор. У МК AVR основним обчислювальним вузлом або «ядром» мікроконтролера, є центральний процесорний пристрій (CPU – Central Processing Unit). CPU складається з арифметично-логічного пристрою (АЛП), з приєднаним до нього набором регістрів для обчислення і зберігання проміжних результатів, регістра стану (статусу) процесора і ряду пристроїв управління, забезпечуючих виконання операцій і «керуючих» роботою АЛП. За допомогою АЛП за командами здійснюються арифметичні, логічні і порозрядні операції над операндами. АЛП може виконувати операції складання, віднімання, пересилки, зсуву (зрушення), а також логічні операції «І», «АБО» і «Виключаюче АБО». Всі МК AVR мають розширені можливості АЛП набір регістрів (блок, банк), що складається з 32 «робочих регістрів загального призначення» (РЗП) – («General Purpose Working Register»). Банк регістрів називається «регістровим файлом» (Register File). РЗП служать внутрішньою пам'яттю МК і використовуються для часового зберігання даних. Все РЗП МК AVR восьмирозрядні, тобто більшість операцій МК можуть проводити з восьмирозрядними двійковими числами. РЗП зв'язані між собою і АЛП за допомогою внутрішньої шини даних – групою ліній передачі інформації, об'єднаної загальною функціональною ознакою. Розрядність внутрішньої шини даних, тобто кількість передаваних одночасно (паралельно) бітів числа, відповідає восьми (розрядності РЗП). Щоб регістри можна було використовувати в програмі, кожен має своє власне ім'я: R0, R1, R2 – R31. Шість регістрів (R26 – R31) здатні об'єднуватися в регістрові пари. Така пара в деяких операціях виступає як самостійний

16-розрядний регістр. Об'єднані попарно регістри можна використовувати для обробки шістнадцятирозрядних чисел. При цьому не втрачається можливість читання кожного регістра пари окремо. Регістрові пари мають свою назву: X (R26, R27), Y (R28, R29), Z (R30, R31).

Усі команди перетворення даних (складання, віднімання і т. д.) МК AVR побудовані таким чином, що обов'язково використовують РЗП. Кожна команда як операнди використовує або вміст двох різних РЗП, або вміст РЗП і константу. Результат обчислень також поміщається в РЗП. Наприклад, команда ADD R0,R1 проводить складання вмісту регістрів R0 і R1. Сума поміщається в R0. Усі регістрові команди звертаються до регістрів протягом одного такту роботи МК. За один тактовий цикл з файлу регістрів вибираються два операнди, в АЛП над ними виконується операція, і результат знов повертається у файл регістрів. РЗП використовуються також в командах переміщення даних. Переміщати дані можна з одного РЗП в інший, з РЗП в елемент пам'яті і у зворотному напрямі. Переміщення даних можливо також між РЗП і регістрами введення – виведення. Деякі команди мають обмеження з використання РЗП. Наприклад, всі команди обміну інформацією з регістрами введення – виведенняня можуть використовувати РОП з номерами R0 – R15. Логічні і арифметичні операції над константами (SBCI, SUBI, CPI, ANDI), операція між константою і вмістом регістра ORI і команда безпосереднього завантаження константи LDI використовують тільки другу половину регістрів регістрового файлу (R16 – R31).

АЛП безпосередньо пов'язаний з регістром SREG статусу (стану) або, по-іншому, регістром ознак. У відповідних розрядах цього регістра фіксуються особливості виконання в АЛП кожної операції: нульовий результат, переповнювання, перенесення із старшого розряду та ін. Для позначення результату виконання операції використовується 8 різних біт умов. Всі вони можуть бути перевірені («лічені») за допомогою команд, що визначають подальший хід виконання програми.

3.2.3. Пристрій пам’яті для зберігання програм МК і підсистема забезпечення виконання програм

У МК є «пам'ять», використовувана для зберігання представленої в двійковому вигляді оброблюваної інформації, запам'ятовування резуль-

татів виконання операцій або окремих команд, збереження іншої корисної інформації про хід процесу обчислень.

Пристрій постійного запам'ятовування (ППЗ), використовуваний в МК AVR, відповідно до концепції Гарвардської архітектури (звернення до команд здійснюється незалежно від доступу до даних), складається з двох компонентів і включає дві роздільні області – пам'ять програм і пам'ять даних. Пам'ять програм МК AVR є репрограмувальною. Інформація, що зберігається в ній, може стиратися кілька разів і вона допускає перепрограмування. Пам'ять програм МК AVR, призначена для зберігання програми управління, реалізована на основі програмованої і електрично стираної флеш-технології (Flash – EPROM). У флеш-пам'яті як елемент пам'яті (ЕП) використовується елемент, що запам'ятовує, на МОП-транзисторі з додатковим «плаваючим затвором», який при програмуванні заряджується негативними зарядами. Залежно від наявності або відсутності заряду на плаваючому затворі комірка пристрою, що запам'ятовує, може інтерпретуватися як логічний «0» або логічна «1».

У МК AVR елементи пам'яті організовуються в 16-розрядні «слова», тому більшість команд МК AVR мають формат 16-розрядного (двобайтового) слова. За кожною адресою пам'яті розміщується одна команда (двійкове число або – код). Звичайно команда МК містить інформацію про відповідні їй дії, що управляють, – код операції (КОП) і адреси операндів, над якими потрібно виконати певні дії. Об'єм пам'яті для різних МК різний і складає від 1 до 64 кбайт. У могутніх МК для адресації використовується 12 розрядів і в пам'яті може зберігається 4 Кб командних 16-розрядних слів. Інші МК використовують 10 або 9 розрядів для адресації. Адресний простір пам'яті програм МК показаний на рис. 3.2. Адресний простір починається з комірки, що зберігає команду з 16 розрядів, яка має нульову адресу $0000. З цієї адреси починається виконання програми після системного скидання. Весь адресний простір включає комірки флеш-пам'яті з адресами від $0000 до $1FFF. Адреса останнього елементу пам'яті буде різною для різних типів МК, тому йому часто приписують умовну адресу F_END.

Для різних МК пам'ять програм має різний об'єм. Об'єм програмної області пам'яті можна дізнатися з довідкових даних, наведених в графі «Flash». Адреса останнього елементу пам'яті завжди на одиницю менше об'єму цієї пам'яті. Так, якщо об'єм програмної пам'яті рівний 64 кілобайтам, тобто 65536 байтам, то в шістнадцятирічному уявленні одержимо

$10000. Враховуючи, що адресація починається з нульової адреси, то для адресації такої кількості комірок ми повинні використовувати адреси з 0 по 65535. Або в шістнадцятирічному вигляді від $0000 до $FFFF. Адресний простір ділиться на дві секції – секцію користувальницької програми (Application Flash Section) і завантажувальну секцію (Boot Flash Section). Програмна пам'ять, що іменується у фірмовій документації на МК, як Flash, допускає стирання записаної туди інформації і повторний запис (до 1000 циклів запису – стирання). Програмування пам'яті при цьому може вестися, як в паралельному режимі, так і в послідовному.

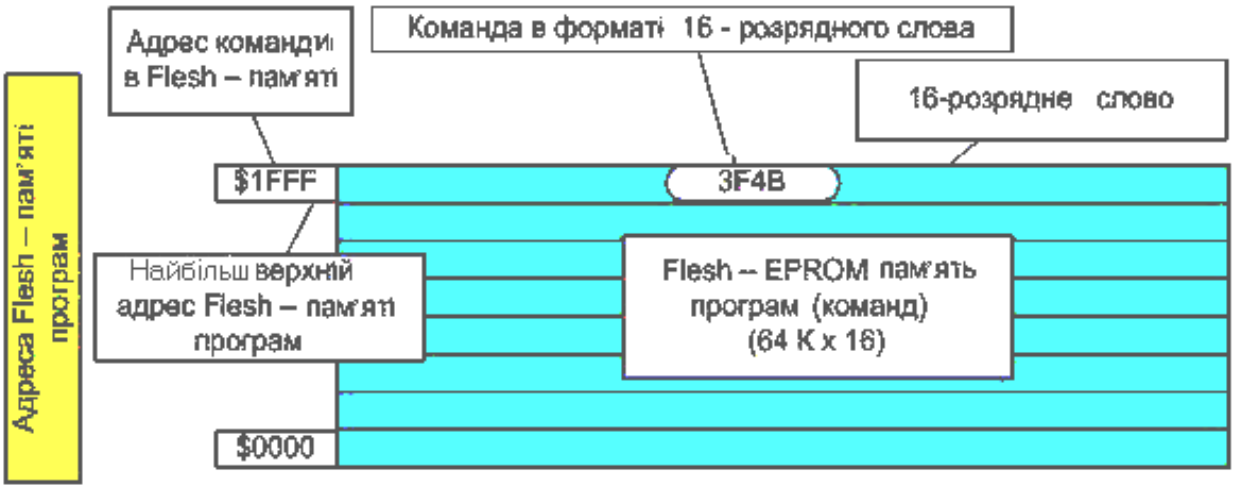


Рис. 3.2. **Графічне зображення адресного простору програмної пам'яті мікроконтролера**

Програма МК – це послідовність операцій, яку він повинен виконувати. Кожна операція, яку здатний виконати МК, кодується 16-розрядним числом, так званим кодом операції. Програма записується в пам'ять у вигляді послідовності кодів. Тому програма роботи МК представлена в пам'яті послідовністю кодів операцій, що відображають кожну команду, закодовану у вигляді двобайтового слова. Для виконання програми в МК послідовно читаються коди операцій і проводиться їх виконання.

Для того, щоб можна було послідовно читати команди з пам'яті програм, в МК є лічильник команд або програмний лічильник (PC – Program Counter) – це спеціалізований внутрішній (не доступний для програміста) регістр, в якому зберігається адреса поточної виконуваної команди. Завдання лічильника полягає в тому, щоб утворювати бінарне слово, що

представляє в пам'яті програм адресу команди. Звичайно такий лічильник має спеціальний інкрементний пристрій, призначення якого – додавати одиницю до поточної адреси і таким чином одержувати наступну адресу. Розмір лічильника команд складає для різних МК від 9 до 13. Кількість розрядів лічильника команд залежить від розміру програмної пам'яті МК, що адресується.

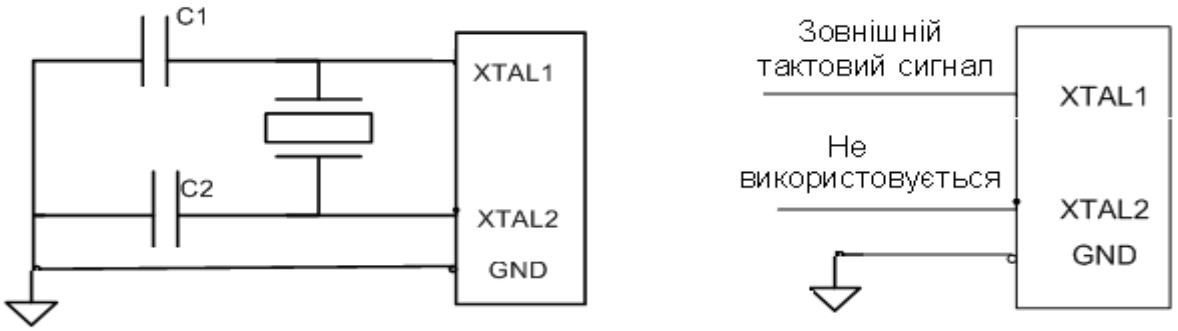
Звернення до пам'яті програм, коли програма рішення конкретної задачі введена виготівником або користувачем в Flash – EPROM в МК, у загальних рисах, здійснюється таким чином. Робота МК завжди починається з процедури «початкового скидання», при якій всі регістри, у тому числі і лічильник команд, встановлюється в початковий стан. По закінченню процесу початкового скидання починається виконання програми. Після виконання першої команди значення лічильника команд (PC – Program Counter) збільшується на одиницю і починається процес читання і виконання наступної команди, записаної за першою адресою. Аналогічно продовжується робота МК і далі. МК AVR циклічно виконує послідовність дій: почергове витягання поточних команд з пам'яті і їх виконання. Інакше кажучи, процедура автоматичного рішення задачі (обробки даних) включає ряд робочих циклів, званих циклами команди, які повторюються, до тих пір, поки не буде виконана вся сукупність команд програми. У процесі виконання програми лічильник команд PC завжди указує на поточну виконувану програму. При прочитуванні чергового коду команди значення лічильника збільшується на одиницю. Таким чином, після скидання нормально працюючий МК завжди знаходиться в стані «покрокового» виконання програми. При цьому програміст не може управляти описаною циклічною послідовністю. Виконання команд користувача, зчитаних з їх пам'яті програм, виконується в строгій послідовності. Існує декілька виключень, коли вміст програмного лічильника може бути змінений за допомогою підпрограм, спеціальних команд управління або шляхом завдання порядку виконання команд. Вміст лічильника команд, а, отже, і послідовність виконання програми, різко міняється при виконанні команд безумовного і умовного переходів. Вміст лічильника команд може скидатися за спеціальною командою управління – сигналом сторожового таймера, що забезпечує можливість відновлення правильного ходу обчислювального процесу в збійних ситуаціях. Вміст лічильника команд може скидатися по спеціальній команді управління – сигналу схеми BOD (МК має вбудовану схему контролю напруги живлення BOD, яка відсте-

жує короткочасне зниження напруги живлення і гарантує формування сигналу системного скидання, якщо напруга живлення впаде нижче за рівень, при якому не гарантується нормальна робота МК).

У МК AVR є сукупність засобів (система синхронізації), які здійснюють правильну передачу інформації від однієї регістрової структури до іншої і формують послідовність сигналів управління, забезпечують синхронне виконання окремих дій в строгі моменти часу (такти). Система синхронізації спільно з пристроєм, що дешифрує код операції, є основою пристрою управління, що забезпечує роботу МК у цілому. За допомогою системи синхронізації формують стабільну, строго синхронізовану в часі періодичну послідовність імпульсів і задають цикл команди – інтервал часу, необхідний для прочитування команди з пам'яті програм і її виконання. Цикл команди складається з певної послідовності елементарних дій – тактів. Тактові сигнали служать для синхронізації функціонування всієї системи в цілому і вони визначають швидкість виконання команд. Для генерування послідовності сигналів (тактів системної синхронізації), що управляють, в МК використовується окремий модуль синхронізації. Він програмується і, залежно від коду, може мати різні варіанти конфігурації.

Як генератор синхроімпульсів (послідовності прямокутних імпульсів) в МК може використовуватися інтегрований в мікросхему МК генератор тактових імпульсів на основі підсилювача з позитивним зворотним зв'язком. При цьому є можливість використовувати в якості елемента, який задає час, цього генератора або зовнішній кварцевий, або керамічний резонатор (осцилятор), або RC-ланцюг. Як альтернативний варіант управління подіями в часі може також використовуватися зовнішній тактовий сигнал (зовнішня синхронізація). За умовчанням в МК використовується внутрішній RC-генератор, який не використовує зовнішні компоненти. Вбудований RC-генератор виробляє коливання фіксованої частоти f_{CLK}, наприклад 8 Мгц. Шляхом програмування цю частоту можна зменшити, наприклад, у 8 разів. Оскільки стабільність RC-генератора не висока, то в МК AVR використовують його калібрування (коди корегувань для зміни частоти або байт калібрування) або, найчастіше, схеми синхронізації з кварцевим або керамічним резонатором. Схема синхрогенератора МК AVR із зовнішнім резонатором показана на рис. 3.3 а. Синхронізація МК здійснюється з використанням внутрішнього інвертуючого

підсилювача, який перетворюється на синхрогенератор за допомогою підключення до виводів XTAL1 і XTAL2 зовнішнього резонатора. Вивід XTAL1 є входом, а вивід XTAL2 виходом внутрішнього підсилювача. Величина місткості конденсаторів C1 і C2 вибирається залежно від типу резонатора (кварцевий або керамічний) і частоти (f_{CLK} <200 кГц, f_{CLK} < 4МГц, f_{CLK} <20 Мгц), на якій працює генератор. Значення місткостей можна знайти в довідкових даних. Схема тактування МК з використанням зовнішнього синхросигнала показана на рис. 3.3 б. Щоб можна було використовувати сигнал від зовнішнього генератора, необхідно програмно перевести МК у режим синхронізації від зовнішнього сигналу і подати імпульсний сигнал синхронізації на вхід XTAL1. Слід, при цьому, добитися того, щоб формовані зовнішнім генератором рівні відповідали логічним рівням МК. Щоб гарантувати стійку роботу центрального процесора при використанні зовнішнього тактового генератора необхідно уникати раптових змін частоти зовнішнього сигналу.



а) б)

Рис. 3.3. **Схеми синхрогенератора МК AVR із зовнішнім резонатором (а) і з використанням зовнішнього синхросигнала (б)**

Оскільки максимально досяжна продуктивність, швидкодія, при якій окремі частини МК здатні надійно виконувати задані функції, залежать від частоти проходження тактових імпульсів, то при описі команд, послідовності дій і подій, що відображають роботу МК, важливо мати на увазі наступне. Вказана у довідкових даних частота синхронізації звичайно відповідає частоті кварцевого резонатора f_{CLK}, що підключається. Разом з тим, функціонування ядра AVR, швидкість виконання команд визначає частота тактових імпульсів по внутрішніх магістралях управління, адреси і даних f_{BUS}. Відношення f_{CLK} і f_{BUS} може бути менше одиниці,

із-за чого у деяких МК при заявленій однаковій частоті тактового генератора цикл виконання команди може займати декілька періодів тактового генератора і час виконання команди може бути більше.

Важливо, що у МК AVR співвідношення f_{CLK} і f_{BUS} рівне 1. А оскільки інтервал часу, що виробляється синхрогенератором, так званий машинний такт, відповідає одному періоду тактових імпульсів на внутрішніх магістралях, то, отже, тривалість циклу синхронізації рівна періоду тактування внутрішніх магістралей. Це дозволяє виконувати майже кожну команду за один машинний такт, збільшуючи лічильник команд на кожному такті. Виконання за один такт системного генератора по одній команді дозволяє організувати багаторівневу конвейєрну обробку інформації, при якій кожна команда з пам'яті програм виконується за один машинний такт. Доступ до 32 восьмирозрядних регістрів загального призначення також здійснюється за один такт системного генератора, тому регістровий файл часто називають файлом регістрів швидкого доступу. За один такт генератора в АЛУ здійснюються відразу три дії: з регістрового файлу читаються два операнди, виконується операція, і результат знову ж таки зберігається у файлі регістрів. При заявленій максимальній тактовій частоті 20 Мгц у МК AVR досягається швидкодія 20 MIPS.

3.2.4. Пристрій пам’яті МК для зберігання даних

Для даних, які повинні зберігатися після відключення робочої напруги, наприклад, для значень фіксованих налаштуваньк обладнання, використовується електрично стиранє, програмуєме користувачем ПЗП – EEPROM (Electrically Erasable Programmable ROM). На схемі МК це пам'ять даних. Програма управління МК AVR може у процесі свого виконання записати дані в EEPROM і прочитати їх звідти. Пам'ять даних допускає до 100 000 циклів стирання-запису. Цей вид незалежної пам'яті відрізняється від Flash-пам'яті тим, що допускає перезапис одного байта без стирання інших. Комірка EEPROM займає значно більше місця на кристалі, чим комірка Flash-пам'яті. Об'єм пам'яті EEPROM порівняно невеликий. Для різних МК AVR він складає від 64 байт до 2 кілобайт. Конкретне значення об'єму пам'яті МК можна дізнатися з довідкових даних (графа EEPROM). Адресний простір пам'яті незалежних даних і констант МК показаний на рис. 3.4. Він, представляючи деяку область, починаєть-

ся з комірки (слова), яка має нульову адресу $0000. Вона зберігає якийсь параметр у вигляді двійкового числа з 8 розрядів. Адреса останнього елементу пам'яті буде різною для різних типів МК. Всі комірки призначені для довготривалого зберігання даних.

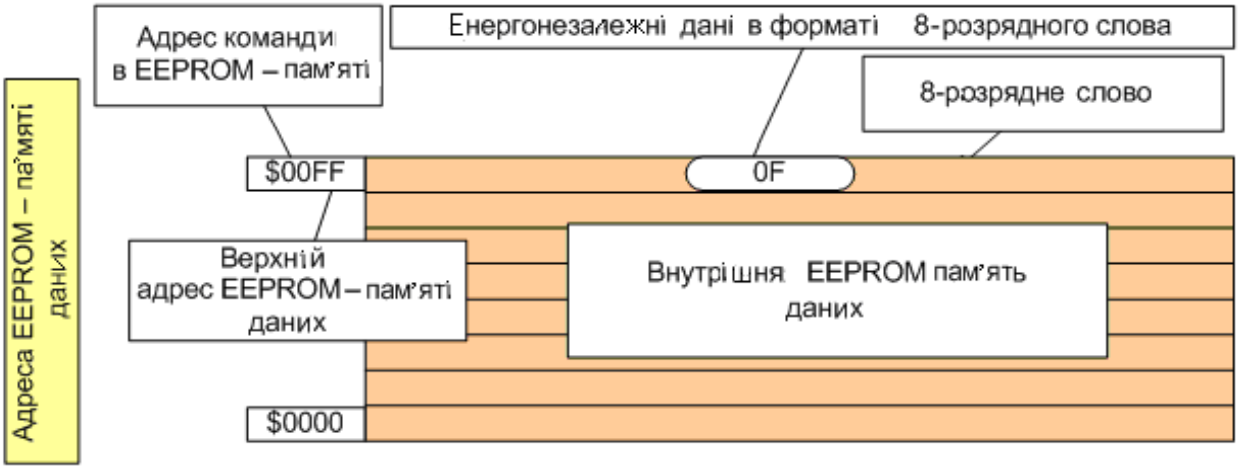


Рис. 3.4. **Графічне зображення адресного простору незалежної пам'яті даних мікроконтролера**

Записувати інформацію в EEPROM можна за допомогою спеціального пристрою – програматора, що забезпечує задану послідовність програмуючих імпульсів з амплітудою 10 – 25 В.

Слід мати на увазі, що у центрального процесорного елемента відсутня можливість звертатися до пам'яті даних, як до пам'яті програм, оскільки він «не бачить» адресного простору EEPROM. До цього виду пам'яті МК AVR може «звертатися» тільки за допомогою спеціальних регістрів введення – виведення. Для здійснення програмування безпосередньо в ході виконання призначеної для користувача програми в МК AVR використовуються три 8-розрядні регістри EEPROM: регістр адреси EEAR (МК вибирає комірку, куди потрібно записати або звідки можна прочитати дані), регістр даних EEDR (через нього поступає записуваний байт і процесор одержує байт при читанні з EEPROM) і регістр управління EECR (визначає режими роботи пам'яті).

3.2.5. Підсистема (модуль) введення – виведення інформації

Система управління з МК обов'язково містить підсистему введення – виведення даних, в якій пристрій введення здійснює введення даних,

що підлягають обробці, а пристрій виведення перетворить результат обробки інформації у форму, зручну для сприйняття пристроями, зовнішніми, відносно до МК. Треба уявляти собі, що підсистема введення – виведення МК – це деяка сукупність програмних і схемотехнічних засобів, призначена для узгодження електричних, логічних і часових параметрів сигналів при обміні інформацією між процесором і периферійними пристроями. Враховуючи однотипність багатьох периферійних пристроїв, що підключаються до МК, в сучасних МК для взаємодії з периферією, звичайно використовують здатні легко адаптуватися до різного термінального обладнання, програмовані підсистеми (модулі) введення – виведення. До модулів такого роду, що виконують операції обміну з термінальним обладнанням, в першу чергу відносяться паралельні «порти» введення – виведення, тобто спеціальні інтерфейсні схеми тієї або іншої розрядності, через кожну з яких підключається периферійний пристрій

Порти введення – виведення – це обов'язковий атрибут МК AVR. У своєму складі МК AVR можуть мати від одного до семи портів введення – виведення. Кожен порт має своє ім'я. Вони іменуються (нумеруються) латинськими буквами від A до G. Кожен вивід порту може бути індивідуально конфігурований, як на вхід (на прийом інформації), так і на вихід (на видачу сигналів). Порт введення – виведення, в першому наближенні, це логічне об'єднання виводів МК, що ідентифікується одним ім'ям (назвою порту), через яке процесор МК може одночасно взаємодіяти зі всіма пинами порту. Відповідно до такого поняття порту в МК деяка кількість ліній введення – виведення (пинів, металевих провідників на корпусі мікросхеми, які можна використовувати для прийому і видачі електричних сигналів), об'єднують в групу, якій додають деякий сенс. Логічне об'єднання груп ліній в порти дозволяє організувати звернення до них як до елементів пам'яті, що зручно при організації обміну даними в паралельному форматі.

При прочитуванні інформації з ліній введення – виведення логічним сигналам, які зараз присутні на ніжках (на пінах) мікросхеми, додають сенс регістра PINx з певною кількістю розрядів. При цьому вважають, що кожен розряд регістра PINx, який служить для обміну інформацією із зовнішніми пристроями, приєднаний до одного з металевих виводів (фізичних контактів), розташованих на корпусі мікросхеми. Регістр PINx доступний тільки для читання даних і його називають часто регістром безпосереднього читання стану ліній порту. При цьому МК проводить прочиту-

вання даних з регістра PINx в моменти часу, які визначаються програмою, і ніяк не пов'язані з моментами зміни даних на лініях порту.

Порти введення – виведення МК AVR є двонаправленими, при цьому у разі потреби кожна лінія порту може бути конфігурована індивідуально. Для забезпечення двонаправленности в МК є регістр напряму передачі DDRx (під «x» тут мається на увазі конкретна буква – ім'я порту), за допомогою якого виконується необхідна конфігурація, і драйвери ліній із змінною схемотехнікою.

Якщо лінія порту конфігурована відповідно до DDRx на вхід, то при установці біта регістра PORTx в одиницю драйвер підключає внутрішній резистор, який одержав назву «підтягаючого резистора» (він створює на ніжці потенціал, близький до напруги джерела живлення або, по-іншому, «підтягає» потенціал піна до величини напруги джерела живлення). Підключення такого резистора переслідує мету – скоротити число навісних елементів при використанні МК. Резистор, який «підтягає до одиниці або до напруги живлення» (pull-up), дає можливість забезпечити рівень логічної одиниці на металевому контакті, що дозволяє розширити здібності порту, оскільки створює витікаючий струм для зовнішніх компонентів, підключених між контактом порту і загальним дротом. Якщо лінія порту налаштована на вихід (пін шляхом запису в регістр DDRx одиниці конфігурований на вихід), то для видачі інформації на зовнішні контакти порту МК, треба записати байт даних в регістр PORTx. Вміст відповідного біта цього байта з'явиться на металевому виводі у вигляді рівня напруги і присутній там, поки не буде замінено іншим, або дана лінія порту не буде перемкнута на введення.

Режими роботи портів введення – виведення, можливі напрями передачі інформації, конфігурації підтягаючих резисторів відповідно до логічних рівнів регістрів PORTx і DDRx, представлені в табл. 3.1.

Звичайно порти МК AVR восьмирозрядні, але в деяких випадках окремі розряди не використовуються. Процесор завжди пише у восьмирозрядні порти або читає з них повноцінний байт інформації. Невживані розряди при читанні байта з порту рівні нулю. Незастосовувані біти при записі в порт просто втрачаються.

Оскільки паралельні порти можуть застосовуватися для управління світлодіодними або іншими цифровими індикаторами, оптронами, реле, то розглядаючи порти і особливості драйверів ліній введення-виведення, зупинимося на понятті здатності, навантаження лінії. Цим поняттям ви-

значають граничний струм, який може протікати через виведення МК, не виводячи його з ладу. Якщо йдеться про нормальну здатність навантаження, то допустима сила струму кожного виводу паралельного порту може складати 20 мА, що достатньо для того, щоб безпосередньо управляти роботою рідкокристалічних індикаторів або забезпечувати свічення одного світлодіода. Сумарний струм порту на повинен перевищувати 50 мА. З цієї причини, якщо потрібно управляти більш ніж чотирма звичайними світлодіодами, оптопарами, або використовуються надяскраві світлодіоди, що працюють при струмі 100 мА, то в таких випадках застосовують інтегральні мікросхеми з підсилювачами струму. Навіть найменші реле споживають струм значної сили. Тому для управління ними необхідні спеціальні підсилювачі з використанням транзисторів.

Таблиця 3.1.

Режими роботи портів введення – виведення

Розряд n-регістра DDRx	Розряд n-регістра PORTx	Напрям передачі даних	Чи підключений резистор,що «підтягає»	Стан виводу порту
0	0	Ввід	Ні	Контакт відключений (Z-стан)
0	1	Ввід	Так	Резистор навантажує вхід і контакт є джерелом струму
1	0	Вивід	Ні	Логічний «0»
1	1	Вивід	Ні	Логічна «1»

Логічна організація паралельних портів достатньо проста і очевидна: пересилка даних у регістри введення-виведення будь-якого порту відображається на його роботі протягом одного такту мікроконтролера. До їх істотних властивостей можна віднести як повну взаємну незалежність портів, так і повну незалежність окремих ліній введення-виведення: функції кожної лінії введення-виведення можна програмувати і використовувати абсолютно незалежно.

Окрім роботи як цифрові лінії введення – виведення загального призначення, вивводи всіх портів можуть виконувати відповідну альтер-

нативну функцію. При цьому через ці ж лінії введення – виведення передаються сигнали інших інтерфейсних засобів, наприклад АЦП, або аналогового компаратора. Передачу цих сигналів надалі називатимемо альтернативними функціями портів. При використанні альтернативних функцій портів введення – виведення в регістрах управління повинні задаватися відповідні цим функціям напрями передачі сигналів. Одночасне програмування декількох функцій для однієї і тієї ж лінії введення – виведення не можливо.

3.2.6. Оперативні пристрої пам’яті МК і система адресації регістрів

Відомо, що «мозком» сучасного МК є процесор і пам'ять. Оперативний запам'ятовуючий пристрій (ОЗП), за своєю ідеєю, призначено для забезпечення центрального процесорного пристрою CPU, що працює у високому темпі, «оперативними» даними. У ОЗП МК зберігається інформація, що оновлюється у ході роботи МК, – часові дані, необхідні для виконання якоїсь частини програми, проміжні і кінцеві результати обробки. Такі дані МК повинен швидко прочитувати і, за певними командами, записувати для часового зберігання. При цьому в режимах читання і запису ОЗП повинен функціонувати з високою швидкодією. Час запису і читання слова в ОЗП повинен бути порівнянним з тривалістю тактових імпульсів і складати долі мікросекунд. При цьому оперативна пам'ять для зберігання інформації вимагає електричну енергію і навіть при короткочасному відключенні джерел живлення ОЗПУ втрачає інформацію.

Оперативна пам'ять, що допускає з високою швидкістю запис і прочитування двійкових чисел, які не повинні зберігатися після відключення робочої напруги, в МК AVR реалізована у вигляді статичної пам'яті – SRAM. Оскільки елементарною одиницею інформації є біт, то оперативну пам'ять SRAM можна розглядати як якийсь набір комірок, кожна з яких може зберігати лише один інформаційний біт. Пам'ять SRAM виконана у вигляді окремих комірок – RS трігерів на основі КМОП-інверторів і може працювати при зменшенні тактової частоти аж до 0 Гц. RS-трігери, як ви пам'ятаєте, це цифрова схема з двома стійкими станами. Після запису біта в таку комірку вона може перебувати в одному з цих станів і зберігати записаний біт скільки завгодно довго – необхідна тільки наявність на-

пруги живлення. Звідси і назва пам'яті – статична, тобто що перебуває у незмінному стані.

Перевагою статичної пам'яті є її швидкодія, а недоліком – високе енергоспоживання і низька питома щільність даних, оскільки ода комірка трігера складається з декількох транзисторів і, отже, займає немало місця на кристалі. Елементи пам'яті в ОЗП розташовані у формі 2-D матриці. Кожа комірка має унікальну позицію, що задається рядком і стовпцем, на перетині яких вона лежить. Комірки організовані в набори слів, коли всі біти одного слова прочитуються одночасно. При послівній організації виділяється жорстко фіксоване число елементарних комірок, в яких може одночасно записуватися або прочитуватися певна порція інформації. Кожному рядку (слову, порції інформації) за її положенням у матриці привласнена адреса – ціле число. Структурне розташування комірок рядка (слова) у пам'яті ОЗП – по 8 позицій для розміщення біт (нулів і одиниць). Запам'ятовуючий пристрій, що складається з набору трігерів (елементів) з двома стійкими станам, називається регістром. Тому пам'ять SRAM є сукупність функціонально однакових 8-розрядних регістрів з місткістю по 1 байт. Кожен регістр має свій ідентифікаційний номер (код), адреса, яка однозначно визначає той або інший регістр усередині ОЗП. Сукупність регістрів, до яких може звертатися МК, називається адресним простором ОЗП. Адресний простір ОЗП доступний центральному процесору. Він визначає потенційні можливості ОЗП – число можливих відмітних один від одного кодових комбінацій (адрес). Адресний простір графічно зображається у вигляді прямокутника з деякого числа рядків (комірок), рівного числу фактично використовуваних регістрів. Кожен регістр в адресному просторі пронумерований і має свою адресу з номером у вигляді двійкового коду. Адресний простір ОЗП МК показаний на рис. 3.5.

Адресний простір, що представляє деяку область, розташовану за адресами з $0000 по $0FFF, починається з регістра, який має нульову адресу $0000. При графічному уявленні нумерація регістрів адресного простору, призначених для часового зберігання даних, проводиться від низу до верху. Адреса останнього елементу пам'яті буде різною для різних типів МК. У регістрах SRAM можуть зберігатися якісь параметри у вигляді двійкового числа з 8 розрядів. Різні області адресного простору ОЗП звичайно групуються в блоки, утворюючи якусь «карту пам'яті». Поняття адресного простору з різними ділянками «карти пам'яті» дозволяє

наочно представити розміщення в ньому адрес різних програмно-доступних об'єктів.

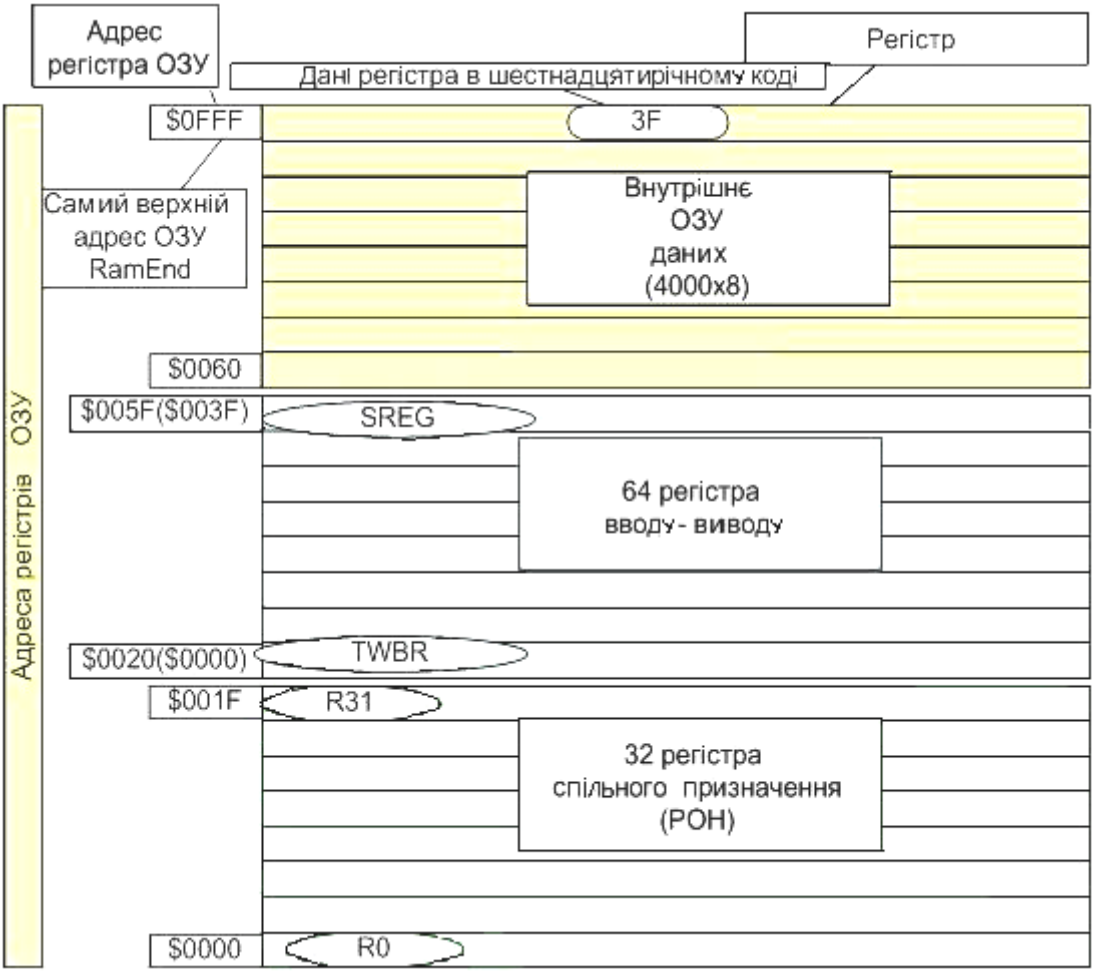


Рис. 3.5. **Графічне зображення адресного простору ОЗП мікроконтролера**

Адресний простір ОЗП (SRAM), що є простіром з крізною адресацією, можна розділити на 3 області. 1. Область пам'яті, суміщена з регістрами загального призначення (РОП). Ця сама нижня область пам'яті ОЗП утворена регістровим файлом з 32 робочими регістрами. Її займають РОП з адресами з $0000 по $001F. Всі РОП пов'язані з АЛП і доступом до них може бути виконаний протягом одного такту системної синхронізації. Це означає, що за час такту в АЛП вводяться два операнди з регістрового файлу, виконується операція, а результат запам'ятовується в регістрі загального призначення (записуючи байт даних в елемент пам'яті з адресою $0000, ми насправді поміщаємо байт в регістр R0 і на-

впаки). Подвійний доступ до РОП істотно збільшує можливості при написанні програм. 2. Область пам'яті SRAM, суміщена з регістрами введення – виведення (РВВ). Цю область займають регістри, призначені для програмування, управління і сигналізації про всі периферійні функції МК, з адресами з $0020 по $005F. Кожному такому регістру введення – виведення відповідає елемент пам'яті в ОЗП. Для команд роботи з регістрами простору введення – виведення використовуються вказані в дужках адреси в просторі введення – виведення (з $0000 по $003F). Адреса елементу пам'яті ОЗП при наскрізній адресації завжди більше номера відповідного РВВ на постійну величину 32 ($0020). Адреса регістра ОЗП наводиться у круглих дужках після безпосередньої адреси області простору введення – виведення. Розміщення регістрів введення – виведення у кожній моделі МК фіксовано і адреси приводяться в довідкових даних до МК. 3. Область пам'яті SRAM, звана внутрішнім ОЗП. Цю область займають регістри, призначені для оперативного зберігання даних. Ця область простору починається з адреси $0060 і продовжується у бік збільшення адрес до останнього елементу пам'яті. Адреса останньої комірки залежить від побудови внутрішнього ОЗП конкретного типу МК. Цей елемент пам'яті має адресу, яку називають RamEnd.

У МК простір пам'яті SRAM можна збільшувати (аж до 64 кілобайт) за допомогою підключення зовнішніх блоків пам'яті. Для цієї мети використовують порти, наприклад, A і C.

3.3. Організація взаємодії між приймачами і джерелами інформації і переривання в МК AVR

3.3.1. Початкові відомості про управління обладнанням у реальному часі

При роботі комп'ютеризованого обладнання всі його робочі механізми повинні виконувати певні закони руху, а послідовність переміщень їх робочих органів (включення, виключення, рухи) повинна бути пов'язана між собою в часі (синхронізована) і за положенням у просторі. Коли в контурі управління «працює» МК, що уявляє собою обчислювальну машину управління, точне узгодження роботи виконавчих механізмів у просторі і синхронність функціонування у часі особливо важливі. Більшість завдань управління, які покладаються на МК в комп'ютеризованому об-

ладнанні, повинні виконуватися у реальному часі. При цьому для ефективного вирішення подібних завдань важливо вибрати принцип зв'язку МК з об'єктом (обміну інформацією із зовнішніми пристроями). У простому випадку, коли обладнання працює циклічно, технологічні процеси характеризуються періодичністю, енергетичні й інформаційні потоки супроводжуються повторюваністю у часі, це може бути синхронний принцип зв'язку МК з об'єктом, при якому процес управління розбивається на цикли рівної тривалості за допомогою тактуючих імпульсів, що генеруються «електронними годинниками». На початку кожного циклу проводиться послідовне опитування і перетворення у цифрову форму сигналів датчиків, а потім, після вироблення дії управління і перетворення його в доцільні переміщення, повороти робочого органу, силової дії на керовані механізми, МК «припиняє» роботу до приходу нового тактуючого імпульсу. Робота більшості робочих органів обладнання і механізмів друкарських машин характеризується зміною в широких межах умов технологічних процесів і різних діючих на обладнання обурюючих дій. Для забезпечення якнайкращого ходу такого технологічного процесу при змінних збуреннях і умовах роботи потрібно використовувати асинхронний принцип зв'язку МК з об'єктом. Замість тактуючих імпульсів МК використовує сигнали «переривання», що поступають від найбільш важливих датчиків (наприклад, датчиків аварійного стану). Кожен імпульс переривання еквівалентний вимозі про припинення обчислень дій управління і переходу до виконання розрахунків, відповідних даному сигналу переривання. У загальному випадку може застосовуватися комбінований спосіб – синхронізації «плюс» датчики аварійного стану, що переводять МК у режим обслуговування переривань (наприклад, у режим аварійної роботи).

У асинхронних і комбінованих системах управління складними об'єктами багато завдань обміну інформацією МК з тими, що працюють з різною швидкістю передачі даних зовнішніми пристроями, можна вирішувати роздільно, забезпечуючи роботу системи управління з розділенням у часі. Якщо, крім розділення в часі численних сигналів від різноманітних датчиків стану об'єктів, застосовувати одну і ту ж апаратуру в різні моменти часу, то можна забезпечити і мінімальну кількість апаратури. Це характерно, наприклад, при рішенні задач обробки безлічі вхідних логічних сигналів, які виникають від різних типів джерел інформації і поступають у пристрій обробки в різний час.

Відомо, що в комп'ютеризованому поліграфічному обладнанні є різні функціональні вузли, що працюють абсолютно по-різному і з різною швидкістю. Частина з них функціонує таким чином, що протягом тривалого часу «чекає» «появи події – сигналу на початок роботи», наприклад, свого включення або появи якихось даних. Інша група функціональних вузлів сама формує сигнали, причому, тільки в тому випадку, якщо істотним чином міняється їх стан (наприклад, відкрилися дверці, закінчилася фарба або охолоджувальна рідина). Третя група функціональних вузлів включаються в роботу періодично, після закінчення певного проміжку часу. У періодичному режимі працюють вали поліграфічної машини, що обертаються. У цих пристроях дані з датчика (наприклад, про те, що зроблений повний оберет валу) «прочитуються» тільки в певні проміжки часу (коли магнітний маркер на валу проходить мимо датчика Холу). Для раціонального вирішення подібних завдань треба розділити робочий час на пасивні і активні інтервали і зробити так, щоб процесор МК у певний («активний») проміжок часу здійснював «сеанс взаємодії» і виконував «роботу з управління», а решта часу, протягом «пасивного» проміжку часу, виконував інші корисні операції (функції) в системі управління. При цьому МК повинен вирішувати дані завдання у масштабі реального часу, коли дані для введення в МК виробляються дуже швидко і не можна у жодному випадку їх втратити. Стосовно конкретного об'єкта управління це означає, що управління обладнанням будується на основі розбиття технологічного процесу на тимчасові інтервали (технологічні цикли) необхідної тривалості, при яких центральний процесор МК здатний виконувати декілька функцій управління з розділенням їх у часі. При такому управлінні є можливість переходу на конкретні вибрані інтервали, за визначених наперед умов, і видачі на виконавчі пристрої на кожному такому часовому інтервалі своїх дій управління. Управління у реальному часі дозволяє почати процес з першого інтервалу, зупинити, у разі потреби, його на будь-якому інтервалі, пустити його знов або повернути програму управління в початкове положення. При цьому, функції управління звичайно реалізуються як процеси, що починаються в певні моменти часу, або, по-іншому, «прив'язані» до часової сітки з елементарним інтервалом часу, і що продовжуються протягом певного часу. При цьому слід мати на увазі, що в системі управління деякі важливіші події можуть «переривати» виконання менш важливих завдань і викликати певні дії, які повинні бути виконані раніше, до певного часу. При рішенні задач управ-

ління, які повинні виконуватися у реальному часі, можливо, використання квазіпаралельних процесів, коли на одному і тому ж інтервалі часу виконується декілька функцій управління. Але організація майже паралельних процесів управління вимагає дуже ретельного їх планування: забезпечення необхідних ресурсів, механізмів взаємодії, точного розрахунку часу виконання операторів кожного з процесів. Організація взаємодії квазіпаралельних процесів вимагає значних ресурсів МК.

У системі управління обладнанням звичайно багато зовнішніх (периферійних) пристроїв, які взаємодіють з процесорним ядром. Тому для управління конкретним пристроєм обладнання у реальному часі також правильно організувати взаємодію із зовнішніми пристроями. Процесор повинен обслуговувати периферійні пристрої тоді, коли вони «готові» до цього. При цьому процесор може звертатися тільки до одного з них і виключно на своєму інтервалі часу. Якість управління у реальному часі істотно залежить не тільки від «швидкості обчислень в процесорі, але також і від того наскільки ефективно розділене завдання управління на підзадачі, а підзадача управління – між різними модулями МК.

Таким чином, в системах управління з МК у реальному часі, коли периферійні пристрої володіють різною швидкістю передачі даних і часом обміну інформацією, коли не можна втратити інформацію, що поступає на вхід МК, для ефективного управління МК повинен, принаймні, уміти вирішувати наступні задачі: а) встановлювати «порядок» взаємодії в часі центрального процесорного пристрою з периферійними пристроями; б) формувати, для синхронізації всіх процесів у МК, «часову сітку» у вигляді послідовності імпульсів із заданим елементарним часовим інтервалом; в) визначати порядок «запрошення» периферійних пристроїв до роботи, час доступу їх до центрального процесора; г) уміти сприймати сигнали, які «повідомляють» процесор про «готовність» окремих пристроїв до дії; д) формувати необхідні «затримки» для встановлення деякого часу «очікування» для обслуговування конкретного пристрою.

3.3.2. Початкові відомості про виконання МК переривань і вкладених підпрограм

Найважливіше значення, якщо говорити про взаємодію центрального процесорного пристрою з «периферією», для більшості МК має здатність перервати виконувану програму у відповідь на «зовнішню» (від-

носно до центрального процесора) подію і виконати програму, спеціально призначену для обробки цієї події. Це називається перериванням поточної програми за запитом зовнішнього пристрою. Переривання (Interrupt) – це асинхронна подія, що полягає у порушенні послідовного ходу виконання програми МК і вимагає обробки, після чого виконання програми може бути продовжено в звичайному порядку. Переривання забезпечують мікроконтролеру можливість реагувати на всілякі події як всередині, так і поза ним, і при цьому ефективніше використовувати час процесора, а периферійним пристроям – здатність автономно функціонувати до того моменту поки не потрібно буде примусити центральний процесор обернути на себе увагу. Механізм переривань дає можливість МК використовувати вільні проміжки часу для виконання іншої програми. У результаті застосування механізму переривання досягається «ефект незалежності» двох процесів: програмно-визначуваних і незалежних (викликаних сторонніми від програми джерелами і моменти появи яких невідомі) подій. Виходить, що основна програма виконується МК як би сама по собі, а процедура обробки зовнішніх подій – окремо. Час роботи МК розподіляється між двома цими процесами таким чином, що обидва вони виконуються майже незалежно один від одного. Кажучи про роботу центрального процесорного пристрою, слід зазначити, що, в загальному випадку, програми виконуються не тільки в послідовності заданої користувачем, а процесор функціонує в трьох режимах: а) читання інформації з програмно-доступних елементів – регістрів, як елементів пам'яті; б) записи інформації в программно-доступні елементи; в) переривання поточної (основної, фонової) програми за сигналом зовнішнього пристрою з переходом на підпрограму обслуговування. Звичайно в МК системах управління запити на переривання можуть поступати від декількох зовнішніх пристроїв. Тому, по-перше, виникає проблема ідентифікації пристрою, що прислав запит, з тим, щоб можна було виконати дії з обслуговування саме цього пристрою. По-друге, при аналізі декількох запитів переривань МК виникає проблема визначення того, який з одночасно виниклих запитів є найбільш важливим або, по-іншому, пріоритетним. Існує декілька методів рішення вказаних проблем. Для виконання різних завдань МК на пріоритетній основі використовуються пріоритетні системи переривання. Інформацію, яка ідентифікує пристрій, що прислав запит в МК на переривання, визначають на основі векторної системи, при якій кожному перериванню відповідає так званий вектор переривання. У векторній сис-

темі переривань пристрій, що прислав запит у МК на переривання, ідентифікується за допомогою методу, суть якого полягає в тому, що після завершення контекстного перемикання МК одержує і передає в лічильник команд код, відповідний адресі того пристрою, який вимагає переривання. Код адреси, відповідний адресі того пристрою, який вимагає переривання, і що представляє безліч впорядкованих наборів з n двійкових чисел, може тлумачити як вектор в n-мірному просторі. «Вектор переривання» характеризується початком – початковою адресою підпрограми обробника переривання (кодом адреси елементу пам'яті, де записана перша команда підпрограми переривання) і «напрямом» розвитку обробки переривання, оскільки код прямо указує, куди треба рухатися, щоб досягти місця розташування підпрограми переривання. Векторні переривання мають дуже високу швидкість відгуку на запити переривання, хоча досягається це за рахунок значних апаратурних витрат.

Для характеристики можливості МК і систем управління на його основі зі здійснення переривань вводять поняття «рівні переривання». Нульовий рівень переривання відповідає МК, у якого не можна перервати поточну програму. Багато МК мають засоби для забезпечення декількох рівнів переривань. Це означає, що є можливість багатократного утворення переривань, у тому числі і переривання усередині самих переривань, або здійснення вкладених переривань, що ведуть до появи послідовностей вкладених підпрограм.

Після дозволу переривання, перш ніж переходити до початку підпрограми переривання, необхідно виконати послідовність дій, що дозволяють повернутися до вірного продовження програми, що переривається, після завершення підпрограми переривання. Ефективний програмно-апаратний засіб для збереження і повернення даних при контекстному перемиканні і здійсненні вкладених переривань – стек. Стек (Stack) – це область пам'яті, до якої можна адресуватися за допомогою спеціального покажчика. Звернення до стека проводиться відповідно до принципу «останній надійшов обслуговується першим». Призначення стека в тому, щоб зберегти поточний зміст необхідних регістрів, якщо відбувається переривання основної програми. Апаратний стек є сукупність елементів пам'яті, організованих так, що звернення до деякої послідовності (списку) слів може відбуватися у зворотному записі. За допомогою стека можна організувати як вкладені підпрограми, так і обслуговування переривань. Тому стек – це перш за все ефективний програмно-апаратний засіб для

збереження і повернення даних при роботі з підпрограмами. З його допомогою можна організувати вкладені підпрограми, коли основна програма викликає підпрограму, яка може викликати нову підпрограму, і т. д.

Іншим призначенням стека є обслуговування переривань. Для створення стека в МК використовується спеціально відведена послідовність елементів пам'яті ОЗП. При цьому реалізується така адресація, що завжди запис або читання відбувається в «верхній» комірці ОЗУ. Коли переривання буде завершено, програма повертається до верхівки стека і «забирає» дані з його поверхні, дотримуючи принцип «останнім увійшов – першим вийшов» Дані із стека вибираються до тих пір, поки не буде відновлене положення до переривання. В результаті програма, яка була не виконана повністю, може повернутися до завершення перерваних дій.

Покажчик стека (SP – Stack Pointer) – це регістр, який знаходиться у центральному процесорному пристрої. Вміст цього регістра показує адресу верхнього елементу списку даних в пам'яті ОЗП. Область стека заповнюється від старших адрес до молодших і, з даної точки зору, стек зростає «вгору».

Кожне звернення до стека для занесення в нього слова даних супроводжується автоматичним зменшенням вмісту покажчика стека на 1, а кожне звернення для витягання інформації із стека супроводжується додаванням 1 до вмісту покажчика стека. Після операції витягання вміст елементів пам'яті у верхівці стека не руйнується, а зберігається незмінним до включення в стек нової інформації.

Оскільки запити на переривання можуть поступати в будь-які моменти часу виконання поточної програми, то це може привести до перекручування одержаних результатів. У системах переривання МК передбачено програмно-управляюче перемикання режимів роботи підсистеми переривань і навіть відключення механізму переривань. Це дає програмісту можливість отримання гнучкого управління процесом обміну інформацією між різними пристроями.

Управління перериваннями, у загальному випадку, здійснюється за допомогою спеціальних команд «переривання дозволені» і «переривання заборонені» і за допомогою так званої «маски» переривань. «Маскування», тобто заборона виконання переривань певних рівнів, досягається за рахунок посилки коду маски в спеціальний регістр. Код «маски» вирішує виконання переривань тільки певним пристроям. Переривання за допомогою «маски» можуть бути також повністю відключені. «Маску»

можна вставити в програму скрізь, де це необхідно, і тим самим відмінити переривання на той час, поки вони можуть вплинути на точність виконання поточної програми або якої-небудь підпрограми.

3.3.3. Загальні відомості про підсистему переривань МК AVR

Підсистема, здатна обслуговувати декілька джерел переривань, присутня в будь-якому МК AVR. Вона працює як з внутрішніми, так із зовнішніми джерелами переривань. Кількість джерел переривань у різних типів МК різна. Якщо ви звернетеся до довідкових даних (графа «кількість переривань Внутрішніх/Зовнішніх»), то побачите, що мінімальна кількість джерел переривання має МК ATtiny11. Він здатний «реагувати» на одне «зовнішнє» переривання, що поступає на ніжку МК з ім'ям INT0 (INTerrupt 0), і на два внутрішні джерела переривань (від таймера лічильника і від вбудованого компаратора). Крім того, його джерелом переривань є початкове скидання МК. Вектор початкового скидання включають в таблицю векторів переривань. МК AVR інших типів мають складніші підсистеми переривань. Найрозвиненіша система переривань у мікроконтролера ATmega1281. Цей МК здатний обслуговувати 17 зовнішніх джерел переривань і 48 внутрішніх (джерелами переривань служать всі вбудовані таймери, компаратори, аналогово-цифрові перетворювачі, підсистеми управління пам'яттю, каналами передачі даних і т. п.). Всі переривання і механізм скидання характеризуються окремими векторами переривань, які, в сукупності, утворюють таблицю переривань. Ця таблиця розташовується за молодшими адресами простору пам'яті програм і для кожного МК вона своя. Її можна знайти в довідкових даних для МК. Управління підсистемою переривань здійснюється за допомогою групи регістрів введення-виведення. Визначальним тут є регістр стану SREG.

Звичайно одночасно всі переривання не використовуються. Для того, щоб вирішити одні переривання і заборонити інші застосовуються так звані маскуючі регістри (регістри маски). Регістр маски – це регістр, розташований у просторі введення – виведення ОЗПУ, який служить для управління окремими джерелами переривань. Кожному біту в регістрі маски відповідає одне джерело переривань. Якщо біт у такому регістрі скинутий в нуль, то переривання цього вигляду заборонені. Якщо біт встановлений в одиничний стан, то переривання дозволені. У МК AVR звичайно використовуються два регістри маски, що відповідають за об-

робку окремих переривань. Перший з них управляє всіма видами переривань, окрім переривань від таймерів. У деяких МК сімейства «MEGA» цей регістр називається головним регістром маски переривань GIMSK (General Interrupt Mask Register), в інших головним регістром маски управління перериваннями GICR (General Interrupt Control Register) або зовнішнім регістром маски управління перериваннями EICR (External Interrupt Control Register). Для управління перериваннями від таймерів є спеціальний регістр маски переривань від таймерів – лічильників TIMSK (Timer/Counter Interrupt Mask Register). У деяких типах МК використовують також регістр маски переривань зі зміною сигналу на будь-якому з контактів PCMSK.

Для управління процесом виконання переривання, окрім регістрів маски, існують ще регістри прапорів переривань. Кожен біт такого регістра – це прапор, що сигналізує про надходження одного з видів переривань. Під час вступу запиту на переривання прапор встановлюється в одиницю. За станом прапора можна судити про наявність запиту. У МК AVR є два регістри прапорів, розташованих у просторі введення-виведення ОЗП.

Ті ж переривання, що і регістр GIMSK, обслуговує регістр прапорів глобальних переривань GIRF (General Interrupt Flag Register), що іменується також, в деяких типах МК, як регістр прапорів зовнішніх переривань EIRF (External Interrupt Flag Register). Прапори переривань від таймерів і лічильників знаходяться у регістрі прапорів переривань таймерів/лічильників TIRF (Timer/Counter Interrupt Flag Register).

Переривання можуть стати активними тільки тоді, коли в регістрі стану встановлений розряд загального дозволу переривань I («прапор I»). Якщо це має місце, і наступає подія переривання, то виконання поточної програми припиняється і переходить за відповідною адресою.

Загальний алгоритм роботи системи переривань наступний.

Після включення і, відповідно скидання МК, всі переривання заборонені і прапори дозволу скинуті в нуль. Якщо планується використовувати один з видів переривань, то необхідно передбачити в програмі включення цього переривання. Для глобального включення переривань необхідно встановити прапор I в регістрі стану SREG в одиницю і записати в регістр маски такий код, який дозволить лише потрібні в даний момент переривання. Вирішивши, таким чином переривання, програма може приступити до виконання свого головного завдання.

Під час вступу запиту на переривання і заданій масці встановлюється прапор відповідного переривання (встановлюється навіть у тому випадку, якщо глобальне переривання заборонене). Якщо переривання дозволене, то МК приступає до його обробки. Поточна програма тимчасово припиняється, і управління передається на адресу відповідного вектора переривання. У той же момент прапор I у регістрі стану SREG автоматично скидається в 0, тим самим забороняючи обробку інших переривань. Прапор у відповідному регістрі прапорів переривань, що відповідає викликаному перериванню, також скидається в 0, сигналізуючи про те, що МК вже приступив до його обробки. Оскільки всі підпрограми обробки переривань обов'язково закінчуються командою повернення з переривання RETI, то досягши цієї ділянки підпрограми, управління передається в ту точку основної програми, в якій перервалася її робота. Прапор I в регістрі стану SREG автоматично встановлюється в 1, вирішуючи нові переривання. Помітимо, що без вживання спеціальних заходів неможливі вкладені переривання, оскільки поки обробляється одне переривання, вся решта переривань заборонена. Щоб всі запити, що поступили, на переривання були оброблені, в МК під час вступу запиту на переривання обов'язково встановлюється в 1 відповідний прапор у регістрі прапорів. У цьому стані він буде знаходиться до тих пір, поки дане переривання не буде оброблено. Після закінчення обробки чергового переривання відбувається перевірка всієї решти прапорів, і якщо є хоч би одне необроблене переривання, МК відразу ж переходить до його обробки.

Запити на входах зовнішніх переривань можуть фіксуватися і відповідним чином ідентифікуватися таким чином. По-перше, запит переривання може ідентифікуватися за низьким рівнем напруги на вході. По-друге, запит переривання може ідентифікуватися за зростаючою ділянкою сигналу (фронтом імпульсу), або за спадаючим (спадом імпульсу).

Час реагування, що пройшов від настання переривання до виконання відповідної підпрограми переривання, для всіх видів переривань складає мінімум чотири цикли такту системної синхронізації. Протягом цих чотирьох тактів два байти адреси повернення будуть збережені в стеку, покажчик стека буде декрементований на 2, а також буде виконаний перехід за адресою відповідної підпрограми. Якщо переривання наступає під час виконання команди, що складається з декількох циклів, то у будь-якому випадку виконання цієї команди буде закінчено, а потім буде оброблене переривання. Повернення з підпрограми обробки перери-

вання (як і при поверненні із звичайної підпрограми) виконується також за чотири цикли. Протягом цих чотирьох тактів системної синхронізації два байти адреси повернення будуть перенесені із стека в лічильник команд, і покажчик стека буде інкрементований на 2. Повернення з підпрограми обробки переривання реалізоване, як правило, за допомогою команди RETI. Ця команда приводить до повернення до місця переривання програми, але також встановлює прапор I в одиницю у регістрі стану SREG для того, щоб знову вирішити переривання. Вміст регістра стану SREG ні при перериваннях, ні при виконанні звичайних підпрограм автоматично не зберігається. Збереження вмісту регістра SREG, а також його відновлення при виході з підпрограми у разі потреби повинно виконуватися за допомогою спеціальних дій, передбачених у програмі.

3.4. Таймері і процесори подій МК AVR

3.4.1. Загальні відомості про таймери і процесори подій МК AVR

Вже відзначалося, що управління обладнанням у комп'ютеризованих системах повинне здійснюватися у реальному часі. Для цього в 102 пристроях автоматики і комп'ютеризованих системах доводиться вирішувати наступні задачі: генерувати імпульсні сигнали, у тому числі і змінної частоти; вимірювати тривалість сигналу; формувати імпульсні сигнали, затримані у часі на програмоване значення; виконувати вимірювання інтервалів часу; формувати імпульсні сигнали з програмованою частотою і програмованим коефіцієнтом заповнення. Крім того, часто доводиться вести підрахунок числа зовнішніх подій і визначати число імпульсів зовнішнього сигналу на заданому часовому інтервалі.

Багато з названих завдань окремо, у принципі, можуть бути, вирішені програмним шляхом, за рахунок використання команд декрементирування (інкрементування) вмісту РОП. Проте, оскільки в ході рішення таких задач використовується центральний процесорний пристрій, то під час виконання таких операцій МК не можна використовувати для інших цілей. Тому всі подібні програмні рішення, пов'язані із завантаженням процесора «рутинною роботою», будуть при рішенні задач управління володіти таким істотним недоліком, як неефективність використання

процесора. Крім того, точність цього методу при формуванні часових інтервалів невелика.

Набагато кращі результати можуть бути одержані, якщо для виконання функцій, пов'язаних з управлінням у реальному часі, використовуються спеціалізовані апаратні засоби, наявні в МК, які називають таймерами/лічильниками. Вони дозволяють не тільки підвищити ефективність використання процесорного ядра МК, але і забезпечити значно кращі можливості для МК при синхронізації зовнішніх пристроїв.

МК серії AVR звичайно містять декілька вбудованих таймерів, які за призначенням можна розділити на дві групи. До першої групи відносять сторожовий таймер. До другої групи належать таймери-лічильники загального призначення. Сторожовий таймер в мікросхемі один і призначений для автоматичного перезапуску МК у разі «зависання» (зациклення програми в результаті помилки, допущеної програмістом, або в результаті дії зовнішньої перешкоди) його програм і відновлення правильного ходу обчислювального процесу при такого роду відмовах. Таймери загального призначення використовуються при рішенні багатьох наукових і технічних проблем, пов'язаних з вимірюванням і формуванням інтервалів часу, що розділяють два характерні моменти якого-небудь процеса. Вимірювання інтервалів часу необхідні для створення «затримок» сигналів у часі, для синхронізації роботи окремих вузлів обладнання, формування прямокутних імпульсів із заданими параметрами. Крім того, таймери загального призначення можуть працювати в режимі лічильника і підраховувати кількість тактових імпульсів заданої частоти, вимірювати тривалість зовнішніх сигналів або кількість подій за певний проміжок часу подій. Часто таймери називають «таймерами-лічильниками». Для вимірювання і формування інтервалів часу в таймерах-лічильниках використовується метод дискретного рахунку, суть якого, стисло зводиться до наступного. Шуканий інтервал часу Δt_X заповнюється імпульсами з відомим зразковим періодом проходження $T_{ОБР} \ll \Delta t_X$, тобто часовий проміжок перетвориться у сукупність періодично наступних імпульсів, число m яких, пропорційно Δt_X, підраховується. Імпульси, що заповнюють інтервал, прийнято називати рахунковими. Проміжок часу Δt_X, що «вирізує» зі всієї імпульсної послідовності імпульсів тільки потрібну і задаючий тривалість рахунку, прийнято називати «часовими воротами». Якщо період проходження рахункових імпульсів $T_{ОБР}$ (частота проходження $F_{ОБР}$), то

за часовий інтервал Δt_X через часові ворота пройде імпульсів $m = \Delta t_X / T_{ОБР} = \Delta t_X F_{ОБР}$ і, отже, шуканий інтервал $\Delta t_X = m / F_{ОБР} = m T_{ОБР}$.

У МК AVR застосовуються як восьмирозрядні, так і шістнадцятирозрядні таймери-лічильники (ТС). Їх кількість залежить від типу МК і змінюється від двох до чотирьох. Точна кількість ТС для конкретного типу МК серії AVR можна визначити в довідкових даних (у графі таблиці «Таймери 8/16 біт»).

3.4.2. Початкові відомості про восьмибітові таймери-лічильники МК

Восьмибітовий ТС, у першому наближенні, є восьмирозрядний рахунковий модуль з схемою управління, який під час вступу на його вхід електричних імпульсів змінює у бік збільшення вміст рахункового регістра, і у вигляді двійкового коду зберігає у собі поточне значення вмісту. Рахунок імпульсів проводиться у двійковому коді, тому максимальне значення імпульсів, яке може підрахувати лічильник без переповнювання, складає $N = 2^8 = 256$ Щоб ТС почав працювати в режимі рахунку, на його вхід повинні поступати рахункові імпульси. Після приходу кожного такого імпульсу вміст рахункового регістра збільшується на одиницю. Рахунковими електричними імпульсами можуть служити як спеціальні тактові імпульси, що виробляються усередині самого МК, так і зовнішні імпульси, що поступають на спеціальні входи мікросхеми МК. При переповнюванні рахункового регістра його вміст обнуляється, і рахунок починається спочатку. У МК AVR є можливість записати в рахунковий регістр число у будь-який момент часу, а також прочитати вміст рахункового регістра. При цьому з погляду програміста рахунковий регістр є програмно-доступним елементом, який «розташовується» у просторі введення – виведення ОЗП і має свою адресу. Рахунковий регістр – це восьмирозрядний лічильник з переповнюванням, на вході якого може бути програмно включений ще один лічильник за допомогою якого можна встановити необхідний коефіцієнт ділення. Помітимо, що 16-розрядний ТС мають шістнадцятирозрядний рахунковий модуль. При цьому відмінність їх у тому, що кожен 16-розрядний рахунковий регістр МК фактично представляє два регістри, призначені для зберігання двох байтів. Один з них призна-

чений для зберігання старших бітів (TH) числа, а другий – для зберігання молодших бітів (TL) вмісту. Всі ТС мають своє «ім'я з номером».

Залежно від програмних настройок ТС може використовувати одне з наступних джерел тактування: а) імпульсну послідовність з частотою внутрішнього тактового генератора f_{BUS}; б) імпульсну послідовність з частотою, визначуваною, діленою на коефіцієнт, встановлений у керованому переддільнику частоти ($f_{BUS}/8, f_{BUS}/64, f_{BUS}/256, f_{BUS}/1024$); в) зовнішню імпульсну послідовність, що поступає на один з входів МК. У випадку а) і б) говорять, що ТС працює у режимі таймера, а у випадку в) – у режимі лічильника подій. У ТС є схема вибору джерела тактового сигналу, яка пропускає тактові імпульси вибраного джерела на вхід ТС; кожен рахунковий імпульс збільшує (зменшує) значення рахункового регістра. Використання переддільника дозволяє управляти часовим масштабом проходження імпульсів і максимальним значенням формованого (вимірюваного) інтервалу. Чим більший коефіцієнт ділення переддільника, тим більше максимальне значення формованого (вимірюваного) часового інтервалу $\Delta t_X = m/F_{ОБР} = k_{ДЕЛ} m/f_{BUS}$. При роботі ТС від зовнішнього сигналу останній синхронізується з сигналом тактового генератора. При цьому мінімальний період зовнішнього сигналу повинен бути більше періоду тактового генератора.

Восьмирозрядний ТС, іменований T/C0 (Timer/Counter 0), присутній у всіх МК базової серії сімейства AVR. У спрощеному вигляді, типова структурна схема T/C0, показана на рис. 3.6.

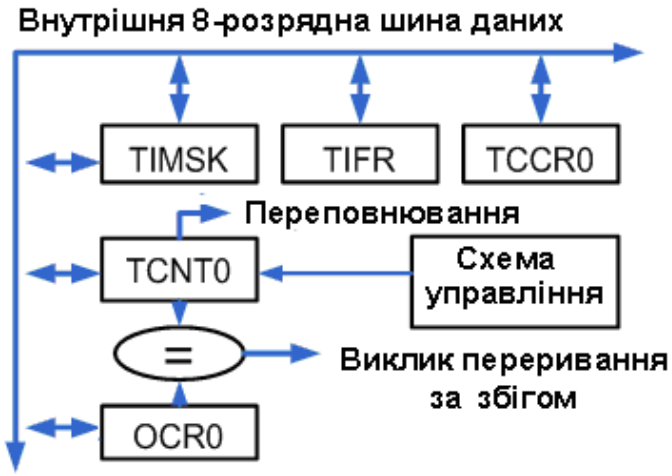


Рис. 3.6. **Типова структурна схема модуля T/C0**

Основний функціональний вузол ТС, – восьмирозрядний рахунковий регістр TCNT0, – реалізований як наростаючий лічильник з переповнюванням і з можливістю читання і запису. Під час його роботи використовуються наступні сигнали: clear – установка всіх бітів TCNT0 у нуль (очищення лічильника); count – функціонування у режимі, коли вміст лічильника збільшується або зменшується на 1. При досягненні рахунковим регістром TCNT0 максимального значення виникає внутрішній сигнал top, і, відповідно до режиму роботи, встановлюється прапор переповнювання TOV0 (Timer/Counter0 Overflow Flag – прапор переривання по переповнюванню ТС №0). Прапор TOV0 може бути використаний для сповіщення процесора про виникнення переривання з переповнювання рахункового регістра. Процесорний пристрій може «звертатися» до значення регістра TCNT0. При цьому команда запису, що поступає від процесора, має пріоритет над всіма іншими операціями (очищення лічильника і операціями рахунку). Роботою ТС№0 управляють апаратно, за допомогою схеми управління, і програмно за допомогою декількох регістрів, які програмно доступні і розташовані в просторі введення-виведення ОЗП. Таймер 0 використовує також регістри TCCRO і TCNTO з файлу регістрів введення-виведення. Прапор TOV0 при виконанні умов, вказаних вище, викликає вектор переривання TIM0_OVF з адресою $009.

Регістр TCCR0 управляє роботою таймера. Біти регістра управління TCR0 (TCCR0 – Timer/Counter 0 Control Register) управляють коефіцієнтом попереднього дільника і вибором тактового сигналу для ТС№0. Регістр управління TCCR0 у базовій серії МК AVR використовує тільки біти 0 – 2. Тільки вони доступні для читання і запису. Решта розрядів зарезервована і їх можна тільки прочитувати (завжди містять логічний нуль). У момент запуску МК, після надходження сигналу скидання у регістр управління TCCR0 заноситься число $00. Коефіцієнт ділення встановлюється записом у регістр TCCR0 однієї з наведених нижче в табл.3.2. комбінацій бітів CS, що управляють (Clock Select – вибір біта установки коефіцієнта ділення переддільника).

При формуванні часових інтервалів використовується регістр збігу ТС №0 (OCR0 – Output Compare Register 0). При його допомозі здійснюється функція порівняння за виходом. У нього записується число, з яким постійно порівнюється вміст рахункового регістра TCNT0. Кожного разу, коли вміст OCR0 виявляється рівним вмісту регістра TCNT0, тобто при їх збігу, компаратором генерується сигнал збігу. Цей сигнал встановлює

відповідний прапор OCF0 (Output Compare Flag 0 – прапор переривання за збігом ТС). Якщо відповідне переривання дозволене, установка прапора OCF0 збіги викликає переривання за збігом ТС№0. При запуску процедури обробки переривання прапор збігу автоматично скидається. Прапор може бути «очищений» програмно шляхом запису логічного 0.

Таблиця 3.2.

Комбінації бітів CS, що управляють

CS02	CS01	CS00	Опис
0	0	0	Таймер/лічильник зупинений
0	0	1	Тактова частота рівна f_{TAKT}
0	1	0	$f_{TAKT}/8$ (від переддільника)
0	1	1	$f_{TAKT}/64$ (від переддільника)
1	0	0	$f_{TAKT}/256$ (від переддільника)
1	0	1	$f_{TAKT}/1024$ (від переддільника)
1	1	0	До виводу Т0 підключене зовнішнє джерело імпульсів. Синхронізація роботи здійснюється за заднім фронтом імпульсів
1	1	1	До виводу Т0 підключене зовнішнє джерело імпульсів. Синхронізація роботи здійснюється за переднім фронтом імпульсів

Для формування широтно-імпульсно модульованих (ШІМ) сигналів у деяких типах МК AVR (з підтримкою режиму ШІМ) використовують два регістри збігу OCR0A і OCR0B. У цьому випадку вміст рахункового регістра TCNT0 безперервно порівнюється з вмістом кожного з двох регістрів збігу (OCR0A і OCR0B).

Дозволом переривань від ТС управляє регістр масок переривань TIMSK (Timer/Counter Interrupt Mask Register). Регістри управління TIMSK в МК AVR можуть, залежно від конструкції, використовувати різну кількість розрядів. У МК ATmega16 використовуються всі 8 розрядів регістра. Ці розряди доступні для читання і запису. У МК, наприклад, ATtiny2313, четвертий біт зарезервований і його можна тільки прочитувати. У момент запуску МК після надходження сигналу скидання у регістр управління

TIMSK заноситься число $00. Зміст розрядів регістра масок переривань від ТС для МК ATmega16 представлено в табл. 3.3.

Таблиця 3.3.

Зміст розрядів регістра масок переривань від ТС для МК ATmega16

Біт	7	6	5	4	3	2	1	0
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

Розряди регістра TIMSK характеризують наступне. Біт 0 – TOIE0 (Timer/Counter0 Overflow Interrupt Enable) – біт дозволу переривання з переповнювання ТС №0. Біт 1 – OCIE0 (Timer/Counter0 Output Compare Match Interrupt Enable) – біт дозволу переривання за збігом ТС №0. Біт 2 – TOIE1(T/C1 Overflow Interrupt Enable) – біт дозволу переривання з переповнювання ТС №1. Біт 3 – OCIE1B(T/C1 Output Compare B Match Interrupt Enable) – біт дозволу переривання за збігом В ТС №1. Біт 4 – OCIE1A (T/C0 Output Compare Match A Interrupt Enable) – біт дозволу переривання за збігом А ТС №1. Біт 5 – TICIE1(Timer/Counter1, Input Capture Interrupt Enable) – біт дозволу переривання за входом захоплення ТС №1. Біт 6 – TOIE2(Timer/Counter2 Overflow Interrupt Enable) біт дозволу переривання з переповнювання ТС №2. Біт 7 – OCIE2 (Timer/Counter2 Output Compare Match Interrupt Enable) – біт дозволу переривання за збігом А ТС №2. Біти регістра прапорів переривань від ТС TIFR (Timer/Counter Interrupt Flag Register) указують на виникле переривання.

Таблиця 3.4.

Зміст розрядів регістра TIFR

Бит	7	6	5	4	3	2	1	0
	OCF 2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

Розряди регістра TIFR характеризують наступне (табл. 3.4). Біт 0 – TOV0 (Timer/Counter0 Overflow Flag) – прапор переривання з переповнювання ТС №0. Біт 1 – OCF0 (Output Compare Flag 0) – прапор перериван-

ня за збігом ТС №0. Біт 2 – TOV1 (Timer/Counter1 Overflow Flag)) – прапор переривання з переповнювання ТС №1. Біт 3 – OCF1B (Timer/Counter1, Output Compare B Match Flag) – прапор переривання за збігом В ТС №1. Біт 4 – OCF1A (Timer/Counter1, Output Compare A Match Flag)) – прапор переривання за збігом А ТС №1. Біт 5 – ICF1 (Timer/Counter1, Input Capture Flag) – прапор переривання за входом захоплення ТС №1. Біт 6 – TOV2 (Timer/Counter2 Overflow Flag) – прапор перериванняз переповнювання ТС №2. Біт 7 – OCF2 (Output Compare Flag 2) – прапор переривання за збігом ТС №2. Помітимо, що в МК ATmega16 ТС №2 практично у всьому є аналогом ТС №0, за винятком того, що ТС №2 оснащений своїм власним попереднім дільником і він може працювати асихронно від внутрішнього тактового генератора, оптимізованого під кварцевий резонатор на частоту 32768 Гц (це дозволяє використовувати ТС №2 як годинник реального часу – RTC). При цьому, звичайно, регістри управління, збігу ТС №2, рахунковий регістр і інші програмно доступні елементи розміщені по інших, ніж у ТС №0, адресам у просторі введення – виведення ОЗП. Переривання ТС №2 також управляються регістром масок переривань TIMSK від ТС, а їх виникнення відображається бітами регістра прапорів переривань TIFR від ТС.

3.4.2. Початкові відомості про шістнадцятирозрядні таймери/лічильники

Шістнадцятирозрядний таймер/лічильник (ТС №1) має 16-розрядну структуру і призначений для формування часових інтервалів, генерування періодичних електричних сигналів, створення імпульсів заданої тривалості з високою точністю. Рахунок імпульсів ТС №1 проводиться у двійковому коді, тому максимальне число імпульсів, яке він може «підрахувати» складає $N = 2^{16} = 65536$. Окремі його вузли функціонують подібно до того, як це описано для ТС №0. ТС№1 може тактуватися безпосередньо сигналом внутрішнього тактового генератора, або імпульсами, одержаними після дільника, або імпульсним сигналом, одержаним із зовнішнього виводу. Всі сигнали запиту на переривання відображаються у регістрі прапорів переривань TIFR. Всі переривання ТС №1 можуть бути індивідуально замасковані за допомогою регістра масок переривань TIMSK. ТС №1 може бути засынхронизований, як від внутрішнього тактового сигналу через попереднього дільника, так і від зовнішнього тактово-

го електричного сигналу, що поступає на вхід T1. При зовнішньому тактуванні ТС №1 зовнішній сигнал «прив'язується» до частоти процесора. Для правильної роботи ТС №1 від зовнішнього тактового сигналу мінімальний час між двома перемиканнями цього сигналу повинен бути не менше одного періоду тактового сигналу. Якнайкращі точність і дозвіл ТС№1 забезпечує при найменшому коефіцієнті попереднього ділення. З іншого боку, високий коефіцієнт попереднього ділення зручний при реалізації низькошвидкісних функцій або синхронізації подій, що рідко відбуваються. У ТС №1 є блок управління логікою вибору тактового сигналу. При цьому, як і раніше, якщо не одне джерело тактового сигналу не вибране, то ТС №1 зупиняється. ТС №1 підтримує дві функції порівняння, використовуючи регістри OCR1A, OCR1B. Функції порівняння включають опції очищення лічильника за збігом порівняння А і зміни стану на зовнішніх виводах при збігах порівняння А і В. ТС №1 може бути використаний як 8-, 9- або 10-розрядний широтно-імпульсний модулятор. У цьому режимі рахунковий регістр і регістри OCR1A, OCR1B працюють як здвоєний самостійний ШІМ з центрованими імпульсами.

Регістри управління ТС №1 (TCCR1A, TCCR1B) восьмирозрядні і доступ до них проводиться через програмний простір введення – виведення ОЗП, аналогічно тому, як було описано раніше. Біти регістра управління ТС1 (TCCR1 – Timer/Counter 1 Control Register) управляють поведінкою виходів сигнала збігу (OC1A і OC1B) і визначають режими генератора сигнала (послідовністю підрахунку імпульсів рахунковим регістром, максимальна межа рахунку (TOP) і спосіб генерації сигналу). У момент запуску МК, після надходження сигналу скидання у регістри управління ТС №1 (TCCR1A, TCCR1B) заноситься число $00. Вміст регістра А управління таймером/лічильником 1 представлене в табл. 3.5.

Таблиця 3.5.

Вміст регістра А управління таймером/лічильником 1

Бит	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0			WGM11	WGM10

Біти COM1A1 і COM1A0 управляють поведінкою виводів (OC1A і OC1B) і визначають характер сигналу на виході після збігу при порівнян-

ні. Щоб вихідний сигнал поступав на вихід OC1A, у регістрі управління повинен бути встановлений в 1 відповідний біт управління. Розряди регістра 2, 3 регістри управління (TCCR1A, TCCR1B) зарезервовані. Коефіцієнт ділення встановлюється записом в регістр TCCR1B однієї з наведених у табл. 3.6. комбінацій бітів CS управління (Clock Select – вибір біта установки коефіцієнта ділення переддільника).

Таблиця 3.6.

Комбінації бітів CS управління

CS02	CS01	CS00	Опис
0	0	0	Таймер/лічильник зупинений
0	0	1	Тактова частота рівна $f_{ТАКТ}$
0	1	0	$f_{ТАКТ}$ / 8 (від переддільника)
0	1	1	$f_{ТАКТ}$ / 64 (від переддільника)
1	0	0	$f_{ТАКТ}$ / 256 (від переддільника)
1	0	1	$f_{ТАКТ}$ / 1024 (від переддільника)
1	1	0	До T0 підключене зовнішнє джерело імпульсів. Синхронізація – за заднім фронтом імпульсів
1	1	1	До T0 підключене зовнішнє джерело імпульсів. Синхронізація – за переднім фронтом імпульсів

При роботі з ТС №1 необхідно враховувати ряд його специфічних особливостей. Перша особливість полягає в тому, що рахунковий регістр (TCNT1), регістри порівняння (OCR1A, OCR1B) і регістр захоплення (ICR1) – це 16-розрядні регістри з так званою подвійною буферизацією, які при доступі до них, як до програмно-доступних елементів простору введення – виведення, вимагають дотримання певної послідовності (процедури). Це пов'язано з тим, що центральний процесорний пристрій може звертатися до таких 16-розрядних регістрів тільки за допомогою 8-розрядної шини даних. Тому доступ до кожного 16-розрядного регістра може бути не чим іншим як послідовним читанням або записом двох (молодшого і старшого) байтів інформації. Щоб виконати побайтне безпомилкове виконання операцій із старшим і молодшим байтом, як говорять «на льоту», 16-розрядний ТС №1 має додатковий 8-розрядний ре-

гістр для часового зберігання старшого байта. Він не доступний програмісту і функціонує в автоматичному режимі, здійснюючи доступ до кожного 16-розрядного регістра послідовним читанням або записуванням байтів інформації. При читанні або записі старшого байта інформація насправді заноситься в тимчасовий регістр або читається з чосового регістра. Читання/запис всіх шістнадцяти розрядів відбуваються у момент читання/запису його молодшого байта. Тому при написанні програми слід дотримуватися наступних правил, які називають «правилами подвійної буферизації». Щоб провести повний запис числа в 16-розрядний регістр, необхідно спочатку записати старший байт, а потім молодший. При читанні 16-розрядного регістра спочатку потрібно зчитати молодший байт, а потім старший. Правила подвійної буферизації дозволяють центральному процесорному пристрою, використовуючи восьмирозрядну шину даних, читати або записувати повне 16-розрядне число за один такт.

Основним функціональним вузлом ТС №1 є 16-розрядний рахунковий регістр TCNT1. Структурна схема рахункового регістра TCNT1 з системою управління показана на рис. 3.7.

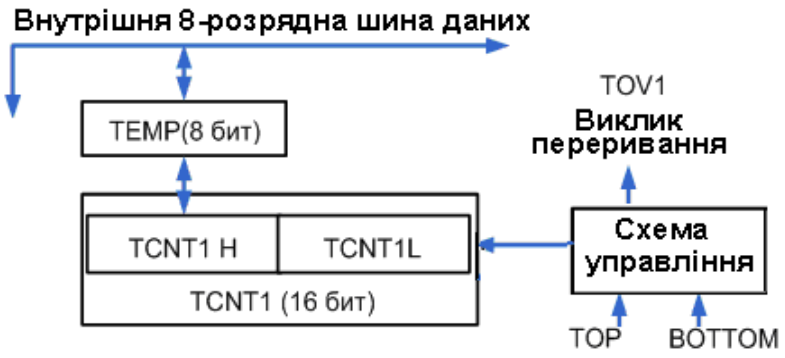


Рис. 3.7. **Структурна схема регістра TCNT1 з системою управління**

Він реалізований як 16-розрядний реверсивний лічильник з переповнюванням і з можливістю читання і запису. Після переповнювання рахункового регістра TCNT1 його робота продовжується. При цьому послідовність зміни кодів наступна $FFFF, $0000, $0001. Під управлінням програми можуть бути виконані пуск і зупинка рахункового регістра TCNT1, установка старшого і молодшого байта в необхідний стан, прочитування поточного коду лічильника. Для управління його роботою використовуються наступні сигнали: clear – установка всіх бітів TCNT1 у нуль (очи-

щення лічильника); count – функціонування у режимі, коли вміст лічильника збільшується або зменшується на 1. При досягненні рахунковим регістром TCNT1 максимального значення виникає внутрішній сигнал top, і відповідно до режиму роботи, встановлюється прапор переповнювання TOV1 (Timer/Counter0 Overflow Flag – флаг переривання з переповнювання ТС). Прапор TOV1 може бути використаний для сповіщення процесора про виникнення переривання з переповнювання рахункового регістра. Коли TCNT1 досягає мінімального значення (нуля) активується сигнал Bottom.

16-розрядний рахунковий регістр відображений у просторі введення – виведення як два 8-розрядні регістри: регістра старшого байта TCNT1H рахункового регістра, який працює з 8 старшими бітами; регістра молодшого байта TCNT1L рахункового регістра, який містить 8 молодших бітів. Регістр часового зберігання Temp зберігає вміст TCNT1H в той момент, коли відбувається читання регістра TCNT1L. І, навпаки, в регістр TCNT1H записується вміст регістра Temp в той момент, коли відбувається запис регістра TCNT1L. Таке функціонування регістрів дозволяє центральному процесору читати або записувати 16 – розрядне число за один такт, використовуючи восьмирозрядну шину даних.

При формуванні часових інтервалів використовується функціональний вузол (модуль) збігу. Його показана структурна схема на рис. 3.8.

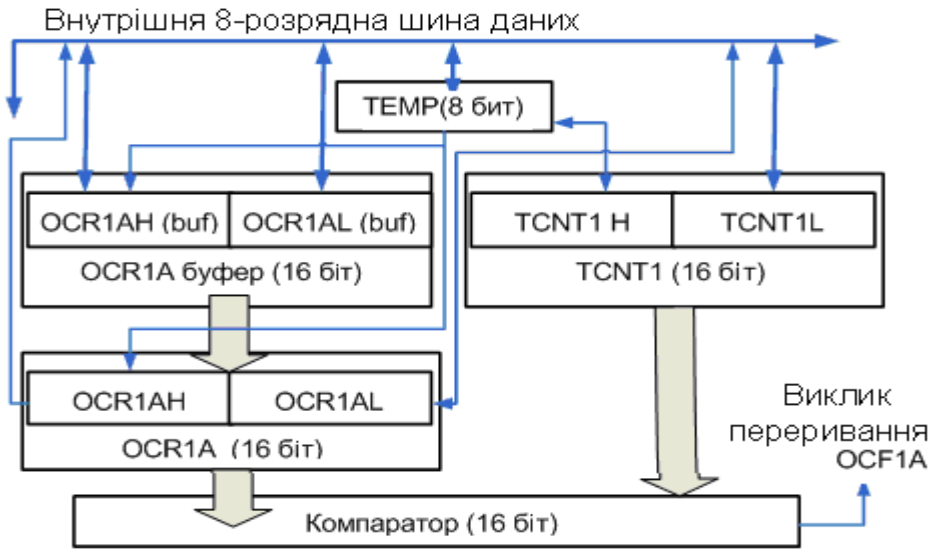


Рис. 3.8. **Структурна схема модуля збігу**

У программно-логічній моделі модуля передбачені засоби для забезпечення режиму буферированного порівняння при якому 16-розрядний регістр збігу (OCR1AH OCR1AL) дублюється, із-за чого в кожен момент до входу компаратора виявляється підключеним один з регістрів даних, а для запису виявляється доступним іншій. У момент настання події вихідного порівняння регістри автоматично міняються місцями.

За допомогою даного модуля здійснюється функція порівняння за виходом (вміст рахункового шістнадцятиричного регістра TCNT1 безперервно порівнюється з числом, записаним у регістр збігу, наприклад OCR1A). Кожного разу, коли вміст OCR1A виявляється рівним вмісту регістра TCNT1, тобто при їх збігу, компаратором генерується сигнал збігу. Цей сигнал у наступному циклі роботи таймера встановлює відповідний прапор OCF1A (Output Compare Flag 1A – прапор переривання за збігом ТС №1). Якщо відповідне переривання дозволене (OCIE1A = 1), установка прапора OCF1A збігу викликає переривання за збігом ТС1. При запуску процедури обробки переривання прапор збігу автоматично скидається. Прапор також може бути «очищений» програмно шляхом запису логічної одиниці. Для того, щоб, не дивлячись на внутрішню 8-розрядну шину даних, в 16-бітовий регістр OCR1A або OCR1B можна було завантажити одночасно обидва байти, використовується внутрішній регістр Temp. Спочатку в регістр OCR1AH (OCR1BH) повинен бути записаний старший байт у допоміжний регістр Temp для проміжного зберігання. Услід за цим за допомогою програми записують у регістр OCR1AL (OCR1BL) молодший байт, який внутрішньо об'єднується з байтом в регістрі Temp. Після чого обидва байти одночасно записуються в 16-розрядний регістр OCR1A (OCR1B). Регістр збігу OCR1A, крім того, має подвійну буферизацію збігу, яка може бути відключена. Коли подвійна буферизація дозволена, центральний процесор звертається до регістра OCR1A через буфер. Подвійна буферизація синхронізує момент оновлення регістра OCR1A з моментом досягнення таймером верхньої або нижньої меж (top і Bottom). Це необхідно для формування ШІМ-сигналів. Можливість програмно задавати верхнє значення (top) ТС №1 дозволяє задавати період вихідного сигналу і коефіцієнт його перерахунку. Вміст регістра OCR1A (з буфером або без буфера) змінюється тільки в процесі виконання команди запису. Тому при читанні старшого байта регістра OCR1A не використовується буфер часового зберігання Temp. Проте буде правильно, якщо, як і при доступі до будь-якого 16-розрядного регі-

стра, на підставі правила подвійної буферизації, спочатку буде прочитаний молодший байт, а потім старший. Запис у регістр OCR1A проводиться через буфер Temp, оскільки порівняння всіх 16 бітів повинне проводиться безперервно. Тому старший байт OCR1AH повинен бути записаний першим. Коли центральний процесор проводить запис за адресою, де знаходиться регістр старшого байта, насправді ця інформація записується у регістр Temp. Коли ж відбувається запис молодшого байта OCR1AL, одночасно старший байт буде скопійований у верхні 8 бітів буфера OCR1AH з регістра Temp в тому ж самому циклі системного генератора.

Контрольні запитання

1. Поясніть і стисло охарактеризуйте типову структурну схему мікроконтролера AVR.
2. Назвіть і охарактеризуйте склад і особливості мікроконтролерів серії AVR.
3. Охарактеризуйте поняття: внутрішня пам'ять, пам'ять програм, пам'ять даних.
4. Енергозалежні види пам'яті.
5. Охарактеризуйте особливості портів вводу – виводу.
6. Що таке лічильник команд і стекова пам'ять?
7. Що таке таймер? Назвіть і поясніть основні режими роботи таймерів.
8. Стисло охарактеризуйте вбудовані периферійні пристрої.
9. Моніторинг напруг живлення у системах з мікроконтролерами.
10. Апаратні і програмні рішення з підвищення надійності роботи МК.
11. Поясніть і стисло охарактеризуйте основні режими роботи портів вводу – виводу.
12. Оперативні пристрої пам'яті МК і система адресації регістрів.
13. Графічне зображення адресного простору ОЗП МК.
14. Наведіть відомості про підсистему переривань МК AVR.
15. Охарактеризуйте загальний алгоритм роботи системи переривань.
16. Поняття програмно-логічної моделі модуля.
17. Як виконується організація вкладених переривань і підпрограм?

Модуль 2. Програмне забезпечення мікропроцесорної техніки, вживаної в поліграфічному обладнанні

Тема 4. Початкові відомості про програмне забезпечення мікроконтролерів

4.1. Загальні відомості про прикладне програмне забезпечення комп'ютеризованого поліграфічного обладнання

4.1.1. Необхідність освоєння програмного забезпечення комп'ютеризованого поліграфічного обладнання

Вже відзначалося, що сучасне комп'ютеризоване обладнання є «симбіоз» апаратних засобів (hardware) і програмного забезпечення (software), які тісно взаємозв'язані і даремні один без одного. Через це при створенні обладнання завжди виникає завдання про правильний розподіл переваг між апаратурними засобами і програмним забезпеченням. Якщо виходити з того, що використання апаратури забезпечує високу швидкодію обладнання, але при цьому зв'язано зі збільшенням вартості і споживаної потужності, а велика питома вага програмного забезпечення допускає модернізацію обладнання простими засобами (шляхом перепрограмування), дозволяє скоротити число компонентів системи і її вартість, то досить часто перевага віддається програмним засобам. У даний час має місце тенденція переміщення основного об'єму витрат з області розробки апаратури в сферу розробки програмного забезпечення. Причому в сферу програм для цілей управління технологічним обладнанням, для збору інформації про хід виробничого процесу, які часто називають прикладними (цільовими, призначеними для користувача) програмами. При цьому, як свідчить досвід, виникає дві проблеми. 1. Фахівець з обладнання часто виявляється непідготовленим до того, що в даний час часто перевага віддається програмному забезпеченню. У нього відсутні прості знання і уміння в області сучасних IT-технологій, він не може освоїти ті програмні засоби, за рахунок яких

реалізуються основні функції обладнання. Такий фахівець не розуміє логіку роботи сучасного обладнання, оскільки та її частина, яка описує послідовність дій, звичайно «прихована» в програмі. Такий представник обслуговуючого персоналу не здатний швидко виявити несправність, оскільки він не може контролювати функціонування комп'ютерної програми під час її виконання. 2. Якщо для обслуговування обладнання привертають програмістів високої кваліфікації, то виявляється, що фахівець, що володіє «таємницями ремесла» в конкретній наочній області, не завжди може викласти програмісту сенс прикладного завдання, що вирішується якимсь вузлом обладнання. У результаті програміст виявляється не у змозі допомогти усунути несправність в обладнанні, а в разі модернізації обладнання програмуються найбільш очевидні, найбільш тривіальні прикладні завдання. Найцікавіші професійні завдання, що дозволяють істотним чином поліпшити роботу комп'ютеризованого обладнання, залишаються за межами досяжності. До того ж, часто фахівець з обладнання так формулює завдання, що прикладні програми виявляються непрацездатними, причому не із-за помилки в кодах, не із-за помилок у програмах, а із-за не вірної формалізації прикладного завдання. Трудомісткість усунення цих помилок, створених парою «професійний програміст – непрограмуючий професіонал» така велика, що часто доводиться приступати до розробки програмного забезпечення наново і іншими засобами. Крім того, помилка програмування може, кінець кінцем, обійтися дуже дорого. Якщо погано уявляєш собі програму функціонування того або іншого об'єкта, то це може бути чревато травмами або навіть фатальним результатом.

Справжній фахівець з комп'ютеризованого обладнання може сам розробляти прикладне програмне забезпечення і відладжувати свої програми. Тому в нинішніх умовах для проектування і експлуатації сучасного комп'ютеризованого обладнання необхідний тандем «професійний програміст – професіонал, що має певні знання і уміння в області програмування». Тому не випадково, набуває поширення тенденція, коли супровід програм, їх розробка, особливо в умовах стислих термінів модернізації виробу, виконується силами «програмуючих» професіоналів в області обладнання. Як виявляється, в таких випадках, хоча прикладні програми, створені професіоналом з обладнання, з погляду професійного програміста виглядають незграбними і нетраціозними, та зате вони володіють безперечною гідністю – вони дійсно працюють, що не можна

сказати про дев'ять з кожних десяти «витончених» програм, створених професійним програмістом (який за визначенням не може бути професіоналом у конкретній області знань).

З урахуванням сказаного і перспектив застосування комп'ютеризованого обладнання представляється важливим, щоб сучасні фахівці, що працюють у своїх предметних областях знань, освоїли «кухню» програмної реалізації завдань, що найбільш часто зустрічаються, для об'єктів автоматизації обладнання. Це дозволить їм добітися якісної роботи комп'ютеризованого обладнання, швидко знаходити несправності, а в необхідних випадках, одержати необхідну підтримку професійних програмістів при рішенні актуальних задач. До подібної постановки питання організації розробки і супроводу прикладного програмного забезпечення приводить і очевидне міркування, що швидке зростання числа мікроконтролерів, що вбудовуються в комп'ютеризоване обладнання, не може супроводжуватися відповідним зростанням числа програмістів.

4.1.2. Аналіз предметної області – перший етап у створенні прикладного програмного забезпечення комп'ютеризованого поліграфічного обладнання

При створенні прикладного програмного забезпечення для комп'ютеризованого обладнання можна виділити три стадії: 1. Аналіз предметної області з метою вироблення базової концепції системи управління, визначення завдань, які можуть бути в комп'ютеризованому обладнанні ефективно вирішені при використанні МК. 2. Розробка алгоритму рішення поставленої задачі (комплексу завдань). 3. Властиво програмування, або точніше, супровід розробки прикладних програм. Розподіл трудовитрат у відсотках за цими трьома стадіям виглядає приблизно так: 40 – 50 – 10. Це означає, що якщо перша стадія роботи виконана, тобто якщо завдання поставлене, то найбільш складною стадією роботи, що слабо формалізується і трудомісткою, є стадія аналізу завдання, її інженерна інтерпретація і розробка «функціональної специфікації» програми для формування алгоритму рішення поставленої задачі. Очевидно, що основне творче навантаження при розробці прикладного програмного забезпечення несе не професійний програміст, а фахівець в даній області знань. Якщо до того ж цей фахівець володіє ос-

новами програмування, то можна чекати, що процес формалізації його професійних знань протікатиме набагато результативнише, чим якби це робив програміст.

Першим кроком на шляху створення системи управління об'єктами обладнання і технологічними процесами є уточнення цілей, а також завдань, які система повинна вирішувати. Вироблення базової концепції (загального уявлення, первинної ідеї, що визначає принцип дії і метод реалізації) побудови обладнання починається з того, що приймається рішення про те, яка створюється система управління і яку участь в ній бере людина-оператор. На цьому етапі звичайно аналізується проблема, що підлягає рішенню. Творці обладнання і творці програмного забезпечення разом повинні визначити, які необхідні засоби управління і які повинні здійснюватися дії управління.

У всіх реальних технологічних процесах використовується базова концепція системи управління, коли при виконанні автоматичного управління окремими вузлами і модулями обладнання у системі працює людина, яка стежить за ходом процесу і бере на себе управління при виникненні ситуацій, не передбачених програмою. Система при цьому значно ускладнюється, оскільки потрібні пристрої індикації, управління, модифікації програм і ін.

Крім планування послідовності процедур вимірювання, відображення інформації, вироблення управлінських рішень, вироблення базової концепції включає і з'ясування того, які види сигналів повинні бути використані в системі при введенні і виведенні інформації. Потрібно також заздалегідь визначити в якій формі, у безперервній (аналогової), або дискретній (цифровій) повинні бути представлені параметри технологічних процесів. На стадії вироблення базової концепції потрібно вирішити і питання, в якій саме мірі в системі управління використовуватиметься МК. Однією з простих форм використання МК є застосування його як системи збору інформації. Параметри виробничого процесу, що в цьому випадку цікавлять технолога, перетворяться у цифрову форму, сприймаються підсистемою введення даних МК. Сам МК проводить обробку інформації, що поступила, і результати цього виводяться на екран дисплея у формі, зручній для сприйняття людиною-оператором. Збір даних для аналізу виробничих ситуацій завжди включається як одне із завдань навіть у найдосконаліших системах автоматичного управління з використанням ЕОМ. Іншою формою використання МК є його робота в

режимі порадника оператора. Як правило, в цьому випадку виводи МК не пов'язані з органами управління технологічними агрегатами, а приєднані до засобів відображення інформації. Разом з тим, МК за закладеною в нього програмою обчислює управляючи дії, необхідні для наближення режиму роботи обладнання до оптимального. Ці рекомендації виводяться на засоби індикації. Управляє системою оператор. Він ухвалює рішення, або враховуючи рекомендації, які йому дає МК, або ігноруючи їх, і, на основі власного досвіду і інтуїції, управляє технологічними агрегатами. У таких ситуаціях говорять, що МК працює в розімкненому контурі управління. Він виконує прості функції управління.

Наступною за складності застосування МК в системі управління є форма супервізорного управління. При цьому МК працює в замкнутому контурі, тобто виводи системи управління пов'язані з технологічними агрегатами і МК впливає на виконавчі механізми. У цьому випадку після визначення МК дій управління, вони перетворюються в сигнали, які впливають на керований об'єкт. Оскільки контур управління замкнутий, функції оператора зводяться до спостереження і його втручання потрібне лише при виникненні відмов у системі або непередбачених ситуацій. Нарешті, формою застосування МК у системі управління може бути безпосереднє цифрове управління. У цьому випадку сигнали, використовувані для приведення в дію виконавських органів, поступають безпосередньо від МК. При цьому МК розраховує реальні дії і передає відповідні сигнали безпосередньо на органи управління. Це робіться для кожного органу управління. Очевидно, що така система обов'язково повинна працювати в реальному масштабі часу, тому часові затримки в кожному контурі управління потрібно ретельно аналізувати.

Наступним кроком після вироблення базових концепцій є визначення конфігурації системи, її основних модулів і того, які необхідно зробіти дії і яким способом, щоб сформувати вхідну і вихідну інформацію. У результаті виконання цього кроку повинні бути встановлені вимоги до МК, як первинному елементу апаратури, і в строго формалізованій формі представлені завдання на програму, включаючі повний і точний опис зовнішнього ефекту програми. Слід також вказати які у програми початкові дані і результати, обмеження на область її застосування.

На практиці нерідкі випадки, коли комп'ютеризовані системи обладнання є вельми складними. Для такого обладнання програміст

зобов'язаний розбіти довгу програму на безліч невеликих сегментів. Кращий спосіб реалізувати раціональне розбиття – це використовувати метод проектування зверху «вниз». При цьому система управління розбивається на підсистеми, які в свою чергу розбиваються на субсистеми і т. д., поки не будуть одержані фрагменти, що легко піддаються контролю. Одержана структура повинна бути включена в документацію на систему управління і використана як основа для програмування. Важливою перевагою даного методу є і те, що докладно деталізовану структуру системи з ясно певними сигналами можна розподілити для подальшої роботи між декількома програмістами.

Відзначимо, що використовувані в комп'ютеризованому обладнанні прикладні програми можуть бути автономними, тобто такими, які пишуться одним програмістом і використовуються окремо і незалежно від інших програм, і системними (програмними комплексами), тобто великими програмами, що складаються з ряду окремих взаємозв'язаних модулів. Автономні програми використовуються для управління якимнебудь відносно простим об'єктом, наприклад, стендом визначення параметрів фарби. Програмні комплекси відрізняються від автономних програм складністю завдань. Розробкою програмних комплексів займаються колективи програмістів. У результаті аналізу системи на різних рівнях, її деталізації і структуризації, застосування функціонально-топологічного методу, використання структурного підходу, програмний комплекс розбивається на модулі і кожен окремий програміст має справу з одним програмним модулем.

4.1.3. Розробка алгоритму рішення поставленої задачі – другий етап у створенні прикладного програмного забезпечення комп'ютеризованого поліграфічного обладнання

З моменту отримання завдання автор автономної програми або розробник програмного модуля приступають до вибору і розробки алгоритму (методу обчислень), який використовуватиметься в програмі. Алгоритм – це точно визначена процедура, приписуюча МК однозначно визначені дії з перетворення, так би мовити, «сирих» початкових даних у оброблені вихідні дані. Часто одне і те ж завдання може бути вирішене різними методами. Тому на мові алгоритмів розробник описує не тільки метод, вибраний для вирішення поставленого завдання, але і спосіб його

рішення, вибраний на етапі інженерної інтерпретації завдання. Для правильної алгоритмізації необхідно з'ясувати особливості кожного алгоритму (методу), застосування якого можливо в даному випадку, як він вирішує поставлену задачу. Від того якій вибраний алгоритм залежить не тільки якість прикладної програми, що розробляється, але і показники всієї комп'ютеризованої системи.

Алгоритм може бути достатньо простий і надзвичайно складний. При цьому треба враховувати, що навіть для щодо простої системи управління обладнанням або технологічним процесом відразу важко розробіти алгоритм, що охоплює всі «задуми і тонкість» системи. Тому секрет успіху розробки прикладної програми полягає у використанні методу декомпозиції, при якому все завдання послідовно розділяється на менші функціональні модулі, кожний з яких можна аналізувати, розробляти і відладжувати окремо від інших. Розділення завдання на підзадачі, декомпозиція складних алгоритмів на складові частини, кожна з яких виявиться відносно простою, виконується послідовно до такого рівня, коли розробка алгоритму стає простою справою.

Розробка алгоритму схожа на розробку функціональних схем. У основу його розробки покладена та ж сама процедура модульного проектування, яка традиційно використовується розробниками апаратурних засобів цифрової електроніки. Відмінність полягає в тому, що при розробці апаратурних засобів як «будівельний» матеріал використовуються функціональні вузли (логічні схеми, трігери, регістри і т. д), а при створенні програмного забезпечення розробник оперує командами, підпрограмами, таблицями й іншими програмними засобами з арсеналу обробки даних.

Для роботи з алгоритмами є важливим поняття виконавця. Виконавець – це якась абстракція (робот, здатний виконувати деякі команди, механічний або електронний пристрій і т. д.), що уміє виконувати деякий цілком визначений набір дій. З погляду програміста виконавець – це програми, що працюють над даними. При цьому дії, що виконуються виконавцем за тією або іншою командою, можуть залежати від передісторії і від обстановки, в якій поступила команда. Кожна дія, яку здатний виконати виконавець, називається розпорядженням, а вся сукупність таких дій – системою розпоряджень виконавця. Наказ на виконання дії, виражений формальним наперед фіксованим способом, називається викликом роз-

порядження. Словом «формальним» підкреслюється, що опис розпорядження повинен проводитися відповідно до цілком певних правил.

Зручно уявити собі виконавця у вигляді пристроїв з кнопками на передній панелі. Біля кожної кнопки при цьому написано ім'я розпорядження, яке буде виконане при натисненні на цю кнопку. Система розпоряджень такого виконавця співпадає з сукупністю кнопок на його передній панелі. Щоб написати її на папері, досить переписати імена, написані біля кнопок. Виклик розпорядження потрібно уявляти собі як натиснення на кнопку, після якого виконавець виконує відповідну дію. Всяка дія проводиться над деякими об'єктами і полягає в зміні стану цих об'єктів.

Командуючи виконавцями, ми досягаємо бажаного результату, причому не одним викликом якого-небудь розпорядження, а певною послідовністю викликів різних розпоряджень. Така послідовність (деякий формальний її опис) називається програмою. Щоб стало зрозуміліше, про що йде мова, наведемо приклад виконавця і алгоритму для нього. У окремому випадку виконавець уявляє собою вакуумну головку самонаклада з системами присосів — чотирма підйомними присосами, що забезпечують надійне відділяння тільки верхнього листа стапеля, і п'ятьма транспортними присосами, за допомогою яких проводиться транспортування листа в подаючі ролики. Головку можна піднімати, опускати, а також переміщати щодо листів паперу (управо, вліво, вгору, вниз).

Систему розпоряджень виконавця компактно можна записати в наступному вигляді: 1) почати роботу; 2) підняти головку вгору (так /ні); 3) виконати кроки вправо/влево; 4) опустити головку вниз (так /ні); 5) створити вакуум (так/нні); 6) підняти головку вгору (так /ні); 7) виконати кроки вправо/вліво; 8) опустити головку вниз (так /ні); 9) прибрати вакуум (так /ні); 10) закінчити роботу. Запис (так /ні) означає, що у розпорядження є один вихідний об'єкт і цей об'єкт може приймати один з двох станів: або «так», або «ні»

Для алгоритму є істотним спосіб його завдання. Найпростішим є запис алгоритму у вигляді набору висловів на природній, розмовній мові. Але всі розмовні мови володіють надмірністю і неоднозначністю, тому переважнішою є його графічна форма. Програміст обов'язково готує «схему програми» у вигляді прямокутників і ромбів, з’єднаних стрілками (схему алгоритму або, як ще говорять, блок-схему алгоритму). Сьогодні, коли програмування ведеться на мовах високого рівня, використання

подібних схем, видно, не має переваг перед програмою. Проте докладний геометричний опис алгоритмів, коли кожен оператор програми поміщений в рамку, є все ж таки вельми корисним, особливо при необхідності використання мов низького рівня. При графічному представленні алго-ритмів у вигляді блок-схеми доцільно дотримуватися наступних правил: 1) схема алгоритму повинна використовувати загальноприйняті позначення (позначення Держстандартів); 2) ступінь подробиці схеми повинен бути приблизно 1:5. Тобто блоку схеми повинне відповідати приблизно 5 операторів програми, хоча в логічних програмах схема опиняється дрібнішою, а в арифметичних крупнішою. На кожну підпрограму і головну програму складається окрема схема; 3) схема повинна бути читана без додаткових пояснень автора. Побудова алгоритму повинна виконуватися із строгого набору об'єктів, кожний з яких реалізує певну процедуру обробки даних. Модуль алгоритму повинен мати тільки одну «точку» входу і одну «точку» виходу. Використовувані у деякому блоці змінні повинні набувати значення в якомусь іншому або в тому ж блоці. Вхідні і вихідні блоки процедур модуля повинні містити вхідні і, відповідно, вихідні параметри; 4) блоки можна об'єднувати в крупніші блоки пунктирними лініями. Об'єднані блоки потрібно коментувати: описувати їх призначення. Запис у блоках повинен бути словесним або математичним, а не у вигляді операторів мови; 5) після складання програми над лівою частиною блоків ставляться мітки, якими в програмі помічені оператори, відповідні даному блоку. У розриві лівої частини верхньої лінії ставляться номери блоків. Основними функціональними блоками схеми алгоритму є наступні графічні символи (еліпси, прямокутники, ромби, стрілки) (рис. 4.1 а – г).

У алгоритма, як деякої послідовності дій або викликів розпоряджень, вироблюваних МК, щоб досягти необхідного результату, начало (рис. 4.1 а) і кінець (рис. 4.1 г) прийнято позначати еліпсом або прямокутником з округляючими боками. Прямокутником (рис. 4.1 б) позначений функціональний оператор, який здійснює безпосереднє перетворення інформації і визначає різні дії, що виконуються програмою. Усередині прямокутника на мові, близькій до природної, пишеться коротка характеристика тих розпоряджень, які здатний виконувати даний оператор (суть виконуваної дії). Послідовність виконання дій показується стрілками. Логічний оператор позначається ромбом, усередині якого також записується його характеристика (рис. 4.1 в). Такий оператор реалізує

операцію вибору і здійснює розгалуження програми. Логічний оператор має один вхід і два виходи, на лініях яких записується «так», коли умова виконана (умова істинно), і «ні», коли не виконується. Якщо умова істинно, то подальше виконання програми продовжиться по шляху, позначеному «так». Якщо умова не виконана, то програма піде по іншому шляху, позначеному стрілкою з написом «ні».

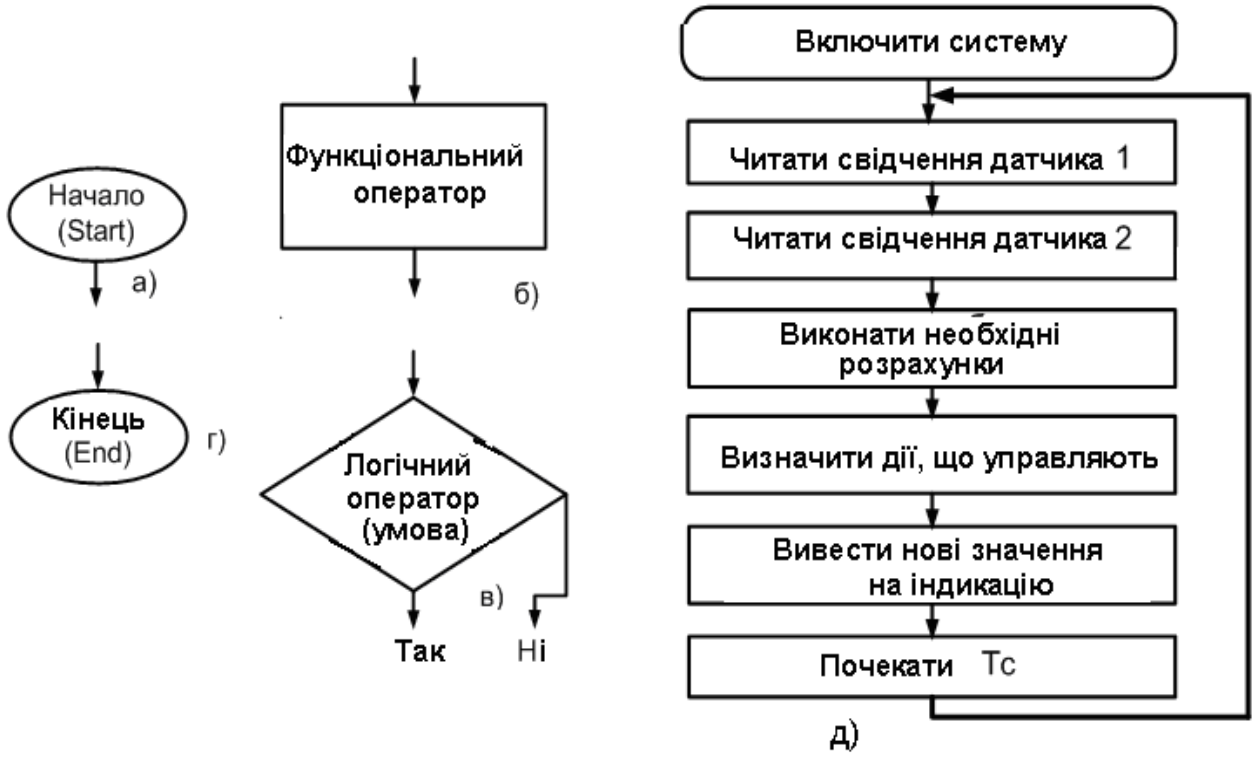


Рис. 4.1. **Зображення основних функціональних блоків алгоритмів (а – г) і схема алгоритму періодичного читання (збору) інформації з цифрових датчиків і відображення її на пристрої індикації (д)**

Для ілюстрації техніки розробки алгоритму на рис. 4.1 д показана схема алгоритму періодичного читання (збору) інформації з цифрових датчиків і відображення її на пристрої індикації. Вирішувана виконавцем задача полягала у тому, щоб «прочитати» бінарні показники датчиків у тому порядку, в якому вони слідують, і порівняти їх з відліками, зробленими на попередніх етапах. Якщо є відмінності, то вони повинні бути підраховані і виведені на пристрій індикації. На початку роботи, згідно алгоритму, прочитуються значення параметрів датчиків (наприклад, у

вигляді восьмирозрядних «слів») і проводиться їх порівняння з попередніми відліками. У системі необхідний пристрій, який би зберігав попередні дані. Потім проводиться порівняння попередніх і нових бінарних даних, і визначаються відмінності. Якщо відмінності відсутні, то вміст програмного лічильника збільшується, управляючи дії не виробляються, індикація не міняється. Далі слідує програмне повернення до початкової точки для проведення наступного циклу прочитування. Якщо ж у показниках датчиків є відмінності між даним результатом і попереднім відліком, то виконується дослідження, щоб визначити в якій групі розрядів розрізняються біти. На цій підставі виробляються дії управління для індикатора і на пристрій відображення інформації виводяться нові дані. Потім, як і раніше, слідує програмне повернення до початкової точки для проведення наступного циклу прочитування. Нижня частина алгоритму є розпорядження, що визначає періодичність опиту датчиків, інтервали часу, через які необхідно повторно фіксувати їх свідчення. Для формування часової затримки повинен бути розроблений спеціальний алгоритм (підпрограма).

Будь-яка прикладна програма управління обладнанням або технологічним процесом може бути побудована шляхом комбінації чотирьох базових модулів: присвоювання, функціональних (послідовних), циклу (повторення), розгалуження (альтернативного рішення). Використання цих чотирьох базових блоків алгоритмів дозволяє вводити дані, організовувати в програмі цикли, здійснювати галуження у програмах і реалізовувати підпрограми. Розглянемо семантику (сенс) блоків і їх роботу. Модуль присвоювання – це атом програми. З його допомогою в програму вводяться числа і символи. Послідовна структура розпоряджень, що виконується зверху вниз і показана на рис. 4.1 д, також широко поширена. Вона являє собою формальну програму дій і описує, з урахуванням всіх умов, які можуть скластися при виконанні програми, послідовність викликів розпоряджень, що виконуються один за одним.

У комп'ютеризованому обладнанні часто виникають ситуації, коли певні команди потрібно повторювати по кілька разів підряд, проводячи при цьому їх незначні модифікації. У таких випадках можна скоротити об'єм програми і заощадити пам'ять програм, якщо використовувати одну і ту ж програму. Такий процес називають організацією циклу. Завдяки циклам можна користуватися командами по кілька разів, не дублюючи їх у пам'яті, тим самим скорочуючи розмір програми. Цикл у програмі – це

засіб, що дозволяє за допомогою однієї і тієї ж послідовності команд виконувати якісь дії багато разів. У загальному випадку цикл – це група команд, виконання яких багато разів повторюється за рахунок того, що в цій групі є команда, що повертає управління знову на першу команду в групі, поки не виконається деяка умова закінчення циклу.

У циклі завжди можна виділити чотири блоки: підготовки, основних дій (тіло циклу), підготовки до наступного циклу (модифікації), перевірки умови. Всі ці блоки обов'язково присутні в кожному циклі, але порядок їх проходження може бути різним. На рис. 4.2 а показана схема алгоритму з постперевіркою (цикл «До»), при використанні якого програма виконується до тих пір, поки не буде задоволена умова виходу з циклу.

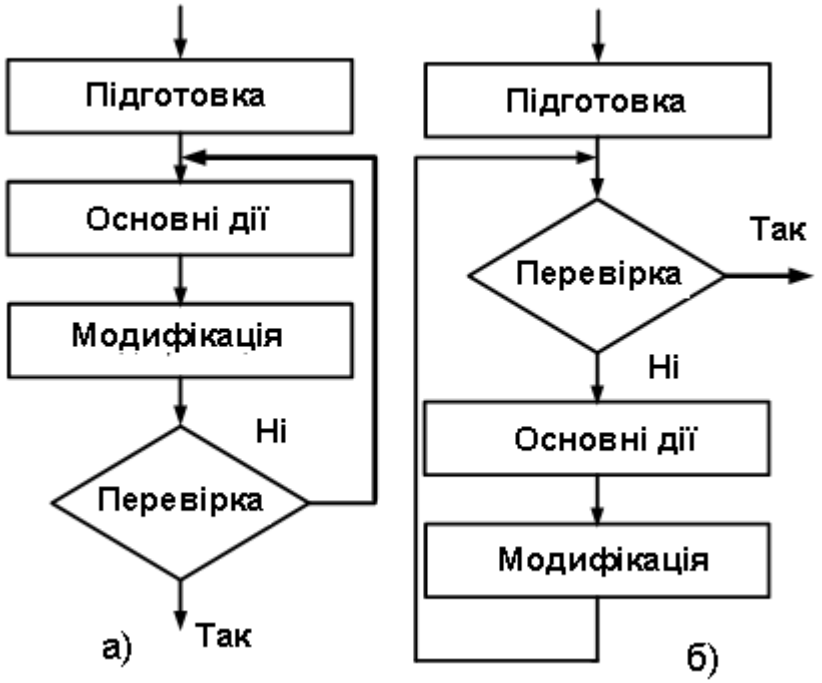


Рис. 4.2. **Схеми алгоритмів організації циклів: а – з постперевіркою; б –з передперевіркою.**

Дана конструкція управління викликає повторення тіла циклу (тобто розпоряджень, представлених блоками «основні дії – модифікація») кінцеве число раз або, взагалі кажучи, нескінченне число раз. У алгоритмі аналізується умова і виробляється відповідь «так/ні». Якщо умова виконана, (відповідь «так»), то виконання циклу закінчується. Інакше виконується повторення тіла циклу, представленого функціональними вузлами «основні дії – модифікація». У схемі з

передперевіркою (рис. 4.2 б) умова перевіряється відразу ж після підготовки (цикл «Поки»). Основна відмінність обох форм алгоритму полягає у тому, що під час постперевірці цикл виконується принаймні один раз, а під час передперевірці може вийти так, що цикл не виконуватиметься жодного разу. Цикл «До» застосовується при обробці однаковим чином елементів масиву, або повторенні обчислень до виконання деякої умови. Щоб пояснити ідею циклу, розглянемо завдання обчислення суми N чисел. Для її вирішення треба послідовно додавати кожне нове число до поточної приватної суми. Щоб вирішити завдання при великому N розумно використовувати алгоритм, коли замість N послідовностей команд збільшення буде використана одна послідовність, виконання якої повториться N разів.

При використанні циклів потрібно вирішувати дві проблеми. По-перше, потрібно підраховувати число вже зроблених повторень, для того, щоб процес підсумовування своєчасно закінчився. По-друге, – забезпечити при кожному проходженні циклу вибірку наступного нового числа. Управляючи конструкції циклів «До» і «Поки» дозволяють вирішувати дані проблеми. Хай розв'язується завдання пошуку в масиві {A} з 100 однобайтових чисел такого, яке має найбільше значення. Схема такого алгоритму показана на рис. 4.3 а.

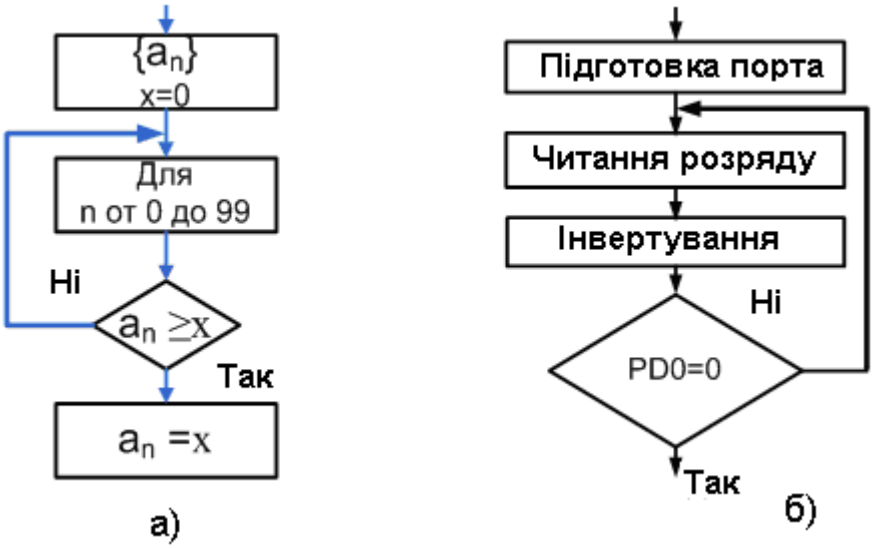


Рис. 4.3**. Схеми алгоритму обчислення максимального елементу в одновимірному масиві (а) і алгоритму перевірки стану молодшого розряду порту D (б)**

Алгоритм використовує циклічний процес. При кожному повторенні тіла циклу проводиться порівняння значення чергового числа a_n, вибраного в масиві {A}, з максимальним із значень змінних, переглянутих у попередніх повтореннях тіла циклу. Якщо рівність $a_n > x$ виконується, то x привласнюється значення a_n. Після стократного повторення тіла циклу відбувається вихід з циклу. При цьому значення x – є шукане рішення задачі. Наступний алгоритм демонструє використання циклу для перевірки стану молодшого розряду 0 порту D. Як видно з рис. 4.3 б, після підготовки порту до роботи «читається» стан пина 0 (молодшого розряду порту D) і його значення інвертується. Потім, виконується операція порівняння (логічне значення розряду перевіряється на рівність 0), після виконання якої програма переходить до початку циклу, якщо умова не виконується або здійснюється вихід з циклу.

Наступним базовим модулем алгоритмів є розгалуження програми (альтернативне рішення). Для опису розгалужень використовують оператора IF. Ця конструкція алгоритму управління стосується ухвалення рішень, застосовується тоді, коли залежно від умови, необхідно виконати або одне, або інше розпорядження (рис. 4.4. а).

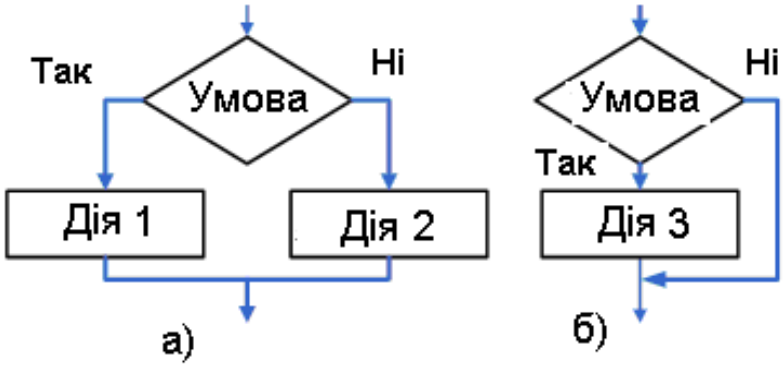


Рис. 4.4. **Схеми алгоритмів розгалуження програм: а) з галуженням; б) з обходом**

Вона забезпечує вибір між двома напрямами ходу програми. Сама по собі вона не приводить до викликів нових розпоряджень, а лише управляє порядком їх виклику. Для визначення напряму, в якому піде подальше виконання програми, робіться перевірка умови. Кожний з шляхів веде до точки злиття, так що програма виконуватиметься незалежно від того, який був шлях вибраний.

Управляюча конструкція алгоритму обходу (рис. 4.4 б). є окремим випадком алгоритму галуження з однією «порожньою» гілкою. Вона застосовується тоді, коли дія, що управляє, повинна бути виконана тільки при виконанні однієї умови. Як приклад приведемо алгоритм, що дозволяє описувати процес обчислення $y = x^2 / R$, де x – змінна, представлена графіком на рис. 4.5, а. На початку алгоритму визначається на якому інтервалі часу (аргументу t) проводиться обчислення *y*. Ця умова тестується функціональним блоком «перехід за умови». Залежно від того, яке значення приймає величина x, в наступному блоці алгоритму обчислюється значення всієї функції y.

Алгоритм розгалуження можна узагальнити на випадок великого числа гілок (випадок складніших розгалужень). Одним з видів складного розгалуження програми є конструкція (рис. 4.5 в), що також управляє, з багатьма умовами і діями (багатоальтернативного рішення). При позитивному результаті порівняння з першою умовою управління передається на першу гілку програми і виконується дія 1. При негативному – проводиться порівняння з другою умовою і т. д. У результаті виконується тільки одна з багатьох дія, номер якого задовольняє умові з тим же номером.

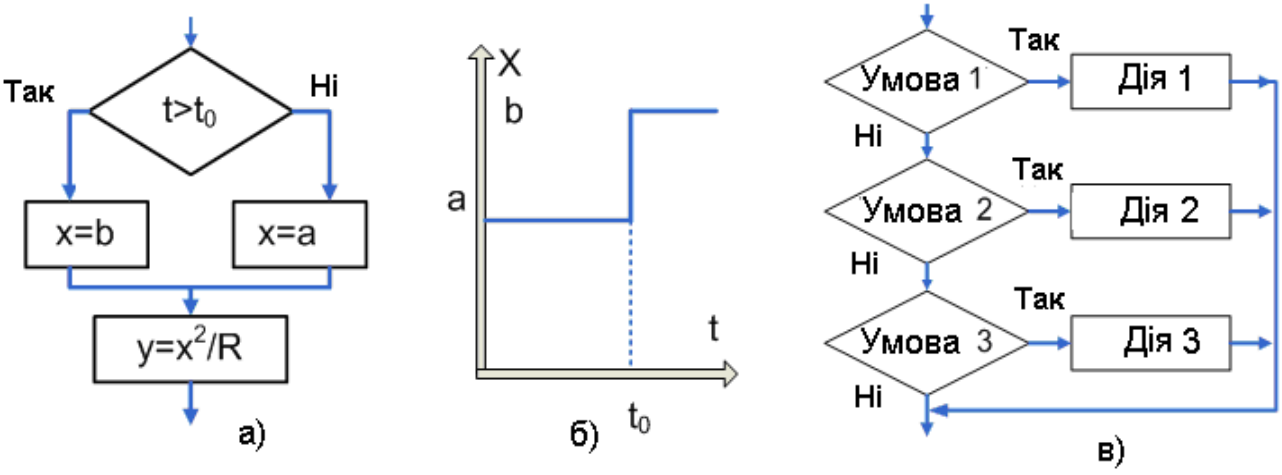


Рис. 4.5. **Схеми алгоритмів розгалуження програм з галуженням (а); графік змінної x, залежної від t (б) і схеми алгоритму багатоальтернативного рішення (в)**

У комп'ютеризованих системах управління обладнанням і технологічними процесами часто виникає потреба вирішувати одну і ту ж

задачу, як у декількох місцях програми, так і на різних етапах виконання. Наприклад, часто потрібно провести відлік показників декількох подібних датчиків. Оскільки виконання будь-якої програми полягає в проведенні якихось дій над якимись об'єктами, то в цьому випадку можна підготувати одну незалежну програму і звертатися до неї в міру необхідності, не повторюючи одні і ті ж команди в різних місцях програми. Така незалежна частина програми називається підпрограмою і розміщується у області пам'яті окремо від основної програми. Якщо говорити про більш загальний випадок, то виявиться що майже завжди при створенні програм зручно розбіти загальне початкове завдання на підзадачі і, вважаючи, що підзадачі вирішені, записати для кожної підзадачі окрему підпрограму. Розбіття початкового завдання на підзадачі фактично є придумуванням нових проміжних виконавців. Природно, що кожну підзадачу можна, у свою чергу, розбіти на підзадачі. Чим складніше початкове завдання, тим більше буде рівнів ієрархії і тим більше, врешті-решт, вийде окремих підпрограм.

Перехід до підпрограми прийнято називати викликом підпрограми. Звернення до підпрограми здійснюється за командою виклику CALL MARK, де MARK – символічне ім'я підпрограми. У самому понятті підпрограми передбачається, що її можна викликати в різних точках головної програми. Тому очевидно, що в МК, що працює з підпрограмами, повинні існувати засоби для збереження адреси тієї точки в головній програмі, в яку треба повернутися (адреси повернення).

Одним із засобів цього є використання стека, вбудованого в МК. Перед командою CALL MARK у стеку необхідно зберегти значення лічильника команд. У цьому випадку повернення з підпрограми здійснюватиметься в те місце основної програми, звідки був здійснений виклик. Для цього будь-яка підпрограма повинна закінчуватися командою повернення з підпрограми RET.

Основна програма у разі потреби може мати декілька підпрограм, а підпрограми, у свою чергу, можуть мати свої підпрограми. Бажано, щоб кожна підпрограма могла звернутися до іншої підпрограми. Такі багатократні звернення називаються вкладеними, а кожен черговий виклик називають рівнем вкладення.

Для організації вкладених звернень використовується стек. При використанні стека допускається найзагальніша форма вкладених переривань. Стек дозволяє одній підпрограмі звертатися не тільки до інших

підпрограм, але і до самої себе. Це можливо тому, що при кожному вході в підпрограму адреса повернення поміщається у вершину стека, а всі адреси, що раніше запам'ятали, опускаються в стек і тим самим зберігаються. При поверненнях адреси, що запам'ятали, вибираються із стека в порядку, зворотному тому, в якому вони запам'ятовувалися. Звичайно, число рівнів вкладеності (тобто число незавершених підпрограм) обмежується розмірами стека, оскільки кожна адреса повернення займає в стеку 2 регістри.

При зверненні до підпрограми часто виникає необхідність зберегти не тільки адресу повернення в основну програму, але і вміст окремих робочих регістрів. У зв'язку з цим важливий аспект поняття підпрограми пов'язаний із способом передачі даних, які у цьому випадку називаються параметрами або аргументами, між головною програмою і підпрограмою. Підпрограма, якій потрібна додаткова інформація у вигляді параметрів її настройки або операндів, називається такою, що параметризується.

Для успішної роботи такої підпрограми необхідно однозначно визначити спосіб передачі в не початкових даних і спосіб виведення результатів її роботи. У МК набули поширення три способи передачі параметрів: через пам'ять; через регістри загального призначення і, іноді, через регістр ознак; через стек. При передачі вхідних параметрів через пам'ять основна програма обов'язково містить команди завантаження деяких елементів пам'яті, а підпрограма – команди прочитування з цих комірок. При передачі вихідних параметрів підпрограма повинна завантажити деякі елементи пам'яті, а основна програма – зчитати.

Другий спосіб передачі даних припускає, що перед входом у підпрограму аргументи завантажуються у загальні регістри, які підпрограмою не використовуються. Підпрограма виконується і залишає свої результати також у загальних регістрах, але інших. Головна програма після повернення з підпрограми використовує і ті та інші результати із загальних регістрів. Передачу параметрів через регістр ознак зручно використовувати при передачі вихідних параметрів, наприклад, в підпрограмах порівняння чисел. У цьому випадку підпрограма повинна встановити в 1 (скинути в 0) відповідні прапори (ознаки), а основна програма – проаналізувати їх значення.

Передача даних може також здійснюватися також з використанням стека. Цей спосіб дозволяє використовувати як параметр вміст лічильника команд.

4.1.4. Програмування – важливий етап у створенні прикладного програмного забезпечення комп'ютеризованого поліграфічного обладнання

Після розробки алгоритму рішення поставленої задачі приступають до власне програмування. Щоб обробіти інформацію на МК, потрібно крок за кроком описати весь процес її обробки. Такий опис називається програмою. Реальна програма повинна складатися тільки з тих команд, які передбачені конструкцією МК. Сукупність цих команд складає так звану машинну мову, або мову машинних команд для МК. Сукупність програм, написаних для МК, складає його програмне забезпечення. Програмне забезпечення повинно бути представлене в тій формі, в якій її сприймає МК. Апаратна частина МК може сприймати тільки два рівні напруги U_1 = 5 В і U_0 = 0 В. Тому необхідне, щоб всі команди і дані в МК були представлені у відповідній формі, тобто в двійковій, у вигляді послідовностей з 0 і 1, оскільки це єдина зрозуміла МК форма. Двійкове представлення команд у МК називається машинною мовою, а кодування команд – програмуванням на машинній мові.

У МК сімейства AVR команди в двійковому коді використовують 16 розрядів і командне слово складається з декількох полів. На початку команди йде так званий код операції (OP – Operations Code), незмінна частина командного слова, на підставі якої дешифратор МК розпізнає команду. За ним слідують операнд або операнди, до яких звертається команда. Наприклад, для команди INC, що збільшує на одиницю вміст регістра r15, командне слово матиме вигляд (рис. 4.6 а). Символами d помічені розряди, які вказують в двійковому коді адресу регістра РЗП з номером 15 (1111 – в двійковому коді).

Аналогічним чином, використовуючи коди операцій і розміщення операндів, що приводяться в довідкових даних, у машинних кодах можна запрограмувати й інші команди програми. Оскільки писати програму в двійкових кодах не зовсім зручно із-за її громіздкості, вважають за краще використовувати шістнадцятирічні коди. Представлення програми в шістнадцятирічному вигляді дозволяє оперувати тими ж машинними кодами, стежити за всіма одиницями і нулями, але при цьому позбавивши від необхідності писати їх. Якщо розбити командне слово на чотирьохрозрядні нібли (півслова), а потім кожен фрагмент записати в шістнадцятиричному коді, то для командного слова одержимо замість

ланцюжка нулів і одиниць (рис. 4.6 а) число 94f3. Таке командне слово легше безпомилково записати і такий запис легше запам'ятати, ніж довгі послідовності 0 і 1. При бажанні всю програму можна порівняно легко представити в шістнадцятиричному коді.

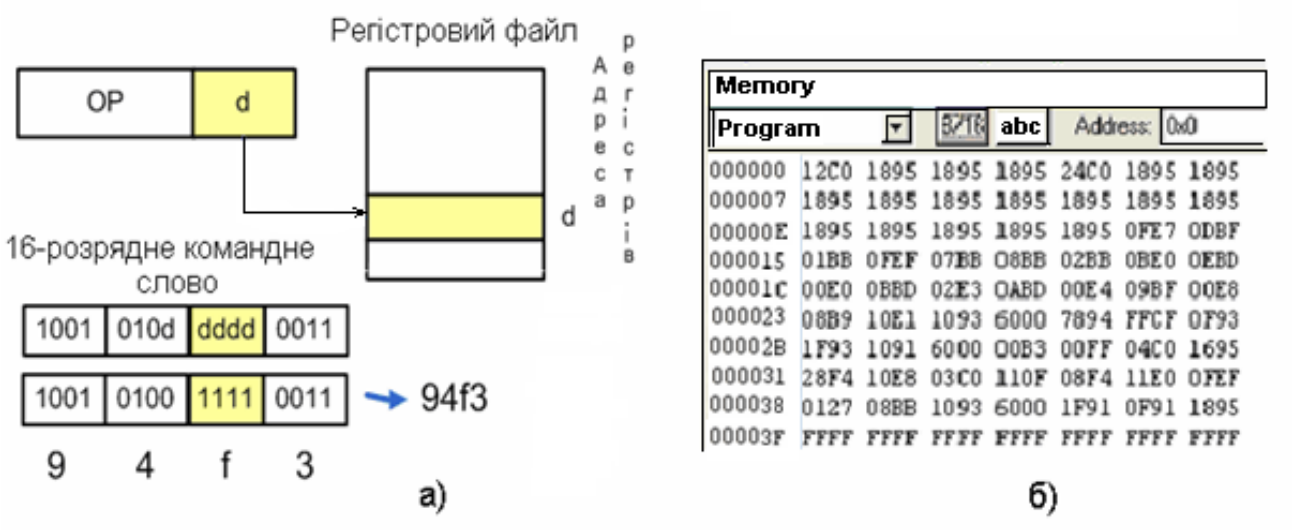


Рис. 4.6. **Вміст командного слова для команди INC, що збільшує на одиницю вміст регістра r15 (а) і програма МК у машинних кодах (у вигляді сукупності командних слів), при представленні команд в елементах пам'яті програм у шістнадцятирічній системі числення (б)**

Треба мати на увазі, що перелік того, що може робіти МК, званий набором команд з вказівкою кодів операцій і операндів, за допомогою яких ці дії здійснюються, звичайно додається до МК. Для програмування безпосередньо на машинній мові (у кодах команд) використовують довідкові таблиці, які в зручній формі містять перелік кодів команди (звичайно в шістнадцятирічному коді) і іншу довідкову інформацію.

Шістнадцятирічна система числення використовується і при представленні команд в елементах пам'яті програм (рис. 4.6 б). Якщо відомі адреси елементів пам'яті, де зберігаються команди і коди необхідних операцій, а також адреси, на які в них є посилання, то у принципі використовуючи машинний код можна розібратися, що відбувається усередині МК згідно поміщеної в пам'ять програми.

Очевидно те, що із збільшенням довжини програми, представленої в машинних кодах, все важче стає запам'ятовувати коди різних операцій

і списки адрес даних, що зберігаються або обчислюваних. Тому програмування на машинній мові, що сприймається апаратними засобами МК, справа трудомістка і кропітка навіть для не складних завдань. Якщо до того ж, під час написання програми або коли програма вже написана, виявиться, що її бажано змінити або відредагувати, то необхідне при цьому коректування адрес наново може стати справжньою мукою. Тому потрібні спеціальні засоби, що полегшують підготовку програм на машинній мові. Набагато легше писати програму, якщо кодам операцій і адресам привсвоювати не числові значення, а символічні буквені імена у вигляді простих буквенних кодів (слів), що мають зрозумілий сенс. Зручніше замість 1001010011110011 або 94f3 написати текст INC R15, розуміючи, що він позначає машинну команду «збільшити вміст регістра R15 на одиницю». Слово INC – це скорочення назви виконуваної операції на англійській мові (Increment – приріст, крок), що відображає основний зміст виконуваної операції на природній, розмовній мові (мнемокод). Зручний для запам'ятовування команд мнемокод є скорочене слово або послідовність букв, замінюючих повне слово або фразу. Використання мнемокодів у програмі значно полегшують роботу програміста і зменшують кількість виникаючих при програмуванні помилок. Оскільки програма, написана з використанням скорочень, не сприймається безпосередньо МК, нескладно здогадатися, що програмування стане ще зручнішим, якщо використовуючи умовні скорочення назв виконуваних операцій, мнемокоди ми матимемо в своєму розпорядженні також деякий засіб, що автоматично підставляє відповідні чисельні значення замість символічних імен (операцій і адрес), після того, як програма написана. Саме таким засобом, що перетворює всі символічні імена і адреси в зрозумілу МК машинну мову, що розміщує програму в правильному порядку в пам'яті, що бере на себе «турботу» про правильне використання всіх адрес при проходженні програми і є спеціальна програма, яка одержала назву «асемблер». Слово «асемблер» відображає, як і «асамблея» те, що йде збір або збірка чого-небудь і при цьому виходять принципово нові властивості, до цього відсутні у компонентів, що беруть участь у збірці.

Програма, написана в новій зручній формі, з використанням мнемокодів і умовних скорочень назв виконуваних операцій, називається програмою на мові Асемблера. Причому мова асемблера ізоморфна машинній мові. Іншими словами, яка рятуєа від необхідності оперувати кодами команд програма на мові Асемблера – це, по суті, та ж програма на

машинній мові, але з символічними іменами для адрес операндів, з символічними іменами для адрес команд і мнемонічними кодами операцій. Завдання програми «асемблера» або просто асемблера – замінити кожне текстове символічне ім'я відповідним двійковим кодом, відвести комірки для кожної команди і кожного операнда, підставити замість символічних адрес їх чисельні значення. Причому треба це зробіти так, щоб між текстом на мові асемблера і командами на машинній мові існувала взаємно однозначна відповідність. На рис. 4.7 показана схема використання програми «асемблера».

Рис. 4.7. **Схема використання асемблера**

Програма, написана користувачем з використанням символічних імен, називається початковою програмою (Source program). Початковий код такої програми – це файл, який має розширення *.asm. Вона поступає в пристрій трансляції або, як то кажуть, «завантажується» спільно із спеціальною «оброблювальною» її програмою. Початкова програма виступає в ролі «даних», використовуваних оброблювальною програмою – асемблером для своєї роботи з зміни текстових фрагментів програми. У ході роботи з перетворення початкової програми в машинний код асемблер, перш за все, формує таблиці для присвоювання чисельних значень мнемонічним кодам операцій і символічним адресам. Для того, щоб присвоїти числове значення кожній символічній адресі, асемблер спочатку складає так звану таблицю імен, в яку заносить зустрінуті в програмі символічні адреси. Потім він відводить область пам'яті для програми, обчислює значення всіх символічних адрес і після цього виконує підстановку. У результаті цих дій над початковою програмою асемблера, або як часто говорять у результаті асемблювання, генерується програма на машинній мові, яку називають об'єктною програмою (Object code). Код такої об'єктної програми – це файл, який має розширення *.obj. Звичайно під час трансляції також створюється лістинг програми (цей файл має

розширення *.lst), за допомогою якого користувач може визначити адреси і машинний код, призначений асемблером своїм командам.

Оскільки для МК є характерним обмеженість пам'яті і відсутність «власного» системного програмного забезпечення, то для асемблювання початкових програм використовується не сам МК, а якась «велика» ЕОМ з обширною пам'яттю і розвиненим програмним забезпеченням.

У програмі на мові асемблера, як у послідовності операторів, взагалі кажучи, використовуються два типи операторів: мнемокоди команд і директиви. Основна відмінність директив і мнемокодів команд полягає у тому, що в процесі асемблювання директиви представляють програмі-асемблеру допоміжну інформацію, яка використовується в процесі асемблювання і служить для опису типів даних, резервування пам'яті, організації процедур і т. д.

Асемблер, забезпечуючи значне підвищення праці програміста в порівнянні з продуктивністю при програмуванні в кодах команд, в той же час зберігає можливість доступу до всіх апаратурних можливостей програмованого МК. З цієї причини саме на мові Асемблера висококваліфікованим програмістом можуть бути написані оптимальним чином використовуючі можливості МК і периферійних пристроїв програми, найефективніші з погляду часу виконання і об'єму займаної пам'яті. Одна з головних переваг написання початкових програм на мові Асемблера полягає також у легкості, з якою робляться вставки і видалення додатків в програмі. При написанні програми часто виявляється, що в деяку точку вже написаної частини програми потрібно вставити додатково групу команд. У програмі на машинній мові це зажадає зрушення всіх слів програми нижче вставлених команд і обов'язкової корекції використовуваних адрес. При «зрушенні» може опинитися так, що адресні поля деяких команд у програмі тепер повинні посилатися на команди, що перемістилися. Це значить, що всі такі адресні поля потрібно відкоректувати. У програмі, написаній на мові Асемблера, подібні проблеми відсутні, оскільки спочатку адресам не привласнюються чисельні значення. Отже, при вставках нових команд ніяких труднощів не виникає. Точно також уникають труднощів і при видаленні команд.

Інша корисна властивість Асемблера, як мови символічного кодування інструкцій МК, а також транслятора з цієї мови, – це виявлення помилок. Асемблер не може виявляти «смислових», логічних помилок у самому задумі програми. Проте він може перевіряти дотримання певних

синтаксичних обмежень, накладених на мову і повідомляти про ці порушення. Слід мати на увазі, що специфіка мови Асемблер полягає і в тому, що набір операторів для цієї мови залежить від системи команд, використовуваних конкретним МК. З цієї причини МК, що випускаються різними компаніями, можуть мати різну систему команд і, відповідно, «свою» мову Асемблера. Тому слід мати на увазі, що надалі ми орієнтуватимемося на Асемблер для мікроконтролерів AVR.

Хоча мова Асемблера значно зручніше для програмування, чим машинна, проте, вона теж не є ідеальною. Для багатьох завдань програми на мові Асемблера виявляються такими, що складаються з тисяч операторів – символічних аналогів машинних команд. Такі програми погано доступні для огляду, важко «читані» і складні у відладці. Не можна не враховувати і той факт, що різке розширення сфери застосування комп'ютеризованих систем призвело до появи великої групи користувачів, малокваліфікованих у області створення програм і навіть практично не знайомих з програмуванням. Ця ситуація особливо гостро поставила проблему використання таких засобів програмування, які б дозволили задавати алгоритми на професійній мові, в термінах, звичніших для непрограміста, а не в термінах мало зрозумілої йому мови Асемблера. Подальше скорочення трудомісткості і ще більшої зручності при написанні програм дає використання мов високого рівня. При написанні програм на них не тільки не треба «думати» про дійсні адреси елементів пам'яті, але у ряді випадків виявляється достатнім всього лише вказати завдання, що підлягають рішенню, не особливо піклуючись про те, як вони розв'язуватимуться. Працюючи з такими мовами програмування, користувач лише визначає, які складні математичні дії або операції над даними повинні бути проведені, не займаючись «програмуванням деталей» в звичайному сенсі цього слова. Ці мови свого часу були розроблені для «великих» комп'ютерів і відрізняються тим, що вони значно більше орієнтовані на людину і з цієї причини відносно слабо пов'язані з конкретною структурою процесора і його конкретними командами. Такі мови оперують не вмістом регістрів, а звичними нам десятковими числами, а також змінними, константами та іншими елементами, знайомими нам з математики. Використовуючи відносно невелике число конструкцій, за допомогою мови високого рівня можна будувати складні програми, застосовуючи при цьому в додатках стандартні бібліотеки.

Якщо програміст пише програму на мові високого рівня, то переклад (компіляцію) програм з цієї мови в машинні коди можна здійснити за допомогою «великої» ЕОМ, подібно тому, як це робилося при асемблюванні. Звичайно, при цьому компілятор з мови високого рівня проводить складніші перетворення, ніж при асемблюванні і компілятори з мов високого рівня – це досить складні програми. Компілятор перетворить початкову програму не прямо в коди, а в якусь проміжну форму, звану об'єктним кодом. Тому після компіляції проміжний об'єктний код доповнюють, використовуючи редактора зв'язків, і вся програма «збирається» цілком. На цьому ж етапі відбувається перетворення імен, що залишилися, на адреси і підключення стандартних підпрограм. У результаті виходить програма в машинних кодах. Компілятор використовує всі ресурси процесора на свій розсуд. Програміст не бере участь у процесі визначення того, в яких регістрах або елементах пам'яті зберігатимуться описи змінних, яким способом здійснюватимуться обчислення та ін. Програма-компілятор вибирає все це сама. Відповідно і «відповідальність» за ефективність машинних кодів, одержаних в результаті компіляції, цілком лягає на програму-компілятор і редактор зв'язків. При цьому виникають декілька проблем. По-перше, перехід на вищий рівень програмування призводить до того, що програміст втрачає доступ до тих апаратних засобів, до яких він мав на мові асемблера. Через це програміст позбавляється можливості поліпшити якість своєї програми, маніпулюючи командами, що працюють зі всіма апаратними засобами МК. Крім того, виникає те, що називають семантичним розривом між архітектурою комп'ютера (або МК) і середовищем її використання. Семантичний розрив породжує помилки за рахунок не завжди коректного використання конструкцій мови програмування, збільшує непродуктивні часові витрати, погіршує надійність всього програмного забезпечення. По-друге, перехід на мову високого рівня вимагає розробки складного компілятора. Відомо, що компілятори гірше справляються з оптимізацією програми, ніж висококваліфіковані програмісти, що позначається на якості одержуваних після компіляції програм. Це виражається у великому об'ємі програми в машинних кодах, у гірших її часових характеристиках. Програми, написані на мовах високого рівня, звичайно вимагають в 2 – 10 разів більше інформаційної місткості в порівнянні з оптимізованими програмами на мові і виконуються в 2 – 100 разів повільніше. Ефективність одержуваної програми безпосередньо визначається якістю використовуваного компілятора і не-

прямим чином залежить від вхідної мови. Творці компілятора, у принципі, не можуть знайти розумний компроміс у розподілі функцій між апаратною і програмною реалізаціями. Це пов'язано з тим, що чим більше «відокремлені» оператори вхідної мови високого рівня від апаратурних засобів МК, чим менше вони орієнтовані на ці засоби, тим складніше побудувати компілятор, що генерує якісну об'єктну програму. При створенні реальних програм для складного комп'ютеризованого обладнання прагнуть «обійти» цей «зворотний бік медалі» мов високого рівня. Наприклад. Сучасною тенденцією комп'ютеризованих систем управління є передача все більшого числа функцій з обслуговування операцій введення – виведення периферійним пристроям і різного роду контроллерам. Програми в таких периферійних пристроях, у зв'язку з жорсткими вимогами до часу їх виконання, обмеженнями на об'єм пам'яті, прагнуть писати на мові Асемблер. Майже всі підпрограми драйверів пристроїв введення – виведення пишуться на мові Асемблера. У зв'язку з цим відзначимо наступне. Оскільки МК AVR були повністю новою розробкою, в якій передбачалося використання мов високого рівня, то при їх проектуванні істотну увагу приділили питанням поліпшення програмно-апаратного інтерфейсу (семантичного зв'язку між програмним забезпеченням і апаратними засобами). Із самого початку в розробку апаратних засобів МК AVR були залучені фахівці з мов високого рівня – перш за все, з мови програмування Сі. У результаті була одержана архітектура МК, спеціально розрахована на складання програм на мові Сі. Мова Сі (була розроблена Деннісом Річи на початку 70-х років двадцятого сторіччя) завоювала особливу популярність у програмістів завдяки унікальному поєднанню можливостей мов високого і низького рівня. Більш предметно про можливості цієї мови програмування ми поговоримо в розділі 4.3.

Отже, останніми роками при введенні до ладу нових систем комп'ютеризованого обладнання з МК AVR для отримання кодів програм використовують транслятори з асемблера і компілятор з мови Сі.

4.2. Мова Асемблера для AVR

Хороша програма для МК – це програма, яка правильно реалізує заданий алгоритм в мінімальний час, займає невеликий об'єм пам'яті і у яку легко вносити зміни. Цим умовам відповідає програмування на мові асемблера. Розробнику надається можливість написання програм в

мнемонічній формі з використанням символічних адрес і посилань. Асемблер містить засоби, що дозволяють у процесі компіляції початкової програми виявляти і відзначати всі вислови, побудовані з порушенням синтаксичних правил мови. Крім того асемблер видає лістинг програми, на якому представлені обидві версії програми (початкова і об'єктна). У МК AVR використовується асемблер від компанії Atmel. Це розширена мова Асемблера, в якої, крім іншого, передбачається можливість присвоєння імені деякої послідовності команд і в будь-яких місцях програми, в яких повинна бути використана ця послідовність. Тобто, йдеться про наступний рівень мови програмування, про макроасемблер, де частини програми, що часто повторюються, які не повинні виконуватися у вигляді підпрограм, можуть визначатися як макроси. Використання мови макроасемблера скорочує запис програми (у середньому на 5 – 20%) і покращує її видимість. Прикладна програма МК, написана на мові Асемблера, містить наступні типи висловів: мнемокоди команд, які транслюються в об'єктні коди; директиви асемблера, які управляють трансляцією; коментарі, які використовуються при документуванні програми.

Початкова програма користувача на Асемблері складається з командних рядків (пропозицій, висловів) довжиною максимум 120 знаків. Кожен рядок відповідає одній пропозиції на Асемблері і, як правило, він є закінченим висловом. Кожна пропозиція мови Асемблера містить чотири фіксовані поля.

1. Поле мітки. Воно служить для вказівки (у вигляді символічного імені) адреси команди МК або набору поміщених у його пам'ять даних, що знаходяться в даному рядку, якщо такі посилання необхідні. У цьому полі, коли є посилання в програмі, можуть розташовуватися символічні імена адрес команд (мітки) і змінні (символьні константи) – символічні адреси першого байта визначеного директивами DB (DW). Оскільки в програмі доводиться посилатися далеко не на кожну команду, те поле мітки може залишатися порожнім. Кінець поля мітки на мові асемблера, як правило, наголошується роздільником – двокрапкою. Визначені в тексті програми імена адрес для позначення адреси команди і змінних (символьних констант) для вказівки адреси першого байта представляються у вигляді алфавітно-цифрової послідовності літер (букв латинського алфавіту і цифр). Як символічне ім'я уживається послідовність з буквено-цифрових знаків, максимальна довжина якої не повинна перевищувати 8 символів. Причому ім'я (мітка) повинне починатися з букви, а не з цифри,

і при цьому в полі мітки не можна вживати деякі зарезервовані слова (наприклад, мнемоніки виконуваних операцій). Кожен командний рядок може починатися з мітки. Звичайно іменами забезпечуються пропозиції, на які проводиться умовний або безумовний перехід. Мітка служить як точка призначення при галуженнях у програмі, переходах до певної точки програми і викликах підпрограм. При цьому одне і те ж ім'я не повинне зустрічатися в полі мітки більше одного разу. Мітка визначається на початку рядка, як буквено-цифрове ім'я з двокрапкою. Оскільки мітка це символічна адреса, то її ім'я потрібно вибирати так, щоб програма легко «читалася» і був зрозумілий сенс виконуваних дій. Наприклад, мітка LOOP (Loop – петля, цикл) добре підходить для позначення початку циклу, тобто крапки, на яку посилається команда переходу за умовою. Аналогічно вибирають імена для наборів даних. Змінна (символьна константа) KONST1 задає адресу першого байта з 5 байт – $1A, $2F, 14, 12, 2/3, визначених директивою DB. Пригадайте, що прочитуючи початковий текст програми, асемблер виявляє імена, введені програмістом (тобто мітки і змінні), для кожного імені визначає числове значення і записує цю інформацію в таблицю імен. Наступні два поля – це поля мнемоніки команд (символічного представлення машинних команд). У цих полях задають послідовності визначених компанією Atmel символів, що позначають у кожному конкретному випадку команду і операнди, до яких повинна бути застосована дана команда.

2. Поле операції. Це поле в пропозиції на мові асемблера містить мнемонічний код операції і визначає операцію, що виконується даною командою. Число символів у мнемонічному буквеному запису операції може варіювати залежно від типу операції. Наприклад, в даному полі може стояти мнемонічний код ADC (мнемокод машинної команди) операції складання з урахуванням перенесення Add with Carry, який легше запам'ятати і вживати в порівнянні з числовим кодом 000111rddddrrrr (r, d – номери використовуваних регістрів).

3. Поле операндів. У цьому полі міститься вся інформація, яка потрібна для того, щоб повністю визначити виконувану команду, зокрема, вказати на джерела і приймачі даних, що беруть участь в операції. Воно відводиться для одного або декількох операндів. Наприклад, це поле може містити як операнди номера регістрів, що беруть участь в операції складання (Rr, Rd), або безпосередньо дані (числа). Як операнди можуть бути символічні імена, числа, вирази, причому у виразах можуть бути ви-

користані знаки арифметичних операцій. Для адресації операндів (тобто вказівки, де знаходиться число для операції, або куди поступає результат) можуть використовуватися регістрова (адресація позначається назвою регістрів), безпосередня (операнд заданий в самій команді), пряма, непряма (у команді указується адреса елементу пам'яті, де знаходиться операнд) адресація даних у пам'яті. Для обчислення адреси операнда або для команд-переходів можна записувати арифметичні формули, що визначають число команд, через яке потрібно «перестрибнути» вгору або вниз. Як символічні адреси в полі операндів уживаються імена, які зустрічаються в полі мітки. Числові дані в полі операндів можуть представлятися в різних системах числення. Дані в полі операндів можуть бути задані в двійковій або шістнадцятирічній системі. Щоб асемблер міг визначити, в якій системі числення представлені дані указують однобуквений код системи (b – Binary – двійковий, h – Hexadecimal – шістнадцятирічний). Якщо поле операндів складається з декількох частин, то вони повинні бути розділені комами.

4. Поле коментаря. Це останнє поле в пропозиції на мові Асемблера дає можливість програмісту забезпечити пропозицію будь-якими поясненнями, що полегшують читання і розуміння програми. Запис у коментарі потрібен лише програмісту. Це поле ігнорується асемблером у процесі трансляції початкової програми в об'єктну. Єдина вимога до цього поля полягає в тому, щоб йому передував обмежувач (крапка з комою). Для ілюстрації розглянемо фрагмент програми написаної на мові асемблера:

Поле метки	Поле операції	Поле операндів	Комментари
main1:	ldi	rab, 0b00010000	;Запис значення
m1:	rjmp	m1	; Порожній нескінченний цикл
p3:	ldi	temp, 0xFF	;Записати в temp число $FF
	ldi	R18, 0xAA	; Записуємо 170 в r18
	ldi	R23, 6	; Записуємо 6 в r23
	ldi	R25, 0b10001000	; Записуємо 136 в r25
	ldi	temp, (1<<7)	; Зсув 1 в 7-ий біт

У програмі послідовність з буквено-цифрових знаків «main1:» у полі мітки це мітка, яка визначає символічну адресу виконуваної команди «ldi rab, 0b00010000». Мнемонічний код ldi у полі операції задає операцію, що виконується даною командою. Мітки «m1:» і «p3:» у полі мітки слу-

жать як точка призначення при переходах до певних ділянок програми. Як операнди, задані в полі операндів, в програмі, написаній на мові Асемблера, використані числа (0xAA, 6, 0b10001000), представлені в шістнадцятирічній, десяткової і двійкової системах числення, символічні імена регістрів (R18, R23, R25, rab, temp), вирази (1<<7). Запис вигляду (1<<7) можна пояснити таким чином: занести одиницю в молодший біт байта (тобто одержимо двійкове число 00000001) і зрушити цю одиницю на 7 позицій (одержимо 10000000, або, якщо коректно записати, 0b10000000). Команда «rjmp m1» у полі операцій визначає, що після виконання команди rjmp треба «перестрибнути» до команди, поміченої меткой«m1».

Коди операції, виконуваних даною командою МК і потрібні при цьому операнди, можна знайти в довідкових даних на МК.

AVR асемблер може також обробляти вирази, поміщені в початковому файлі. Вирази – це комбінація знаків операцій і операндів, результатом якої є певне значення. Знаки операцій визначають ті дії, які повинні бути виконані над операндами. Операнд – це константа, літерал, ідентифікатор, виклик функції, вираз вибору елементу та ін. Кожен операнд у виразі може бути виразом, сформованим комбінацією операндів, знаків операцій і круглих дужок. Вирази складаються з операндів, функцій і операторів. Всі вирази займають у пам'яті по 4 байти (32 біта).

У початковому файлі для обробки виразів можуть застосовуватися наступні операнди: 1) визначені користувачем мітки. Міткам асемблер призначає те значення, яке містить лічильник команд у місці їх появи в початковому файлі; 2) змінні, визначені користувачем за допомогою директиви «SET»; 3) змінні, визначені користувачем за допомогою директиви «EQU»; 4) цілочисельні константи. При цьому в початковому файлі для обробки констант у виразах допустимі наступні формати: а) десяткові (за замовчуванням) – 10,225; б) шістнадцятиричні – 0x1A, 0xFF або $1A, $FF; в) двійкові 0b00001010, 0b11111111; г) символи ASCII – ‘A’, ‘R’, ‘a’; д) послідовності символів ASCII (рядки) – ‘stroka’; е) вміст лічильника команд у поточний момент часу.

Можуть застосовуватися наступні функції: 1) LOW (exp) – повертає молодший байт у виразі (exp). Приклад: LOW ($9ABC) повертає $BC; 2) HIGH (exp) – повертає старший байт у виразі (exp). Приклад: HIGH ($9ABC) повертає ($9A); 3) BYTE2 (exp) – має те ж призначення, що і HIGH (exp). Приклад: BYTE2 ($9ABC) повертає ($9A); 4) BYTE3 (exp) –

повертає третій байт у виразі (exp). Приклад: BYTE3 ($87654321) повертає ($65); 5) BYTE4 (exp) – повертає четвертий байт у виразі (exp). Приклад: BYTE4 ($87654321) повертає ($87); 6) LWRD (exp) – повертає розряди 0 ... 15 у виразі (exp). Приклад: LWRD ($87654321) повертає ($4321); 7) HWRD (exp) – повертає розряди 16 ... 31 у виразі (exp). Приклад: HWRD ($87654321) повертає ($8765); 8) PAGE (exp) – повертає розряди 16 ... 21 у виразі (exp). Приклад: PAGE ($7654321) повертає $5; 9) EXP (exp) – повертає значення 2^{EXP} у виразі (exp). Приклад: EXP (4) повертає значення 2^4 = 16; 10) LOG2 (exp) – повертає цілу частину $\log_2$(exp). Приклад: LOG2 (16) повертає цілу частину $\log_2(5) = 2$.

У початковому файлі для обробки виразів асемблер підтримує наступні оператори. Пріоритетність операторів зменшується у міру їх проходження в списку. Вирази, поміщені в дужки, завжди виконуються в першу чергу. 1. Унарний оператор (тобто що складається з оператора і попереднього йому знаку унарної операції) логічне «Ні» (логічного заперечення), що позначається символом «!», повертає 1, якщо значення виразу рівне нулю, а в решті випадків – 0 (пріоритет 14). При цьому унарні операції виконуються справа наліво. Приклад: ldi r16,!0xf0 ; завантажується 0x00 в r16. 2. Унарний оператор порозрядне «Ні» (доповнення до одиниці), що позначається символом «~», повертає всі розряди виразу інвертованими (пріоритет 14). Приклад: ldi r16,~0xf0 ; завантажується 0x0f в r16. Після дії унарного оператора двійкового доповнення («~») всі одиниці в двійковому коді значення операнда міняються на 0, а все 0 – на 1. 3. Оператор унарний мінус (доповнення до двох), що позначається символом «–», повертає арифметичне заперечення виразу (пріоритет 14). Приклад: ldi r16, –2 ; завантажується 0xfe (0b11111110) в r16. 4. Бінарний оператор множення (тобто що складається з двох операндів, розділених знаком бінарної операції), що позначається символом «*», повертає результат множення двох виразів (пріоритет 13). Приклад: ldi r16, labelA*2 ; завантажується labelA*2 у r16. 5. Бінарний оператор ділення, що позначається символом «/», повертає результат ділення виразу, вказаного зліва, на вираз, вказаний справа (пріоритет 13). Приклад: ldi r16, labelA/2 ; завантажується labelA/2 у r16. 6. Бінарний оператор складання, що позначається символом «+», повертає суму двох виразів (пріоритет 12). Приклад: ldi r16, k1+k2 ; завантажується k1+k2 у r16. 7. Бінарний оператор віднімання, що позначається символом «–», повертає різницю двох виразів (пріоритет 12). Приклад: ldi r16, k1–k2 ; завантажу-

ється k1–k2 у r16. 8. Бінарний оператор «зрушення вліво», що позначається символом «<<», зрушує вліво вираз, вказаний ліворуч на кількість розрядів, вказану справа (пріоритет 11). Приклад: ldi r16, 1<<5 ; завантажується $20 в r16. Одиниця – двійкове число 0b00000001, зрушується вліво на 5 позицій – 0b00100000. 9. Бінарний оператор «зрушення вправо», що позначається символом «>>», зрушує вправо вираз, вказаний ліворуч на кількість розрядів, вказане праворуч (пріоритет 11). Приклад: ldi r16, 8>>1 ; завантажується $04 в r16. Вісім – двійкове число 0b00001000, зрушується вправо на одну позицію – 0b00000100. Бінарні оператори повинні бути цілими величинами. Вони часто використовуються для виконання арифметичних операцій перетворення (зрушення вліво відповідає множенню першого операнда на ступінь числа два, рівну другому операнду, а зрушення вправо відповідає діленню першого операнда на 2 у ступені, рівному другому операнду). 10. Бінарний оператор «менше», що позначається символом «<», повертає 1, якщо вираз із знаком, вказаний зліва, менше виразу із знаком, вказаного справа, інакше – 0 (пріоритет 10). Пример: ori r16, k1<k2; завантажується $00 в r16, якщо k1<k2. 11. Бінарний оператор «менше або рівно», що позначається символом «<=», повертає 1, якщо вираз із знаком, вказаний ліворуч, менше або рівно виразу із знаком, вказаного праворуч, інакше – 0 (пріоритет 10). Приклад: ori r16, k1<=k2; завантажується $00 в r16, якщо k1<=k2. 0b00001000, зрушується вправо на одну позицію – 0b00000100. 12. Бінарний оператор «більше», що позначається символом «>», повертає 1, якщо вираз із знаком, вказаний ліворуч, більше виразу із знаком, вказаного праворуч, інакше – 0 (пріоритет 10). Приклад: ori r16, k1>k2; завантажується $00 в r16, якщо k1>k2. 13. Бінарний оператор «більше або рівно», що позначається символом «>=», повертає одиницю, якщо вираз із знаком, вказаний ліворуч, більше або рівно виразу із знаком, вказаного праворуч, інакше – 0 (пріоритет 10). Приклад: ori r16, k1>=k2; завантажується $00 в r16, якщо k1>=k2. 14. Бінарний оператор «рівно», що позначається символом «==», повертає одиницю, якщо вираз із знаком, вказаний ліворуч, рівний виразу із знаком, вказаному праворуч, інакше – 0 (пріоритет 9). Приклад: ori r16, k1==k2; завантажується $00 в r16, якщо k1==k2. 15. Бінарний оператор «не рівно», що позначається символом «!=», повертає 1, якщо вираз із знаком, вказаний ліворуч, не рівний виразу із знаком, вказаному праворуч, інакше – 0 (пріоритет 9). Приклад: ori r16, k1!=k2; завантажується $00 в r16, якщо k1!=k2. 16. Бінарний опера-

тор «порозрядне І», що позначається символом «&», повертає результат логічної операції «І» над кожною парою відповідних розрядів виразів, вказаних ліворуч і праворуч (пріоритет 8). Приклад:ldi r16, $d2&91; завантажується $90 в r16. Операція побітового логічного «І», що позначається символом «&», порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо обидва порівнюваних біта одиниці, то відповідний біт встановлюється в 1, інакше – в 0. 17. Бінарний оператор «порозрядне виключаючє АБО», що позначається символом «^», повертає результат логічної операції що «виключаючє АБО» над кожною парою відповідних розрядів виразів, вказаних ліворуч і праворуч (пріоритет 7). Приклад: ldi r16, $d2^91; завантажується $43 в r16. Операція «побітного виключачого АБО», що позначається символом «^», порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо один з порівнюваних бітів рівний 0, а другий біт рівний 1, то відповідний біт результату встановлюється в 1, інакше, тобто коли обидва біти рівні 1 або 0, відповідний біт результату встановлюється в 0. 18. Бінарний оператор «порозрядне АБО», що позначається символом «|», повертає результат логічної операції «АБО» над кожною парою відповідних розрядів виразів, вказаних ліворуч і праворуч (пріоритет 6). Приклад: ldi r16, $d2|91; завантажується $d3 у r16. Операція «побітного логічного АБО», що позначається символом «|», порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо будь-який (або обидва) з порівнюваних бітів рівний 1, то відповідний біт результату встановлюється в 1, інакше результуючий біт рівний нулю. 19. Бінарний оператор «логічне І», що позначається символом «&&», повертає 1, якщо обидва вирази не рівні 0, інакше – 0 (пріоритет 5). Приклад: ldi r16, $d2&&04; завантажується $01 в r16. Операція «логічного І», що позначається символом «&&», виробляє значення 1, якщо обидва операнди мають не нульові значення. Якщо один з операндів рівний 0, то результат також рівний 0. 20. Бінарний оператор «логічне АБО», що позначається символом «||», повертає 1, якщо хоч би один з двох виразів не рівний 0, інакше – 0 (пріоритет 4). Приклад:ldi r16, $d2||00; завантажується $01 в r16. Операція «логічного АБО», що позначається символом «||», виробляє значення 0, якщо обидва операнди мають значення 0. Якщо який-небудь з операндів має не нульове значення, то результат операції рівний 1.

Важливу роль у мові Асемблера грають директиви (вказівки управління для асемблера). Якщо команди програми визначають дії, що вико-

нуються над даними в процесі рішення задачі, то директиви не пов'язані з діями над даними, а лише повідомляють асемблеру відомості, необхідні в процесі асемблювання. Вони служать для управління спеціальними, типовими для всіх асемблерів функціями: ініціалізували ділянки пам'яті, визначають константи в пам'яті і т. д. Директиви, що задаються в програмі на мові асемблера, містять інформацію, потрібну для управління асемблюванням з початкової на об'єктну мову. Вони є послідовністю символів, визначених компанією Atmel, які завжди починаються з крапки, використовуються лише в процесі асемблювання і не транслюються в машинні коди. Директивам привласнені мнемонічні коди, які записуються в полі операцій. Директиви асемблера перераховані табл. 4.1.

Перші три директивиі в таблиці визначають вибраний для роботи сегмент пам'яті в адресному просторі. Початковий файл може використовувати декілька блоків різних видів пам'яті, які під час асемблювання об'єднуються в один. Тому, щоб МК «знав» з яким блоком (сегментом) пам'яті йому треба працювати при виконанні якого-небудь фрагмента програми, цей сегмент заздалегідь треба оголосити. Після оголошення сегмент стає поточним. Це значить, що всі подальші оператори асемблера відноситимуться до оголошеного сегменту пам'яті. Оголошений сегмент залишатиметься поточним до тих пір, поки не буде оголошений який-небудь інший сегмент. Якщо жоден сегмент пам'яті не вказаний, то за замовчаванням ним є сегмент CSEG. Цей сегмент, оголошений за умовчанням або директивою CSEG, описує команди, які потім у вигляді кодів будуть поміщені в пам'ять програм. Оскільки команди в програмній пам'яті повинні розташовуватися по порядку, то процес їх розміщення автоматизований. Програміст просто послідовно пише команди, не указуючи адресу, за якою ці команди будуть розміщені (табл. 4.1.).

Під час асемблювання команди автоматично розміщуються у флэш-пам'яті. Для забезпечення даного процесу використовується віртуальний лічильник адрес (покажчик поточної адреси), вказуючий на поточний 16-розрядний елемент пам'яті програм. Покажчик дозволяє розмістити всі команди по елементах пам'яті. За замовчуванням вважається, що на початку програми значення адреси поточного покажчика рівне нулю (встановити лічильник на нульову адресу можна також записавши два рядки . CSEG .ORG 0). Якщо перша команда буде розміщена за нульовою адресою, то у міру асемблювання програми покажчик зміщується у бік збільшення адреси.

Таблиця 4.1.

Директиви асемблера

Директива	Призначення директиви і короткий опис
.CSEG	Встановлює початок поточного сегменту коду у флеш-пам'яті команд
.DSEG	Встановлює початок сегменту коду в SRAM (ОЗУ)
.ESEG	Встановлює початок сегменту коду в EEPROM
.ORG	Визначає абсолютну адресу
.BYTE	Відводить для змінної місце в пам'яті
.DB	Визначає однобайтову (і) константу (и)
.DW	Визначає двобайтову (і) константу (и)
.DEF	Призначає регістру символічне ім'я
.EQU	Зіставляє символічне ім'я з виразом
.SET	Призначає символічне ім'я деякому виразу
.DEVICE	Повідомляє асемблер про тип використовуваного МК
.INCLUDE	Вставляє початковий код з іншого файлу
.LIST	Створює файл лістингу
.NONLIST	Відміняє створення файлу лістингу
.MACRO	Начало макросу
.ENDMACRO	Кінець макросу
.LISTMAC	Відображає код макросу в лістингу
.EXIT	Завершує асемблювання файлу

Директива «ORG» (від ORiGin – начало) задає абсолютну адресу в об'єктній програмі. Вона просто указує асемблеру, де начало програми у флеш-пам'яті програм. Крім того, директивою «ORG» у сегменті даних, позначеному директивою DSEG, встановлюється віртуальний лічильник адреси в пам'яті SRAM, а в сегменті EEPROM, позначеному директивою ESEG – віртуальний лічильник адреси в пам'яті EEPROM. Директива «ORG» встановлює лічильник адреси поточного сегменту на абсолютну адресу exp. Якщо дана директива не вказана, то лічильник адреси команд і лічильник адреси в EEPROM спочатку містять 0, тоді як лічильник адреси в пам'яті SRAM починається із значення 96 ($60). Лічильники ад-

реси в EEPROM і SRAM адресуються до 8-розрядних комірок, тоді як лічильник адреси команд адресується до 16-розрядних елементів пам'яті.

Директива «DSEG» визначає факт використання сегменту оперативної пам'яті ОЗУ (SRAM). У цьому випадку сегмент даних складається з директив .BYTE і міток. При цьому також використовується власний віртуальний лічильник адреси, вказуючий на восьмирозрядні елементи пам'яті (байти). За допомогою директиви «ORG» лічильник може бути встановлений на певну абсолютну адресу пам'яті.

Директива «BYTE» застосовується в сегменті даних DSEG і ESEG. Вона відводить змінній місце в пам'яті. Її формат [мітка: .BYTE exp]. Мітка є ім'ям змінної. За замовчуванням змінна «прив'язується» до адреси першого зарезервованого елементу пам'яті простору DSEG. Кількість зарезервованих байтів в пам'яті визначається по значенню виразу exp. Число резервованих комірок указується в полі операндів, а символічна адреса першої комірки групи – в полі мітки. Приклад використання директив «DSEG» і «BYTE»:

```
.DSEG
.ORG    0x60
Var1:   .BYTE 1 ; резервуємо 1 байт для змінної
Var2:    .BYTE 16 ; резервуємо таблицю із16 байтів
```

Для першого байта відводиться елемент пам'яті з адресою Var1 (0x60). Цю адресу пам'яті неможливо змінити, або скинути на 0. Наступний вільний елемент пам'яті знаходитиметься за адресою Var2 = Var1+1. Оскільки потім резервується 16 елементів пам'яті, то адреса чергової комірки знаходитиметься за адресою «метка+вираз», тобто, за адресою 1+16.

Директива «ESEG» визначає факт використання сегменту пам'яті даних EEPROM. В цьому випадку сегмент даних складається виключно з директив .BYTE і міток. Сегмент пам'яті даних EEPROM використовує власний віртуальний лічильник адреси, вказуючий на восьмирозрядні елементи пам'яті. За допомогою директиви «ORG» лічильник може бути встановлений на певну адресу пам'яті EEPROM. Приклад використання директив «DSEG» і «BYTE»:

```
. ESEG
.ORG    0x08
Var1:   .BYTE 1 ; резервуємо 1 байт для змінної
Var2:    .BYTE 10 ; резервуємо таблицю із10 байтів
```

Директива опису даних «DB» (Define a Byte – визначити байт) дозволяє асемблеру сформувати константу або константи і помістити її (їх)

в елемент(и) пам'яті програм або EEPROM. За допомогою директиви такого типу можна зарезервувати групу елементів пам'яті і дати цій групі символічне ім'я. Це виявляється вельми корисним, коли потрібно відвести місце для вхідних, вихідних або проміжних даних. Формат директиви опису даних «DB» – [мітка: .DB exp -list]. Як exp –list може виступати: а) ряд розділених комами констант або виразів, які можуть містити арифметичні і логічні операції і повинні знаходитися в діапазоні від – 128 до 255 (негативні вирази записуються в пам'яті у вигляді доповнення до двох); б) послідовність ASCII-символів, поміщених у лапки. Значення окремих байтів, які потрібно помістити в пам'ять, записуються праворуч від директиви «DB» через кому. Директива «DB» може застосовуватися у сегменті флеш-пам'яті програм CSEG і в сегменті ESEG незалежної пам'яті даних EEPROM. Якщо байти повинні зберігатися в шістнадцятирозрядної пам'яті CSEG, комірка якої вміщає два байти, а exp – list містить два вирази (парне число), то перший вираз зберігається в молодшому байті слова, другий – в старшому. Якщо кількість виразів непарна, наприклад, одне, то воно запам'ятовується в молодшому байті слова, а старший байт заповнюється нулями. Треба мати на увазі наступне. Якщо необхідно записати в пам'ять велику, але непарну кількість значень, то треба так розбіти всі дані на декілька рядків, щоб у кожному рядку було парне число байтів. Тільки в останньому рядку допускається залишити непарне число значень – байтів. Приклад:

```
.CSEG
konst1:   .db $1A, $2F, 14, 12, 2/3, “ABCD“
konst2:   .db 1, 2, 3, 4, 5, 6, 7, 8
            .db 9
Konst3:   .db 4+$1A $2F &$1A, 14, 0b01100011<<1
. ESEG
Konst4:   .db -3
```

Директива опису даних «DW» (Define a Word – визначити слово) дозволяє асемблеру сформувати константу або константи розміром у слово (16 біт) і помістити її (їх) в елемент(и) пам'яті програм або в сегменті пам'яті EEPROM. За допомогою директиви такого типу можна зарезервувати групу елементів пам'яті і дати цій групі символічне ім'я. Це виявляється вельми корисним, коли потрібно відвести місце для вхідних, вихідних або проміжних даних. Формат директиви опису даних «DW» – [мітка: .DW exp– list]. Як exp – list може виступати ряд розділених комами виразів, які можуть містити арифметичні і логічні операції і повинні зна-

ходитися в діапазоні від – 32768 до 65535 (негативні вирази записуються в пам'яті у вигляді доповнення до двох). Значення окремих слів, які потрібно помістити в пам'ять, записуються праворуч від директиви «DW» через кому. Директива «DW» може застосовуватися в сегменті флеш-пам'яті програм CSEG і в сегменті ESEG незалежної пам'яті даних EEPROM. Якщо у восьмирозрядні комірки ESEG записуються слова, то спочатку записується молодший байт слова, а старший байт слова розміщується за адресою, наступною за адресою молодшого байта. Приклад використання директив «DW»:

```
.CSEG
Konst5:   .dw $1A2F,196512/3
Konst6:   .dw 1024+$0B $FF2F&$1A24, $1A2F>> 4
. ESEG
Konst7:   .dw -3, 100, $100
```

Директива «DEF» **(Define** – визначити) призначає регістру символічне ім'я, за яким решта частини програми може звертатися до цього регістра. Регістр може мати декілька символічних імен, а позначення регістра може бути пізніше в програмі перевизначено. Директива дозволяє присвоювати різним регістрам МК будь-які осмислені імена, спрощуючи читання і розуміння тексту програми. Наприклад, регістру, призначеному для часового зберігання даних, зручно присвоїти ім'я temp. Формат директиви опису даних «DEF» –[Позначення = регістр]. Приклад:

```
.def    temp = r16    ; Визначення головного робітника регістра
.def    work = r16    ; До r16 можна звертатися по двох іменах
.def    temp = r17    ; Перевизначення головного робітника регістра
```

Директива «EQU» (Equate – вважати рівним) призначає виразу деяке символічне ім'я, за яким решта частини програми може звертатися до цього виразу. Формат директиви «EQU» – [Позначення = exp], де exp – простий арифметичний або логічний вираз. Директива «EQU» присвоює позначенню деяке значення. Решта частини програми може оперувати з позначенням як з виразом. Оскільки в даному випадку йдеться про визначення константи, то позначенню не можна пізніше в програмі присвоїти нове значення. Приклад використання:

```
.equ    offset = $0100+100
.equ    divv = 256/8
.equ    next = divv<<2
```

Директива «SET» (Set – встановити в певний стан) призначає виразу деяке символічне ім'я, за яким надалі можна звертатися як до значен-

ня. Формат директиви «SET» –[Позначення = exp], де exp – простий арифметичний або логічний вираз. Директива «SET» присвоює позначенню деяке значення. Решта частини програми може оперувати з позначенням як з виразом. Приклад використання:

```
. set    divv = 256/8
. set    next = divv<<2
```

Директива «DEVICE» (Device – пристрій) повідомляє асемблер про те, на якому типі мікроконтролера повинна виконуватися дана програма. Якщо в початковому файлі з'являється команда, не підтримувана даним мікроконтролером, або відбувається вихід за межі пам'яті, то асемблер видає в своєму вікні повідомлень і в лістингу програми відповідне попередження. Якщо директива «DEVICE» відсутня, то асемблер допускає, що дозволені всі команди, а межі пам'яті відсутні. Приклад використання:

```
. device        AT90S8515
```

Директива «INCLUDE» (Inclusion – включення) повідомляє асемблер про те, що в початковий файл повинен бути вставлений іншій, додатковий файл. Вміст файлу, що включається, треба вставити в те місце програми, де використана дана директива. Формат директиви «INCLUDE» – [«INCLUDE» ім'я файлу]. Директива «INCLUDE» вставляє в цьому місці програми вміст іншого початкового файлу, а потім указує асемблеру транслювати файл, що вставляється, до його закінчення або до директиви «EXIT» і додати одержаний об'єктний код до початкової програми. Файл, що вставляється, у свою чергу, також може включати власні файли, що вставляються. Приклад:

```
.include "2313def.inc"        ; Приєднання файлу описів МК.
```

Директива «LIST» (List – вести список) повідомляє асемблер про те, що з цієї миті повинно бути почато створення лістингу програми. Лістинг – це створюваний разом з об'єктним і завантажувальним спеціальний файл, який містить команди з початкового файлу разом з відповідними їм кодами операцій і адресами. У цьому файлі також відображається весь хід асемблювання програми. У лістингу повторюється текст програми, включаючи всі приєднані фрагменти. Проти кожного рядка програми, що містить реальну команду, поміщаються відповідні їй машинні коди. Там же показуються знайдені в процесі трансляції помилки. За замовчуванням лістинг не формується, тому якщо він потрібен, то повинна бути використана відповідна директива. Приклад використання:

```
.list                                ; Включення лістингу
NONLIST
```

Директива NONLIST» відключає створення лістингу програми. Приклад використання:

```
. nonlist                    ; Відключення лістингу
```

Директива «MACRO» повідомляє асемблер про те, що в цьому місці програми починається макрос – набір команд, що виконуються при виклику макросу. Макрос подібний підпрограмі, але на відміну від підпрограми, яка записується в пам'ять тільки один раз, команди макросу розміщуються в пам'яті стільки разів, скільки викликається макрос. Макроси звичайно застосовуються у тих фрагментах програми, які критичні за часом їх виконання. Тобто тоді, де три тактові імпульси для виконання команди виклику підпрограми recall і чотири для виконання команди виходу з підпрограми ret недозволена розкіш. Макроси застосовуються також в тих випадках, коли повинні виконуватися кращі і максимально ефективніші власні команди користувача. Макрос складається з двох об'єктів – макроозначення і макропідстановки. У макроозначенні задана послідовність операторів і їй привласнено призначене для користувача ім'я. Після того, як макрокоманда визначена, ім'я макросу задає підстановку послідовності операторів з визначення. Визначення складається з імені, ключового слова macro, «тіла визначення» і завершшальної директиви endmacro. Завжди, коли в програмі зустрічається ім'я макросу, то макрос (що представляє послідовність команд) як би «розгортається» (тобто його команди вставляються в програму). При виклику макросу до нього можуть додаватися параметри. Параметри в макросі дозволяють настроювати підстановку, тобто змінювати формований фрагмент залежно від параметрів виклику макросу. Формальні параметри, тобто параметри у визначенні макросу, мають порядкові номери. Макрос може приймати до десяти параметрів. Ці параметри у визначенні макросу пронумеровані @0 до @9 у порядку їх проходження. При виклику макросу параметри передаються у вигляді окремого списку і відділяються один від одного комами. При виклику макросу фактичні параметри записуються після імені макросу. Визначення макросу закінчується директивою «ENDMACRO». Якщо команди макросу повинні відображатися в кожному місці його виклику, то в початковому файлі повинна бути вказана директива «LISTMAC», яка повідомляє асемблер про те, що в цьому місці лістингу програми при виклику макросу повинні бути вставлені всі його команди. За замовчуванням указуються імена і параметри макросу. Приклад:

```
.macro subi16                   ; Початок макроозначення
        subi @1, low(@0)        Відняти константу – слово
        sbci @2, high(@0)       ; з регістрової пари
.endmacro                       ; Кінець  макроозначення
.listmac
```

Директива «EXIT» завершує асемблювання файлу. Вона указує асемблеру завершити в цьому місці процес асемблювання, не досягнувши кінця файлу, як це відбувається в звичайному режимі.

4.3. Програмне забезпечення мікроконтролера AVR на мові CI

4.3.1. Роль мови високого рівня CI в прикладному програмному забезпеченні мікроконтролера

Обробка даних – найважливіше завдання МК. Відомо, що для того, щоб обробіти інформацію на МК, потрібно крок за кроком описати весь процес обробки. Такий опис у вигляді послідовності команд називається програмою. Кожна команда в загальному випадку містить вказівку того, що повинне бути зроблене (код операції), і визначення дії, тобто операнд(и). Сукупність програм, написаних для МК, складає його програмне забезпечення. Програма, що виконується МК, повинна складатися тільки з тих команд, які передбачені його конструкцією. Сукупність цих команд складає так звану машинну мову, або мову машинних команд даного МК.

Нескладно здогадатися, що при програмуванні «незграбним» з погляду людини способом, на машинній мові низького рівня (у машинних або шістнадцятирічних кодах), навіть нескладні завдання зажадають трудомісткої і кропіткої праці. Тому були створені спеціальні засоби, що полегшують підготовку програм на машинній мові. Складати програми набагато зручніше, якщо разом з символічними іменами для команд, званих мнемокодами, можна вживати для позначення адрес у командах також символічні імена. Тому набагато легше опинилося писати програму на мові Асемблера. Хоча мова Асемблера значно зручніше для програмування, чим машинна, проте, і вона теж не є ідеальною. Процес програмування на Асемблері – заняття вельми кропітке, стмлююче і часто займає вельми багато часу. Подальше скорочення трудомісткості і ще більші зручності при написанні програм дає використання мов високого рівня. Працюючи з такими мовами програмування, користувач лише ви-

значає, які складні математичні дії або операції над даними повинні бути проведені, не займаючись «програмуванням деталей» у звичайному сенсі цього слова. Ці мови були розроблені для «великих» комп'ютерів і відрізняються тим, що вони значно більше орієнтовані на людину і з цієї причини відносно слабо пов'язані з конкретною структурою процесора і його конкретними командами. Такі мови оперують не вмістом регістрів, а звичнішими нам десятковими числами, змінними, константами і іншими елементами. При написанні програм на мовах високого рівня константи і змінні можуть приймати звичні для нас значення, бути позитивними, негативними, дробовими і т. д. Більш того, деякі команди МК можуть бути згруповані разом, як якийсь вираз, форматом в один рядок. Використовуючи відносно невелике число конструкцій, застосовуючи при цьому в додатках стандартні бібліотеки, за допомогою мови високого рівня можна будувати крупні і складні програми. МК, що використовує мову високого рівня, набуває здатності реалізовувати в програмі складні команди, такі, як наприклад, обчислення складних функцій, управління за допомогою галужень, логічні і умовні переходи, а не тільки виконувати звичайний набір арифметичних операцій з восьмирозрядними двійковими числами і слідувати строго фіксованому порядку виконання команд.

Ми вже відзначали, що спочатку при розробці апаратних засобів МК AVR була одержана архітектура МК, спеціально розрахована на складання програм на мові Сі. Мова Сі (була розроблена Деннісом Річи на початку 70-х років двадцятого сторіччя) завоювала особливу популярність у програмістів завдяки унікальному поєднанню можливостей мов високого і низького рівня. Основна причина, за якою Сі став популярним, полягає у високій швидкості виконання одержуваного коду і його компактності, що особливо цінно для професійних додатків, орієнтованих на використання в МК, для яких і до теперішнього часу швидкодія і об'єм пам'яті є критичними показниками. Сі можна назвати мовою асемблера високого рівня. Це цілком правомірно, оскільки Сі відкриває програмісту доступ до «нутрощів» МК – бітам, байтам і регістрам, що управляють роботою центрального процесорного ядра і зовнішніх пристроїв. Але Сі все-таки представляє щось більше, ніж мова асемблера високого рівня. Блокова структура програми на Сі забезпечує як захист даних, так і високий рівень контролю за областями дії і видимості змінних. Сі підтримує багато важливих структур даних високого рівня і структури, що управля-

ють, стали звичними в сучасних мовах програмування. На Сі можна створювати великі і складні програми.

Слід мати на увазі, що мова СІ – не найкраща мова для початкового ознайомлення з програмуванням. Існують простіші і могутніші мови, такі як Бейсік, використання яких дозволяє освоїти «хороший» стиль програмування, але не вимагає від програміста детального знайомства з конкретним МК. Могутні засоби вимагають від програміста обережності, акуратності і хорошого знання мови зі всіма його перевагами і недоліками. У порівнянні з менш могутніми і простішими мовами СІ представляє програмісту високий рівень гнучкості, покладаючи на нього високу відповідальність. Збільшення гнучкості може сприяти побудові компактного і ефективного коду, але може привести до заплутаної програми, яка трудно читається. Слід зазначити ще одну проблему мови СІ – недостатню «читабельність тексту» програми. Але, ймовірно, це не проблема самого СІ, а швидше того, хто програмує на СІ.

Для того, щоб початкова програма на СІ була відтрансльована і переведена у виконуваний машинний код МК, вона повинна пройти через три процеси: препроцессорування, компіляцію і завантаження (збірку). У завдання препроцесора входить підключення при необхідності до даної програми на СІ зовнішніх файлів, що указуються за допомогою директив, які мають на початку символ «#». Препроцесор (макропроцесор) – це складова частина мови СІ, яка обробляє початковий текст програми до того, як він пройде через компілятор. Препроцесор читає рядки тексту і виконує дії, визначені командними рядками. Якщо першим символом у рядку, відмінним від пропуску, є символ #, то такий рядок розглядається препроцесором, як командна. Командні рядки називаються директивами препроцесора. Директиви препроцесора дозволяють: 1) включати в програму текст з інших файлів; 2) передавати компілятору спеціальні директиви; 3) визначати макроси, які полегшують програмування і покращують читаність початкового коду; 4) задавати умови компіляції для налагоджувальних цілей і/або для зменшення розміру одержуваного коду. Компілятор за кілька разів транслює те, що виробляє препроцесор, в об'єктний файл, який містить оптимізований машинний код, за умови, що не зустрілися синтаксичні і семантичні помилки. Якщо в початковому файлі з програмою на СІ виявляються помилки, то програмісту видається їх список, в якому помилки прив'язуються до номера рядка, в якому вони з'явилися. Програміст циклічно виконує дії з редагування/компіляції, поки

не будуть усунені всі помилки в початковому файлі. Завантажувач зв'язує між собою об'єктний файл, що одержується від компілятора, з програмами з необхідних бібліотек і, можливо, з іншими файлами. У результаті збірки виходить файл, який може бути запущений для виконання.

Отже, при експлуатації сучасних поліграфічних машин і введенні до ладу нових систем комп'ютеризованого обладнання з МК доведеться «зіткнутися» з тим, що коди програм у них можуть бути написані на мові зрозумілій для користувачів помірної кваліфікації у області програмування – на мові високого рівня СІ.

4.4. Можливості мови СІ в прикладному програмному забезпеченні мікроконтролера AVR

4.4.1. Структура програми на мові СІ, константи і директиви препроцесора

Як відомо, будь-яка програма являє якусь послідовність інструкцій машинного коду, керуючих поведінкою МК, як програмно – апаратного засобу. На відміну від Асемблера програма на мові СІ абстрагована від системи команд МК і основні оператори мови СІ не «прив'язані» до простих інструкцій МК. Тому навіть для реалізації однієї команди на мові СІ компілятор породжує декілька команд МК, а іноді, невелику програму. У мові СІ, щоб полегшити працю програміста, «працюють» з «крупними категоріями». Для кожної команди СІ використовують «вирази», «операторів», функції. При цьому форма написання команд у СІ наближається до форми, прийнятої в математиці. У результаті програмісту не доводиться вдаватися в «дрібні подробиці» і він може зосередитися на головному – на суті програми. Зокрема, мова СІ надає незвичайно високу гнучкість для «логічної» організації програми або програмної системи. Типова логічна організація простої програми на мові СІ показана на рис. 4.8. Фрагменти програми, помічені штрихами, можуть бути відсутніми.

Текст програми на мові СІ називають початковим («ісходником»). Він звичайно починається «шапкою» з назвою програми, що дозволяє відрізняти програми. У «шапку» можна помістити версію програми, дату написання і інші потрібні відомості. «Шапка» представляє декілька рядків коментаря. Особливо важливо прокоментувати свої дії, якщо програма

досить складна, в ній використані нетривіальні алгоритми. Наявність у тексті програми коментаря важливо і для складних комп'ютеризованих систем, які в процесі свого життєвого циклу можуть кілька разів модернізуватися. Треба, щоб про програму могли одержати відомості люди, що проектують інші модулі обладнання, фахівці, що експлуатують і модернізують систему. Щоб уникнути неприємностей, пов'язаних з відсутністю інформації про програму, в процесі її складання використовують коментарі. Вважається «правилом хорошого тону» вводити докладні текстові коментарі. Компілятор завжди ігнорує текст коментарів, що дозволяє програмісту детально висловлювати відомості про програму.

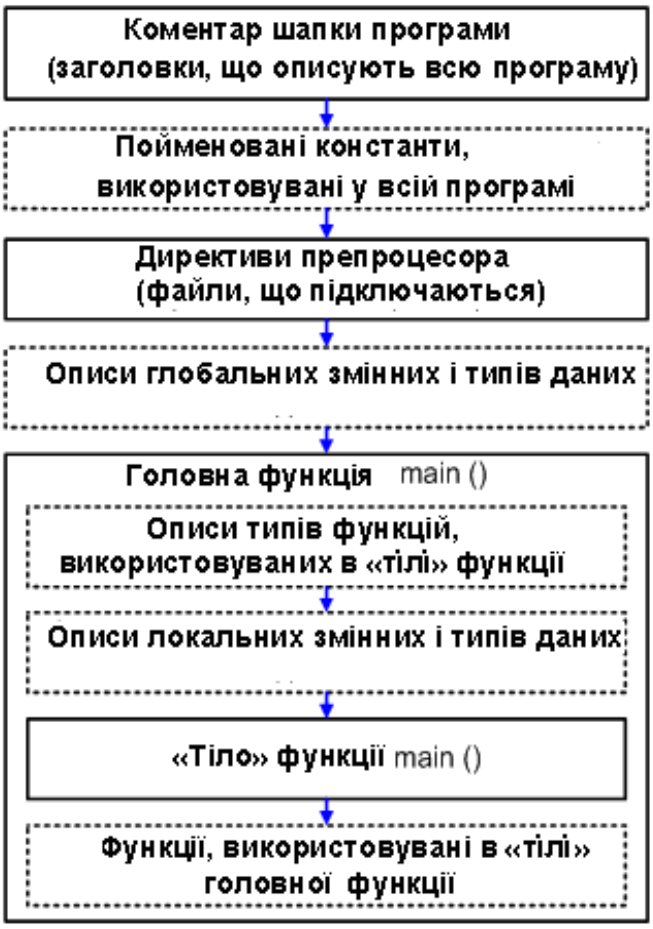


Рис. 4.8. **Логічна організація простої програми на мові СІ**

Розмір коментарів обмежений розміром вільної пам'яті комп'ютера, але при цьому не варто перетворювати текст програми, що прояснює код, у літературний «опис». У мові СІ використовується два різновиди коментарів. Перший, багаторядковий коментар, є блок, що починається з послідовності символів (/*) і закінчується символами (*/). Наприклад:

```
/* Початок коментаря
```

КОМЕНТАР
Кінець коментаря */

Для підвищення читабельності тексту програми в коментарях рекомендується використовувати символи ******.

Другий вид – однорядковий коментар – слідує за «подвійним слешем» (//). Окрім пояснення тексту програми, даний вид коментаря може бути використаний для тимчасового виключення з програми деякої її частини. Цей прийом звичайно використовується при відладці.

За «шапкою» можуть слідувати рядки, що характеризують константи, використовувані у всій програмі. Оскільки константи не можуть бути оголошені у функціях, їх «виносять» за «тіло» функції. Константа, як об'єкт програми, – це заздалегідь певне числове або символьне значення, представлене унікальним ім'ям. При цьому значення, привласнене константі спочатку, не може бути змінено впродовж всієї програми. Не можна в ході програми поміняти ім'я константи. Ім'я константи може містити букви латинського алфавіту (A, B,Z) і цифри (0.9), а також символ підкреслення (_), який вважається буквою. У мові CI можуть бути використані чотири типу констант: цілі константи; константи з плаваючою крапкою; символьні константи; рядкові літерали. Ціла константа – це десяткове, двійкове, вісімкове або шістнадцятирічне число, яке являє собою цілу величину в одній з наступних форм: десяткової, двійкової, вісімкової або шістнадцятирічної. Десяткова константа складається з однієї або декількох десятирічних цифр, причому перша цифра не повинна бути нулем (інакше число буде сприйнято як вісімкове). Наприклад, int_b=15. Двійкова константа починається з обов'язкової послідовності 0b і містить одну або декілька двійкових цифр. Наприклад int_b1=0b101011. Шістнадцятирічна константа починається з обов'язкової послідовності 0x або 0X і містить одну або декілька Шістнадцятирічних цифр. Наприклад, 0xea, 0XEA. Якщо потрібно сформувати негативну цілу константу, то перед записом константи використовують знак «–», який називається унарним мінусом. Кожній цілій константі привласнюється тип, що визначає перетворення, які повинні бути виконані, якщо константа використовується у виразі. Тип константи залежить від її значення.

Щоб цілу константу визначити типом long integer (довга ціла), достатньо в кінці константи поставити суфікс «l» або «L». Наприклад, 12l, 0x3FAL. Щоб цілу константу визначити типом unsigned integer (цілі без знаку), достатньо в кінці константи поставити суфікс «U». Наприклад:

12U, 0b10101U. Для визначення цілої константи типом unsigned long integer (довга ціла без знаку), достатньо в кінці константи поставити суфікс «UL». НаПриклад:12UL.

Константа з плаваючою крапкою (floating point) – десяткове число, представлене у вигляді дійсної величини з десятковою крапкою або експонентою. Формат має вигляд: [цифри].[цифри] [E|e][+|-]цифри] 1.6E3. Числа з плаваючою крапкою складається з цілої і дробової частини і/або експоненти. Константи з плаваючою крапкою представляють позитивні величини подвоєної точності (мають тип double). Для визначення негативної величини слід сформувати константний вираз, що складається із знака мінуса і позитивної константи. Наприклад, -1.6E-3. Щоб константу визначити типом floating point достатньо в кінці константи поставити суфікс «F». Наприклад, 9.14 F.

Символьна константа (Chracter) – представляється символом, поміщеним в одиночні лапки. Значенням символьної константи є числовий код символу. Наприклад, 'g'.

Рядкова константа (String) або літерал – послідовність символів, включаючи маленькі і великі букви російського і латинського алфавіту, а також цифри, поміщені в лапки. Наприклад, «Сидоров», «SIDOROV1224». У кінець кожного рядкового літерала компілятором додається нульовий символ.

Рядковий літерал має тип char [ ]. Це означає, що рядок розглядається як масив символів. Слід вказати на важливу особливість: кількість елементів масиву дорівнює кількості символів у рядку плюс один, оскільки нульовий символ (символ кінця рядка) також є елементом масиву.

Відповідно до Гарвардської архітектури константи можуть зберігатися в трьох видах пам'яті. Щоб визначити константи, що зберігаються у FLASH-пам'яті, слід використовувати ключові слова flash або const. Для визначення констант, що зберігаються в EEPROM-пам'яті, використовують ключове слово eeprom. Для визначення константи, яка зберігається в регістрі, використовують ключове слово register. Приклад:

```
    /* Ці константи будуть розташовані в EEPROM */
eeprom int int_const1=12-7; // Ціла константа int_const1 буде рівна 5
eeprom char char_const='b'; //Символ. константа char_const буде рівна
// числовому коду ASCII символу 'b', тобто 0x62
/* Ці константи будуть розташовані в FLASH пам'яті */
flash long long_int_const=136L;  // Ціла довга константа
// long_int_const буде рівна 136
```

```
flash int int_array[10]={9,16};       // Цей цілочисельний масив
// має 10 членів, причому перші два
// це 9 і 16, а решта нулів
const int int_const2=0123; // Ціла константа int_const2 буде рівна 123
// у вісімковому, або 83 в десятковому вигляді
/* Ця константа буде розташована в регістрі */
register int int_const3=0xl0;         // Ціла константа int_const3 буде рівна
// 16 у десятковому, або 10 в
// шістнадцятирічному вигляді
/* Ці константи будуть розташовані в SRAM */
int int_const4=0bl0;          // Ціла константа int_const4 буде рівна
// 2 у десятковому, або 10 у двійковому вигляді
char string_const[]="Hello!";         // Це рядкова константа
```

У тексті програми можуть бути рядки, помічені символом **#**. Це командні рядки, що використовуються препроцесором. Коли препроцесор читає ці рядки тексту, то він починає виконувати дії, визначені командними рядками. Командні рядки називаються директивами препроцесора. Всі директиви починаються зі знаку **#.** Після директив препроцесора крапка з комою не ставиться. У багато програм на СІ підставляються один або декілька файлів, що полегшують процес програмування. Вказівку на те, що в певне місце програми необхідно додати файл, дає директива #include. Ця можливість препроцесора дозволяє слідувати в мові СІ ідеям структурного програмування, згідно яким велика програма розчленовується на логічно завершені частини, і потім кожна оформляється як самостійна функція. Після відладки кожна з них оформляється у вигляді окремого файлу і при необхідності включається у відладжену програму директивою #include. Поява директив:

```
#include <файл1>
#include <файл2>
#include <файл3>
```

призводить до того, що препроцесор підставляє на місце цих директив тексти файлів: файл1, файл2, файл3. Директива #include включає в програму вміст вказаних файлів. Директива #include широко використовується для включення в програму так званих заголовних файлів (файлів з розширенням .h, або так званих «хидерів»), що містять визначення периферійних пристроїв і векторів переривань використовуваного мікроконтролера, прототипи функцій визначених користувачем. Приєднання файлів до основного тексту програми виглядатимуть таким чином: текстів описів для МК ATtyni2313 #include <tyni2313.h>, файлів бібліотеки

функцій, що реалізовують затримку #include delay.h. Зустрівши #include <tyni2313.h>, препроцесор, перед компіляцією, вставить замість цього рядка вміст (текст) заголовного файлу «хидера» МК ATtyni2313. Цей файл містить перелік регістрів, що є у МК ATtyni2313 і відповідність їх назв їх фізичним адресам у МК. Замість строчки #include delay.h, препроцесор, перед компіляцією, вставить текст «хидера» delay.h. Цей файл містить функції для створення пауз. Після під'єднування такого «хидера» для того, щоб зробіти паузу в програмі досить буде лише написати: delay_us (N) – зробіти паузу в N мікросекунд; delay_ms (x) – зробіти паузу в x мілісекунд.

Директива #include може зустрічатися в будь-якому місці програми, але звичайно всі включення розміщуються на початку початкового файлу. Директива #include має дві синтаксичні форми:

#include <файл1>
#include “файл2”.

Якщо ім'я файлу задане в кутових дужках, то компілятор шукатиме цей файл у деякому стандартному каталозі. Якщо ж ім'я файлу вказане в лапках, то компілятор шукатиме цей файл у поточному каталозі (у якому міститься основний файл початкового тексту).

Для заміни одних лексографічних одиниць мови СІ (констант, ключових слів, виразів і операторів), що часто використовуються за інші, звані ідентифікаторами (ідентифікатор – ім'я, що складається з букв латинського алфавіту і цифр, яке дається змінній, функції, мітці або іншому об'єкту), служить директива #define. Ідентифікатори, замінюючі текстові або числові константи, називають іменованими константами. Ідентифікатори, замінюючі фрагменти програм, називають макроозначеннями (макроозначення можуть мати аргументи). Директива #define має дві синтаксичні форми:

#define ідентифікатор текст
#define ідентифікатор (список параметрів) текст.

У обох синтаксичних формах текст може являти собою будь-який фрагмент програми на СІ. За допомогою першої синтаксичної форми директиви #define, услід за якою пишеться ідентифікатор (ім'я макро) і текст (значення макро), можна вказати процесору, щоб він, при будь-якій появі в початковому файлі даного ідентифікатора, замінив його ім'я на відповідне значення тексту (на відповідне значення макро). Наприклад, директива #define pi 3.1415926, пов'язує ідентифікатор pi із значенням

3.1415926 (підставить замість змінної в тексті програми число). За допомогою директиви директиви #define можна визначати навіть складні функції. Наприклад, відповідно до директив #define A 15, #define B (A+20), препроцесор замінить всі ідентифікатори, що зустрічаються в програмі на відповідний їм текст (A на число 15, а B - на вираз 15+20). Такий процес називається макропідстановкою. У другій синтаксичній формі в директиві #define є список формальних параметрів, який може містити один або декілька ідентифікаторів, розділених комами. Формальні параметри в тексті макроозначення відзначають позиції, на які повинні бути підставлені фактичні аргументи макровиклику. Кожен формальний параметр може з'явитися в тексті макроозначення кілька разів. При макровиклику услід за ідентифікатором записується список фактичних аргументів, кількість яких повинна співпадати з кількістю формальних параметрів. Наприклад, #define X(a,b,c) ((a)*(b)-(c)). Препроцесор відповідно до цієї директиви замінить фрагмент Y=X(r+m,r-m,n) Y=X((r+m) *(r-m)-(n)).

Важливим завданням при програмуванні є вивід у файл. У Сі такий вивід реалізується через зовнішні функції і макропідстановки. Щоб перетворити параметр макроозначення в символьний рядок використовують функцію printf(#x). Наприклад, якщо записати #define PRINT_STRING(x) printf(#x), то препроцесор відповідно до цієї директиви замінить фрагмент PRINT_STRING(Good) на фрагмент printf(“Good “). Щоб з'єднати двох операторів в один, можна використовувати оператора **##**. Наприклад, при #define NAME(a,b) a ## b препроцесор замінить фрагмент char NAME(Hello,World=1) на фрагмент char HelloWorld =1. Директиву #define можна перенести на новий рядок, використовуючи символ \. Наприклад:

```
#define STRING "Це дуже \
довгий рядок..."
```

Директиви #asm можна використовувати для включення в будь-якому місці початкової програми асемблерного коду. Наприклад:

```
/* Вставка асемблерного коду */
void main(void)
{
#asm  // початок асемблерного коду
sei     ; дозволяємо глобальні переривання
#endasm      // закінчення асемблерного коду
}
```

```
#asm("sei") // Дозволити глобально всі переривання
#asm("cli") // Заборонити глобально всі переривання
```

```
#asm("nop") // Пауза в 1 такт процесора
#asm("wdr") // Скинути сторожовий таймер
```

Слід при цьому звернути увагу, що коментар в асемблерній частині коду відділяється крапкою з комою. У довідниках можна знайти й інші директиви препроцесора. Після опису директив препроцесора слідує блок описів змінних і типів даних.

4.4.2. Змінні і типи даних мови CI

Суть будь-якої програми зводиться до введення, зберігання модифікування і виведення деякої інформації. Щоб програма могла впродовж свого виконання зберігати певні дані і оперувати з ними, служать змінні і константи (заздалегідь визначені і незмінні числові або символьні значення, представлені унікальним ім'ям). Представлена унікальним ім'ям змінна, як об'єкт програми, є заздалегідь певне числове або символьне значення, що займає в загальному випадку декілька елементів пам'яті, покликаних зберігати дані. Тому змінна володіє ім'ям, розміром і рядом інших, властивих їй властивостей. Наприклад, у програмі може використовуватися змінна з ім'ям "x". Під час виконання програми змінні, на відміну від констант, можуть безліч разів змінювати свої значення. Ім'я у змінної постійно і незмінно, а значення її може мінятися. Над змінною можуть виконуватися різні дії (різні операції). Тому, щоб охарактеризувати ту безліч станів, яка може приймати змінна і безліч операцій, які над ним допустимі, вводять поняття типу змінної. Так наприклад, з одного боку, станом (значенням) змінної може бути число (наприклад, будь-яке дійсне число), над яким допустимі будь-які арифметичні операції. З іншого боку, в програмі можуть використовуватися букви. Тоді над ними арифметичні операції не допустимі. У програмі треба заздалегідь указати, що уявляють собою використовувані змінні, який їх тип. По-іншому, перед використанням змінної в програмі CI треба вказати компілятору як називається змінна, і який тип даних вона може зберігати.

Процес оголошення змінної здійснюється, у першу чергу, за рахунок створення її ідентифікатора. Ідентифікатором є ім'я, яке дається змінній, що містить букви латинського алфавіту (A, B .,Z, a, b .,z) і цифри (0.9), а також символ підкреслення, який вважається буквою. Ідентифікатори (наприклад, з іменами identiv1, temp) можуть починатися тільки з букви або із знаку підкреслення. При цьому бажано використовувати

змістовні імена для позначення змінних, наприклад speed_1 для змінної, що позначає швидкість першого об'єкта. При написанні імені ідентифікатора має значення великі або малі це букви. Наприклад, імена temp, Temp, TEMP сприйматимуться компілятором як різні ідентифікатори. Як ідентифікатор не можна використовувати зарезервовані для компілятора ключові слова, їх список можна знайти в довідковій літературі. При оголошенні змінної для неї резервується деяка область пам'яті, розмір якої залежить від того, який тип привласнений змінній. Щоб уникнути проблем, пов'язаних з переповнюванням або неправильною інтерпретацією даних, треба завжди чітко представляти, скільки байт у пам'яті займатиме оголошена змінна. Типи даних, підтримуваних компілятором CI, їх розміри і діапазони можливих значень наведені в табл. 4.2.

Таблиця 4.2

Типи даних, підтримуваних компілятором CI, їх розміри і діапазони можливих значень

Тип	Назва	Розмір у бітах	Діапазон значень
bit	Біт	1	0, 1
char	Символ	8	-128.127
unsigned char	Символ без знаку	8	0.255
signed char	Символ із знаком	8	-128.127
int	Ціле	16	-32768.32767
sort int	Коротке ціле	16	-32768.32767
unsigned int	Ціле без знаку	16	0.65535
signed int	Ціле із знаком	16	-32768.32767
long int	Довге ціле	32	-2147483648. 2147483648
unsigned long int	Довге ціле без знаку	32	0.4294967295
signed long int	Довге ціле із знаком	32	-2147483648. 2147483648
float	З плаваючою крапкою	32	±1.175e-38.±3.402e38
double	Подвійне	32	±1.175e-38.±3.402e38

Якщо ви припускаєте оперувати числами від 0 до 255 (позитивними числами розміром у байт без знаку), то доцільно використовувати unsigned char. При використанні чисел від 0 до 65536 (позитивними числами розміром у два байти), треба застосовувати тип даних unsigned int. Якщо вам потрібен знаковий байт, то оголошуйте його як signed char. Всі змінні повинні бути описані до їх використання. Опис задає тип, за яким слідує список однієї або більш змінних цього типу. Оголошення змінної починається з ключового слова, що визначає його тип, за яким слідує власне ім'я змінної. Наприклад, змінні, які зберігатимуть цілі числа (... – 2, – 1, 0, 1, 2, 3, ...), оголошують так:

```
   int змінна1;
       int змінна2;
```

Фрагмент програми, що оголошує змінні, може мати вигляд:

```
/*
Type     Size (Bits)        Range  from CodeVisionAVR help
bit      1          0, 1
char     8          -128 to 127
unsigned char   8          0 to 255
signed char     8          -128 to 127
int      16         -32768 to 32767
short int 16        -32768 to 32767
unsigned int    16         0 to 65535
signed int      16         -32768 to 32767
long int  32        -2147483648 to 2147483647
unsigned long int32        0 to 4294967295
signed long int   32       -2147483648 to 2147483647
float    32         ±1.175e-38 to ±3.402e38
double   32         ±1.175e-38 to ±3.402e38
*/
int a, b;  // Оголошення змінних a і b, як цілих
int temp;          // Оголошення змінної temp, як цілої
```

У ряді випадків процес оголошення змінних зручно суміщати з операцією присвоювання. Приклад 1:

```
/* Змінним в описах можна задавати початкові значення
  об'єднуючи, таким чином, опис і оператор присвоювання */
int x=1;  // Змінна x оголошується, як ціла
          // і їй присвоюється початкове значення 1
```

Присвоювання (=) означає, що треба присвоїти значення константи (вирази) праворуч від оператора присвоювання тієї змінної, що вказана зліва від нього. Приклад 2:

```
/*
Type     Size (Bits)       Range  from CodeVisionAVR help
unsigned char    8         0 to 255
unsigned int     16        0 to 65535
*/
unsigned char my_peremen = 34;
unsigned int big_peremen = 34034;
```
/* Це оголошені дві змінні та їм привласнені значення. Перша my_peremen – символьного типу – це 1 байт, вона може зберігати число від 0 до 255. У даному випадку в ній зберігається число 34. Друга big_peremen – цілого типу, два байти, в ній може зберігається число від 0 до 65535, а в прикладі в неї помістили десяткове число 34034. */

Читаність програми підвищується, якщо при завданні списку змінних у кожному рядку програми розміщується по одному імені змінної і всі імена вирівнюються по першому символу. При використанні такої угоди услід за ім'ям змінної залишається місце для коментаря. Одне ключове слово дозволяє оголосити дещо змінних одного і того ж типа. Дещо змінних можуть бути відразу оголошені в одному рядку так:

```
int змінна1, змінна2.
int x, y, z;
```

Цілочисельні змінні можуть бути знаковими і беззнаковими. Знакові змінні можуть представляти як позитивні, так і негативні числа. Для цього в їх уявленні найстарший біт відводиться під знак. Беззнакові змінні приймають тільки позитивні значення. Ключові слова signed і unsigned указують на те, як інтерпретується найстарший біт оголошуваної змінної. Старший біт для числа із знаком (signed) визначається як знак числа. Якщо вказано ключове слово unsigned, тобто число беззнакове, то старший біт інтерпретується як частина числа. За відсутності ключового слова unsigned змінна вважається знаковою. Якщо оголошено негативне число, то компілятор генерує так званий зворотний код. Відмінність в інтерпретації компілятором чисел із знаком і без знаку демонструє приклад:

```
/*
Type     Size (Bits)       Range  from CodeVisionAVR help
unsigned char    8         0 to 255
signed char      8         -128 to 127
*/
signed char x, a;          // Оголошення змінних x і a, як символьних
// із знаком
unsigned char y, b;        // Оголошення змінних y і b, як символьних
// без знаку
```

```
void main(void)
{
x=0xFF;
/* Оскільки змінна x має тип signed char (символьне
   знаком), то компілятор інтерпретує її значення (0xFF =
   0b 1111 1111), як – 1, оскільки 1 в самому старшому розряді означає "–"
   а решта семи одиниць в додатковому коді означає 1 */
y=0xFF;
/* Оскільки змінна y має тип unsigned char (символьне без знаку)
   то компілятор інтерпретує її значення (0xFF=0b 1111 1111)
   як 255, оскільки всі 1 є частиною числа */
a=x+y;
b=x+y;
/* У обох випадках обчислюється сума: 0b 1111 1111 + 0b 1111 1111 =
   0b 1 1111 1110. Одиниця в найстаршому розряді пропадає, оскільки під змінну типу char (як signed, так і unsigned) в пам'яті відводиться 8 біт (1 байт). Таким чином, в обох випадках результат складання: 0b 1111 1110, але компілятором він інтерпретується по-різному. Оскільки змінна a має тип signed char (символьне із знаком), то компілятор інтерпретує її значення, як –2, оскільки 1 в найстаршому розряді означає "–", а решта семи розрядів в додатковому коді означає 2. Оскільки змінна b має тип unsigned char (символьне без знаку), то компілятор інтерпретує її значення, як 254, оскільки всі розряди є частиною числа. Таким чином, після виконання функції main: x=-1, y=255, a=-2, b=254 */
}
```

У випадку, якщо на початку стоїть ключове слово signed або unsigned і далі слідує ідентифікатор змінної, то вони розглядатимуться як змінна int. Приклад:

```
unsigned int n;    // n – беззнакова ціла
unsigned char b;           // b - беззнакова символьна
int з;      // мається на увазі signed int з
unsigned d;        // мається на увазі unsigned int d
signed f;          // мається на увазі signed int f
```

Символьний тип даних char застосовується часто у випадках, коли змінна повинна нести інформацію про код ASCII. Цей тип даних використовується для побудови таких складних конструкцій, як рядки, символьні масиви та інше. Дані типу char також можуть бути знаковими і беззнаковими. Наприклад:

```
char з;   // Оголошення змінної з, як символьної
unsigned char b;           // b – беззнакова символьна
```

Для представлення чисел з плаваючою комою застосовується тип даних float. Цей тип даних, як правило, використовується для зберігання не дуже великих дробових чисел.

Невідповідність типів даних при роботі на CI є однією з помилок, що найбільш часто зустрічаються. Компілятор не завжди може запобігти подібним недолікам. Як наслідок, програма виконує не ті дії, які від неї чекає програміст.

Змінні можуть бути видимими (тобто доступними) у всьому файлі (модулі), в якому вони описані і невидимими, глобальними і локальними. Змінні можуть бути згруповані в масиви. Масиви – це група елементів однакового типу (double, Float, int і т. д.). При оголошенні масиву компілятор одержує інформацію про тип елементів масиву та їх кількість. Синтаксис оголошення масиву:

[<модификатор місця зберігання>] <тип> <идентификатор масиву> [<константне _вираз>] ;

Специфікатор типу задає тип елементів оголошуваного масиву. Елементами масиву не можуть бути функції та елементи типу void (невизначені). Константний вираз у квадратних дужках задає кількість елементів масиву. Константний вираз при оголошенні масиву може бути опущений у наступних випадках: 1) при оголошенні масив ініціалізувався (у цьому випадку компілятор сам автоматично визначає розмір масиву; 2) масив оголошений як формальний параметр функції; 3) масив оголошений як посилання на масив, явно визначений в іншому файлі.

Компілятор підтримує багатовимірні масиви, які можуть мати до 8 вимірювань. Кожен константний вираз у квадратних дужках визначає число елементів за даним вимірюванням масиву (при оголошенні одновимірного масиву містить один константний вираз, при оголошенні двомірного – 2, тривимірного – 3). Приклади:

```
/* Приклади глобальних масивів */

int glob_array1 [10];          // Оголошення цілочисельного глобального
// масиву glob_array1
// При цьому всі елементи глобального
// масиву автоматично ініціалізувалися
// із значенням 0

int glob_array2[3]={1, 2, 3};          // Оголошення та ініціалізація
// цілочисельного глобального
// масиву glob_array2
```

```
int glob_array3[10]={4,28,16,35};          // Оголошення та ініціалізація
// цілочисельного глобального
// масиву glob_array3
// При цьому ініціалізують
// тільки перші 4 елементи масиву
// Решту 6 елементів будуть 0

int glob_array4[]={1,2,3,4};          // Оголошення та ініціалізація
// цілочисельного глобального
// масиву glob_array4.
// Компілятор автоматично визначить
// розмір масиву glob_array4[4]

int glob_array5[3][2]={5,0},{8,4},{3,0};     // Оголошення та ініціалізація
// глобального цілочисельного
// багатовимірного
// масиву glob_array5
char glob_array6[]="Hello!";         // Оголошення і ініціалізація символьного
// глобального масиву (рядкового
// літерала) glob_array6
```

Структури – це складовий об'єкт, визначений користувачем, в якого входять елементи будь-яких типів за винятком функцій. На відміну від масиву, який є однорідним об'єктом, структура може бути неоднорідною. Приклад:

```
/* Глобальна структура, розташована в SRAM */
/* Змінна str_ram визначається як структура,
   що складається з шести компонент: цілої змінної, а
   символьних змінних b і з, цілочисельних масивів d[15] і
   e[20] і символьної змінної, розташованої за покажчиком next */
struct structure_ram
{
int a;
char c,b;
int d[15],e[20];
char *next;
} str_ram;
```

4.4.3. Функції мови CI

У багатьох областях діяльності людини широко використовують системний підхід, який грунтується на тому, що вирішувана проблема

завдяки функціональній декомпозиції, заздалегідь розбивається на декілька більш-менш крупних завдань, які, в свою чергу діляться на дрібні завдання. Функціональна декомпозиція, послідовне розчленовування складного завдання на набір простіших задач і завдань, є важливим аспектом розробки програмного забезпечення. Більшість мов програмування високого рівня своїми засобами «підтримують» реалізацію концепції функціональної декомпозиції. Ключовим елементом подібної концепції при програмуванні на СІ виступає функція. На відміну від інших мов програмування високого рівня, в мові СІ немає ділення на процедури, програми і функції. Тут вся програма будується тільки з функцій. Для програміста функція – це програма, що працює в деякому середовищі, яке не вимагає спеціальних зусиль: досить «звернутися» до неї і будуть «вироблені» бажані дії. Функція в мові СІ – це програмний об'єкт, що несе закінчене смислове навантаження і що має унікальне ім'я, що виконує перетворення своїх аргументів і при цьому повертає результати перетворень. Функція – сукупність оголошень і операторів, звичайно призначених для вирішення певного завдання. Функція – це фрагмент програми, в який передаються параметри, і який повертає значення (або нічого). Щоб функція виконала певні дії, вона повинна бути викликана в програмі. При зверненні до функції вона виконує поставлене завдання, а по закінченню роботи повертає як результат деяке значення. Повернення результату – відмінна риса функцій. Коли функція не повертає значення (точніше повертає значення типу void – порожньо), то вона все одно служить для того, щоб змінювати свої параметри або глобальні для функції змінні. Функція, в загальному випадку, має список аргументів (параметрів).

У мові СІ з використанням функцій зв'язані три поняття: визначення функції (опис дій, що виконуються функцією), оголошення функції (завдання форми звернення до функції) і виклик функції. Визначення функції складається з її заголовка і власне «тіла», яке поміщене у фігурні дужки і несе смислове навантаження. Кожна функція, яку передбачається використовувати в програмі, повинна бути оголошена. Синтаксис для оголошення функції:

[<клас пам'яті>] [<тип>] ім'я_функції ([<список параметрів>]
{
Тіло функції
}

Необов'язковий специфікатор класу пам'яті [<класс памяти>] задає клас пам'яті функції, який може бути static або extern. Специфікатор типу функції [<тип>] задає тип повертаного значення і може задавати будь-який тип. Якщо специфікатор типу не заданий, то передбачається, що функція повертає значення типу int. Для функцій, що не повертають ніяких значень, повинен бути використаний тип void (порожньо), вказуючий на відсутність повертаного значення. Ім'я функції (имя_функции) це деякий набір символів, вибраних за вашим бажанням і написаних маленькими буквами. Всі програми на СІ починаються з головної функції main (). Дана функція є обов'язковою. При цьому бажано давати осмислені імена функціям, що нагадують про їх призначення. Прийнято, у ряді випадків, імена функцій (окрім main, звичайно) починати з двох символів підкреслення. Наприклад __moja_funkziya. Список формальних параметрів [<список параметров>] – це послідовність оголошень формальних параметрів (аргументів), розділена комами. Кожен аргумент функції являє собою змінну, вираз або константу, передавані в тіло функції для подальшого використання в обчислювальному процесі. Список аргументів може бути порожнім. Формальні параметри – це змінні, використовувані усередині тіла функції і одержуючі значення при виклику функції шляхом копіювання в них значень відповідних фактичних параметрів. Список формальних параметрів може закінчуватися комою (,) або комою з крапками (,…). Це означає, що число аргументів функції змінне. Проте передбачається, що функція має, принаймні, стільки обов'язкових аргументів, скільки формальних параметрів. Якщо тип формального параметра не вказаний, то цьому параметру присвоюється тип int. Якщо функція не використовує аргументи, то наявність круглих дужок обов'язкова, а замість списку параметрів рекомендується вказати слово void (порожньо). Визначення простої функції виглядає так:

```
    int  __func(int x)
{
          /* Один або декілька операторів
             що завершуються оператором return(щось);
           */
          return x+1;
 }
```

Визначення функції задає тип поверненого значення, ім'я функції, типи і число формальних параметрів, а також оголошення змінних і операторів, званих «тілом» функції і що визначають дію функції. Якщо функ-

ція повертає значення, відмінне типу void (порожньо), в теле функції обов'язково повинен бути присутнім оператор return з параметрами того ж типу, що і повернене значення. У випадку, якщо повернене значення не використовуватиметься надалі в програмі (void), оператор return слідує без параметра і взагалі може бути опущений. У цьому випадку повернення з функції здійснюється після досягнення останньої фігурної дужки тіла функції, що закривається. Інший приклад визначення функції:

```
/* Основна функція програми */
void main(void)
{
int x=5, y=6, z=10;          // Оголошуємо таініціалізуємо цілі
          // змінні x, y і z
int result;          // Оголошуємо цілу змінну result
result=function(x,y,z);       // Викличемо функцію function з фактичними
          // параметрами x,y і z. У результаті функція
          // поверне значення суми цих параметрів:
          // result=5+6+10=21
}
```

Кожна функція, яку передбачається використовувати в програмі, повинна бути в ній оголошена. Оголошення функції схоже з визначенням функції, але з тією лише різницею, що «тіло» в оголошенні відсутнє, і імена формальних параметрів теж можуть бути опущені. У звичайному варіанті функції використовуються після їх визначення. Бувають випадки, коли функції викликають один одного, і організувати їх правильне визначення скрутно. Обійти подібну проблему дозволяють прототипи функцій, які уявляють собою оголошення до визначення. Щоб оголосити функцію, можна використовувати функціональні прототипи. Прототип – це явне оголошення функції, яке передує її визначенню. Прототип функції показує зразок того, як застосовувати функцію в програмі, які значення в ній передаються і, якщо вона повертає якесь значення, то прототип указує тип повернених даних. Прототип функції становить заголовок функції, причому в списку параметрів указують тільки типи, без ідентифікаторів. Прототип не має дужок { }, а після дужок ( ) ставиться крапка з комою. Тип повертаного значення при оголошенні функції повинен відповідати типу повертаного значення у визначенні функції. Прототип функції оголошується так: возв_тип __func_name (список оголошуваних параметрів). Приклад:

```
/* Прототипи функції */
int f1(int );
```

```
void port_init(void);
void timer0_init(void);
void timer0_ovf_isr(void);
void uart0_init(void);
void int0_isr(void);
void init_devices(void);
void get_period(void);
void save_result(void);
int function(int a, int b, int з)          // Оголошення функції function, яка
// повертає ціле значення, оскільки
// перед ім'ям функції коштує int
...
/* Фактичне визначення функції може бути написано
   де-небудь у іншому місці програми */

int function(int a, int b, int з)
{
[група операторів]          // тіло функції
}
```

При виклику функції їй за допомогою аргументів (формальних параметрів) можуть бути передані деякі значення (фактичні параметри), використовувані під час виконання функції. Аргументи (фактичні значення) у МК AVR передаються функції через стек. Функція може повертати деяке (одне) значення. Це повернене значення і є результат виконання функції, який при виконанні програми підставляється в точку виклику функції, де б цей виклик не зустрівся. Значення функцій в МК AVR повертаються в регістри r30, r31, r22 і r23 (від молодшого байта до старшого). Приклад:

```
/* Виклик функції */
 /* Визначення функції з формальними параметрами a, b, і з */
int function(int a, int b, int з)
{
return (a+b+c);      // тіло функції
}
/* Основна функція програми */
void main(void)
{
int x=5, y=6, z=10;// Оголошуємо та ініціалізуємо цілі
// змінні x, y і z
int result;// Оголошуємо цілу змінну result
result=function(x,y,z);          // Викличемо функцію function з фактичними
// параметрами x,y і z. У результаті функція
// поверне значення суми цих параметрів:
```

```
// result=5+6+10=21
}
```

Функція може викликати інші функції (одну або декілька), які вирішують невелику і специфічну частину загального завдання, а ті, у свою чергу, проводити виклик третіх. При цьому структура кожної функції, що викликається, подібна структурі головної програми.

```
/*  ================================
Зручно над функцією зробіти заголовок що детально пояснює призначення функції !
Це буде функція в якій описано початкову  конфігурацію МК відповідно до  поставленого завдання
==================================  */
(void)__init_mk(void) {
/* Спочатку будь-якій функції оголошуються
ЛОКАЛЬНІ ЗМІННІ – якщо звичайно вони вам потрібні */
/* void – означає порожньо.  Перед назвою функції – void – означає що функція не повертає ніякого значення. А в дужках після назви означає що при виклику у функцію не передаються ніякі значення.
*/
// ініціалізація Port_B
DDRB=0xFF;      // всі ніжки зробіти виходами
PORTB=0xFF;    // вивести на всі ніжки "1"
```

Функція може викликати сама себе. Це явище в програмуванні називається рекурсією.

Будь-яка програма на CI, у принципі, може складатися з великого числа функцій [main (), function1(x1, y1), function2(x2, y2, z2),., functionN(xn, yn)], але при цьому вона обов'язково включає функцію main (). Англійське слово «main» означає «головна». Саме з цієї функції завжди починається виконання програми. Як тільки буде здійснений виклик функції FUNCTION1(x,y), яка вирішує приватне завдання, то управління програмою буде передано функції FUNCTION1(x,y) і до оператора return виконуватиметься тіло FUNCTION1, після чого управління повертається в тіло функції main (). Після цього продовжується виконання функції main () до виклику функції FUNCTION2(a,b,c). Далі виконується тіло FUNCTION2(a,b,c) і знову здійснюється передача управління в тіло функції main (). Так продовжується поки не буде виконана повністю головна функція.

Під час виконання програми, в процесі обчислення функцій, змінні можуть безліч разів міняти свої значення, а крім того, то використовуватися, то ні. Для того, щоб можна було управляти «потребою» змінних у

програмі і часом їх «життя», використовують ряд понять, зокрема, область дії, область видимості і час життя кожної змінної. Під областю дії змінної розуміють область програми, в якій змінна доступна для використання. З цим поняттям тісно зв'язано поняття області видимості змінної. Змінна знаходиться у «області видимості», якщо до неї можна дістати доступ. Якщо змінна виходить з області дії, стає «не потрібною», то вона повинна стати «невидимою». У зв'язку з цим, змінні, використовувані у функціях, розділяють на глобальні і локальні. Глобальні змінні – це змінні, доступні у всіх функціях програми. Вони є видимими (доступними) у всьому файлі (модулі) програми. Глобальні змінні оголошуються поза тілом якої-небудь функції і діють впродовж всієї програми. Такі змінні доступні в будь-якій з функцій програми, яка описана після оголошення глобальної змінної. Глобальні змінні оголошуються до появи в тексті програми будь-якої функції. Імена глобальних змінних не повинні співпадати. Локальні змінні доступні тільки в окремих функціях { functionN(xn.,yn)}, де вони оголошені. Локальні змінні оголошуються на початку тієї функції, де їх передбачається використовувати, тобто відразу після фігурної дужки «{ ». У МК такі змінні звичайно розміщуються в стеку. Локальні змінні мають своєю областю видимості функцію або блок, в яких вони оголошені. Область дії локальної змінної може виключати внутрішній блок програми, якщо в ньому оголошена глобальна змінна з тим же ім'ям. Тобто, оголошення локальної змінної приховує глобальну змінну з тим же ім'ям і всі звернення до імені глобальної змінної в межах області локального оголошення викликають звернення до локальної змінної. Звичайну локальну змінну видно тільки з тієї функції, в якій вона оголошена. Час життя локальної змінної (інтервал виконання програми, протягом якого вона існує) визначається часом виконання функції, в якому вона оголошена. При поверненні управління до зухвалої функції, пам'ять, що відводиться під локальні змінні, звільняється. Значення локальної змінної зберігається поки виконується ця функція. Приклад:

```
/* Оголошення глобальних змінних */
long x;
char y;

/* Ініціалізація глобальних змінних */
x=10;
/* Оголошення з одночасною ініціалізацією глобальних змінних */
int c=0xfa;
```

```
...
void main(void)
{
/* Оголошення локальних змінних
тобто змінних тільки усередині цієї функції */
int a;
long b;
/* Ініціалізація локальних змінних */
a=0x2c;
/* Оголошення з одночасною ініціалізацією локальних змінних */
char c='s';
...
}
```

Слово ініціалізація означає присвоєння змінної початкового значення. Початкове значення змінної присвоюють при її оголошенні, приєднавши ініціалізатор до описувача. Приклад:

```
/* Ініціалізація даних */
char symbol= 'F';              // Змінна symbol ініціалізувалася символом 'F'
const long variable = (512 * 1024);              // змінна, що Немодифікується
// (константа) variable
// ініціалізувався константним
// виразом, після чого вона
// не може бути змінена
eeprom int array1[2][2] = {1,2,3,4};              // Ініціалізувався двомірний
// масив array1 цілих величин
// розташований в eeprom
// елементам масиву привласнюються
// значення із списку
eeprom int array1[2][2] = {1,2},{3,4}; // Та ж сама ініціалізація
int array2[2][2] = {1,2},{3};              //якщо при ініціалізації вказане менше
// значень для елементів масиву, то
// елементи, що залишилися, ініціалізувалися
// 0, тобто в даному випадку елементи
// першого рядка набудуть значення 1 і 2
// а другий 3 і 0
```

Корисно якомога рідше використовувати глобальні змінні і не використовувати локальні змінні в головній функції main.

Щоб забезпечити можливість зберігати значення локальної змінної при виході з функції і використовувати це значення при повторному виклику цієї функції, локальні змінні повинні бути оголошені із специфікатором класу пам'яті static (статичні). Така змінна має глобальний час життя

і область видимості усередині функції, в якій вона оголошена. Статичні змінні можна використовувати для зберігання значень змінних впродовж часу роботи програми. При виконанні програми статичні змінні ініціалізувалися тільки один раз, навіть якщо функція, де вони ініціалізувалися, викликається багато разів. Приклад:

```
/* Оголошення локальних змінних x
   з класом пам'яті static */
int function1(void)   //
{       // Визначення функції function1
static int x=2;         // з локальною змінною x, з класом
return (x=x+5);       // пам'яті static
}                                       //
int function2(void)   //
{       // Визначення функції function2
int x=2;        // з локальною змінною x
return (x=x+5);       //
}                                       //
int function3(void)   //
{       // Визначення функції function3
static int x=3;         // з локальною змінною x, з класом
...       // пам'яті static
}                                       //
/* Основна функція програми */
void main(void)
{
...
int y;   // Оголошення цілої локальної змінної y
/* функція function1 повертає значення 7, оскільки
   початкове значення x=2, а після return x=x+5, x=7 */
y=function1();
/* функція function1 повертає значення 12, оскільки
   значення x запам'яталося, завдяки static
   і при виклику цієї функції x=7
   а після return x=x+5, x=7+5=12 */
y=function1();
/* функція function2 повертає значення 7, оскільки
   початкове значення x=2, а після return x=x+5, x=7 */
y=function2();
/* функція function2 знову повертає значення 7, оскільки
   значення x не запам'яталося і при виклику цієї функції
   x знову ініціалізувався 2, а після return x=x+5, x=7 */
y=function2();
```

```
...}
```

Щоб використовувати змінні, які глобально оголошені в інших початкових файлах проекту, вони повинні бути оголошені із специфікатором класу пам'яті extern (зовнішній). Мета такого оголошення полягає в тому, щоб зробіти визначення змінної глобального рівня видимим усередині функції. Приклад:

```
/* Початковий файл file1.c */
/* Оголошення змінної a */
extern int a;  // Оголошення змінної a
/* тепер підключимо файл, який містить
   визначення змінної a */
#include <file.h> // Підключення файлу file.h
void main(void)
{
...}
```

4.4.4. Операції, оператори, вирази мови CI, використовувані в «тілі» функції

Для здійснення маніпуляцій з даними мова CI має в своєму розпорядженні широкий набір операцій. Операції становлять деяку дію, що виконується над одним (унарна) або декількома (бінарна, тернарна операції) операндами, результатом якого є повертане значення. Дії, які повинні бути виконані над операндами, конкретизують і визначають знаки операцій (наприклад +, - *, /). Тут операнд – це константа, літерал, ідентифікатор або складніший вираз, сформований комбінацією операндів, знаків операцій і круглих дужок. Кожен операнд має тип.

Спеціальне позначення для певної операції над даними – операндами називається оператором. У програмах часто використовують програмні об'єкти, які встановлюють його в певний стан. З історичних причин така установка стану (значення) об'єкту називається присвоюваням, а програмний об'єкт, що виконує дану дію – оператором присвоєння. Наприклад, дію оператора c=a*b можна описати словами: «з присвоюється значення a, помножене на b». У програмах можуть використовуватися оператори викликів функцій і оператори управіння.

Комбінація знаків операцій і операндів, результатом якої є певне значення, називається виразом. Вираз в CI є послідовність операторів, операндів і знаків пунктуації, що сприймається компілятором, як керівни-

цтво до певної дії над даними. Всякий вираз, за яким йде крапка з комою, утворює пропозицію або інструкцію мови.

Змінну можна змінювати за допомогою операції простого присвоєння (=). Операція простого присвоєння використовується для заміни значення лівого операнда значенням правого операнда. У мові СІ присвоєння також є виразом, і значення такого виразу є величина, яка присвоюється. У мові СІ операція простого присвоєння позначається знаком рівності, наприклад, x = 12. Це читається не як "ікс рівне 12", а як "присвоїти змінній ікс значення 12". Такий рядок є і простим оператором присвоєння, тобто дією. У кінці операторів ставиться крапка з комою.

Розглянемо оператор x = x + 3;. Це не рівняння (якщо розглядати цей рядок як математичне рівняння, воно не має рішень). Насправді тут написано:1) «узяти значення змінної ІКС»: 2) «додати до нього 3»; 3) «записати нове значення в змінну ІКС», стерши в ній колишнє значення. Таким чином, оскільки у оператора присвоєння є дві частини: ліва і права і є вираз { ліва_частина = права_частина;}, то це означає, що в лівій частині стоїть просто ім'я змінної, в яку записується обчислений справа результат. Якщо ім'я змінної зустрічається в правій частині, то це означає «підставити сюди поточне значення цієї змінної». При цьому поточне значення самою змінною не змінюється, береться його копія. Наприклад. Хай маємо оператора присвоєння (x = x + 3 ;) і початкове значення x є 12. Спочатку обчислюється права частина оператора присвоєння. Це означає, що треба узяти копію значення (число 12) з ім'ям «ікс» провести складання (12 + 3) і виконати саме присвоєння x = 15. Таким чином, після дії оператора присвоєння в змінній з тим же ім'ям тепер знаходиться нове значення x = 15.

Окрім простого присвоєння, є група операцій присвоєння (*=, /=, – =, + =), які об'єднують просте присвоєння з однією з бінарних операцій. Такі укорочені форми запису операції присвоєння називаються складеними операціями присвоєння і мають вигляд: (операнд1) (бінарна операція) = (операнд2). Складене присвоєння за результатом еквівалентно наступному простому присвоєнню: (операнд1) = (операнд1) (бінарна операція) (операнд2). Приклад:

```
/* Бінарні операції (*=, /=, +=) */
int a=5;   // Оголошуємо і ініціалізували цілу змінну a
int b=4;  // Оголошуємо і ініціалізували цілу змінну b
a+=3;                  // еквівалентно   a=a+3, в результаті a=8
```

```
a/=b;          // еквівалентно   a=a/b, в результаті a=8/4=2
b*=a;          // еквівалентно   b=b*a, в результаті b=4*2=8
```

Над операндами визначеного типу допустимі групи операцій: арифметичні, операції відношення, логічні операції, побітові операції. До арифметичних операцій відносяться: складання (+), віднімання (–), ділення(/), множення (*) і визначення залишку (%). Над цілими числами можна проводити такі арифметичні операції: x + y – складання, x – y – віднімання, x * y – множення, x / y – ділення націло (тобто із залишком; результат – ціле), x % y – обчислити залишок від ділення без остачі. Для останніх двох операцій: 5/2 дасть 2, а 5%2 дасть 1. Операція залишок від ділення показує число, яке виходить у вигляді залишку від ділення першого операнда на другій. Операндами цієї операції можуть бути тільки цілі числа. Приклади:

```
/* Бінарні операції (*), (/) і (%) (*/)
int a=37,b=10;    // Оголошуємо і ініціалізували цілі
        //змінні a і b
        int x, y, z;          // Оголошуємо цілі змінні x, y, і z
        x=a*b;   // x=370
        y=a/b;   // y=3
        z=a%b;  // z=7
/* Бінарні операції (+) і (-) (*/)
int a=37, b=10;   // Оголошуємо і ініціалізували цілі
        //змінні a і b
int x, y;  // Оголошуємо цілі змінні x і y
x=a+b;   // x=47
y=a-b;   // y=27
```

У арифметичних операторах присвоєння допускаються скорочення:

Довгий запис	Сенс	Скорочується до
x = x + 1;	"збільшити на 1"	x++; (або ++x;)
x = x - 1;	"зменшити на 1"	x--; (або --x;)
x = x + y;	"додати y"	x += y;
x = x * y;	"помножити на y"	x *= y;
x = x / y;	"поділити на y"	x /= y;

Важливо, що при виконанні операцій (при застосуванні операторів інкремента і декремента) ++ і -- з'являється побічний ефект – змінюється значення змінної, використовуваної як операнд. Крім того, відносно до

операндів оператори інкремента і декремента можуть бути префіксними і постфіксними. Префіксний оператор застосовується до операнда перед використанням одержаного результату. Наприклад, при index= -- curent змінна curent спочатку зменшується на одиницю, після чого результат присвоюється змінній index. Постфіксний оператор застосовується до операнда *у* після використання операнда. Наприклад, при x=y++ спочатку змінної *у* присвоюється значення змінної x, а потім результат збільшується на 1.

На практиці доводиться регулярно порівнювати значення деяких об'єктів, щоб визначити момент настання деякої події. Для того, щоб була можливість порівнювати між собою значення яких-небудь змінних у мові СІ визначені наступні операції відношення: перевірка на рівність (==), перевірка на нерівність (!=), менше (<), менше або рівно (<=), більше (>), більше або рівно(>=). Оператори порівняння (порівнювати можна змінні, вирази, константи) можуть використовуватися як умови: менше (x < y), більше (x > y), менше або рівно (x <= y), більше або рівно (x >= y), рівно (x == y), не рівно (x!=y). У результаті роботи операторів порівняння повертається логічне значення «істина», якщо умова, що перевіряється, вірна, або «неправда» у противному випадку. Всі ці оператори як результат операції порівняння видають 1, якщо порівняння істиннє, або 0, якщо воно помилковє.

У мові СІ є 3 логічних операції: «І» (&&), «АБО» (||) і заперечення (!). Аргументами логічних операцій можуть бути будь-які числа, включаючи ті, що задаються аргументами типу char. Результат логічної операції – 1, якщо істина, і 0, якщо неправда. Слід мати на увазі, що логічні операції мають низький пріоритет. Обчислення виразів, що містять логічні операції, проводиться зліва направо і припиняється, як тільки вдається визначити результат. Якщо вираз складений з логічних тверджень (тобто з виразів, що виробляють значення типу int), з'єднаних між собою операцією «І» (&&), то обчислення виразу припиняється, як тільки хоч би в одному логічному твердженні виробляється значення нуль. Якщо вираз складений з логічних тверджень, з'єднаних між собою операцією «АБО» (||), то обчислення виразу припиняється, як тільки хоч би в одному логічному твердженні виробляється ненульове значення. Приклад:

```
/* Бінарні операції (&& і ||) */
int a=0xA5C8;     // Оголошуємо та ініціалізуємо цілу змінну a
int b=0x0B8D; // Оголошуємо та ініціалізуємо цілу змінну b
```

```
int c=0, d=0;   // Оголошуємо та ініціалізуємо цілі змінні з і d
int result;           // Оголошуємо цілу змінну result
result = a&&b;
          // result = 1
result = a&&c;     // result = 0
result = c&&d;     // result = 0
result = a||b;       // result = 1
result = a||c;       // result = 1
result = c||d;       // result = 0
```

На практиці часто доводиться відстежувати стани різних програмних об'єктів за допомогою «прапорів» і двійкових датчиків. Для цієї мети можна використовувати булеві змінні. Проте, якщо використовується багато логічних ознак, зручніше як прапори використовувати окремі біти змінних. Для того, щоб можна було адресуватися до окремих біт, використовуються так звані порозрядні (побітові) логічні операції. При цьому операнд, або операнди побітових операцій повинні бути цілого типа. Побітові операції звичайно використовуються у додатках, що вимагають доступу до апаратних засобів на рівні маніпуляції з бітами (наприклад, до вимикачів, перемикачів, кінцевих і путніх вимикачів, бінарних датчиків). Тому, щоб оперувати конкретнішими даними, припустимо, що на вхід МК поступають двійкові сигнали від 8 ключів: SW0, SW1, SW2, .,SW7. Сигнал від кожного окремого ключа може приймати два стани: «включено» (1); «вимкнено» (0). Ми хочемо, перевіряючи значення вказаних вище сигналів, судити про стан об'єкту, в якому знаходяться ключі. Операція побітного «І» (&) порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо обидва порівнюваних біта одиниці, то відповідний біт встановлюється в 1, інакше – в 0. Приклад:

```
/* Бінарна операція (&)
int a=0xA5C8;     // Оголошуємо та ініціалізуємо цілу змінну a з
              // значенням a=0b1010010111001000 - 1010 0101 1100 1000
int b=0x0B8D; // Оголошуємо та ініціалізуємо цілу змінну b з
          // значенням b=0b101110001101 -   0000 1011 1000 1101
int з;      // Оголошуємо цілу змінну з
з = a&b;            // c=0x0188=0b110001000          -         0000 0001 1000 1000
```

Повернемося до прикладу з ключами. Хай нас цікавить стан п'ятого ключа. Для скидання окремих біт в байті (виключення ключів, що не цікавлять нас) використовують операцію побітове логічне «І». Накладемо «маску», тобто зробимо так, щоб змінна «mask» виглядала, як 00010000,

або в шістнадцятирічному коді 0x10. Стан ключа присвоїмо змінній з ім'ям (ідентифікатором) flag. Тоді:

```
int mask=0x0010;// Ініціалізували цілу змінну mask
int sw=0x0B8D;  // Ініціалізували цілу змінну sw
                // що описує стан ключів
int flag;    // Оголошуємо цілу змінну flag

flag = sw & mask;
```

Операція побітного логічного «АБО» (|) порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо будь-який (або обидва) з порівнюваних бітів рівний 1, то відповідний біт результату встановлюється в 1, інакше – в 0. Приклад:

```
/* Бінарна операція ( | )
int a=0xA5C8;     // Оголошуємо та ініціалізуємо цілу змінну a з
          // значенням a=0b1010010111001000 - 1010 0101 1100 1000
int b=0x0B8D; // Оголошуємо та ініціалізуємо цілу змінну b з
                // значенням b=0b101110001101 -   0000 1011 1000 1101
int з;      // Оголошуємо цілу змінну з
з = a|b;   // c=0xAFCD=0b1010111111001101 -  1010 1111 1100 1101
```

Повернемося до прикладу з ключами. Хай у початковому стані всі ключі були «вимкнені». Допустимо, що тепер нам треба включити третій ключ. Для цього треба зробіти так, щоб змінна SW в двійковому вигляді виглядала 00000100, або в шістнадцятирічному вигляді 0x04. Застосуємо для цього операцію побітового «АБО»:

```
int mask=0x0004;// Ініціалізували цілу змінну mask
int sw;       // Оголошуємо цілу змінну sw
int flag;    // Оголошуємо цілу змінну flag
flag = sw | mask;
```

Операція побітного «Виключачого АБО» (^) порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо один з порівнюваних бітів рівний 0, а другий 1, то відповідний біт результату встановлюється в 1, інакше, тобто коли обидва біти рівні 1 або 0, відповідний біт результату встановлюється в 0. Приклад:

```
/* Бінарні операції (^)
int a=0xA5C8;     // Оголошуємо та ініціалізуємо цілу змінну a з
                // значенням a=0b1010010111001000 - 1010 0101 1100 1000
int b=0x0B8D; // Оголошуємо та ініціалізуємо цілу змінну b з
          // значенням b=0b101110001101 -   0000 1011 1000 1101
int з;      // Оголошуємо цілу змінну з
з = a^b;      // c=0xAE45=0b1010111001000101 -    1010 1110 0100 0101
```

У прикладі з ключами змінимо стан четвертого ключа на протилежний, незалежно від його початкового стану. Застосуємо для цього операцію побітового «Виключаючого АБО»:

```
int mask=0x0008;// Ініціалізували цілу змінну mask
int sw;      // Оголошуємо цілу змінну sw
int flag;    // Оголошуємо цілу змінну flag
flag = sw ^ mask;
```

Слід мати на увазі, що все сказане щодо побітових логічних операцій справедливе при використанні беззнакових типів операндів. При застосуванні побітних операцій із знаковими операндами виникають проблеми сумісності. Вихід з подібної проблеми – перетворити перед виконанням побітових операцій знакові цілі в беззнакові.

Як вже наголошувалося, часто при програмуванні користуються складеною операцією присвоєння, яка об'єднує просте присвоєння з однією з бінарних операцій. На практиці для присвоєння результату порозрядних логічних операцій використовуються скорочені форми запису: (&=) – побітове «І» з присвоєнням, (|=) побітове логічне «АБО» з присвоєнням, (^=) – побітове «Виключаючє АБО» з присвоєнням. Операції (&=), (|=), (^=) часто називають накладенням «маски». Число, що стоїть праворуч від знаку такої операції, називають маскою. При накладенні маски змінюються значення лише певних бітів операнда, що стоїть зліва. Цими операціями користуються в тих випадках, якщо в змінній потрібно змінити один або декілька бітів, а інші залишити без зміни. Приклад:

```
/* Бінарні операції &=, |= і ^=, як операції накладення маски */
unsigned char x=0b00111100; // Оголошуємо та ініціалізуємо //символьні
unsigned char y=0b00111100;      // (байтові) змінні x,y і z
unsigned char z=0b00111100;      //
/* Накладемо на ці числа маску 0b00001111 */
x&=0b00001111;
/* При накладенні маски операцією &= у тих розрядах, де в масці коштує  значення 0, біти початкового числа приймають значення 0, не залежно від  первинного значення. У тих розрядах, де в масці коштує значення 1,  біти початкового числа не змінюються і в результаті: x=0b00001100 */
 y|=0b00001111;
/* При накладенні маски операцією |= у тих розрядах, де в масці коштує  значення 1, біти початкового числа приймають значення 1, не залежно від первинного значення. У тих розрядах, де в масці коштує значення 0, біти початкового числа не змінюються і в результаті: y=0b00111111 */
z^=0b00001111;
```

/* При накладенні маски операцією ^= у тих розрядах, де в масці коштує значення 1, біти початкового числа приймають значення, протилежне первинному, тобто якщо був 0, то стане 1, якщо була 1, то стане 0. У тих розрядах, де в масці коштує значення 0, біти початкового числа не змінюються. Таким чином, в результаті z=0b00110011 */

Для здійснення зрушення послідовності біт вліво або вправо застосовуються операції зсуву, які здійснюють зсув значення першого операнда вліво (<<) або вправо (>>) на кількість біт, що задаються другим операндом. Обидва операнди повинні бути цілими величинами. При зрушенні вліво праві біти, що звільняються, встановлюються в 0. При зрушенні управо метод заповнення лівих бітів, що звільняються, залежить від типу першого операнда. Якщо тип unsigned (беззнаковий), то вільні ліві біти встановлюються в 0. Інакше (якщо тип signed – знаковий) вони заповнюються копією знакового біта. У цьому випадку результат операції зрушення не визначений, якщо другий операнд негативний. Щоб уникнути подібних ситуацій рекомендується перетворювати операнд операції в беззнаковий тип. Операції з беззнаковими операндами можуть застосовуватися для множення або ділення на число k, рівне ступеню числа 2. Приклад:

```
/* Бінарні операції (<<) і (>>) */
int a=0b101001;// Оголошуємо та ініціалізуємо цілу змінну a
        int x, y; // Оголошуємо цілі змінні x і y
        x=a<<2;// зрушуємо значення a на 2 біта вліво і одержане
        // значення привласнюємо змінній x; x=0b10100100
        y=a>>1;// зрушуємо значення a на 1 біт управо і одержане
        // значення привласнюємо змінній y; y=0b10100
// Значення змінної a не змінюється! a=0b101001
```

Розглянемо дуже інформативну і цінну для розуміння операцій зсуву наступний рядок програми:

```
PORTB = (unsigned char) ~(ADCW>>2);
```

У даному рядку обчислюється вираз (unsigned char) ~(ADCW>>2) і набутого значення присвоюється змінній PORTB. Хай ADCW – це змінна (об’явлена у файлі mega16.h двобайтова величина – слово), в якій зберігається 10-бітовий результат аналогово-цифрового перетворення (АЦП). Вважаємо також, що нам потрібно відобразити 8 старших бітів результату ADCW – тобто біти_9_2. Для досягнення цього треба зрушити всі біти слова ADCW управо на 2 позиції. Вираз ADCW >> 2 і виконує дане зрушення. При цьому біти 1 і 0 зрушуються управо і пропадають, а решта біт теж зрушується управо: біт_9 переміщається у позицію біт_7, біт_8 у позицію біт_6 і так далі поки біт_2 не стає біт_0. Тобто 8 біт результату

АЦП переміщаються в позиції в біт 7_0 молодшого байта (LowByte – LB) слова. Тут ADCW >> n означає зрушити всі біти числа управо на n позицій, що рівносильно діленню на 2 в ступені n. До речі, ADCW << n означає зрушити всі біти числа вліво на n позицій, що рівносильно множенню на 2 в ступені n. Далі йде операція побітового інвертування ~. Результатом цього виразу ~(ADCW>>2) будуть інвертовані 8 старших біт результату АЦП, що знаходяться в молодшому (правому –LB) байті двох байтового слова ADCW.

У Сі-змінну можна поміщати тільки той тип даних, який вона може зберігати. Оскільки порт PORTB восьмирозрядний (байт), а ADCW – це два байти, то перш ніж виконати оператор присвоєння (це знак =) потрібно перетворити слово (слово – word – значить 2 байти) ADCW у беззнаковий байт. Перетворення типів даних роблять так: перед тим, що треба перетворити в дужках ( ) записують тип даних, до якого потрібно перетворити (unsigned char) ~(ADCW>>2). Результат цього рядка – один байт і ми можемо помістити його в PORTB.

При програмуванні операцій зрушення використовують складову операцію присвоєння, яка об'єднує просте присвоєння з однією з операцій зрушення. Скорочені форми запису операцій зрушення: (<<=) – зрушення вліво з присвоєнням, (>>=) – зрушення управо з присвоєнням.

Операція «кома» зв'язує між собою декілька виразів таким чином, що останні розглядаються компілятором, як єдиний вираз. Операція послідовного обчислення використовується для обчислення двох і більш виразів там, де по синтаксису допустимо тільки один вираз. Ця операція обчислює два операнди зліва направо. Завдяки використанню даної операції при написанні програм досягається їх висока ефективність. Приклад:

```
/* Бінарна операція (,) */
int a=0xA5C8, b=0x0B8D;    // Послідовно оголошуємо і
                           // ініціалізували цілі змінні a і b
```

У СІ операції виконуються у певному порядку. Так, наприклад, математичні вирази обчислюються зліва направо, тоді як оператор простого присвоєння виконується справа наліво. Для того, щоб компілятор «міг розібратися», яку ж дію здійснювати першою при виконанні операцій, останні володіють важливою властивістю – пріоритетом. Значення величин пріоритетів операцій, використовуваних у СІ, з вказівкою напряму їх виконання, можна знайти в літературі з мови СІ.

4.4.5. Структурі мови СІ, що управляють, використовувані в «тілі» функції

Ми дотепер розглядали приклади програм для МК на мові СІ, які виконувалися у порядку проходження операторів, що закінчуються крапкою з комою. Тобто оператори виконувалися в тому порядку, в якому вони записані в програмі, послідовно. У реальних завданнях управління від програми потрібно, щоб вона могла «ухвалювати рішення» залежно від ситуації, що є на даний момент на об'єкті управління. Мова СІ володіє вичерпним набором конструкцій (структур управління), що дозволяють управляти порядком окремих гілок програми. У ньому можна передати управління в ту, або іншу частину програми залежно від результату перевірки деякої умови, виконати деякий набір операторів задану кількість раз, або поки не виконуватиметься якась умова. Використовувані в мові СІ структури, що управляють, можна розділити на дві групи: структури вибору і цикли. Оператори, вживані в цих структурах, також можна розділити на декілька категорій: умовні оператори; оператори циклу, оператори переходу, інші оператори. Першим прикладом конструкції вибору є умовні оператори, які здійснюють галуження в програмах. Оператор **if** проводить галуження програми залежно від результату перевірки деякої умови на істинність:

```
/*   Умовний оператор */
     if(умова) оператор1;
     ...продовження... оператор2
```

Умова, що перевіряється, може бути будь-яким виразом, але найчастіше містить оператори порівняння. Всі вирази, реалізовуючі умови для конструкції вибору, містяться в круглі дужки. Схема алгоритму умовного оператора if має вигляд, показаний на рис. 4.9 а.

Оператор if працює так: спочатку обчислюється умова (вираз). Якщо умова істинна (тобто значення виразу відмінно від 0, то виконується оператор 1 (або група операторів1), а, потім, виконується продовження. Якщо умова помилкова (значення виразу рівне 0), то оператор1 пропускається (оператор не виконується), і відразу виконується продовження. При цьому слід мати на увазі, що, у Сі працює правило: "Неправда" (False) тільки нуль; "Істина"(True) – не нуль або так (!0). При цьому «!(істина)» дає «неправда», а «!(неправда)» дає "істина".

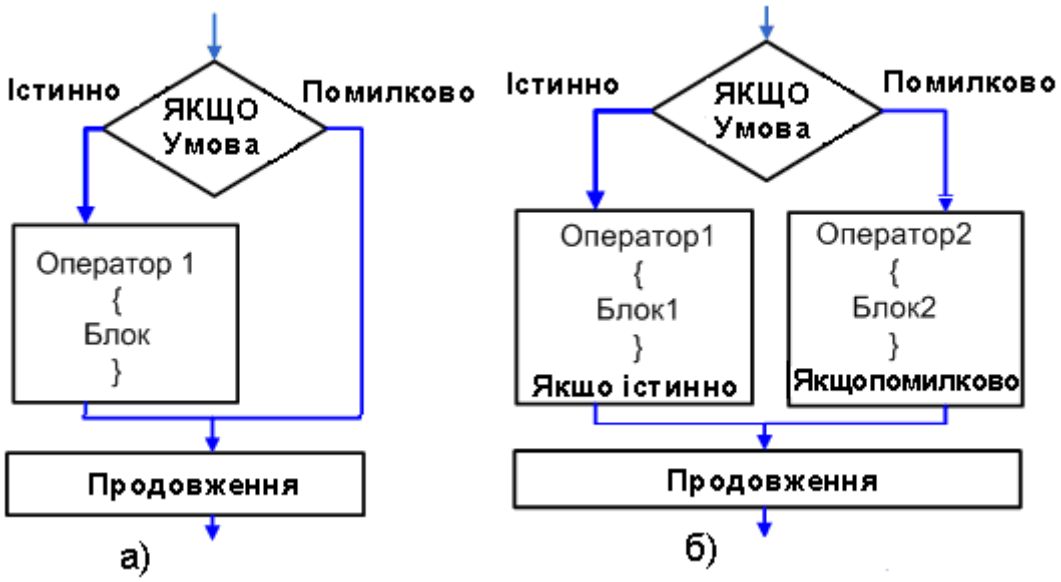


Рис. 4.9. **Схема алгоритму умовного оператора if без (а) і з ключовим словом else (б)**

У конструкції мови СІ оператори можуть бути блоковими. Це означає, що залежно від ухваленого рішення виконується не один, а цілий набір операторів. Якщо треба виконати при істинності умови декілька операторів, ми повинні укласти їх у дужки { ... } – це «складений оператор». Блок починається з фігурної дужки, що відкривається, і закінчується нею. Вміст блоку розглядається компілятором мови як єдиний оператор.

```
    if(умова)
{
        оператор1;
        оператор2;
        ...
 }
    Продовження. операторN
```

Після фігурної дужки, що закривається «}», точка з комою не ставиться. У програмах, пов'язаних з ухваленням рішень, часто проводиться перевірка значення змінної, наприклад *x*, на нерівність 0. Для цього використовується умова if(x!=0). Типовою помилкою при використанні перевірки на 0 є використання в умовних конструкціях оператора присвоєння (=), замість оператора порівняння (==). Слідує також в програмах використовувати if(x != 0) або if(x == 0).

Іншою структурою, яка реалізує вибір і проводить галуження програми залежно від результату перевірки деякої умови на істинність, є

оператор **if** з ключовим словом **else**. Данна форма структури, з частиною "інакше", має вигляд:

```
      if( умова, що перевіряється)
 оператор1_якщо_істинно;
      else
оператор2_якщо_хибно;
```

У структурі вибору "або те, або інше" (але не обидва відразу) можуть використовуватися не тільки один оператор, але і група операторів, або складений оператор (блок). Схема алгоритму умовного оператора if з ключовим словом else має вигляд, показаний на рис. 4.9 б. За аналогією з оператором if, цей умовний оператор працює так: якщо умова прийняла дійсне значення, виконується оператор1 (вираз1), або блок 1, а якщо помилкове – вираз 2 (оператор2, або блок 2). Звичайно повертане значення в продовженні присвоюється якій-небудь змінній. Приклад 1.

```
/* Оператор if-else */
/* Змінною з буде присвоєне значення змінної
   найбільшої a і b  */
if (a > b)// Якщо a > b
з = a;    // те = a
else      // інакше
з = b;    // з = b
```

Приклад 2.

```
int x, y, z;
if(x == 1)
{ y = 2; z = x + y; }
 else
{ y = 1; z = x – y; }
```

Як умови можуть використовуватися оператори порівняння (порівнювати можна змінні, вирази, константи):

```
      x <  y       менше
      x >  y       більше
      x <= y       менше або рівно
      x >= y       більше або рівно
      x == y       рівно
      x != y       не рівно
```

Ці оператори як результат операції порівняння видають 1, якщо порівняння істинно, і 0, якщо воно помилково. Частина з else є необов'язковою. Якщо ця частина відсутня, то виконується наступний за if оператор програми. Іншою структурою, що управляє, призначеною для організації вибору з безлічі різних варіантів і здійснення галуження, служить

конструкція множинного вибору «**switch – case»** (перемикач – випадок). Вона дозволяє вибирати з декількох варіантів і використовується у випадках, коли необхідно проводити порівняння деякої змінної з великим числом однотипних змінних, або констант. Застосування «switch – case» ефективно при використанні в програмі переліків. Конструкція має наступний синтаксис:

```
switch(вираз)
{
[оголошення]
.
[case константне_вираз1]; [група операторів 1]
[case константне_вираження2]; [група операторів2]
[case константне_вираження3]; [група операторів3]
.
 [default ; [група операторівN]
};
```

Вираз, наступний за ключовим словом switch в круглих дужках, може бути будь-яким виразом, допустимим у мові CI, значення якого повинне бути цілим. Значення цього виразу є ключовим для вибору з декількох варіантів. «Тіло» оператора switch складається з декількох операторів, помічених ключовим словом case (випадок) з подальшим константним виразом. Кожен випадок повинен бути помічений або цілим числом, або символьною константою, або константним виразом. Всі константні вирази в операторі switch повинні бути унікальні. Схема алгоритму конструкції «switch – case» має вигляд, показаний на рис. 4.10 а. Конструкція множинного вибору «switch – case» є своєрідний «перемикач».

Він працює таким чином. На першому етапі аналізується вираз, що перевіряється, і здійснюється перехід до операторів тієї гілки програми, для якої його значення співпадає з вказаним константним виразом. Після обчислення виразу в заголовку його результат послідовно порівнюється з константними виразами, починаючи з самого верхнього, поки не буде встановлена їх відповідність. Після виконання послідовності операторів усередині однієї гілки case, відбувається вихід з конструкції «switch – case». Для виходу з конструкції «switch – case» може використовуватися оператор break ("вивалитися з перемикача"). Break краще писати, інакше, знайшовши потрібний варіант, програма перевірятиме і наступні умови case, марно витрачаючи час

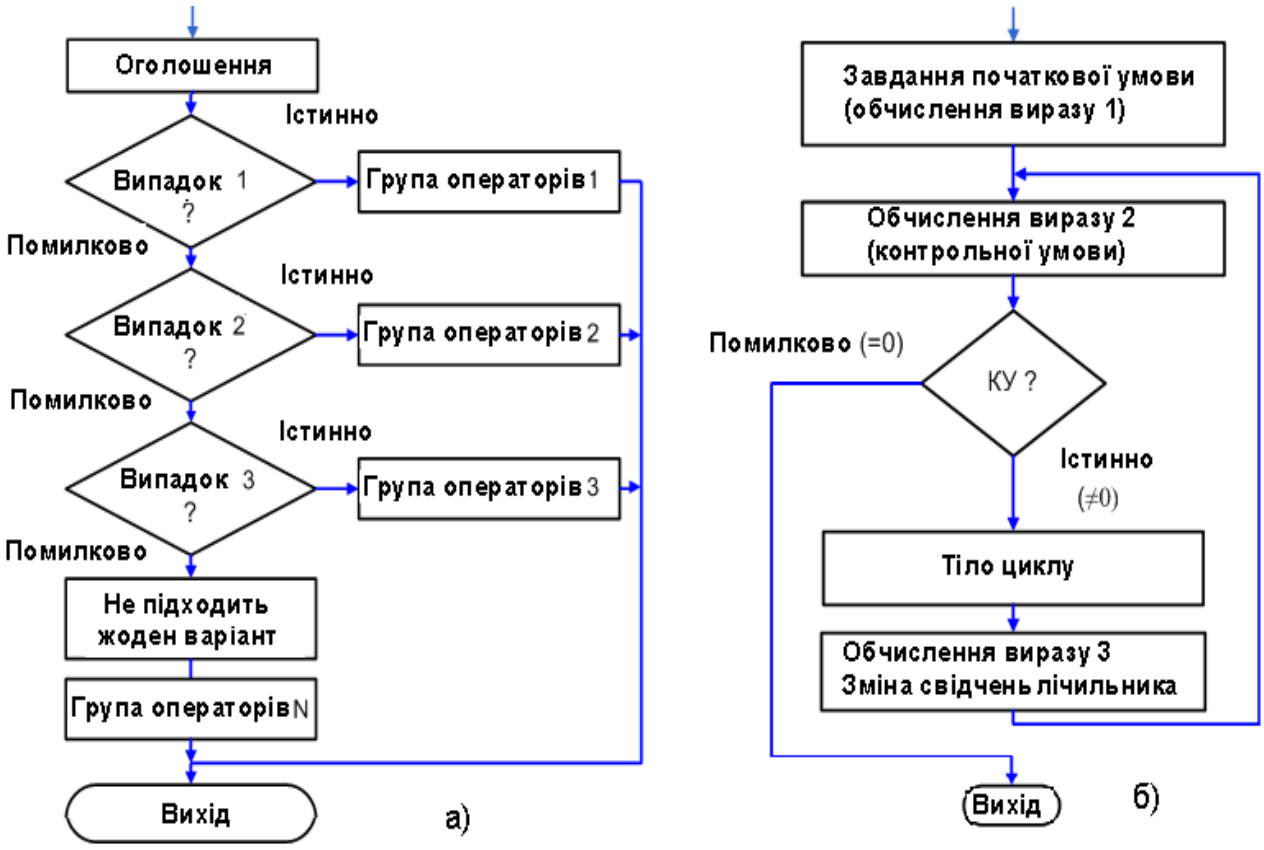


Рис. 4.10. **Схеми алгоритму множинного вибору «switch –** case» (а) і **алгоритму конструкції циклу for (б)**

Якщо жоден з варіантів не підходить то виконується оператор гілки, що стоїть після ключового слова default (за замовчуванням). Наприклад:

```
switch(вираз)
{
case 5:
/* цей код виконуватиметься, якщо результат обчислення виразу рівний числу 5  на цьому робота оператора switch закінчиться */
break;
case -32:
/* цей код виконуватиметься якщо результат обчислення виразу рівний негативному-числу -32  на цьому робота оператора switch закінчиться */
break;
case 'G':
/* цей код виконуватиметься якщо результат обчислення виразу рівний числу відповідному символу G у таблиці ASCII  на цьому робота оператора switch закінчиться */
break;
default:
```

/* цей код виконуватиметься якщо результат обчислення виразу не рівний ні 5 ні -32 ні 'G' на цьому робота оператора switch закінчиться */

};

Приклад:

```
/* Оператор switch */
int x=2;        // Оголошуємо і ініціалізували цілу змінну у
switch (x)      // Оператор switch
{
case 0: x=x-2;        // Тіло оператора switch
case 3: x=x-2;        //
case 2: x=x*5;        //
case 5: x=x/2;        break; // Рядок з оператором break
case 4: x=x+1;        //
default: ;       //
}
[група операторів]
```

/* Виконання оператора switch починається з оператора, поміченого case 2, оскільки x=2. Таким чином, змінна x набуває значення x=2*5=10. Потім виконується оператор, помічений ключовим словом case 5 (по порядку), x набуває значення x=10/2=5. Далі за оператором break відбувається вихід з оператора switch на [групу операторів]. Якби break не було, то ще виконувалися б оператори, помічені case 4 і default, а вже потім [група операторів] */

/* switch закінчений –виконується подальший код програми */

Наступним могутнім механізмом управління ходом послідовності виконання програми є використання циклів (ітераційних структур). Цикл дозволяє повторювати виконання окремих операторів або груп операторів і, тим самим, задає багатократне виконання одного і того ж фрагмента програми. Число повторень у циклі в деяких випадках фіксоване, а в інших визначається в процесі рахунку на основі однієї або декількох перевірок умов. Цикл має точку входження, перевірочну умову і (необов'язково) точку виходу. Цикл, що не має точки виходу, називається нескінченним. Для нескінченного циклу перевірочна умова завжди приймає дійсне значення. Взагалі кажучи, цикли бувають з перевіркою умови перед початком виконання тіла циклу, після закінчення виконання тіла або усередині тіла. Перевірка умови перед виконанням тіла здійснюється в циклах for, while, а після закінчення тіла циклу – do while. Цикли можуть бути вкладеними один в одного довільним чином. Найбільш загальною формою циклу в CI є конструкція циклу **for** ("для кожного"). Синтаксис циклу for має вигляд:

```
for (вираз1 ; вираз 2; вираз 3)
{
«Тіло»- оператор або блок операторів
}
```

Вираз 1 (початкова умова) звичайно використовується для встановлення початкового значення параметра циклу (ініціалізації змінної). Вираз 2 (контрольна умова) – це вираз, що визначає умову, при якій тіло циклу виконуватиметься. Його застосовують як перевірочну умову і на практиці воно, часто, містить вираз з операторами порівняння. Вираз 3 (лічильник) визначає закон зміни параметра циклу після кожного виконання тіла циклу. Він служить для приросту значення лічильника циклів. Наприклад, для for (i=5;i<20;i+=4) i=5 – це початкове выраження1 i<20 – контрольне вираз 2 i+=4 – лічильник. Звичайно вираз 3 це i++, тобто до змінної додається 1 кожен "прогін" циклу. Схема алгоритму конструкції циклу for показана на рис. 4.10 б. При виконанні циклу for спочатку виконується вираз 1, якщо воно присутнє в конструкції. Потім обчислюється величина виразу 2 (якщо воно вказане) і, якщо одержаний результат прийняв дійсне значення, виконується тіло циклу. Після виконання тіла циклу (для значення виразу 2 відмінного від нуля), обчислюється вираз 3, і знову обчислюється вираз 2. Якщо воно, як раніше, відмінно від нуля, то цикл повторюється. Якщо вираз 2 рівне нулю (неправда), то управління передається на оператора, наступного за оператором for. Приклад:

```
/* Оператор for */
/* У даному прикладі обчислюється сума чисел від 1 до 10 */
void main(void)
{
int x;      // Оголошуємо цілу змінну x
int y=0;  // Оголошуємо і ініціалізували цілу змінну y
int z=10;// Оголошуємо і ініціалізували цілу змінну z
for (x=1; x<=z; x=x+1)          // Оператор циклу
{
y=y+x;   // Тіло циклу
}
}
```

/* Тіло циклу (y=y+1) виконуватиметься, починаючи із значення x=1. Після кожного виконання циклу x збільшуватиметься на 1 і тіло циклу виконуватиметься з новим значенням x. Цикл виконуватиметься до тих пір, поки x не перевищить значення z, тобто 10.

Зверніть увагу, що після виконання циклу for, значення x буде рівне 11. Значення y буде рівне 55 */

У програмах часто використовується варіант оператора for, що здійснює нескінченний цикл. Для його організації використовують порожній умовний вираз, а для виходу з циклу використовують додаткову умову і оператор **break** (вийти з циклу). Якщо відсутня перевірка, тобто вираз 2, то вважається, що воно завжди істинно. Приклад:

```
/*  Нескінченний цикл з оператором for  */
for (;;)
{
...
...  break;
...
}
```

Оскільки синтаксис мови CI не допускає пропуску символу крапка з комою (;), то в круглих дужках двічі вказаний порожній оператор (;). Для організації повторень можна використовувати цикл з передумовою **while** ("до тих пір, поки істинно"). Синтаксис конструкції циклу такої:

```
while(умовний вираз)
 «тіло» (оператор);
      ...продовження...
або
while(умовний вираз)
{
 оператори;
            ...
 }
 ...продовження...
```

Як вираз допускається використовувати будь-який вираз мови CI, а як тіло – будь-який оператор, зокрема порожній або складений. Якщо вираз є константа з дійсним значенням, то тіло циклу виконуватиметься завжди і, отже, маємо справу з нескінченним оператором. Нескінченний цикл можна також організувати при допомозі:

```
      while(1)
{
            ...
  }
```

Цикл також виявиться нескінченним, коли умова, визначена у виразі спочатку істинно і ніде далі в тілі циклу не міняється. Схема алгоритму конструкції циклу while показана на рис. 4.11 а.

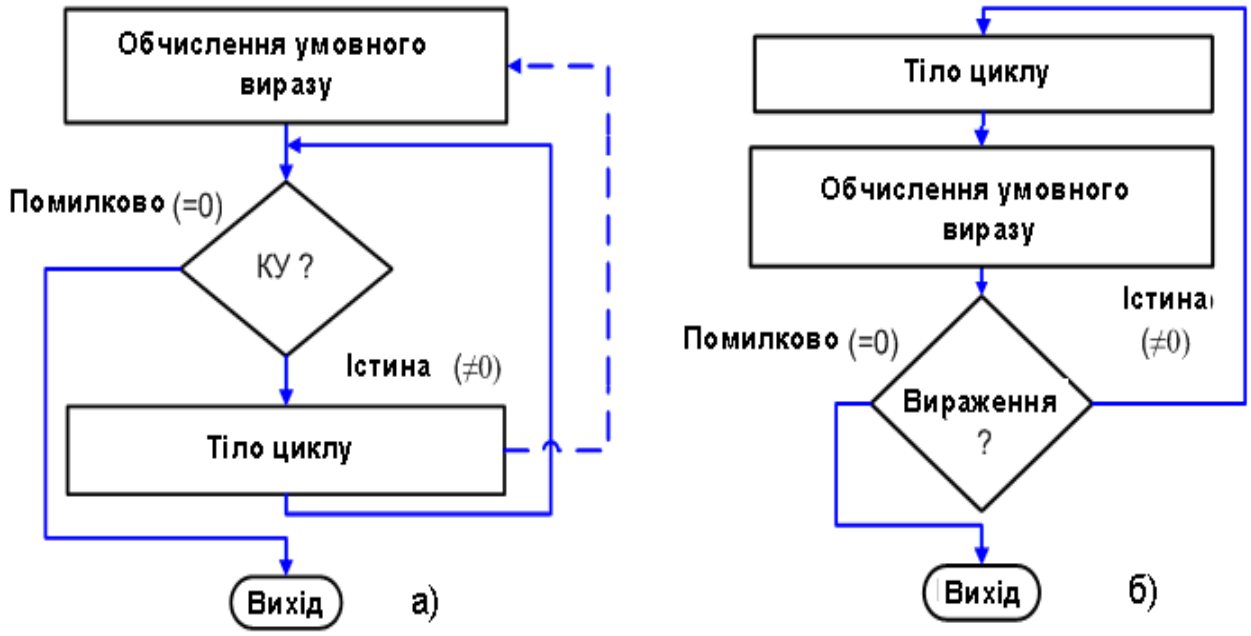


Рис. 4.11. **Схема алгоритму конструкції циклу while (а) і циклу do –while (б)**

У циклі типу while перевірка умови проводиться перед виконанням тіла циклу. Спочатку обчислюється умовний вираз. Якщо результат обчислення умовного виразу рівний нулю (вираз помилковий), то виконання циклу типу while закінчується і виконується наступний за порядком оператор. Інакше, тобто тоді, коли результат обчислення умовного виразу не рівний нулю, то виконується тіло циклу (оператор або група операторів) і процес повторюється спочатку. При цьому оператор або група операторів, що становлять тіло циклу, повинні, як правило, змінювати значення однієї або декількох змінних, що входять в умовний вираз (від яких залежить істинність умови повторень) з тим, щоб, врешті-решт, вираз перетворився на нуль, і цикл завершився.

Приклад1.

```
int i;
i = a;  /* початкова ініціалізація */
while(i < b)
{
        тіло_циклу;
 i += з; /* збільшення лічильника */
 }
       ...продовження...
```

Приклад2.

```
/* Оператор while */
/* У даному прикладі обчислюється факторіал числа 5 */
void main(void)
{
int x=1;  // Оголошуємо і ініціалізували цілу змінну x
int y=1;  // Оголошуємо і ініціалізували цілу змінну y
int z=5;  // Оголошуємо і ініціалізували цілу змінну z
while (x<=z)        // Оператор циклу
{
/* Тіло циклу while */
y=y*x;    // обчислюємо значення y
x=++x;   // інкрементуемо (збільшуємо на 1)
// значення x
}
}
```

/* Тіло циклу (y=y*x, x=++x) виконуватиметься, починаючи із значення x=1. Після кожного виконання циклу x збільшуватиметься на 1 і тіло циклу виконуватиметься з новим значенням x. Цикл виконуватиме до тих пір, поки x не перевищить значення z, тобто 5. Зверніть увагу, що після виконання циклу while, значення x буде рівне 6. Значення y буде рівне 120 */.

Багатократне виконання однієї і тієї ж ділянки програми можна організувати також за допомогою оператора циклу з постумовою «**do – while».** На відміну від оператора «while» цикл «do – while» спочатку перевіряє «тіло» (оператор або блок), а потім вже здійснює перевірку виразу на істинність. Такий оператор може бути застосований у тому випадку, коли необхідно виконати тіло циклу хоч би один раз. Синтаксис конструкції циклу такої:

```
do «тіло» (оператор);
while(умовний вираз)
      ...продовження...
або
do
{
 оператори;
            ...
 }
 while(умовний вираз)
...продовження...
```

Цикл «do – while» припиняє виконуватися, коли умовний вираз звертається в нуль (стає помилковим). Схема алгоритму конструкції циклу «do – while» показана на рис. 4.11 б. Спочатку виконується тіло циклу

«do – while» (може бути складеним оператором), а потім обчислюється умовний вираз. Якщо воно істинно (значення виразу відмінно від 0), то тіло циклу виконується знову. Якщо вираз стає помилковим (значення виразу рівне 0), то виконання оператора «do –while» закінчується і виконується наступний за порядком оператор. Щоб перервати виконання циклу до того, як умова стане помилковою, можна використовувати оператора break. Приклад:

```
/* Оператор do-while */
/* У даному прикладі обчислюється факторіал числа 5 */
void main(void)
{
int x=1;  // Оголошуємо і ініціалізували цілу змінну x
int y=1;  // Оголошуємо і ініціалізували цілу змінну y
int z=5;  // Оголошуємо і ініціалізували цілу змінну z
do        // Оператор циклу
{
/* Тіло циклу do-while */
y=y*x;    // обчислюємо значення y
x=++x;   // інкрементуемо (збільшуємо на 1)
// значення x
} while (x<=z);     // перевірка умови
}
```

/* Тіло циклу (y=y*x, x=++x) виконуватиметься, починаючи із значення x=1. Після кожного виконання циклу x збільшуватиметься на 1, потім перевірятиметься умова (x<=z) і, якщо воно істинно, тіло циклу повторюватиметься з новим значенням x. Цикл виконуватиме до тих пір, поки x не перевищить значення z, тобто 5. Зверніть увагу, що після виконання циклу do-while, значення x буде рівне 6. Значення y буде рівне 120 */

На закінчення розглянемо декілька операторів, які надають допомогу програмісту при організації галужень і циклів.

Оператор **break** забезпечує виконання самого внутрішнього з об'єднуючого його операторів switch, do – while, for, while. Він може зустрічатися у тілі циклу скільки завгодно раз і дає можливість управляти виходом з циклу інакше, ніж перевіркою умови на початку або в кінці циклу. Після виконання оператора break управління передається оператору, наступному за перерваним. Приклад:

```
/* Оператор break */
/* Пошук у заданому масиві елементу, рівного 0 */
/* Підключимо заголовний файл із стандартними функціями введення-висновку */
#include <stdio.h>  // підключаємо  заголовний файл з
```

```
// стандартними функціями введення-висновку
int array[7]={-3,-2,-1,0,1,2,3};       // Оголошуємо і ініціалізували
// цілочисельний масив array
int n=7;         // Оголошуємо і ініціалізували цілу змінну
// n – кількість елементів масиву
void main(void)       // Основна функція програми
{
int i;    // Оголошуємо цілу змінну i, яка
// задаватиме індекс елементу
// масиву array
for(i=0;i<n;i++)        // Цикл for
{
/* Тіло циклу for */
if(array[i]==0)          // Якщо i-ий елемент масиву рівний 0
break;          // те вихід з циклу for по break
}
/* Оператор if-else */
if(i==n)         // Якщо i=n,
printf("%d not found \n",0);          // те виводимо рядок:
// "0 not found "
else    // Якщо i не рівний n
printf("%d on %d place \n",0,i);    // те виводимо рядок:
// "0 on i place ", де
// i – номер місця
}
```

/* Тіло циклу (if(array[i]=0) break) виконуватиметься, починаючи із значення i=0. Після кожного виконання циклу і збільшуватиметься на 1, потім перевірятиметься умова (i<n) і, якщо воно істинно, тіло циклу повторюватиметься з новим значенням i. Цикл виконуватиметься до тих пір, поки і не стане рівне n, або вираз у тілі циклу for не стане істиною, і буде здійснений вихід з циклу по оператору break. У обох випадках буде здійснений перехід на оператора if-else. */

Здійснити переривання можна також за допомогою оператора **continue**. На відміну від оператора break він приводить до початку наступної ітерації охоплюючого циклу (do – while, for, while). У циклах while, do – while це означає безпосередній перехід до виконання перевірочної частини. У циклі for управління передається на крок зміни параметра циклу. Приклад:

```
/* Оператор continue */
/* У цьому фрагменті оброблятимуться тільки позитивні
   елементи масиву array */
...
for (x=0; x<n; x++)  // Цикл for
```

```
{
/* Тіло циклу for */
if (array[x]< 0)        // Якщо елемент масиву менше 0
continue;      // те наступну частину тіла циклу пропускаємо
                                // і переходимо до наступної ітерації циклу for
/* Тут проводиться обробка позитивних елементів масиву */
[група операторів]
}
```
.../* x – індекс елементу масиву array[x]. Виконання циклу for починається з x=0. Перевіряється елемент array[0] масиву і, якщо він менше 0, то по оператору continue здійснюється перехід до наступної ітерації циклу for, тобто x збільшується на 1 (x++) і перевіряється елемент array[1] і т. д. Якщо елемент масиву array[x] більше 0, то оператор continue пропускається і виконується [група операторів], яка обробляє позитивні елементи масиву. Після виконання [групи операторів] також здійснюється перехід до наступної ітерації циклу for і цикл знову повторюється з x=x+1. Цикл for виконуватиметься, поки значення x не стане рівне (або більше) значення n (умова x<n) */

Оператор **return** завершує виконання функції, в якій він заданий, і повертає управління у визивану функцію, в крапку, безпосередньо наступну за викликом. Синтаксис: return (вираз). Значення виразу, якщо воно задане, повертається у визивану функцію як значення функції, що викликається. Якщо вираз опущений, то повертане значення не визначене. Вираз може бути поміщений в круглі дужки (їх наявність необов'язково). Якщо у якої-небудь функції відсутній оператор return, то передача управління в зухвалу функцію відбувається після виконання останнього оператора функції, що викликається. При цьому повертане значення не визначене. Якщо функція не повинна мати повертане значення, то її слід оголосити з типом void. Приклад:

```
/* Оператор return */
/* Функція summa */
int summa (int x, int y)
{
return (x+y);          // Задамо повертане значення
}
/* Основна функція програми */
void main(void)
{
int a=5, b=10;         // Оголошуємо і ініціалізували цілі
// змінні a і b
int z;   // Оголошуємо цілу змінну z
c= summa (a, b);     // Обчислимо значення z через виклик
// функції summa
```

```
}
/* Функція summa має два формальні параметри x і y типу int, і
   повертає значення типу int, про що говорить описувач, що стоїть
   перед ім'ям функції.
   Після виклику функції summa в основній програмі оператор return
   поверне значення, рівне сумі фактичних параметрів a і b, тобто
   c=a+b=5+10=15 */
```

Порожній оператор складається з крапки з комою. При його виконанні нічого не відбувається. Він дозволяє вказати порожнє тіло в операторах циклу, де по синтаксису потрібен хоч би один оператор. Приклад:

```
/* Порожній оператор */
/* Організація нескінченного циклу за допомогою порожнього оператора */
while (1)       // Умова в дужках завжди істинно
{
/* Тіло циклу while */
;       // Порожній оператор
}
```

Контрольні запитання

1. Охарактеризуйте основні особливості програмування мікроконтролера.

2. Що таке призначений для користувача код програми, об'єктний і виконуваний код?

3. Наведіть найпростіший приклад програми на Асемблері.

3. Опис програм.

4. Арифметичні і логічні команди. Арифметичні підпрограми.

5. Охарактеризуйте основні особливості програмування на мові CI.

6. Наведіть найпростіший приклад типової програми на мові CI.

7. Алгоритм. Основні функціональні блоки схеми алгоритму. Побудова алгоритму.

8. Поняття виконавця. Система приписань виконавця.

9. Особливості мови Асемблера для AVR.

10. Можливості мови CI у прикладному програмному забезпеченні мікроконтролера AVR.

11. Структура програми мовою CI.

12. Схеми алгоритмів організації циклів.

13. Схема алгоритму багатоальтернативного рішення.

14. Схема алгоритму умовного оператора if.

15. Схема алгоритму численного вибору «switch – case».

16. Схема алгоритму конструкції циклу do – while.

17. Назвіть основні стадії при створенні прикладного програмного забезпечення для комп'ютеризованого обладнання.

18. Наведіть визначення алгоритму при написанні програми.

19. Наведіть зображення основних функціональних блоків алгоритмів.

20. Наведіть схему алгоритму організації циклів з передперевіркою.

21. Наведіть і пояснить схеми алгоритмів розгалуження програм.

22. Наведіть і пояснить схему використання програми «асемблера».

23. Кожна пропозиція мови Асемблера містить 4 фіксовані поля. Які?

24. Директиви в мові Асемблера. Дайте пояснення.

25. Логічна організація простої програми на мові СІ.

26. Назвіть особливості виводу у файл у мові СІ.

27. Змінні і типи даних мови СІ.

28. Компілятор і компіляція програми. Поясніть.

29. Функції мови СІ. Основні поняття.

30. Назвіть основні операції, оператори, вирази мови СІ, використовувані у «тілі» функції.

31. Назвіть основні структурі мови СІ, що управляють, використовувані у «тілі» функції.

Тема 5. Програмне забезпечення взаємодії людини з комп'ютеризованим обладнанням

5.1. Програмне забезпечення засобів введення – виведення динамічної інформації. Типові апаратні і програмні рішення для введення інформації

5.1.1. Початкові відомості про типові пристрої введення інформації

У поліграфічному обладнанні мікроконтролери часто працюють автономно за допомогою програми, що задається попередньо, без втручання людини. Разом з цим у поліграфії використовуються так звані інтерактивні комп'ютеризовані системи, які включають в контур управління людину – оператора. Інтерактивні управлінські системи вимагають вводити в МК оперативну інформацію і відображати її у формі, зручної для людини. Як пристрої ручного введення знакової інформації найбільшого поширення в комп'ютеризованому обладнанні набули цифрові, алфавітно-цифрові і спеціальні клавіатури (Keyboard). У різних за призначенням і функціональної складності системах управляіння використовуються наступні набори кнопок і види клавіатур: 1. Прості, використовувані для введення команд управління типу «Пуск», «Зупинок» і т. п. 2. Цифрові, призначені для введення даних і команд управління режимами роботи обладнання. Вони звичайно складаються з клавіш, яким відповідають або шістнадцятиричні цифри, або команди управління типу «Відобразити значення температури вузла», «Встановити формат А4», «Start» і т. д. 3. Алфавітно-цифрові, призначені для введення не тільки цифрових, але і текстових даних. За допомогою таких клавіатур оператор «передає» дані і команди МК (комп'ютеру, фотоапарату, відеокамері, мобільному телефону і т. д.) за допомогою введення букв, цифр і спеціальних символів. В якості останніх можуть бути «гарячі клавіші» – службові клавіші, що викликають певні дії програми. У роботі з програмними продуктами професіонали вважають за краще використовувати саме ці клавіші. 4. Спеціалізовані клавіатури, в яких кожній клавіші відповідає певний порядок дій в процесі управління. Наприклад, «Створити вакуум у пристрої самонаклада», «Включити охолоджування інфрачервоної лампи» сушильного

пристрою і т. п. 5. Багатофункціональні клавіатури на основі сенсорів, що доповнюються змінними шильдиками (невеликими лицьовими панельками) з відповідними написами, або спеціальними пристроями відображення подібної інформації на сенсорному екрані. «Програмовані клавіші» (softkeys) являють собою ряд «кнопок» розташованих на екрані, причому їх призначення може бути програмно змінено. Це значно підвищує гнучкість використання клавіатури, дозволяє при невеликій кількості клавіш забезпечити наочність і зручність в управлінні. Багатофункціональні клавіатури на основі сенсорів, за наявності відповідних програмних засобів, дозволяють на одних і тих же апаратних засобах реалізувати набір різноманітних технологічних мов і забезпечити оперативну їх заміну.

Клавіатура, за своєю суттю, є набір ключів, що електрично замикаються або розмикаються. Але при цьому механізм спрацьовування цих ключів може бути різним. У даний час основна маса виробів оснащена натисненними (тактильними), контактними мембранними клавішами, в яких формування електричного сигналу забезпечується замиканням однакових струмопровідних контактів у вигляді двох гнучких мембран, розташованих паралельно один одному на деякій відстані і розділених пластиковою плівкою з отворами напроти контактів. Повернення клавіші після натиснення здійснюється за рахунок купольного резинового буфера, що є аналогом пружини в механічній клавіатурі. Безконтактна клавіатура також є аналогом перетворювача механічних дій в електричні сигнали. «Програмовані кнопки» на сенсорному екрані, поєднуючи в собі орган управління і область зображення, представляючи компактний і досить складний людино-машинний електронний інтерфейс, є безконтактними аналогами ключа. Формування сигналу в сенсорних клавішах забезпечується дотиком оператора до спеціальних пластин – сенсорів. Доторкнувшись до такої «кнопки» на екрані, залежно від використовуваного фізичного принципу (зміни місткості або опору електричного ланцюга при дотику і т. п.) оператор проводить дії управління.

Якщо клавіатура має невелику кількість кнопок N, то до МК кожна клавіша може бути підключена як окрема кнопка до двох розрядів свого порту входу. Число використовуваних ліній для з'єднання і кількість необхідних розрядів портів буде 2N. Якщо в клавіатурі число кнопок перевищує 8, то, число необхідних восьмирозрядних портів буде більше двох (>16) і, оскільки кількість вхідних восьмирозрядних портів у МК звичайно обмежена, то необхідно шукати рішення, яке б дозволило зменшити чис-

ло електричних ліній для з'єднання МК з кнопками і, відповідно, кількість портів входу. Ідея одного з таких рішень полягає в тому, щоб закодувати інформацію про стан N кнопок М-разрядним кодом, у якого $N = 2^M$. Тоді, наприклад, для передачі інформації, про стан 16 кнопок буде потрібно всього 4 лінії і можна використовувати всього 4 розряди порту. Апаратурна реалізація такого способу з мінімальною кількістю з'єднань полягає у використанні в клавіатурі «матриці», складеної з клавіш. У схемі кодування з двокоординатною (X-Y, матричної) адресацією клавіш кожна клавіша знаходиться у вузлі матриці провідників, що складається із m_X стовпців і m_Y рядків, кількість яких вибирають з умови $N = m_X m_Y$. При квадратній матриці $m_X = m_Y = \sqrt{N}$. Загальна кількість виводів з матриці клавіш набагато менша, ніж при використанні для кожної кнопки двох виводів. При N=128 необхідно 256 і 12 виводів, відповідно.

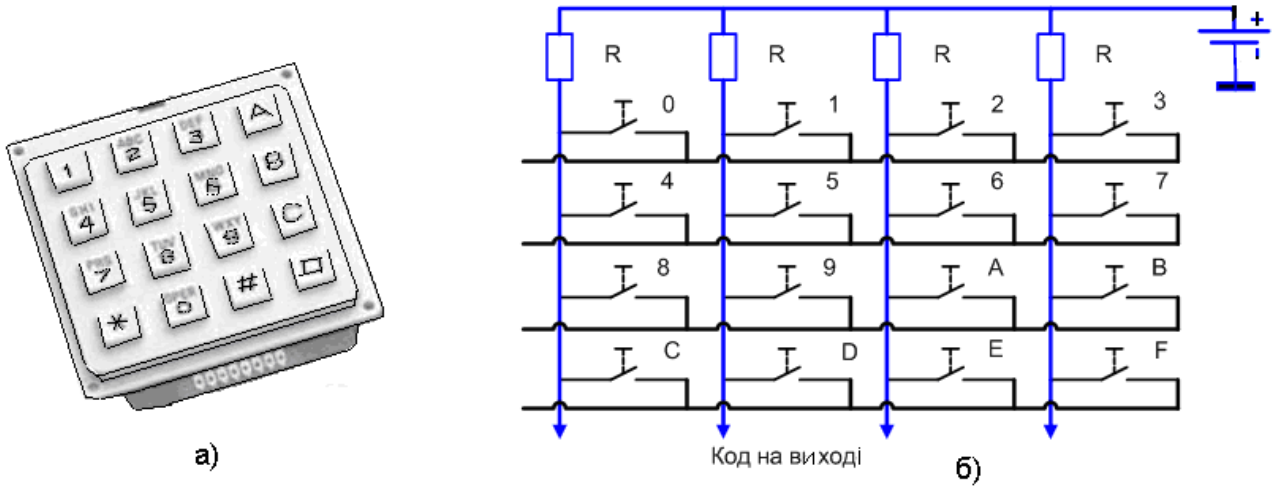


Рис. 5.1. **Зображення шістнадцятипозиційної матричної (4x4) клавіатури (а) і електрична схема некодуючої матричної клавіатури (б)**

Дешеві і прості матричні клавіатури становлять систему двійкових перемикачів (ключів), включених на перетині рядків і колонок якоїсь матриці (X-Y) необхідної розмірності. Часто в клавіатурах використовують матриці розмірністю 4x4 або 5x5. Електрична схема шістнадцятипозиційної матричної (4x4) клавіатури представлена на рис. 5.1. Клавіатура складається з двох систем (горизонтальних і вертикальних) провідників (шин) з нормально розімкненими контактами в місцях перетину. Горизон-

тальні провідники (рядки матриці) є вхідними, а вертикальні шини – вихідними, інформаційними. На вертикальні шини від джерела живлення поданий електричний потенціал близько 5 В. Кожна клавіша, що має з'єднаний з вертикальною і горизонтальною шиною один вивід, розташована в «матриці». Вона має свій номер, відповідний її положенню в координатах X-Y. На цифрові клавіші нанесені позначення, відповідні їх кодам (0,1, .., F).

Принцип формування вихідного коду, відповідного натиснутої клавіші, для простої матричної клавіатури полягає в наступному. За відсутності натиснутих клавіш на вертикальних шинах є потенціали, відповідні рівням логічної одиниці, тому на виході клавіатури (на вертикальних шинах) формується двійковий код 1111. Якщо на горизонтальні шини подати потенціал логічного нуля, то при натисненні, наприклад, клавіші 4 і замиканні контакту, що з'єднанає другу горизонтальну і першу (зліва) вертикальну шини, на виході клавіатури встановляться електричні потенціали, які відповідають вихідному двійковому коду 0111. Аналогічно, формуватимуться коди для інших клавіш, що натискаються. Таким чином, при натисненні однієї з клавіш і за умови наявності відповідних напруг на вертикальних і горизонтальних шинах на виході клавіатури з'явитися набір електричних сигналів, який відповідає певному двійковому коду.

5.1.2. Введення інформації у мікроконтролер з клавіатури

Кодуюча клавіатура може бути з вбудованим контроллером, який електронним способом формує код, відповідний натиснутій клавіші. У цьому випадку вона складається з двох частин: а) набору (матриці) клавіш, відповідних набору знаків алфавіту і команд управління; б) пристрої кодування, який перетворить сигнал натиснутої клавіші в паралельний код обміну інформацією. Управління роботою і ідентифікацію (кодування) натиснутої клавіші в таких клавіатурах здійснюють виконані у вигляді окремої мікросхеми електронні засоби (кодери, шифратори, постійні запам'ятовуючи пристрої – ПЗП і т. п.). На рис. 5.2 а. показаний приклад використання мікросхеми 74C922 в клавіатурі. Контроллер на інтегральній схемі 74C922 кодує стан шістнадцяти кнопок клавіатури. При цьому він усуває «брязкіт» контактів і формує результат навіть при натисненні декількох клавіш (пріоритетний шифратор). Вихідний код знімається з виходів A, B, C, D. Звичайно клавіатури з контроллером є автономний

виріб, який формує на виході код натиснутої клавіші. Для організації інтерфейсу клавіатури з послідовними або паралельними лініями передачі даних використовується спеціалізована мікросхема.

Схема підключення шістнадцятипозиційної матричної клавіатури до МК показана на рис. 5. 2 б. На схемі лінії порту B використовуються для сканування, а лінії порту D – для опиту матриці клавіш. Діоди забезпечують захист від замикання між собою скануючих ліній у разі одночасного натиснення більш ніж однієї клавіші.

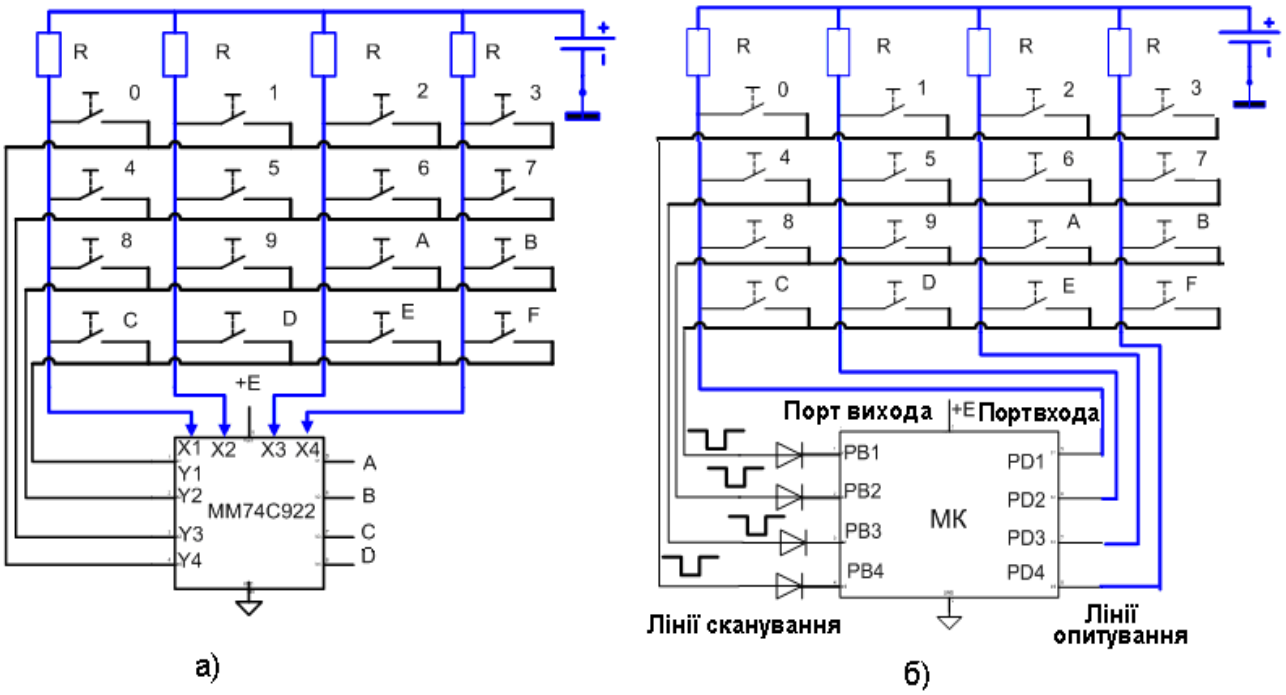


Рис. 5.2. **Електрична схема матричної клавіатури (4x4) з контроллером, виконаному на мікросхемі 74C922 (а) і електрична схема підключення матричної клавіатури (4x4) до МК (б)**

У комп'ютеризованому обладнанні з МК часто використовують дешеві матричні клавіатури без контроллера на мікросхемі, які є просто матрицею двійкових перемикачів, включених на перетині рядків і колонок матриці. Ідентифікація (кодування) натиснутої клавіші в таких клавіатурах виконують за рахунок складання програми МК. При цьому звичайно використовується метод сканування й опитування. Суть методу полягає в наступному. У кожен момент часу програмним шляхом тільки на одній з вихідних горизонтальних ліній матриці формується сигнал логічного 0. На решті горизонтальних ліній у цей час повинен бути рівень логічної 1. Видача сигналу 0 послідовно повторюється для кожної горизонтальної

лінії. У даному прикладі матриці розмірністю 4х4 для горизонтальних шин треба послідовно сформувати чотири вихідних коди: 0111, 1011, 1101, 1110. Після кожної генерації кодів вертикальні лінії матриці «опитуються» МК. Якщо при цьому деяка вертикальна лінія КВ набуває значення 0, то можна програмним шляхом визначити натиснуту клавішу. Адже сигнал на вихідній інформаційній вертикальній лінії матиме значення 0 тільки у випадку, якщо натиснута клавіша з'єднає її з горизонтальною лінією $К_Г$, на якій в даний момент присутній рівень 0.

При обслуговуванні клавіатури програмно треба організувати сканування клавіатури, усунення брязкоту контактів, введення коду і ідентифікацію коду натиснутої клавіші. Сканування служить для виявлення натиснутої клавіші і подальшої її ідентифікації. Схема алгоритму процедури сканування показана на рис. 5.3.

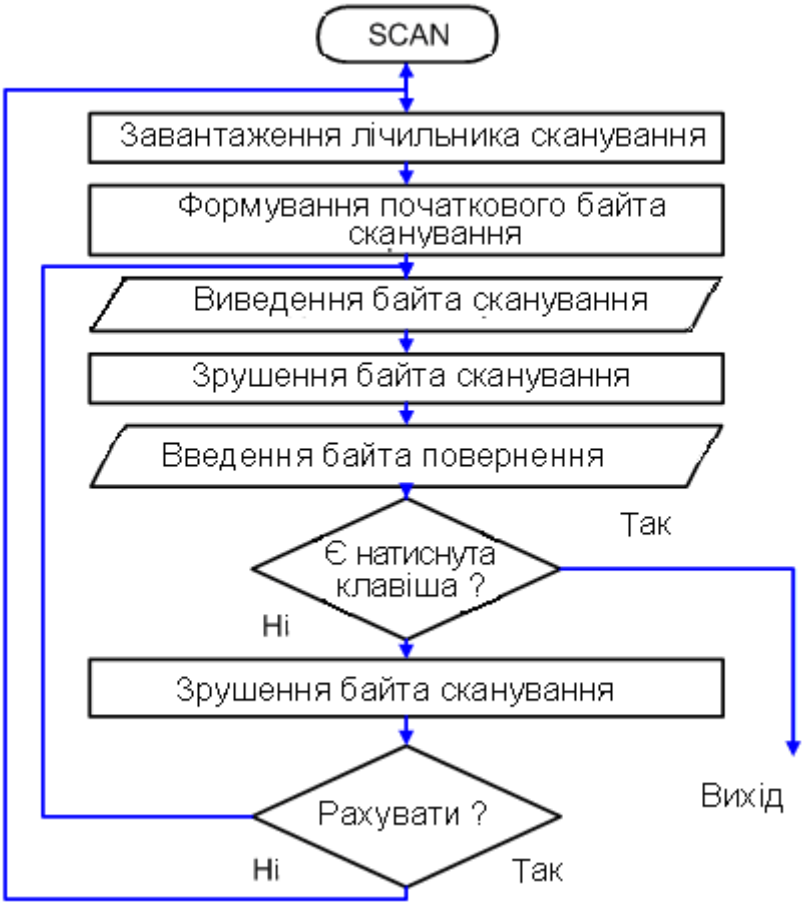


Рис. 5.3. **Схема алгоритму процедури сканування**

Дії зі сканування зводяться до послідовного запису в порт В цифрових комбінацій – байтів сканування, що містять 0 тільки в одному біті. Послідовність байтів сканування становить код «нуль, що біжить», тому

формування чергового байта сканування здійснюється шляхом зрушення попереднього значення. Напрям зрушення визначає послідовність опиту клавіш. Якщо в циклі сканування не виявлено натиснутої клавіші, то сканування повторюється знову. Якщо на перетині лінії сканування і лінії повернення знаходиться натиснута клавіша, то у відповідному біті байта повернення, що приймається портом D, знаходитиметься 0. Наприклад, при подачі напруги низького рівня на другу горизонтальну лінію (у розрядах PB1.PB4 код 1011) на найправішій вертикальній шині при натисненні клавіші 7 з'явиться напруга низького рівня і в циклі опиту в порт D поступить код 1110. При цьому номер горизонтальної шини співпадає з номером циклу сканування в порту B, а номер вертикальної шини визначається положенням нуля в коді порту D. У випадку, якщо хоч би одна клавіша натиснута (байт повернення не всі одиниці) вступає в дію частина програми з умовною назвою «Є натиснута клавіша». У цьому випадку коди портів B і D запам'ятовуються і в дію вступає процедура ідентифікації натиснутої клавіші. Для захисту від брязкоту контактів при натисненні на клавішу вводиться програмний блок часової затримки (5 – 20 мс залежно від механічних характеристик ключів клавіатури).

5.2. Найпростіші типові апаратні і програмні рішення для виведення сигналів управління і інформації

5.2.1. Відображення інформації за допомогою напівпровідникових індикаторів

Початкові відомості. Відомо, що більшу частину інформації (близько 80%) людина одержує за допомогою зору. Якщо інформація створюється і передається електронними засобами, то вона відтворюється за допомогою засобів відображення інформації (ЗВІ). ЗВІ є свого роду «перекладачами» закодованих електричних сигналів у візуальну інформацію. ЗВІ є вельми важливими посередником при організації «діалогу» людини з комп'ютеризованим обладнанням. При виведенні інформації, що зберігається в «пам'яті» комп'ютеризованого обладнання, може створюватися і так звана документальна копія – надрукований на папері текст або зображення, яке можна не тільки бачити очима, але і переносити або зберігати тривалий час. Тому ЗВІ – це пристрої, які використовуються для отримання недокументальних копій. При даній формі копії

інформацію можна бачити лише протягом короткого відрізка часу. Крім того, недокументальна копія не володіє властивістю тривалого зберігання і можливістю транспортування. У цьому сенсі в літературі також використовують термін «дисплей». Дисплеї служать для виведення оператору повідомлень, необхідних даних, або показу якоїсь потрібної для управління графічної інформації. Основними вузлами ЗВІ є індикатори, що перетворюють електричні сигнали у видиме зображення, і інтегральні мікросхеми, що забезпечують «цифрове» управління і «стиковку» електронної й оптичної частин. Індикатор – це прилад, за допомогою якого інформація, призначена для зорового сприйняття, відображається за допомогою одного або сукупності елементів відображення інформації.

Інформація, що є об'єктом індикації різноманітна, тому існує багато типів індикаторів різного призначення. Останнім часом високі технічні характеристики напівпровідникових індикаторів забезпечили їх успішне впровадження в якості елементів індикації в апаратурі і приладах управління обладнанням. Важливим достоїнством напівпровідникових індикаторів є сумісність рівнів їх напруг, що управляють, і споживаних струмів із струмами і напругами логічних рівнів інтегральної електронної техніки. Це дозволяє скоротити об'єми схем управління елементами індикації і підвищити надійність індикаторних пристроїв. Напівпровідникові індикаторні системи є одним з найбільш універсальних класів приладів для відображення інформації різного характеру, які володіють оптимальною сукупністю властивостей, що забезпечують їх широке застосування в пристроях і системах індивідуального і колективного користування. Напівпровідникові індикатори є приладами, де інформація, призначена для зорового сприйняття, відображається за допомогою одного або сукупності дискретних елементів. За виглядом інформації, яка відображається, напівпровідникові індикатори можуть бути класифіковані таким чином.

1. Одиничні індикатори (поширений також термін «світловипромінюючі діоди» – СВД). Вони складаються з одного елементу відображення і призначені, в основному, для представлення інформації у вигляді точки або іншої геометричної фігури. У основі функціонування СВД лежать механізми інжекції неосновних носіїв в гомо – або гетеро p-n структуру з їх подальшою випромінювальною рекомбінацією. Індикатори на основі СВД застосовуються для індикації стану окремих вузлів апаратури (типу «так/ні», «включено/виключено»), підсвічування написів і кнопок, створення табло та ін. СВД працюють тільки при постійному струмі, що

протікає в прямому напрямі, і мають дуже малий диференціальний опір. Тому СВД повинні підключатися до джерел струму, а напруги на них повинні бути більшими від напруги порогу. Схема підключення світлодіода до джерела живлення має вигляд, показаний на рис. 5.3 а. Хоча Е – це джерело напруги, схема підключення світлодіода до джерела напруги, наведена на рис. 5.3 а, може розглядатися як підключення до джерела струму. Це справедливо, якщо (рис. 5.3 б) при протіканні прямого струму $I_{ПР}$ напруга живлення Е більше падіння напруги на світлодіоді U_D, причому, і якщо опір струмообмежуючого резистора R більше диференціального опору світловипромінюючого діода. Щоб забезпечити свічення світлодіода необхідно, щоб через нього протікав постійний струм близько 10 – 15 міліампер. Опір струмообмежуючого резистора в схемі рис. 5.3 а, може бути визначено із співвідношення $R = (E - U_D)/I_{ПР}$. Падіння напруги на світлодіоді U_D залежить від кольору випромінювання (домінуючої довжини хвилі), і це значення беруть з довідкових даних на світлодіод.

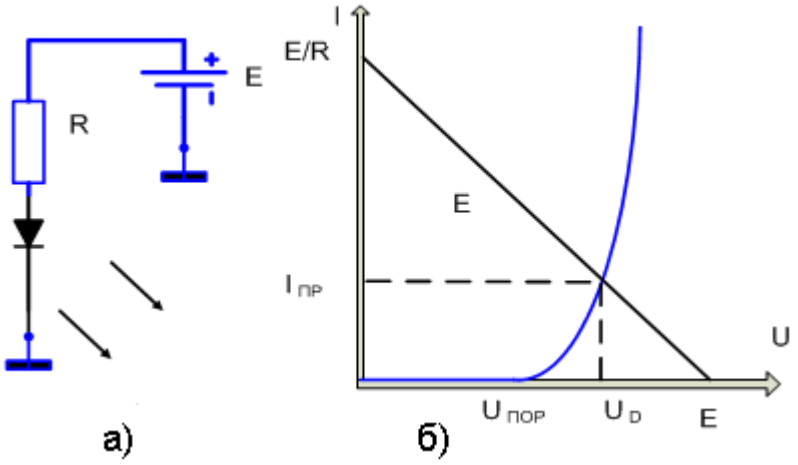


Рис. 5.3. **Схема включення одиничного індикатора (а) і графічний спосіб визначення струмів і напруг на світловипромінюючому діоді (б)**

Величина прямого струму $I_{ПР}$ залежить від того, яку силу світла повинен забезпечити світлодіод у системі «оператор – індикатор». Світотехнічний параметр «сила світла» – це світловий потік, що доводиться на одиницю тілесного кута в напрямі, перпендикулярному площині випромінюючого кристала. Сила світла світлодіода залежить від прямого струму так, як це показано на рис. 5.4. Як свідчать графіки на рис. 5.3 б і рис. 5.4 а силу світла (яскравість випромінювання) світлодіода при проті-

канні постійного струму можна регулювати за рахунок зміни напруги живлення E і, отже, величини протікаючого через світлодіод струму. Регулюючим елементом може служити змінний резистор R, винесений на лицьову панель. Це важливо для забезпечення комфорту прочитування інформації при роботі в приміщенні вночі і при високому рівні зовнішньої освітленості. Іншим варіантом регулювання яскравості свічення індикаторів є використання імпульсного способу, коли через світлодіод струм протікає не постійно, а короткочасно, імпульсно. Як видно з рис. 5.4 б при роботі в імпульсному режимі (при протіканні імпульсного прямого струму) від світлодіода можна одержати набагато більшу силу світла (яскравість свічення). При частоті відновлення інформації на світлодіоді більше 100 Гц, «мерехтіння» випромінювання не будуть помітні для ока людини, а свічення світлодіода сприйматиметься так, ніби він управляється постійним струмом. При протіканні імпульсного струму знижується теплове навантаження на світлодіод (визначається середнім значенням імпульсного струму). Через це напівпровідникові індикатори витримують пікові струми, що в 2 – 3 рази перевершують максимально допустимі при протіканні постійного струму.

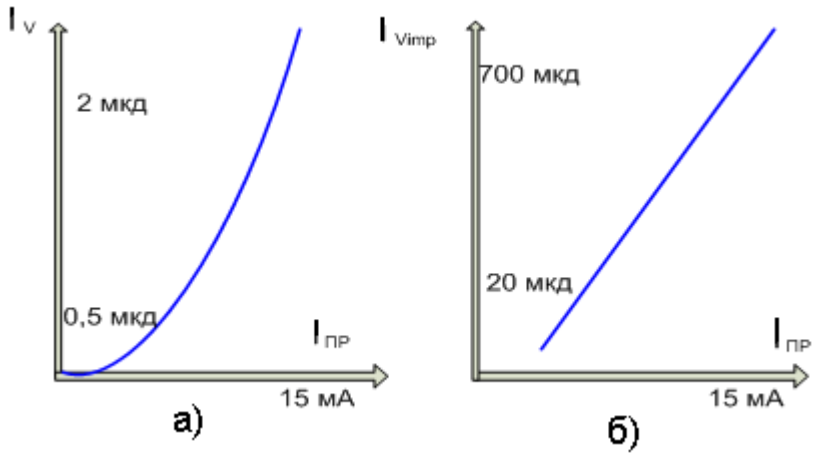


Рис. 5.4. **Залежності сили світла випромінюючого світлодіода від прямого струму: а) при протіканні постійного струму; б) при дії імпульсного струму, що протікає в прямому напрямі**

Дослідженнями встановлено, що відносна ефективність випромінювання збільшується із зростанням амплітуди прямого імпульсного струму. Так, наприклад, яскравість свічення світлодіода, яку він має при протіканні постійного струму 10 мА, можна одержати, якщо через діод

пропускати імпульсний струм 40 мА з частотою 40 Гц при шпаруватості 8. При цьому середній струм через світлодіод складе 5 мА. Таким чином, імпульсний режим живлення дозволяє за допомогою зниження середнього прямого струму через елемент, що світиться, використовувати індикатор без значних втрат сили світла, без порушення гранично допустимого теплового режиму індикатора і при підвищених температурах навколишнього середовища. Все це свідчить і про те, що без погіршення характеристик, яскравості, можна забезпечувати імпульсне управління світлодіодами. При цьому, застосовуючи широтно-імпульсний метод, можна змінювати значення середнього прямого струму, що протікає через світлодіод і, природно, яскравість його свічення.

2. Шкальні індикатори. Вони мають елементи відображення у вигляді правильних багатокутників і призначені для відображення інформації у вигляді рівнів або значень величин. Окрему групу складають так звані лінійні формувачі зображення, в яких індикація інформації може проводитися заповненням шкали від нульового елементу до елементу відповідного максимальному значенню параметра. Крім того індикація інформації може бути типу «хвіст комети» (вид індикації при якому елемент, відповідний максимальному значенню параметра, випромінює максимум світлової енергії; два – три розташованих поряд елементу, відповідні меншим значенням параметра, випромінюють світлову енергію з інтенсивністю, що послідовно зменшується до нуля).

3. Цифрові індикатори. Вони складаються, як правило, з елементів відображення у вигляді сегментів і призначені для відображення цифрової інформації і окремих букв алфавіту. Застосування знаходять 6-, 7-, 8- і дев'ятиелементні (дев'ятисегментні) індикатори (рис. 5.5).

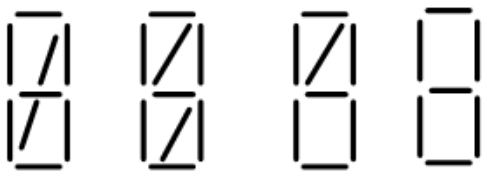


Рис. 5.5. **Рисунки індикаторів, що показують форму і взаємне розташування елементів структурні (зображення, що виникають при включенні всіх елементів індикаторів)**

Найширше (за сумарною оцінкою технологічності виробництва, вартістю, звичністю написання цифр, простотою пристроїв управління)

використовуються для відображення цифрової і буквеної інформації семисегментні індикатори. Сім елементів (сегментів), що відображають, дозволяють висвічувати десяткові і шістнадцятирічні цифри, а також деякі спеціальні знаки. Можливі варіанти виконання семисегментного індикатора з децимальною точкою індикатора показані на рис. 5.6 а. За виконанням семисегментний індикатор становить набір одноелементних напівпровідникових світлодіодів (випромінюючих світло p – n-переходів, виконаних у вигляді сегменту), розміщених на підставі корпусу заданим чином. Сегменти випромінюють світло при проходженні прямого струму. Сім сегментів індикатора позначені буквами від a до г. За допомогою сегменту h відображають крапку, що світиться. Залежно від технології виконання в такій монолітній конструкції напівпровідникові області одного типу можуть з'єднуватися разом, утворюючи індикатор із загальним електродом управління. Можуть бути використані семисегментні індикатори із загальним катодом (рис. 5.6 б), або анодом (рис. 5.6 в). Необхідна комбінація сегментів визначається зовнішньою їх комутацією (підключенням живлячого струму на відповідний сегмент). Схема підключення сегментів індикатора із загальним анодом до порту B МК показана на рис. 5. 6 г.

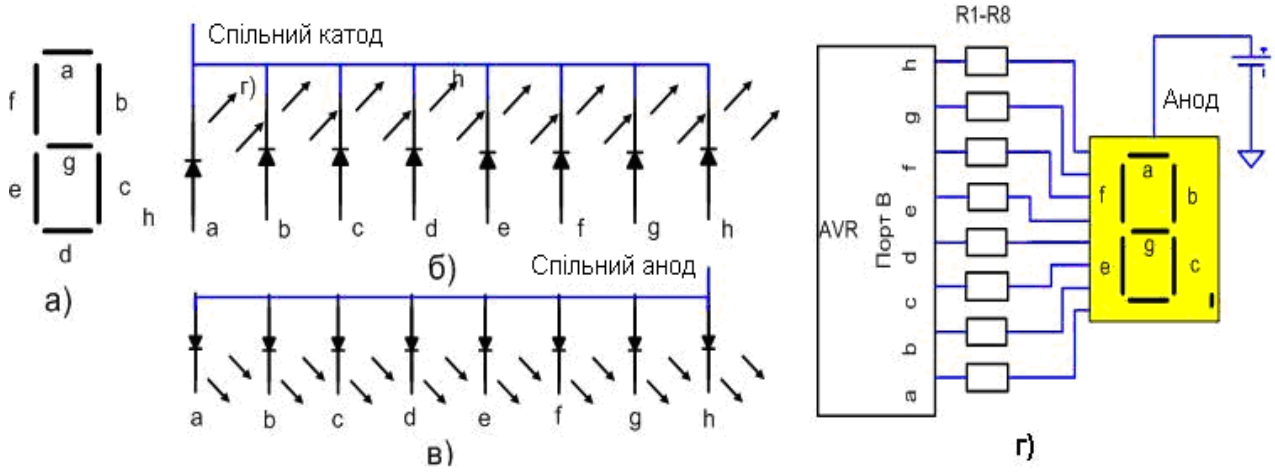


Рис. 5.6. **Буквені позначення виводів і схеми включення семисегментного індикатора із загальним катодом і загальним анодом (а – г)**

Струм від джерела е.р.с. через обмежуючий резистор і випромінюючий світло p – n-перехід, виконаний у вигляді сегменту, втікає в порт і через внутрішній n-канальний польовий МОП-транзистор відводиться на

«землю». Через ніжку порту (пин) МК здатний «поглинати» струм силою до 20 мА. Проте при цьому треба мати на увазі, що загальна сума струмів порту не повинна перевищувати 100 мА. Тобто, в граничному випадку, коли включені всі сегменти індикатора, сила струму, що протікає через світлодіод сегменту, не повинна перевищувати величину 12,5 мА (100 мА/8 = 12,5 мА). Якщо припустити, що пряме падіння напруги на світлодіоді U_D = 1.5 В, то величина опорів R1 – R8 повинна бути $R = \frac{E - U_D - U_{MOS}}{I_{ПР}} = \frac{5B - 1.5B - 0.3B}{12.5mA} = 256Ом$, де U_{MOS} – падіння напруги в каналі польового МОП транзистора. Можна помітити, що при безпосередньому підключенні семисегментного індикатора до порту МК має місце велике «навантаження» порту зі струму. Тому для забезпечення меншого струмового навантаження порту використовуються виконані у вигляді однієї мікросхеми індикаторні формувачі струмів або спеціальні інтегральні схеми перетворювачів кодів для управління семисегментними індикаторами (дешифратори-формувачі). Структурна схема управління напівпровідниковим семисегментним індикатором із загальним катодом показана на рис. 5.7. Функціонування семисегментного індикатора забезпечується дешифратором, який перетворює двійковий код у семирозрядний позиційний код. Оскільки сегменти є струмовими приладами, то для їх нормального функціонування необхідно стабілізувати прямий струм через кожен сегмент. Це завдання виконують формувачі струму.

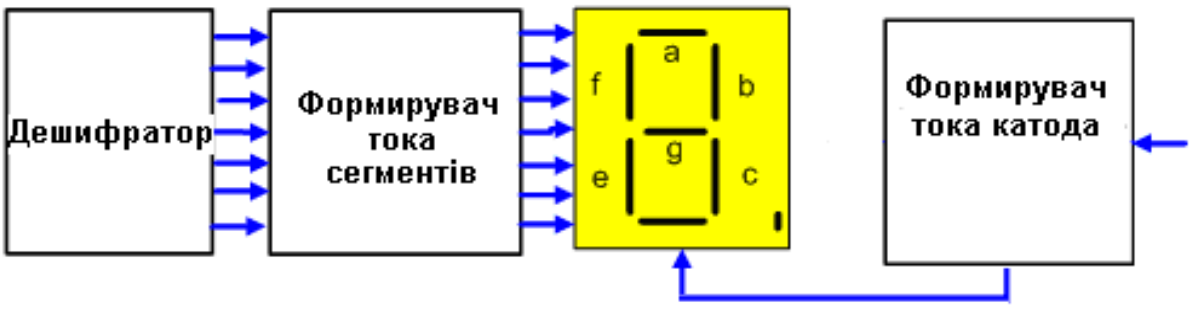


Рис. 5.7. **Структурна схема управління напівпровідниковим семисегментним індикатором із загальним катодом у статичному режимі (режимі постійного струму)**

Індикатор, у якого інформаційне поле дозволяє відобразити тільки один набір знаків (одне знакомісце), називається однорозрядним. У ряді випадків вдається здійснювати безшовну стиковку семисегментних інди-

каторів по горизонталі, що дозволяє створювати інформаційні табло різних розмірів і конфігурації інформаційних полів і необхідної розрядності. У таких монолітних індикаторах є декілька фіксованих знакомісць для відображення інформації і їх називають багаторозрядними (рис. 5.8).

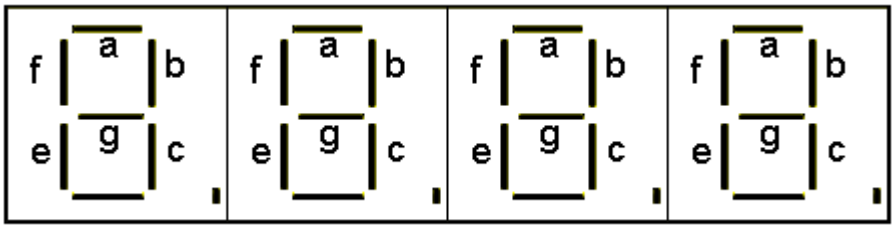


Рис. 5.8. **Чотирьохрозрядне табло, складене з чотирьох семисегментних індикаторів**

На жаль, семисегментні індикатори забезпечують відтворення обмеженого числа знаків (теоретично – 48, а практично – 30). Збільшення числа сегментів, що становлять знак, до 10 – 16, з одного боку дозволяє збільшити число знаків, що відображаються, але, з іншого боку, істотно ускладнює технологію їх виготовлення і схеми управління індикаторами. Істотним недоліком семисегментних індикаторів є і те, що єдина помилка в коді управління, або несправність одного сегменту, призводить до повної неможливості читання цифри. Тому часто замість багатосегментних індикаторів використовують буквено-цифрові індикатори.

4. Буквено-цифрові індикатори. Вони становлять виконану в єдиному корпусі матрицю світлодіодів, в якій одиничні елементи відображення інформації згруповані по рядках і стовпцях. Дослідження показали, що вже 35-елементна матриця дозволяє забезпечити задовільне відтворення знакової інформації, зокрема прописних і заголовних букв російського алфавіту, знаків і цифр, букв грецького і латинського алфавітів. У матричному індикаторі елементи, що світяться, розміщені в m рядках, по n елементів у кожній. При цьому матриця 5x7 діодів, що світяться, повинна мати, принаймні, 36 виводів (35 для кожного одиничного індикатора і один загальний електрод). Для управління матричним індикатором теоретично можна використовувати систему однокоординатної адресації, яка відрізняється тим, що кожен світлодіод має дві незалежні від інших елементів матриці входу. Незалежність один від одного з управління зберігається і тоді, коли для зменшення числа з'єднань другі входи

управління об'єднуються. Оскільки анод і катод у кожного світлодіода незалежні, то при однокоординатній адресації світлодіоди можуть включатися одночасно і на будь-який проміжок часу. Схеми однокоординатної адресації багатоелементних індикаторів мають істотні недоліки – велике число каналів управління. Крім того, для включення одного світлодіода матриці необхідно забезпечити протікання через нього струму величиной 10 – 15 мА. Через це «живлення» матриці постійним струмом зажадає джерела, що забезпечує значний струм, що практично неможливо. Тому, при однокоординатній адресації можливо застосування тільки імпульсного способу живлення, та і то з підключенням матричного індикатора через схеми формування струму.

Для адресації світлодіода в буквених багаторозрядних індикаторах звичайно використовується метод двокоординатної матричної адресації, при якому кожен вихід світлодіода, у відмінності від однокоординатної адресації, під'єднується до виводів інших світлодіодів. При цьому схема управління розбита на частини, з'єднані по рядках і стовпцях з виводами світлодіода. Система матричної адресації забезпечує значне зменшення числа каналів управління і виводів індикатора (за умови їх виконання загальними електродними шинами). Число виводів, необхідних для управління матрицею, визначається таким співвідношенням $N_B = 2[\sqrt{M \bullet (m \bullet n)}] = 2[\sqrt{N_{диод}}]$, де [ ] означає закруглене число в дужках до більшого цілого. Для матричного індикатора з числом діодів $N_{диод}$ = 35х3 одержимо $N_B = 2[\sqrt{105}]$ = 23 замість 105 каналів управління і понад 106 виводів індикатора.

Окремим напрямом світлодіодного ринку стали керовані мікроконтролерами великорозмірні світлодіодні індикатори (панелі) для використання всередині і поза приміщеннями на основі дуже яскравих СВД. Такі панелі візуального відображення інформації дозволяють одержувати близько 16 мільйонів колірних відтінків з кутом огляду близько 120. При цьому використовують СВД червоного, зеленого, синього кольорів свічення. Сила світла окремого світлодіода складає 2 – 10 кандел. Залежно від розмірів системи відстань сприйняття інформації може скласти 150 – 200 метрів.

Все різноманіття схем підключення напівпровідникових індикаторів, вживаних у засобах відображення інформації з МК, можна звести до трьох: 1. Схема паралельного (порозрядного) управління сегментними

індикаторами на постійному струмі. 2. Схема мультиплексного управління сегментними і матричними індикаторами. 3. Схема мультиплексного управління сегментними і матричними індикаторами з пам'яттю. Вибір конкретної схеми включення визначається рядом факторів: характером цифрової інформації, що подається на схему; способом організації інтерфейсу; вибором виду струму живлення і т. д.

5.2.2. Типові апаратні і програмні рішення для відображення інформації невеликої інформаційної ємності багаторозрядними напівпровідниковими семисегментними індикаторами

При організації інтерфейсу МК з семисегментними індикаторами застосовуються статичний (на постійному струмі) і динамічний способи відображення інформації. Для відображення цифрової інформації з малою інформаційною місткістю, відтворюючою одну або дві цифри, може використовуватися однокоординатна адресація (паралельне підключення сегментів) з управлінням на постійному струмі (у статичному режимі). При цьому використовуються індикатори, що мають окремі виводи для кожного сегменту. При управлінні цифровими напівпровідниковими індикаторами в статичному режимі кожен індикатор дисплея забезпечується дешифратором двійкового коду 8-4-2-1 у позиційний код, що сприймається індикатором і формувачами струму (рис. 5.9). Кожен вихід дешифратора управляє через відповідний ключ певним сегментом (a – h).

Інформація, яку треба відобразити, поступає у вигляді логічних рівнів на шини ДК. У сукупності вона являє двійковий код «один з десяти». При цьому кожній чотирьохбітовій двійковій комбінації відповідає відображення однієї цифри на семисегментному індикаторі із загальним анодом. Поступаючі на схему управління дані містять також інформацію про необхідність включення або виключення децимальної крапки на індикаторі. Дешифратор перетворить поступаючу інформацію з двійкового коду (ДК) у семирозрядний позиційний код. Дозвіл на включення того або іншого знакомісця дисплея подається на дешифратор по лініях A1 і A2.

Напівпровідникові сегменти є струмовими приладами і для їх нормального функціонування необхідно стабілізувати прямий струм через кожен елемент. Рішення задачі забезпечення заданого струму через сегмент виконують резистори. З урахуванням неминучого розкиду характеристик різних сегментів і забезпечення рівномірності свічення, харак-

теристики формувачів струму повинні за можливістю наближатися до характеристик ідеального джерела струму. Принципова особливість схеми управління індикатором у статичному режимі полягає в тому, що при двійковому коді, що незмінюється, через сегменти індикаторів, що світяться, тече постійний прямий струм.

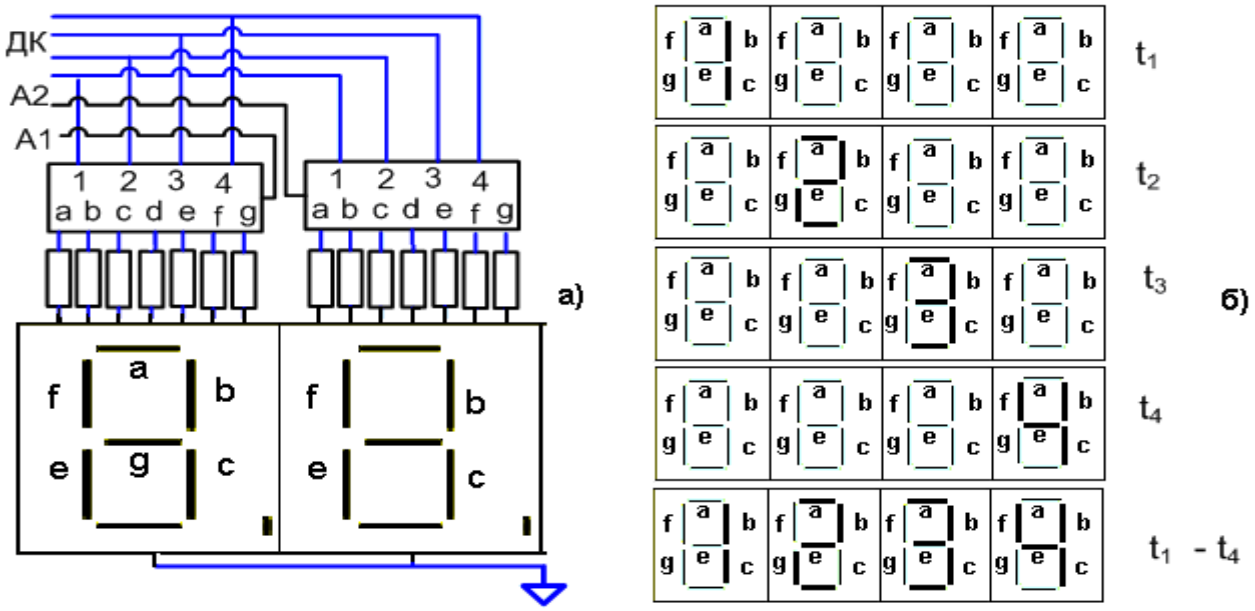


Рис. 5.9. **Структурна схема управління чотирьохрозрядним напівпровідниковим семисегментним індикатором із загальним катодом у статичному режимі режимі постійного струму (а) і зображення, що виникають при динамічному включенні розрядів індикатора, і загальне відображення інформації (внизу), яке сприймає людина (б)**

Окрім монохромних семисегментних індикаторів можуть застосовуватися дисплеї з двома кольорами свічення. Ці індикатори мають вісім анодних виводів (поодинці на кожен сегмент і децимальну точку) і два катодні виводи, кожний з яких об'єднує катоди сегментів різних кольорів. При використанні дешифраторів у подібних пристроях відображення інформації необхідно стежити за тим, щоб виконувалася відповідність між вхідними кодами мікросхеми – дешифратора і вихідними позиційними кодами, відповідними необхідному положенню сегментів, що світяться.

Одним з істотних недоліків схеми управління індикатором у статичному режимі є залежність її роботи від температури навколишнього середовища.

Із збільшенням кількості розрядів (знакомісць) у дисплеї паралельна схема управління, що працює в статичному режимі, стає громіздкою і неекономічною. Тому в пристроях відображення інформації підвищеної і великої інформативності рекомендується застосовувати динамічний спосіб відображення інформації (мультиплексний режим). При цьому в таких пристроях використовують багаторозрядні індикатори монолітної конструкції, у яких виведення управління однойменних сегментів цифр, розміщених в одному корпусі, з'єднаніні разом. Динамічний спосіб відображення інформації заснований на тому, що якщо оновлювати інформацію, що відображається, з частотою більше 20 разів за секунду, то людському оку вона представлятиметься незмінною. Подібний ефект використовується в кіно і в телебаченні. Там, окремі кадри, що змінялися з великою швидкістю, зливаються в одне безперервне зображення. При динамічному режимі розряди індикатора працюють не одночасно, а по черзі. У кожен момент часу «працює» лише одне знакомісце, в якому відображається одна цифра (рис. 5.9 б). Розряди індикатора включаються по черзі, починаючи з першого і закінчуючи останнім. Перемикання цифр у знакомісцях відбувається з великою швидкістю, тому людське око не помічає, що цифри світяться по черзі. Почергові мигтіння цифр зливаються в одну «картину» внаслідок чого людина бачить цифри одночасно у всіх розрядах. Даний спосіб відображення інформації називають мультиплексним, оскільки передача інформації до сприймаючих індикаторів проводиться з «тимчасовим ущільненням», при якому протягом періоду кадру $T_К$ для індикації знакомісця з номером $N_{ЗНАКМЕСТ}$ надається проміжок часу $T_К/N_{ЗНАКМЕСТ}$.

Типова структурна схема управління трьохрозрядним напівпровідниковим семисегментним індикатором в динамічному режимі показана на рис. 5.10. У схемі управління виводи однойменних сегментів (a – g, h) всіх цифр з'єднані між собою і через формувачі струмів сегментів підключені до виводів одного дешифратора. Інформація про цифру, яку потрібно відобразити в певному розряді, у вигляді двійкового коду 8-4-2-1 поступає на вхід дешифратора, де перетвориться в семисегментний позиційний код. Загальні катоди розрядів (знакомісць) через ключі (підсилювачі – формувачі струмів) з певною частотою по черзі під'єднуються до негативного потенціалу виводів скануючого пристрою. Через це в дисплеї завжди світиться тільки один індикатор (індикатор одного знакомісця). Синхронізацію всіх процесів у схемі здійснює генератор тактуючих

імпульсів (на рис. 5.10 не показаний). Для узгодження процесів, що відбуваються, в часі він формує імпульсні сигнали з тактовою частотою f, які поступають на спеціальний вхід дешифратора, де вони за допомогою блоку 2 формують збудливі сигнали сегментів. Синфазно з цим тактові імпульси поступають на скануючий пристрій, який підключає відповідні катоди індикаторів знакомісць.

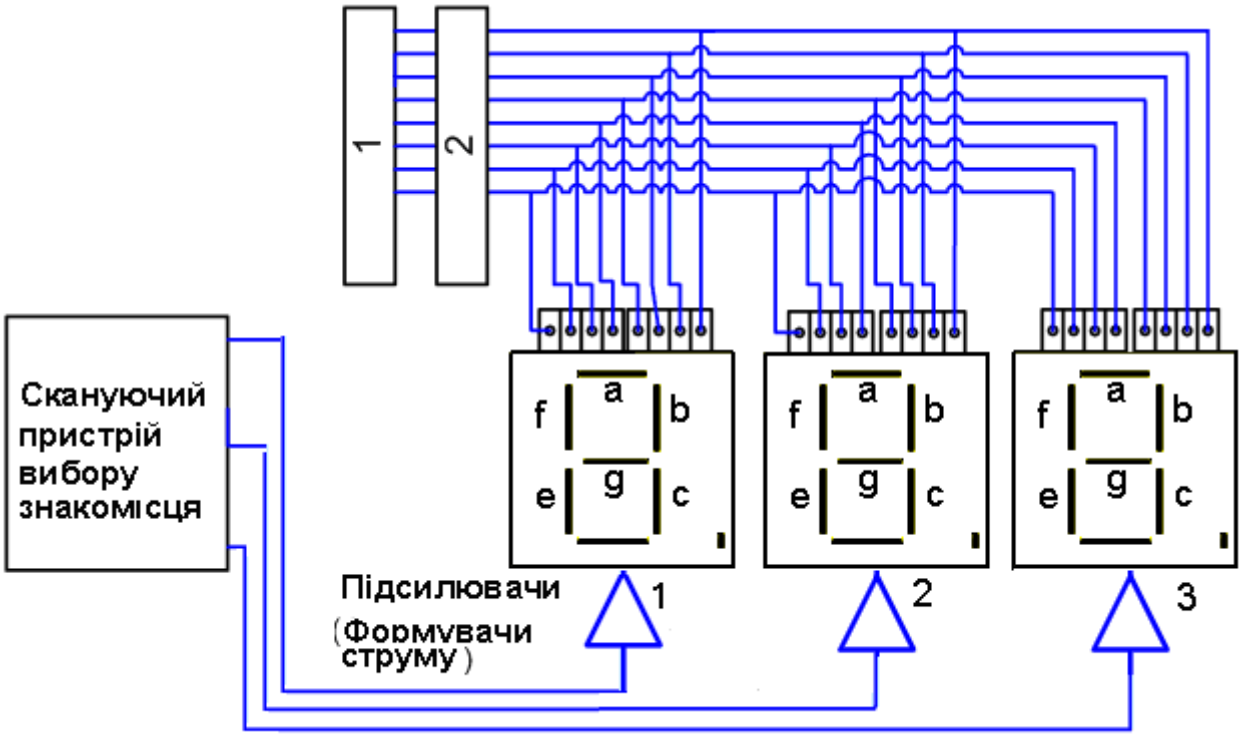


Рис. 5.10. **Структурна схема управління трьома цифровими семисегментними індикаторами в мультиплексному режимі: 1 – дешифратор; 2 – формувач струмів сегментів**

Роботу схеми в динаміці можна представити таким чином. По першому тактуючому імпульсу на вхід дешифратора поступає тетрада двійкового коду для першої цифри. Перетворену дешифратором 1 інформація у вигляді позиційного коду через формувачі струмів сегментів 2 поступає на відповідні сегменти всіх цифр індикаторів. Одночасно, скануючий пристрій замикає струмовий ланцюг катода першої цифри, тому всі сегменти першої цифри починають світитися. За другим тактовим імпульсом скануючий пристрій відключає загальний вивід першої цифри і підключає катод другої цифри. За другим тактовим імпульсом на дешифратор подається тетрада двійкового коду другої цифри. Оскільки за-

мкнутий ланцюг протікання струму для другої цифри, то світяться сегменти другої цифри. Аналогічно відбувається управління і рештою цифр дисплея. Якщо перераховані дії періодично повторювати, то цифри будуть світитися одна за одною.

Управління індикаторами в мультиплексному режимі відбувається циклічно. Кожен кадр відображення інформації розбивається на стільки тактів, скільки знакомісць ($N_{ЗНАКМІСТ}$) містить дисплей відображення інформації. При цьому частота зміни кадрів F_{KADR} =f / $N_{ЗНАКМІСТ}$. Якщо F_{KADR} відповідає частоті 70 – 100 Гц, що відповідає часу циклу 10 – 15 мс, те мерехтіння цифр стає непомітним. Кожний з $N_{ЗНАКМІСТ}$ розрядів включений протягом часу $t_{ВКЛ}$ = 1/f = 1/($N_{ЗНАКМІСТ}$x F_{KADR}). Тобто, час протікання струму через елемент дисплея, що світиться, обернено пропорційний до кількості цифр у керованому наборі і частоті зміни кадрів.

Очевидно, що в динамічному режимі зменшується величина струму, що протікає через сегменти (середній прямий струм). Відповідно зменшується яскравість їх свічення. Для підтримки яскравості їх свічення на колишньому рівні необхідно зберегти середній прямий струм. Зробити це можна за рахунок збільшення амплітуди імпульсного струму.

Мультиплексне управління володіє в порівнянні із статичним способом двома істотними перевагами: 1) такий режим роботи індикатора забезпечує значне зменшення числа каналів управління і виводів індикатора, що дозволяє використовувати невелику кількість виводів (пінів) портів мікроконтролера і елементів управління (ключів); 2) така схема менш енергоємна. Це пояснюється тим, що із зростанням пікового струму індикаторів світловидатність на одиницю струму збільшується.

Мультиплексне управління доцільно застосовувати в пристроях відображення цифрової інформації, що використовують МК. Спрощена схема підключення до МК трьохрозрядного семисегментного дисплея в режимі динамічної індикації показана на рис. 5.11. Управління індикатором здійснюється через порти B і C МК. Вивводи однойменних сегментів індикатора об'єднані і підключені до порту B МК. При цьому лінія, підключена до розряду PB0, управляє сегментом «a», лінія PB1 – сегментом «b» і т. д. За допомогою резисторів, підключених до порту B, створюються джерела струму, які формують струми сегментів. Можливість почергового вибору індикаторів знакомісць забезпечується трьома електронними ключами на біполярних транзисторах, підключеними до порту C. Подавая на розряди порту PC0, PC1, PC2 логічні сигнали, відповідні коду

номера розряда, МК може за допомогою ключа включати відповідний розряд. Решта знакомісць при цьому повинна бути вимкнена. Алгоритм, що реалізовує роботу динамічної індикації, полягає в наступному.

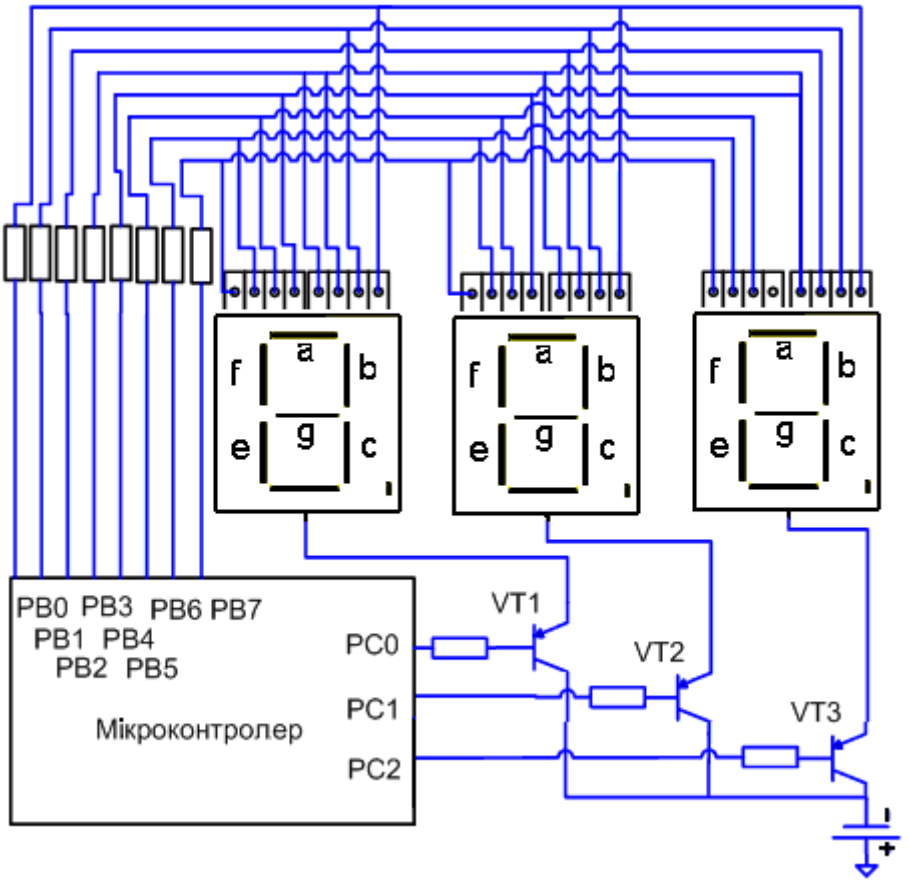


Рис. 5.11. **Структурна схема управління МК трьома цифровими семисегментними індикаторами в мультиплексному режимі**

При динамічній індикації МК повинен «видавати» у порт В байт індикації – двійковий код, відповідний позиційному коду знакомісця. Для цієї мети в МК програмно здійснюється перекодування двійкового коду, відповідного символу, що відображається, в байт позиційного коду, який видається у вихідний порт В. Перекодування здійснюється табличним способом. Цей байт індикації поступає одночасно на всі однойменні входи дисплея. У порті С формується байт вибірки. Звичайно він становить код «нуль, що біжить». Для отримання яскравої і рівної індикації необхідно забезпечити: по-перше, заборону вибірки знакомісць на час зміни байта індикації в порті В (здійснити бланкування); по-друге, регенерацію зображення на кожному знакомісці з частотою, при якій не буде дискомфорту при прочитуванні інформації індикатора. Програма формування

сигналів підключення сегментів і розрядів індикатора, щоб не завантажувати «рутинною роботою» центральний процесорний пристрій, повинна використовувати переривання за таймером.

5.2.3. Типові апаратні і програмні рішення для відображення інформації великої інформаційної ємкості матричними напівпровідниковими буквено-цифровими індикаторами

При організації інтерфейсу МК з напівпровідниковими буквено-цифровими (матричними) індикаторами застосовуються динамічний спосіб відображення інформації і метод двокоординатної матричної адресації. Підключення матричного індикатора до МК здійснюється через схеми формування струму колонок і рядків (рис. 5.12 а).

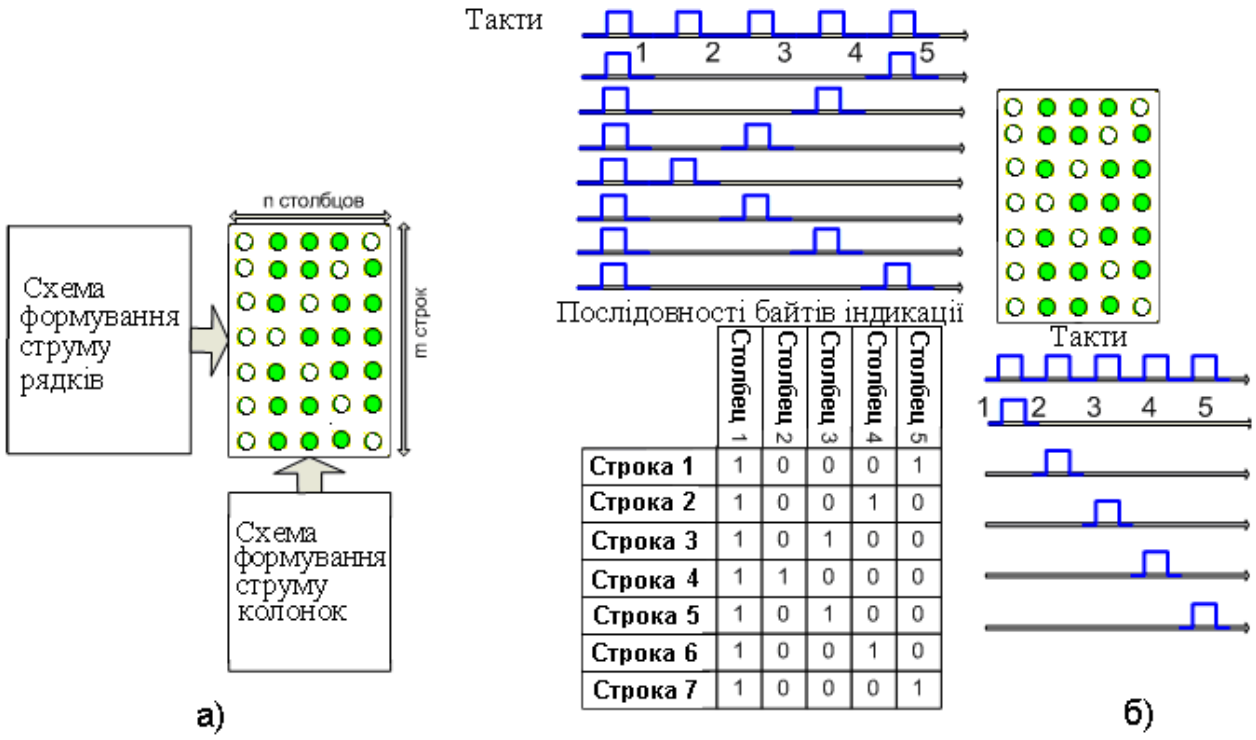


Рис. 5.12. **Схеми підключення матричного однорозрядного індикатора (а) і процедури відображення символу К на знакомісці матричного індикатора (б)**

Матрична структура буквено-цифрових індикаторів дозволяє здійснювати виведення знаку на індикацію шляхом «конструювання» графічного образу символу з набору точок, що світяться. При цьому сама стру-

ктура матриці (mxn) і мультиплексне відображення інформації припускають два можливі способи такого конструювання. При першому способі мультиплексного управління інформація одночасно подається на всі m рядків при послідовному підключенні n стовпців, тобто здійснюється послідовне «підсвічування» матриці за стовпцями (колонками). При другому способі інформація одночасно подається на всі n стовпців при послідовному підключенні всіх m рядків, тобто здійснюється «розгортка зображення» за рядками. Вказані методи індикації одержали відповідні назви – метод стробування за стовпцями і метод стробування за рядками. У режимі стробуванняє («підсвічування» світлодіода шляхом подачі імпульсного сигналу на електрод) за стовпцями, інформація про матрицю розкладається на п'ять (за кількостю стовпців у матриці) семирозрядних (за кількостю рядків у матриці) кодових слів і записується у спеціальну пам'ять, що зберігає ці кодові комбінації. Семирозрядні кодові слова (байти індикації) поступають на всі адресні шини рядків матриці і утримуються в такому стані протягом часу опиту одного стовпця (одного байта вибірки). На кожному такті «засвічуються» необхідні світлодіоди однієї колонки матриці шляхом подачі імпульсного сигналу на відповідний стовпець. У цілому, графічний образ символу «набирається» шляхом розгортки зображення за стовпцями, тобто за рахунок перебору кодів всіх колонок (байтів вибірки). При частоті відновлення інформації на колонці світлодіодів більше 100 Гц свічення матриці сприйматиметься як цілісний образ символу. При підсвічуванні світлодіодів матриці за рядками на кожному такті подається вирішуючий сигнал (сигнал строба) на рядок, а по лінії стовпців подається код стовпця. Інформація, що поступає на адресні шини стовпців, утримується незмінною протягом часу опиту одного рядка матриці. Цей процес повторюється для кожного рядка, внаслідок чого формується графічний образ символу.

Принцип управління матричним індикатором «по колонках» показаний на рис. 5.12 б. Процес формування символу К способом стробування «по колонках» протікає таким чином. Для відображення символу на матричному індикаторі на вхід всіх рядків одночасно подається інформація про те, які світлодіоди стовпця повинні світитися (байти індикації). Байт індикації являє собою код засвічення світлодіодів колонки (див. двійковий код стовпця на рис. 5.12 б). На рис. 5.12 б на рядки подаються імпульсні сигнали високих логічних рівнів, що забезпечують свічення потрібних світлодіодів (у даному прикладі для першого стовпця – всіх семи

(1111111), для другого – середнього (0001000), і т. д.). Звичайно інформація для управління рядками (набір байтів індикації) поступає зі сдвигових регістрів (за числом рядків) і послідовно подається на електроди рядків індикатора. Одночасно з імпульсами, поданими на рядки, на електрод першого стовпця подається сигнал вибірки (сигнал стробування). Цей сигнал для забезпечення свічення колонки формується в результаті подачі на всі електроди стовпців байта вибірки – двійкового коду, у якого тільки на електрод порушуваного стовпця подається імпульсний сигнал високого логічного рівня, а на інші – низького рівня. На першому такті, коли байт вибірки рівний 1000000, починають світитися всі «вибрані» байтом індикації світлодіоди (всі сім світлодіодів першого стовпця матриці). Після закінчення часу експонування першого стовпця інформаційні сигнали (інформаційний байт) і сигнали стробування (байт вибірки) знімаються. На другому такті роботи на вхід всіх рядків подається інформація про висвічення необхідних світлодіодів другого стовпця матриці. У разі символу К на четвертий рядок подається сигнал високого логічного рівня, а на інші низького. Коли на вхід другого стовпця подається стробуючий сигнал, то в результаті висвічується середній світлодіод другої колонки. Висвічення решти елементів відбувається аналогічно. Після кожного такту відбувається зрушення інформації в регістрах, що зберігають байт індикації, і в наступному часовому такті збуджується стробуючий імпульс в черговому стовпці. За п'ять тактів відбувається повна передача інформації, що міститься в регістрах. При частоті відновлення інформації на кожному з рядків не нижче 100 Гц зображення символу К відображатиметься без мигтінь і сприйматися як цілісний образ букви К. Слід мати на увазі, що залежно від кількості рядків і стовпців час протікання струму через кожен світлодіод скорочується в n і m разів з відповідним зниженням яскравості свічення. Для збереження необхідної яскравості свічення необхідно збільшувати середній струм через світлодіоди: імпульсний струм через кожен світлодіод необхідно збільшувати в число раз, відповідне кількості стробованих лінійок. Як відзначалося раніше, напівпровідникові індикатори витримують значні пікові струми, що дозволяє забезпечувати мультиплексне управління матрицями світлодіодів без погіршення характеристик, яскравості.

Спосіб стробування за стовпцями може бути застосований у пристроях відображення інформації на декілька знакомісць. При цьому подальша кількість знакомісць спричиняє за собою зростання шпаруватості

збудливих імпульсів і збільшення, для збереження яскравості свічення, амплітуди імпульсного струму, що протікає через кожен світлодіод матриці. Імпульсний струм не повинен перевищувати максимально допустимої величини, за якою починається зниження квантового виходу напівпровідникового матеріалу. Кількість знакомісць залежить від середнього струму через світлодіод і від максимально допустимого імпульсного струму, тобто від типу індикатора. Спосіб стробування за стовпцями простіший в апаратурному виконанні, чим спосіб стробування за рядками (відзначимо, що процес формування символу К способом стробування «за рядками» протікає аналогічним чином).

5.3. Програмне забезпечення засобів формування звукових сигналів

5.3.1. Початкові відомості про способи формування звукових сигналів

У сучасне життя дуже швидко упроваджуються електронні записники (органайзери), кишенькові персональні комп'ютери (КПК), в яких люди зберігають інформацію про поточні заходи. У простому випадку такі пристрої звуковим сигналом нагадують про момент початку якої-небудь важливої події, виступають у ролі будильника, а в більш загальному – відтворюють складні звукові потоки.

У комп'ютеризованому поліграфічному обладнанні, як і у всіх керованих об'єктах, можуть виникати несправності. Щоб звернути увагу оператора на проблеми, що з'явилися, обладнання оснащують засобами аварійної сигналізації. Разом із засобами візуального відображення небезпеки (у вигляді простої миготливої лампочки або генерування повідомлень на комп'ютерному екрані), застосовують «настирливу» звукову сигналізацію, яка залишається включеною до тих пір, поки оператор не відреагує на аварійний сигнал.

Залучення уваги оператора звуковими сигналами обумовлене тим, що в звичайному житті людина близько 10% всієї поступаючої до нього інформації одержує за допомогою звуків. Слуховий аналізатор людини забезпечує розрізнення не тільки інтенсивності, частоти і тривалості звукових коливань, але і положення джерела звуку в просторі.

Всі звуки можуть бути розділені на прості і складні. Відповідно до цього комп'ютеризоване обладнання може «відтворювати» (генерувати) різні звукові сигнали. Це можуть бути коливання, що відбуваються з однією частотою, які називають чистими тонами. У обладнанні, наприклад, на звукову сигналізацію може виводитися показник інтенсивності якого-небудь процесу, при якому вищому навантаженню відповідає вища частота звуку. Можливо використання сигналів типу «бип – бип» (бип – сигнали, як у першого супутника Землі), різних звуків типу «сирена» (для інформування про настання критичної ситуації). Звуковідтворюючий пристрій, у системах сигналізації, може здійснити генерування складних «голосових і музичних творів». Виявляються ефективними і добре сприймаються оператором керівні вказівки, що подаються «голосом». Для ухвалення коректуючих дій можуть застосовуватися набори звукових файлів, в яких приємним голосом промовляються необхідні в даній ситуації рекомендації. Часто здійснюється дублювання засобів сигналізації: одночасно з голосом з'являється повідомлення на екрані комп'ютера.

Людина здатна розрізняти людську мову, шуми і музичні звуки. Злита мова являє собою послідовність складних звуків тривалістю 20 – 100 мс, спектр яких, як прийнято вважати, зосереджений у смузі 50 – 5000 Гц. При цьому на власних частотах (формантах) голосового тракту спостерігаються резонанси. З погляду механізму створення звукових коливань при вимовленні голосних і деяких дзвінких приголосних голосові зв'язки створюють квазіперіодичні послідовності звукових імпульсів (імпульсів повітря), які порушують резонансні порожнини органів мови і примушують вібрувати голосові зв'язки. Резонансні порожнини визначають резонансні області в спектрі мови, які називають формантами (резонанси особливо виражені на трьох частотах, близько 500, 1 500 і 2 500 Гц). При вимовленні приголосних резонансні порожнини збуджуються шумовим сигналом. Мова є комбінацією складних звуків, змінних за частотою і інтенсивністю. Найбільш високою інтенсивністю характеризуються голосні звуки, а приголосні – менш інтенсивні. Інтенсивність звуку при переході від найбільш гучної голосної до найтихішої приголосної міняється на 30 – 40 дБ. Загальний діапазон зміни інтенсивності мови складає 60 – 100 дБ. Музичні звуки витягуються з музичних інструментів. Існують шумові музичні інструменти (барабани, литаври, кастаньєти та ін.). Крім того, існують звуки типу «гудіння» дротів, «завивання вітру» і інше, в яких можна уловити деяку «музичність».

Для формування всіляких складних звуків (мови і музики) за допомогою електронних і комп'ютерних засобів зараз використовують складні звуковідтворюючі пристрої (процесори звуку), які обробляють звук на програмному і апаратному рівні. Звукові процесори при формуванні звуку можуть створювати різні ефекти, що підвищують виразність і об'ємність звучання. Звуковідтворюючі пристрої, наприклад, в комп'ютерній системі звичайно складаються з двох частин: цифрового контролера і власне кодувальника – декододувальника, який займається перетворенням звукового сигналу їх цифрової форми в аналогову.

Відомо, що для відтворення з високою якістю голосу і музики можуть застосовуватися мобільні мікропроцесорні пристрої з вбудованою і змінною флеш-пам'яттю, відомі як MP3-плейери. Використання системи стиснення MP3 дозволяє не тільки насолоджуватися мелодіями і музичними творами, але і забезпечує невеликий об'єм файлів, в які вони записуються. У MP3-плеєрі мікропроцесор управляє відтворенням музичних файлів, відображає інформацію про поточний звуковий фрагмент на екрані і посилає команди процесору, який «витягує звуки» з пам'яті, формує спеціальні ефекти, управляє алгоритмом стиску і з допомогою ЦАП перетворює цифрові байти на аналогові сигнали.

5.3.2. Формування простих звукових сигналів

У ряді випадків виявляється достатнім, якщо звуковідтворюючим технічним засобом відтворюється не складний музичний твір, а прості звуки. Наприклад, при підтвердженні факту натиснення клавіш необхідні окремі звуки нот від «до» до «сі» (чисті тони з гамми). У інших ситуаціях потрібно відтворити деякі відносно прості мелодії («музичні фрази», подібні тим, коли спрацьовує сигналізація). У цих випадках завдання створення джерела звуку дещо спрощується, оскільки не потрібні складні процесори звуку. Щоб став зрозумілим спосіб створення простих звуків і комбінацій музичних звуків за допомогою електронних звуковідтворюючих засобів, нагадаємо деякі відомості, що стосуються музики. Музичні твори є комбінацію звуків, яку можна характеризувати за допомогою нот. Кожному нотному знаку відповідають певний звук. Використовують наступні назви нот (у порядку, в якому вони відповідають білим клавішам фортепіано): до, ре, мі, фа, соль, ля, сі. Ноти – це символи, що позначають висоту і тривалість музичних звуків. Графічний запис музичного тво-

ру з використанням нотних знаків називається нотним записом. Ноти розташовуються на п'яти лінійках (рис. 5.13), які називаються нотним станом (або нотоносцем).

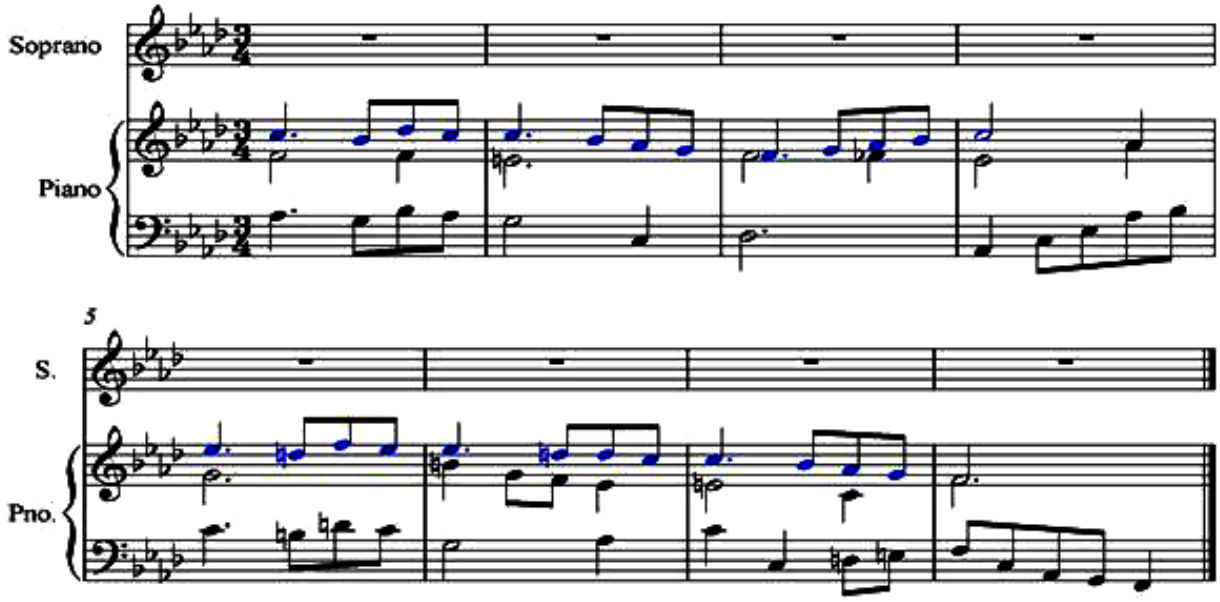


Рис. 5.13. **Розташування нот на нотному стані**

Лінійки рахують від низу до верху. Ноти у вигляді овальних позначок записуються у порядку звучання зліва направо. Кожна нота знаходиться або на якому-небудь рядку нотного стану, або між рядків. Іноді використовуються додаткові лінійки, що розширюють нотний стан вгору або вниз. Додаткові лінійки малюються тільки на таку довжину, яка потрібна для написання нот, що на них знаходяться. Вертикальна позиція ноти (її висота на нотному стані) залежить від висоти її звучання. Кожному рядку (між рядків) нотного стану присвоюється якесь порядкове нотне значення, при цьому порядок нот не змінюється. Наприклад, якщо на другій лінійці знизу знаходиться нота «соль», то між першим і другим рядком розташовується нота «фа», на першій — «мі» і т. д. Таким чином, щоб визначити позиції всіх нот на нотоносці, досить визначити позицію однієї; інші при цьому обчислюються автоматично. Щоб знати, яка нота вибрана як відправна, в музиці існують ключі – спеціальні символи, що записуються в лівому кінці нотоносця. Найбільш поширений скрипковий ключ указує на те, що на другому рядку знизу поставлена нота «соль» першої октави. Існують не тільки «чисті» ноти («до», «ре», «мі», «фа», «сіль», «ля», «сі»), але і їх похідні. Для їх позначення зліва від ноти пи-

шуться символи дієз, бемоль, дубль-дієз і дубль-бемоль, звані знаками альтерації. Дієзи або бемолі можуть стояти на початку кожної строчки нотного стану і означати, що всі ноти даної назви виконуються в дієзному або бемольному варіанті. Колір овалу нот, званого головкою (чорний або білий), і палички, приставлені ним (штилі) і невеликі карлючки на штилях (звані «хвостиками») указують на їх тривалість. Основною тривалістю нот є: а) ціла (біла нота без штилю); б) половина (біла з штилем); в) чверть (чорна з штилем); г) восьма (чорна, штиль якої на кінці має «хвостик»); д) шістнадцята (чорна, штиль з двома «хвостиками»); е) тридцатьдруга (чорна, штиль з трьома «хвостиками»). При цьому тривалість цілої ноти є величина відносна; вона залежить від поточного темпу твору. При завданні тривалості звучання цілої, автоматично задається тривалість звучання решти типів нот. Якщо підряд записані декілька нот тривалістю менше чвертей, то вони записуються під загальним ребром – паличкою, що з'єднує кінці штилів. При цьому, якщо ноти восьмі, ребро одинарне, якщо шістнадцяті – подвійне і т. д. Це ребро іноді називають в'язкою.

У записі нот зустрічається об'єднання нот (найчастіше шістнадцяті, тридцятьдругі і дрібніш) під одне ребро з різних тактів. При цьому над тактовою межею залишається тільки одне ребро. Буває, що потрібно записати ноту тривалістю, наприклад, три восьмих. Для цього є два способи. При першому беруться дві ноти, в сумі що дають три восьмих, і залігговуються, тобто між ними ставиться ліга – дуга, кінцями що майже торкається до овалів нот. При другому способі до довшої тривалості (у нашому випадку – до чверті) праворуч від овалу приписується крапка, що означає, що тривалість ноти збільшена ще на половину. Нарешті, буває необхідним поділити яку-небудь тривалість не на дві половини, а на три, п'ять або іншу кількість рівних частин, не кратну двом. У цьому випадку використовуються триоли, пентоли та інші аналогічні форми запису. Для зміни тривалості ноти у велику сторону на розсуд виконавця в записі використовуються фермата. Крім того, в нотному записі аналогічно самим нотам використовуються паузи для запису ділянок без звучання. Чергування сильних і слабких доль (внутрішнього ритму) мелодії обумовлює її ділення на такти. Такти розділяються тактовою межею, тобто тонкою вертикальною межею, що перетинає всі п'ять лінійок.

Музика складається з різних звуків, які, в першому наближенні, є гармонійними. Звук, який ми чуємо коли джерело його здійснює гармонійні коливання, називається музичним тоном. Наприклад, коливання

камертона близькі до гармонійних. Відхилення від періодичності, обумовлене загасанням, у камертона не велике, тобто амплітуда спадає поволі, протягом дуже великого числа коливань. Його звук відповідає періодичному гармонійному коливанню з певною частотою (певній ноті). Будь-який музичний звук, характеризується певною частотою коливань. Часто за еталон частоти ноти береться нота ля першої октави, частота якої повинна бути рівною 440 Гц. У гармонійного коливання на спектральній діаграмі є одна єдина паличка («стеблинка»), розташована на одній єдиній частоті. Коливання пружної пластинки, затиснутої в лещатах, мають тим більше високу частоту, чим коротше вільний (що коливається) шматок пластинки. Причому зменшення амплітуди, протягом дуже великого числа коливань, в такій звуковій системі відбувається значно швидше. Спектр таких коливань вже складається з набору «стеблинок» на спектральній діаграмі, а в наборі частот можна виділити основну і кратну їй. Частоти створюваних ними звуків знаходяться в смузі спектру від 16 до 5000 Гц. Струни рояля або піаніно утворюють обширний набір коливальних систем з різними власними частотами. Відношення частот двох сусідніх нот для рівномірно темперованого музичного ладу виражається ірраціональним числом, рівним 2 – 1/12. Ноти, відстань між якими кратно октаві, називаються однаково. У кожній октаві формується 12 звуків, віддалених один від одного на півтону (кожна октава ділиться на 12 нот). Від октави до октави частота звукового сигналу міняється в 2 рази.

При формуванні звуків, відповідних певним нотам, у електронних системах необхідно штучним шляхом відтворити акустичний сигнал, від якого б наше вухо одержало враження цієї ноти, подібне тому, яке воно одержує від «природного джерела». Для простої акустичної імітації музичних звуків, як виявилось, необхідно здійснити генерацію періодичної послідовності прямокутних імпульсів, заданої частоти проходження. Якщо імпульси підсилити і перетворити в акустичний сигнал (наприклад, за допомогою п'єзоелектричного перетворювача, телефонного капсуля або гучномовця), то одержимо звуковий сигнал, відповідний певній ноті. При використанні МК для генерації періодичної послідовності імпульсів з шпаруватістю рівної двом (меандр) досить створити в одному розряді порту зміни напруги з 0 на 5 вольт із заданою частотою (рис. 5.14). Значення частот для нот трьох октав, починаючи з ноти «до» першої октави, для імпульсного сигналу з амплітудою майже рівної напрузі 5 вольт наведені в табл. 5.1.

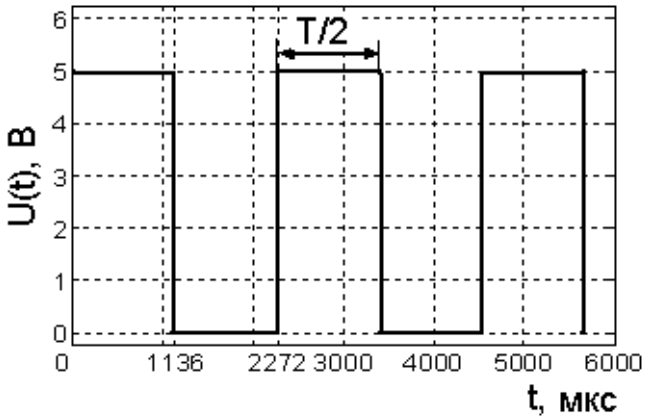


Рис. 5.14. **Імпульсний періодичний сигнал (меандр), який імітує ноту «ля» першої октави з частотою 440 Гц (напівперіод рівно 1136 мкс)**

Таблиця 5.1

Значення частот для нот трьох октав

Нота	Частота, Гц	Число	Нота	Частота, Гц	Число	Нота	Частота, Гц	Число
До1	261,6	4748	До2	523,2	2373	До3	1046,5	1187
До1#	277,2	4480	До2#	554,4	2240	До3#	1108,7	1120
Ре1	293,7	4228	Ре2	587,3	2114	Ре3	1174,6	1057
Ре1#	311,1	3992	Ре2#	622,2	1996	Ре3#	1244,5	998
Мі1	329,6	3768	Мі2	659,2	1884	Мі3	1318,5	942
Фа1	349,2	3556	Фа2	698,4	1778	Фа3	1396,9	889
Фа1#	369,9	3356	Фа2#	739,99	1678	Фа3#	1479,9	839
Соль1	391,9	3168	Соль2	783,99	1584	Соль3	1567,9	792
Соль1	415,3	2990	Соль2	830,6	1495	Соль3	1661,2	747
Ля1	440	2822	Ля2	880	1411	Ля3	1760	705
Ля1#	466,2	2664	Ля2#	932,3	1332	Ля3#	1864,6	666
Сі1	493,9	2514	Сі2	987,8	1257	Сі3	1975,5	628

Генерація імпульсних сигналів, відповідних певним нотам, може грунтуватися на двох способах. При першому способі реалізації імпульсної послідовності використовується центральний процесорний пристрій МК. При другому способі для створення меандра заданої частоти прохо-

дження імпульсів, необхідного для формування звуку, використовують вбудований у МК таймер. Другий спосіб переважніше.

Для формування звукових сигналів із заданими частотами можна використовувати шістнадцятирозрядний таймер лічильник (ТС) №1. Для генерації звуку зручно використовувати режим СТС (скидання за збігом). Слід мати на увазі, що в МК є спеціальний вихід ОС1А (ОС1В), який у момент переривання таймера за збігом перемикає сигнал на виході з логічного 0 у логічну 1 і навпаки. Щоб одержати необхідний звук досить буде в регістр збігу ТС №1 записати число, відповідне частоті імпульсного сигналу для відтворної ноти. Значення тих чисел, які необхідно записати в регістр збігу, для відтворення певної ноти, наведені у табл. 5.1. Аналогічним чином, як створювалися «чисті тони», відповідні окремим нотам, можна за допомогою МК відтворювати одноголосні мелодії (складати з нот «музичні фрази» – послідовності нот). Слід враховувати, що в мелодії кожна нота має свій тон (частоту) і тривалість звучання. З цієї причини, щоб МК міг працювати з мелодією її треба представити в такій формі, щоб її зручно було зберігати в пам'яті і швидко витягувати у разі потреби. Іншими словами, мелодію треба «закодувати». Оскільки будь-яка мелодія складається з нот, то перше, що треба зробити, це закодувати тон (частоту) ноти. Зручно при цьому поступити таким чином: всі ноти пронумерувати по порядку, починаючи з найнижчого тону.

Оскільки в музичному ряду однієї октави 12 нот (сім основних – «білих» і п'ять додаткових – «чорних»), то зручно коди нотам присвоїти у порядку зростання частоти. Тоді ноті «до» першої октави відповідатиме код 1, «ре2 – код 3 і т. д. Код ноти «до» другої октави –13. І так далі, відповідно таблиці 5.2.

Кожна нота має певну тривалість звучання. При цьому тривалість цілої ноти є величина відносна, оскільки вона залежить від поточного темпу твору. А оскільки при завданні тривалості звучання цілої, автоматично задається тривалість звучання решти типів нот, то, зручно, щоб зберегти в мелодії співвідношення між тривалістю, тривалість виражати долями від цілої. В цьому випадку як код для тривалості можна використовувати 7 чисел (таблиця 5.3). Окрім нот, будь-яка мелодія містить музичні паузи – проміжки часу, коли не один звук не звучить. Зручно, в цілях збереження темпу, тривалість музичних пауз приймати такою ж, як і тривалість нот. Зручно представити паузу, як одну з нот. Ноті «пауза», або ноті без звуку зручно привласнити нульовий код.

Таблиця 5.2

Коди нот у порядку зростання частоти

Код ноти	Нота	Код ноти	Нота	Код ноти	Нота
1	До1	13	До2	25	До3
2	До1#	14	До2#	26	До3#
3	Ре1	15	Ре2	27	Ре3
4	Ре1#	16	Ре2#	28	Ре3#
5	Мі1	17	Мі2	29	Мі3
6	Фа1	18	Фа2	30	Фа3
7	Фа1#	19	Фа2#	31	Фа3#
8	Соль1	20	Соль2	31	Соль3
9	Соль1 #	21	Соль2#	33	Соль3#
10	Ля1	22	Ля2	34	Ля3
11	Ля1#	23	Ля2#	35	Ля3#
12	Сі1	24	Сі2	36	Сі3

Таблиця 5.3.

Використання 7 чисел в якості коду для тривалості нот

Код	Тривалість	Число
0	1 (ціла)	64
1	1/2 (половинна)	128
2	1/4(чверть)	256
3	1/8 (восьма)	512
4	1/16 (шістнадцята)	1024
5	1/32 (тридцять друга)	2048
6	1/64 (шістдесят четверта)	4096

Щоб кожну ноту, з своїми тоном і тривалістю звучання, зручно було «розміщувати» в пам'яті МК доцільно кожну ноту представляти у вигляді коду, розмірністю в один байт. Для 32 значень кодів різних нот буде потрібно 5 біт, а для кодування тривалості – 3 розряди. Домовимося, що три старші розряди в байті ми використовуватимемо для кодування три-

валості ноти, а 5 бітів, що залишилися, – для кодування її тону. Наприклад, код ноти «ля» першої октави тривалістю j у двійковому вигляді матиме вигляд, показаний у табл. 5.4. Запрограмовану (закодовану) одноголосну мелодію представлятиме ланцюжок таких кодів.

Таблиця 5.4

Приклад коду ноти «ля» першої октави тривалістю j у двійковому вигляді

Код тривалості			Код тону ноти					Код ноти
0	1	0	0	1	0	1	0	4A

Таким чином, якщо є нотний стан, то, привласнивши кожній ноті і кожній музичній паузі свій код, ми одержимо набір (файл) з чисел, що представляють для МК закодовану одноголосну мелодію. Після запису програми в МК він формуватиме музичні мелодії на підключеному до відповідного порту п'єзовипромінювачі, або в динаміку, з'єднаному за допомогою підсилювача. Аналогічним чином, якщо в МК є достатнє число таймерів, можна відтворити багатоголосу музичну мелодію з числом голосів, визначуваним кількістю використовуваних таймерів.

5.3.3. Формування звуків з використанням синтезаторів мови

Для синтезу мови використовують два способи: синтез за правилами і синтез за зразками (компілятивний синтез). При синтезі за правилами мовний сигнал утворюється тільки за правилами, що зберігаються в пам'яті пристрою, без звернення до якого-небудь виду мови, вимовної людиною. При компілятивному синтезі є «словник» мовних одиниць, спеціально підготовлених з реальних елементів мовного матеріалу, наговореного певним диктором. Цими елементами можуть бути цілі фрази, слова, склади, фонеми – мінімальні мовні одиниці, яким при написанні відповідає одна і та ж буква або символ. Найбільш перспективний напрям – синтез за правилами – дозволяє перетворити довільний орфографічний текст у відповідний мовний сигнал. При цьому пристрій звуковідтворення працює за схемою: текст – фонема – мова. Проте при практично необмеженому словнику таке перетворення в даний час не забез-

печує ще досить високої якості синтезованої мови. Тому найбільш поширеним є компілятивний синтез, де при обмеженому словнику (із-за великого об'єму пам'яті) досягається відносно хороша якість мови. Найпростіша система синтезу мови виходить при компіляції слів. Основу компілятивної системи складає запам'ятовуючий пристрій мовних одиниць. Об'єм даних, необхідних для представлення мовного сигналу на інтервалі в 1 секунду при використанні, наприклад диференціальної імпульсно-кодової модуляції (ДІКМ), складає 20 кбіт. А при адаптивній ДІКМ – 5 кбіт. Істотне зниження об'єму пристрою, що запам'ятовує, для словника мовних одиниць може бути одержано, якщо при компілятивному синтезі за основу замість слів приймаються фонеми або склади. Схема пристрою мовного виведення інформації має вигляд, показаний на рис. 5.14.

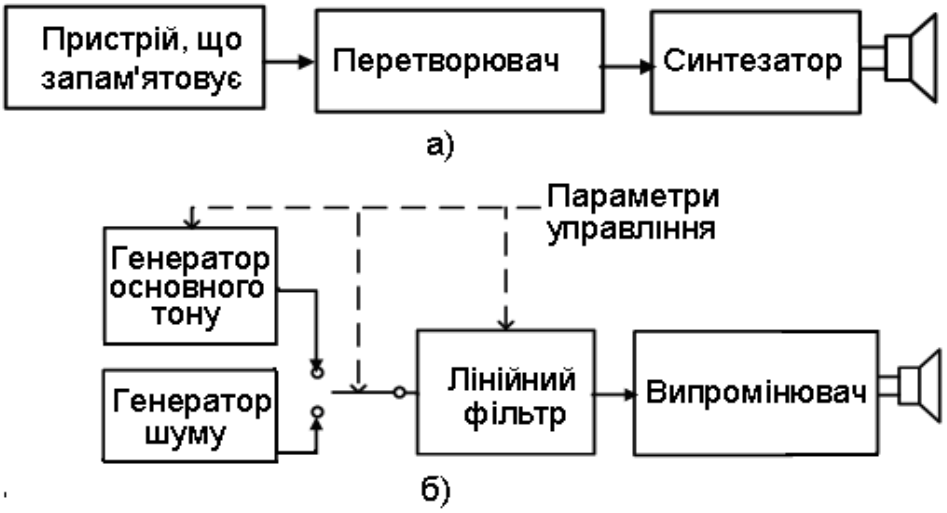


Рис. 5.14. **Структурна схема пристрою виведення мовного сигналу (а) і схема синтезатора мови (б)**

Перетворювач виконує декілька функцій. Якщо в пристрої, що запам'ятовує, міститься інформація у вигляді тексту повідомлення, то перетворювач перекодує текст у фонетичні (відповідні звукам мови) символи, оскільки фонеми найчастіше вимовляються не так як пишуться, а деякі не вимовляються взагалі. Крім того, перетворювач «розставляє» паузи між словами і формує фазові наголоси. Результат перетворення у вигляді послідовності управлінських параметрів поступає в синтезатор, який імітує мовоутворення. У синтезаторі є блоки, які «прагнуть» сформувати миттєвий спектр амплітуд мовного сигналу (рис. 5.14 б). Генератор основного тону імітує роботу голосових зв'язок мовного апарату лю-

дини. Частота основного тону для чоловічого голосу лежить у межах 80 – 90 Гц, для жіночого – у межах 160 – 320 Гц. Генератор шуму з рівномірним спектром використовується для формування таких глухих звуків, як «с», «ш» і ін. При цьому використовується формантне представлення мовного сигналу. Для якісного синтезу звукового сигналу задають параметри декількох формант (резонансних областей у спектрі мови) основного тону. При цьому в процесі синтезу мови частоти і смуги формант безперервно міняються. Міняються також при створенні мови частоти основного тону. Лінійний фільтр під дією параметрів управління переміщає максимуми енергії звукового сигналу, формує положення спектральних компонентів (частоти і амплітуди лінійчатого спектру). Управління блоками синтезатора здійснюється на основі закладених у них характеристик фонем, правил синтезу і т. д.

Контрольні запитання

1. Стисло охарактеризуйте програмне середовище AVR Studio.
2. Поясніть призначення і особливості програмних наладчиків.
3. Що таке апаратні наладчики.
4. Поясніть для чого призначені програми-імітатори електронних пристроїв?
5. Наведіть приклади типових апаратних рішень і наладки програм.
6. Що означає поняття: усунення брязкоту контактів.
7. Поясніть як виконується організація взаємодії мікроконтролера і клавіатури.
8. Поясніть схему пристрою виведення мовного сигналу.
9. Поясніть введення інформації в мікроконтролер із клавіатури.
10. Організація взаємодії мікроконтролера і клавіатури.
11. Назвіть і поясніть основні способи відображення інформації за допомогою напівпровідникових індикаторів.
12. Типові апаратні й програмні рішення для відображення інформації невеликої інформаційної ємності багаторозрядними напівпровідниковими семисегментними індикаторами.
13. Структурна схема керування чотирирозрядним напівпровідниковим семисегментним індикатором.
14. Наведіть і поясніть структурну схему синтезатора мови.

Рекомендована література

1. Аппаратно-программное обеспечение полиграфического оборудования: Межведомственный сб. науч. тр. / Под ред. А.С. Сидоров. – М. : МГУП, 2001. – 178 с.

2. Баранов В. Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Изд. дом «Додэка-XXI», 2004. – 288 с.

3. Волощак І. А. Автоматизований електропривід поліграфічних машин: Підручник / Волощак І. А., Стрепко І. Т. – Львів : Фенікс, 1998. – 239 с.

4. Голубцов М. С. Микроконтроллеры AVR: от простого к сложному. – М.: СОЛОН-Пресс, 2003. – 288 с.

5. Гребнев В. В. Микроконтроллеры семейства AVR фирмы Amtel. М.: ИП РадиоСофт, 2002. – 176 с.

6. Евстифеев А. В. Микроконтроллеры AVR семейства Classic фирмы Amtel – М.: Изд. дом «Додэка-XXI», 2006. – 288 с.

7. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы «ATMEL». – М.: Изд. дом «Додэка-XXI», 2004. – 560 с.

8. Зубков С. В. Assembler для DOS, Windows и UNIX. – 2-е изд., испр. и доп. – М.: ДМК, 2000. – 608 с.

9. Кёниг А. М. Полное руководство по PIC-микроконтроллерам / Пер. с нем. – К.: МК-Пресс, 2007. – 256 с.

10. Киппхан Гельмут Энциклопедия по печатным средствам информации. Технологии и способы производства. – Изд.: Springer Verlag, Heidelberg. Русский перевод. М: Московский Государственный Университет Печати, 2003. – 1280 с.

11. Комп"ютерні технології друкарства: Зб. наук. праць / За ред. С. М. Гунько, О. П. Стецьків, І. Т. Стрепко та ін.- Львів, 1998. – 248 с.

12. Корнеев В. В. Современные микропроцессоры. – 2-е изд. – М.: Нолидж, 2000. – 227 с.

13. Костров Б. В. Микропроцессорные системы и микроконтроллеры. / Б. В. Костров, В. Н. Ручкин– М.: "ТсхБук", 2007. – 320 с.

14. Кузьминов А. Ю. Интерфейс RS232. Связь между компьютером и микроконтроллером. – М.: Радио и связь, 2004. – 168 с.

15. Лебедев М. Б. CodeVisionAVR; пособие для начинающих. – М.: Додэка – XXI, 2008. – 592 с.

16. Микроконтроллеры AVR в радиолюбительской практике. – СПб.:

Наука и Техника, 2007. – 352 с. – (Серия «Радиолюбитель»).

17. Мортон Дж. Микроконтроллеры AVR. Вводный курс. / Пер. с англ. – М.: Издательский дом "Додэка-XXI", 2006. – 272 с.

18. Партала О. Н. Цифровые КМОП микросхемы. Справочник. – СПб: Н и Т, 2001. – 400 с.

19. Петров И. В. Программируемые контроллеры. Стандартные языки и приемы прикладного проектирования / Под ред. В. П. Дьяконова. – М.: СОЛОН-Пресс, 2004. – 256 с.

20. Предко М. Справочник по PIC-микроконтроллерам / Пер. с англ. – М.: ДМК Пресс, 2002. – 512 с.

21. Трамперт В. AVR-RISC микроконтроллеры / Пер. с нем. – К.: МК-Пресс, 2006. – 464 с.

22. Трамперт В. Измерение, управление и регулирование с помощью AVR-микроконтроллеров / Пер. с нем. – К.: «МК-Пресс», 2006. – 208 с.

23. Туманов М. П. Технические средства автоматизации и управления: цифровые средства обработки информации и программное обеспечение: Учебное пособие. – М.: МГИЭМ, 2005. – 71 с.

24. Фрунзе А. В. Микроконтроллеры? Это же просто! Т. 1, т. 2. – М.: ООО "ИД СКИМЕН", 2002. – 336 с.

25. Чехман Я. І. Друкарське устаткування: Підручник / Я. І. Чехман, В. Т. Сенкусь, В. П. Дідич та ін. – Львів: УАД, 2005. – 468 с.

26. Шагурин И. И. Современные микроконтроллеры и микропроцессоры Motorola. – Изд.: Горячая Линия-Телеком. – 2004. – 952 с.

27. Шпак Ю. А. Программирование на языке С для AVR и PIC микроконтроллеров – К.: МК-Пресс, 2006. – 400 с.

28. Ярема С. М. Видавничі поліграфічні технології та обладнання: Загальний курс. Навч. посібн. для студ. вищ. навч. закл., які навчаються за спец. "Видавнича справа та редагування" і "Технологія розроблення, виготовлення та оформлення пакувань" / Відкритий Міжнародний ун-т розвитку людини "Україна". – К.: Університет "Україна", 2003. – 320с.

29. Яценков В. С. Микроконтроллеры Microchip®. Практическое руководство. – 2-е изд. испр. и доп. – М.: Горячая линия-Телеком, 2005. – 280 с.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Гоков Олександр Михайлович
Жидко Євген Анатолійович

КОМП'ЮТЕРИЗОВАНІ СИСТЕМИ ПОЛІГРАФІЧНОГО ОБЛАДНАННЯ

Навчальний посібник

Відповідальний за випуск **Лапта С. І.**
Відповідальний редактор **Сєдова Л. М.**

Редактор **Хижняк Т. М.**
Коректор **Мартовицька-Максимова В. А.**

План 2009 р. Поз. №72-П.
Підп. до друку 5.10.2009 Формат 60 x 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 15,25. Обл.-вид. арк. 19,06. Тираж 400 прим. Зам. № 736
Видавець і виготівник — видавництво ХНЕУ, 61001, м. Харків, пр. Леніна, 9а
Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи
Дк №481 від 13.06.2001 р.

Гоков О. М.
Жидко Є. А.

КОМП'ЮТЕРИЗОВАНІ СИСТЕМИ ПОЛІГРАФІЧНОГО ОБЛАДНАННЯ

Навчальний посібник

Викладено питання, пов'язані з технічним і програмним забезпеченням сучасних комп'ютеризованих систем поліфафічного обладнання. Розглянуто загальні принципи побудови і функціонування комп'ютеризованих систем та вузлів обладнання, обміну інформацією в сучасних системах управління поліграфічним обладнанням і роботи основних функціональних вузлів мікропроцесорів; архітектуру, типові функції та особливості виконання контролерів AVR, алгоритми роботи його типових вузлів і програмне забезпечення; принципи алгоритмічного та програмного забезпечення мехатронних систем; основи написання і виконання програм.

ВИДАВНИЦТВО ХНЕУ
ХАРКІВ 2009