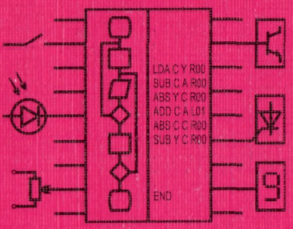


І.Т. Стрепко
О.В. Тимченко
Б.В. Дурняк

ПРОЕКТУВАННЯ СИСТЕМ КЕРУВАННЯ НА ОДНОКРИСТАЛЬНИХ МІКРО-ЕОМ

Київ - 1998

І.Т.Стрепко, О.В.Тимченко, Б.В.Дурняк

ПРОЕКТУВАННЯ СИСТЕМ КЕРУВАННЯ НА ОДНОКРИСТАЛЬНИХ МІКРО-ЕОМ

Київ
Видавництво "Фенікс"
1998

ББК 32.973
С84
УДК 681.3+65-52

Рецензенти: доктор техн. наук *Погрібний В.О.*
доктор фіз.-мат. наук *Яворський І.М.*

Зав. редакцією *Іваник М.Д.*

Стрепко І.Т., Тимченко О.В., Дурняк Б.В.
С84 Проектування систем керування на однокристальних мікро-ЕОМ. - К.: Фенікс, 1998. - 286с.

Розглядається застосування однокристальних мікро-ЕОМ в цифрових системах автоматичного керування, вибір засобів апаратної підтримки функціонування мікро-ЕОМ та розробка програмного забезпечення. Подані багаточисельні приклади і задачі обробки сиґналів, що реалізують типові процедури керування технологічними об'єктами. Висвітлені питання проектування цифрових систем на всіх етапах розробки прикладного програмного забезпечення.

Для наукових працівників, інженерів-проектувальників засобів автоматизації, а також студентів відповідних спеціальностей.

Іл. 91. Табл. 46. Бібліогр.: -40 назв.

ISBN 5-87332-072-1 ББК 32.973

ЗМІСТ

СПИСОК СКОРОЧЕНЬ

А	–	адаптер
АЛП	–	арифметико-логічний пристрій
АЦП	–	аналого-цифровий перетворювач
АЧХ	–	амплітудно-частотна характеристика
ВІС	–	велика інтеґральна схема
ВМ	–	виконавчий механізм
ВСЕ	–	внутрішньосхемний емулятор
Д	–	давач
ЕОМ	–	електронна обчислювальна машина
ЗПД	–	зовнішня пам'ять даних
ЗПП	–	зовнішня пам'ять програм
ІС	–	інтеґральна схема
ІХ	–	імпульсна характеристика
КС	–	керуюче слово
МК	–	мікроконтролер
МП	–	мікропроцесор
ОЗП	–	оперативний запам'ятовуючий пристрій
ОК	–	об'єкт керування
ОМЕОМ	–	однокристальна мікро-ЕОМ
ОС	–	операційна система
ПЗ	–	програмне забезпечення
ПЗП	–	постійний запам'ятовуючий пристрій
ППП	–	блок послідовного інтерфейса і переривань
ПЛМ	–	програмована логічна матриця
ПП	–	пам'ять програм
ППЗП	–	перепрограмовуваний запам'ятовуючий пристрій
Р	–	цифровий резонатор
РЗП	–	реґістр загального призначення
РПД	–	резидентна пам'ять даних
РПЗП	–	репрограмований ПЗП
РПП	–	резидентна пам'ять програм
САК	–	система автоматичного керування
САПР	–	системи автоматизованого проектування
СВЗ	–	схема вибірки-зберігання
СК	–	сиґнали керування
СС	–	сиґнали стану
Т	–	таймер

ТП	–	технологічний процес
УАПП	–	універсальний асинхронний прийомо-передавач
УФТ	–	ультрафіолетовий
ФВЧ	–	фільтр верхніх частот
ФНЧ	–	фільтр нижніх частот
ФР	–	фільтр режекторний
ФС	–	фільтр смуговий
ФСґ	–	формувач сиґналів
ФЧХ	–	фазо-частотна характеристика
ЦАП	–	цифроаналоговий перетворювач
ЦОС	–	цифрова обробка сиґналів
ЦР	–	цифровий реґулятор
ЦСАК	–	цифрова система автоматичного керування
ЦФ	–	цифровий фільтр
ШПФ	–	швидке перетворення Фур'є

ОСНОВНІ УМОВНІ ПОЗНАЧЕННЯ

$\leftarrow$	–	оператор присвоювання (заміщення)
$\leftrightarrow$	–	оператор взаємного обміну
$\wedge$, $\vee$, $\oplus$	–	оператори логічних операцій: І (кон'юкція), АБО (диз'юнкція), виключне АБО
@	–	префікс непрямої адресації
#	–	префікс безпосереднього операнда
inc(·)	–	інкремент
MPL	–	перемножувач
SM	–	суматор
z^{-1}	–	елемент затримки

ПЕРЕДМОВА

Важливою складовою частиною науково-технічного прогресу є комплексна автоматизація технологічних процесів (ТП) на основі мікропроцесорних систем керування. Використання мікропроцесорів (МП) і мікро-ЕОМ в цифрових системах автоматичного керування (ЦСАК) дозволяє більш ефективно, порівняно з традиційними засобами автоматики, вирішувати комплексні завдання автоматизації в різних галузях народного господарства.

Більше того, сучасний рівень мікропроцесорних засобів, який об'єднує в собі жорстко запрограмовані функціональні елементи та змінні перепрограмовувані модулі, значно розширює функціональні можливості цифрових регуляторів (ЦР), побудованих на великих інтегральних мікросхемах (ВІС). Розвиток мікропроцесорних систем керування ТП характеризується підвищеним ступенем інтеграції елементної бази і ускладненням алгоритмів функціонування.

Окремий клас ВІС складають однокристальні мікро-ЕОМ (ОМЕОМ), тобто прилади, які конструктивно виконані у формі однієї мікросхеми і включають в себе всі складові пристрої, необхідні для реалізації ЦСАК мінімальної конфігурації: арифметичний пристрій, оперативну та постійну пам'ять, паралельні та послідовні порти вводу/виводу інформації, таймери, лічильники і т.п. Апаратна підтримка функціонування ОМЕОМ надзвичайно проста і зводиться, як правило, до буферизації портів вводу/виводу, узгодження рівнів сигналів та навантажувальної здатності. Завдяки цьому неперервно зростає промисловий випуск ОМЕОМ, що в наш час складає понад третину від усіх мікропроцесорних засобів, одночасно при цьому розширюється область їх використання. В той же час програмне забезпечення ОМЕОМ є достатньо складним. Тому 90%, а часом і більше вартості ЦСАК на ОМЕОМ становить вартість програмного забезпечення.

В літературі ОМЕОМ часто називають мікроконтролерами (МК). Це обґрунтовують тим, що такі ВІС мають незначну ємність пам'яті, фізичну і логічну розділеність пам'яті програм та пам'яті даних, спрощену і орієнтовану на задачі керування систему команд, примітивні методи адресації команд і даних. Специфічна організація вводу/виводу інформації визначає область їх застосування як спеціалізованих обчислювачів, які включені в контур керування об'єктом або процесом. Структурна організація,

набір команд і апаратно-програмні засоби вводу/виводу інформації цих мікросхем найбільш пристосовані для керування і реґулювання в пристроях і системах автоматики, а не для обробки даних.

Однак ці твердження відображають лише технічний рівень ОМЕОМ на даний момент. Розвиток технології виготовлення ВІС значно підносить ступінь їх інтеґрації, що дозволить у найближчий час використовувати ОМЕОМ і при обробці даних.

Назва МК використовується нами лише для позначення сімейства ОМЕОМ, що об'єднані низкою загальних ознак, наприклад, розрядністю, системою команд, набором функціональних блоків тощо.

Найбільший ефект від впровадження ОМЕОМ досягається в локальних або розподілених ЦСАК.

Високі техніко-економічні показники ЦСАК на базі ОМЕОМ (низька вартість, мала енерґоємність, компактність, висока надійність, точність та швидкодія, можливість реалізації складних алгоритмів керування, які можна легко міняти програмним шляхом) сприяють їх інтенсивному впровадженню не лише у виробничу сферу, але і в побут.

В даній книзі викладені питання проектування ЦСАК на основі ОМЕОМ. За базові вибрано ВІС КР1816ВЕ51 (МК51) та аналоговий МК КМ1813ВЕ1А (ВЕ1). Подано опис структурної організації і системи команд МК51 і ВЕ1; розглянуті численні приклади і задачі обробки сиґналів (починаючи від простих команд передачі даних до складних програм, що реалізують типові процедури керування поліграфічними об'єктами), програмної та апаратної реалізації пристроїв ЦСАК; узагальнено досвід проектування цифрових систем на всіх етапах розробки прикладного програмного забезпечення.

Монографія є колективною працею авторів, в якій використано їх теоретичні розробки та досвід практичної реалізації систем керування та обробки сигналів на однокристальних мікроЕОМ.

Автори висловлюють щиру подяку рецензентам книги доктору технічних наук Погрібному В.О. і доктору фізико-математичних наук Яворському І.М., а також редактору Старовойту М.В. за корисні поради та зауваження, які сприяли поліпшенню цього видання.

Всі зауваження і пропозиції будуть прийняті з вдячністю.

Автори

Розділ 1. СИСТЕМИ КЕРУВАННЯ НА ОДНОКРИСТАЛЬНИХ мікро-ЕОМ

1.1. Переваги і недоліки цифрових систем

Зростаючий інтерес до цифрових систем автоматичного керування обумовлений рядом важливих переваг такого регулювання в порівнянні з аналоговою і дискретно-аналоговою реалізаціями. Ці переваги можна поділити на три групи: принципові, реалізаційні і техніко-експлуатаційні.

До принципових переваг належать:

- багатофункціональність;
- мультиплексна обробка інформації;
- реалізація будь-яких перетворень;
- відсутність принципових обмежень на складність систем.

Реалізаційні переваги складають:

- висока стабільність характеристик;
- відтворюваність характеристик (відсутність індивідуальних ознак систем);
- широкі можливості для адаптації (переналагодження і керування характеристиками систем) у процесі роботи;
- висока точність операндів;
- реалізація за допомогою ВІС, відсутність реактивних елементів;
- повністю цифрова реалізація систем.

Техніко-експлуатаційними перевагами є:

- висока надійність;
- малі габарити і маса;
- мале енергоспоживання;
- широкі можливості уніфікації;
- можливість діагностики;
- низький електромагнітний вплив і зручний захист від нього.

Одночасно цифрові системи мають ряд недоліків:

- сигнали керування є відносно низькочастотними;
- шуми округлення, нелінійні ефекти переповнення і граничні цикли;
- залежність швидкості обробки інформації від необхідної точності (розрядності);
- чутливість до радіоактивного випромінювання, особливо на мікросхемах, виконаних за n-МОН технологією.

Можливість реалізації алгоритму керування на ОМЕОМ дозволяє один і той же об'єкт використовувати різним чином. В

цьому полягає багатофункціональність цифрових систем.

У багатьох випадках практичного застосування частота дискретизації сиґналу виявляється багато раз меншою за тактову частоту ОМЕОМ. Це дозволяє в проміжках між сусідніми тактами керування системи обробляти інші сиґнали (наприклад, інших каналів), забезпечуючи таким чином мультиплексування або виконання інших видів обробки (багатофункціональність).

ЦСАК практично не обмежені у видах реалізовуваних функцій, що властиве аналоговій техніці, і забезпечують реалізацію будь-яких формально описаних перетворень сиґналів. Існує принципова можливість забезпечення необхідної точності при алгоритмах будь-якої складності.

Решта переваг цифрових систем коментарів не потребують. До найбільш суттєвих недоліків цифрових систем належать обмеження на частотний діапазон сиґналів, а також шуми округлення і ефекти переповнення. Уникнути цих вад вдається лише шляхом помітного ускладнення і збільшення обсягу обладнання.

Системи, в яких необхідно проводити фільтрацію, корекцію, стискання інформації, спектральний або кореляційний аналіз, нелінійні перетворення, відносяться до складних систем. У них для обчислення вихідного відліку доводиться виконувати значне число (порядку десятків і сотень) операцій множення і додавання, а часом і таку достатньо складну для реалізації функцію, як операція ділення. Цифрові системи, як правило, включають всі ці види обробки.

Цифрові системи автоматичного керування є різновидністю систем цифрової обробки сиґналів (ЦОС). Практично всі види ЦОС можуть бути представлені композицією рекурсивної і нерекурсивної фільтрації (включаючи коваріацію), спектральних, нелінійних і логічних перетворень. В свою чергу вказані види обробки зв'язані з реалізацією операторів:

а) згортки:

$$y_k = \sum_{m=-(M-1)/2}^{(M-1)/2} x_{k-m} \cdot h_m,$$

де $\{h_m\}$, $m \in [0, M-1]$ - імпульсна характеристика фільтра;

б) рекурсії:

$$h_k = \sum_{m=0}^{M} b_m \cdot x_{k-m} - \sum_{m=1}^{N} a_m \cdot y_{k-m}; k \geq 0$$

в) дискретного перетворення Фур'є (ДПФ):

$$X(k) = \sum_{n=0}^{N-1} x_n e^{-j[2\pi/N]nk}, k \in \{0, N-1\};$$

г) нелінійного (наприклад, поліноміального) перетворення:

$$y_k = \sum_{m=0}^{N} A_m \cdot x_k^m;$$

ґ) логічного перетворення:

$$y_k = F(x_k,\ldots,x_{k-l}),$$

де F - оператор логічного перетворення.

Відзначимо, що в системах автоматичного керування найчастіше використовують операції а), б) і ґ), хоча, як правило, використання всіх інших операторів забезпечує необхідну зручність і інформативність систем керування.

Основу практично всіх алгоритмів цифрового керування складають оператори. В свою чергу, обчислення багатьох з них пов'язане з операціями виду:

$$y = A \cdot x + B,$$

де A,B - деякі проміжні змінні, x - вхідна величина.

Виконання цієї операції на основі ЕОМ з традиційною фон-Нейманівською архітектурою, для якої характерне сумісне зберігання даних, програм і констант, передбачає виконання наступних етапів:

- вибірка операндів;
- вибірка команди множення;
- отримання добутку;
- аналіз результату;
- запис старших і молодших розрядів результату;
- вибір команди додавання;
- отримання суми;
- аналіз результату;
- запис результату;
- перевірка переривання;
- перехід до наступної команди.

Виконання великої кількості таких операцій (наприклад, у відповідності до порядку системи) навіть для порівняно простих алгоритмів керування звужує діапазон частот оброблюваних сигналів до сотень герц.

Із цих же причин малоефективне і використання

універсальних МП.

Тому для подолання вказаних обмежень в цифрових системах використовують:

1. спеціалізовані ВІС, що реалізують типові операції (фільтрація, згортка) або функціонально закінчені вузли;

2. спеціалізовані набори інтеґральних схем для цифрових систем керування;

3. цифрові процесори обробки сиґналів спеціалізованих (як правило, однокристальних) мікро-ЕОМ, орієнтованих на клас ЦСАК.

Останні реалізують не конкретну задачу, а призначені для певного класу задач керування, в основі яких лежать вказані співвідношення. Ця орієнтація пов'язана зі зміною архітектури і схемними модифікаціями, що дозволяє при однакових з універсальними мікро-ЕОМ технології, споживаній потужності і вартості забезпечити підвищення продуктивності на 1-2 порядки.

В цих процесорах використовується гарвардська або модифікована гарвардська архітектура, при якій дані і команди розміщуються в окремих блоках пам'яті. Крім того, широко використовуються такі особливості:

1. безпосереднє використання комірок ОЗП як реґістрів арифметичного блоку; включення вибору команд зчитування і запису операндів в цикл команд;

2. багатошинна архітектура;

3. використання чотирireґістрового АЛП, використання двох акумуляторів або проміжних реґістрів;

4. суміщення в одному циклі команди операцій додавання і множення;

5. короткий командний цикл, який досягається за рахунок використання спеціальних структур основних функціональних вузлів;

6. велика розрядність АЛП для проміжних обчислень;

7. включення вузлів, які забезпечують ввід і вивід аналогових сиґналів.

Перевагами цифрових в порівнянні з аналоговими САК є:

– компактність виробу – ЦСАК реалізується одним конструктивним елементом;

– строга відповідність отриманих функцій і характеристик розрахунковим параметрам; відсутність необхідності вибору, перевірки та узгодження компонентів;

– стабільність характеристик, яка визначається стабільністю

кварцового резонатора;

– незалежність форми характеристик від кліматичних умов, часу, взаємного впливу елементів;

– відсутність обмежень на складність обробки, пов'язаних із проблемою стійкості і розкидом значень дискретних елементів;

– розширення можливостей реалізації різноманітних видів ЦСАК – логічних, нелінійних, адаптивних і т.д.;

– простота обробки аналогових низькочастотних сигналів;

– спрощення процесу проектування і модифікації систем.

1.2. Основні етапи проектування систем автоматичного керування на ОМЕОМ

Ці питання є загальними при розробці методики проектування систем на будь-яких ОМЕОМ та МП. Їх суть наступна:

1. В основі проектування лежить побудова структури математичної моделі процесу обробки сигналу в САК, яка являє собою композицію лінійних, нелінійних та логічних операторів. Така модель повинна бути наслідком вимог по відношенню «вхід-вихід» САК і раціонально (оптимально) узгоджувати можливості апаратури з алгоритмами обчислень.

2. Після побудови структури математичної моделі необхідно виконати розрахунок числових значень її параметрів, наприклад, квантованих внаслідок квантування з обмеженою розрядністю коефіцієнтів передаточних функцій, коефіцієнтів операторів, нелінійних перетворень і т.п. Крім того, необхідно оцінити похибки перетворень і допустиму точність обчислень.

Перераховані задачі складають зміст основних розділів теорії САК. Їх вирішення в рамках конкретної задачі практично нездійсниме без використання САПР. При розрахунку параметрів моделі, як і при побудові її структури, найважливішою є задача узгодження алгоритмічних рішень і апаратних засобів.

Адаптація алгоритмів до можливостей конкретних мікро-ЕОМ суттєво пов'язана з вирішенням нетривіальних задач оптимізації. Відмітимо, що оптимізація алгоритмів з урахуванням особливостей мікро-ЕОМ дозволяє в декілька разів, а часом і на порядок зменшити необхідні обчислювальні ресурси.

При використанні ОМЕОМ та МП алгоритм ЦСАК реалізується у вигляді програми, наладнання якої вимагає відповідних апаратних та програмних засобів.

3. У зв'язку з потребою використання САПР необхідно розв'язувати такі основні задачі:

а) імітаційне моделювання проектованої системи на етапі створення структури математичної моделі;

б) розрахунок і оптимізація операторів моделі або їх параметрів і відповідних обчислювальних алгоритмів;

в) програмування і налагоджування асемблерних програм для ОМЕОМ.

Таким чином, найважливішою особливістю технологій проектування ЦСАК є допоміжна роль розробки математичних моделей і їх налагодження. Що ж стосується схемотехнічних вимог, то звичайно використовується типова схема підключення ОМЕОМ і проектувальник практично звільняється від вирішення схемотехнічних проблем і пов'язаних з ними помилок.

4. До особливостей проектування систем на ОМЕОМ необхідно віднести і специфічні вимоги до підготовки проектувальників. На відміну від традиційної, в значній мірі орієнтованої на розвиток інженерних навичок, тут необхідна глибока підготовка з теорії ЦСАК, прикладної математики і обчислювальної та мікропроцесорної техніки.

Кінцевою метою проектування є створення апаратних та програмних засобів функціонування САК, тобто розробка алгоритмів роботи, структурних і принципових схем системи, а також програмного забезпечення на мові асемблер для реалізації заданого закону керування.

Основними критеріями проектування є час, за який реалізуються один цикл керування, використаний об'єм ОЗП та ПЗП, споживана потужність та інші.

САК за заданим алгоритмом формує аналогові та цифрові сигнали керування (СК), які подаються через цифроаналогові перетворювачі (ЦАП) і формувачі сигналів (ФСг) на виконавчі механізми (ВМ) об'єкта керування (ОК) та пристрої індикації, в залежності від сигналів стану (СС), які формуються за допомогою аналого-цифрових перетворювачів (АЦП) і ФСг з сигналів давачів об'єкта керування (рис. 1.1).

Загальний алгоритм роботи САК (рис. 1.2) включає наступні етапи:

1. Ініціалізація САК, яка зводиться до встановлення початкових значень сигналів керування {Y} і {Q}, ініціалізації схем контролю за роботою САК та внутрішніх таймерів.

2. Ввід і обробка аналогових {X} та бінарних {Z} сигналів

давачів, ініціалізація схем контролю за роботою САК та внутрішніх таймерів.

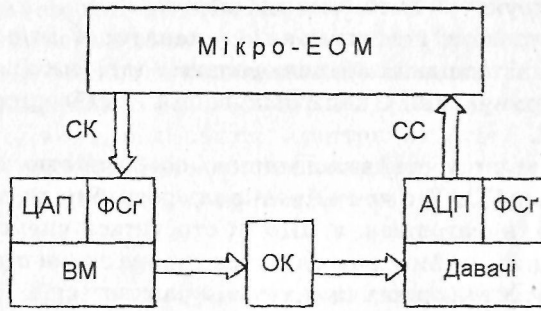


Рис. 1.1. Структура цифрової системи керування на основі мікро-ЕОМ

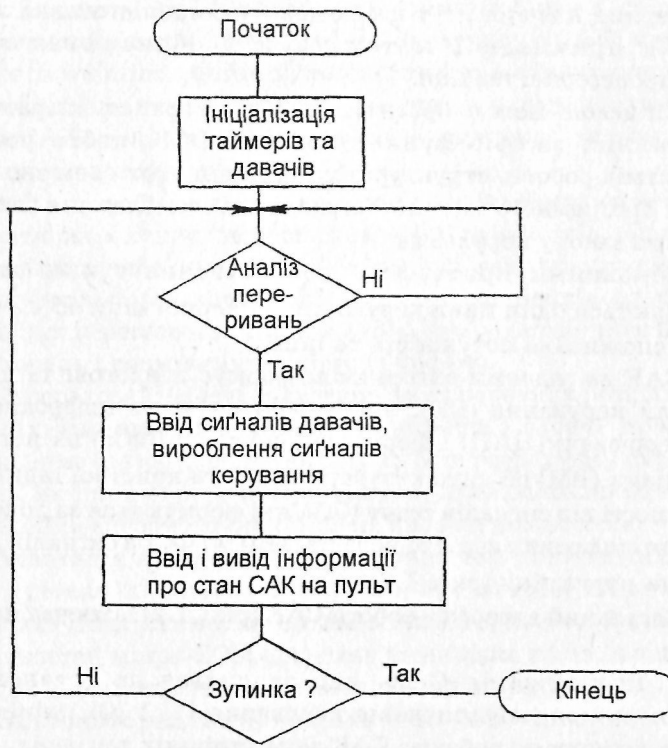


Рис. 1.2. Алгоритм роботи САК

3. Вивід інформації про стан САК на пульт керування і ввід сиґналів від оператора.

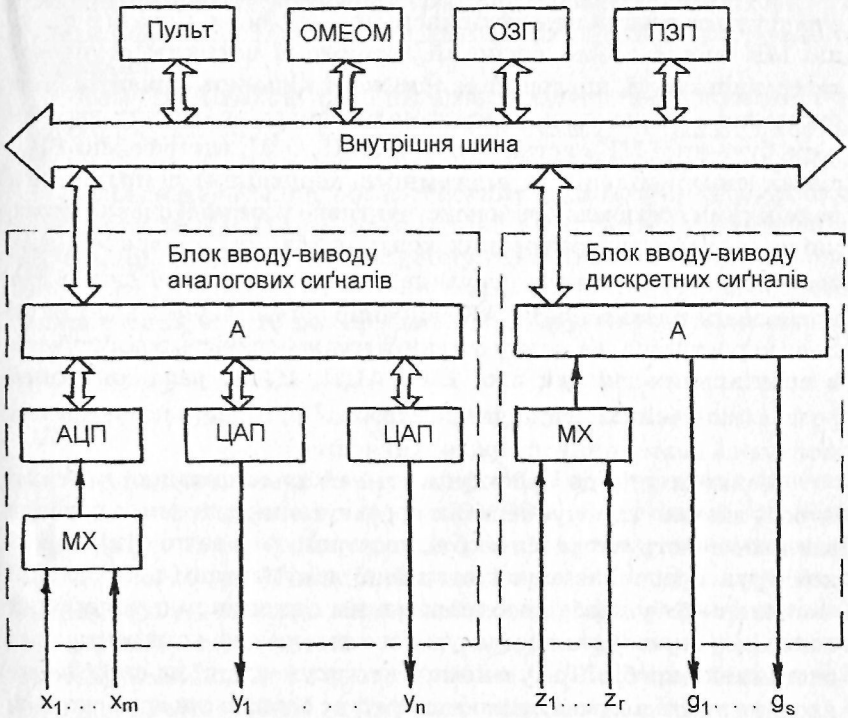


Рис. 1.3. Структурна схема САК

Узагальнена структурна схема САК приведена на рис.1.3 і складається з ОМЕОМ, ОЗП, ПЗП, каналів вводу-виводу аналогових та цифрових сиґналів, пульта оператора. При необхідності в канали вводу-виводу включають адаптери А для запам'ятовування цифрових даних, мультиплексори МХ аналогових та цифрових сиґналів з метою розділення їх в часі для подальшої обробки інформації в ОМЕОМ. В залежності від типу ОМЕОМ і ємності пам'яті програм блоки ОЗП та ПЗП можуть бути відсутні (використовуються внутрішні ОЗП та ПЗП ОМЕОМ).

1.3. Особливості проектування апаратної частини

При розробці ЦСАК об'єктами на основі ОМЕОМ суттєво спрощується етап системотехнічного проектування. Справа в тому, що МК являє собою логічний автомат із високим ступенем детермінованості, що допускає обмежену кількість варіантів його системного включення. Тому типовий склад апаратних засобів ядра будь-якої МК-системи (МК, ППЗП, ОЗП, інтерфейсні ВІС, схеми синхронізації та системного керування) є практично незмінними і оформляються конструктивно у вигляді одноплатних універсальних програмованих контролерів, які призначені для вмонтовування в контур керування технологічним об'єктом. На друкованій платі такої ЦСАК звичайно організовують монтажне поле користувача, на якому останній має можливість розміщувати в спеціальних гніздах свої ВІС, АЦП, ЦАП, реле, оптронні розв'язки та інші специфічні схеми. Часто на таких платах монтують і автономне джерело живлення.

Такий підхід до проектування найбільш складної частини виробу значно скорочує терміни проектування системи, а головне звільняє конструктора від необхідності розробки великої кількості конструкторської та експлуатаційної документації центральної частини виробу. Побудований за таким принципом стандартний контролер переходить з розряду підсистем у розряд комплектуючих системних виробів. Тому в комплект документації на виріб може входити лише документація на апаратуру спряження ядра системи з давачами та виконавчими елементами ОК. Крім того, у більшості випадків для реалізації ядра системи цілком достатньо засобів, які містяться в єдиній мікросхемі ОМЕОМ. Однак спрощення апаратури пов'язане з ускладненням програмного забезпечення. При цьому основним проектним документом на розробку стає лістинг прикладної програми.

Створення мікропроцесорної системи не закінчується розробкою її принципової схеми. Необхідно виконати налагодження її макетного, дослідного та серійного взірців, розробити і налагодити програмне забезпечення. Розширення функціональних можливостей ОМЕОМ приводить до зменшення числа точок схеми, які фізично доступні для безпосередньої дії на них та контролю їх стану. Фізично локалізувати, виділити для перевірки окремий функціональний вузол ядра системи від інших частин схеми стає неможливим. Дії на потрібну точку схеми або функціональний вузол та контроль їх стану набувають

опосередкованого характеру. При неможливості прямої подачі сиґналу на входи функціонального вузла від зовнішнього джерела часто єдиною альтернативою є ґенерація необхідної послідовності іншими функціональними вузлами схеми під дією програмного керування.

Тому важливою особливістю процесу налагодження і перевірки МК-систем є необхідність сумісного налагодження апаратури та програм навіть на перших етапах розробки ЦСАК.

Слід зазначити, що обсяг основних трудозатрат на розробку апаратних засобів МК-систем постійно зменшується у відношенні до сумарних затрат на розробку прикладного програмного забезпечення системи. Практично затрати на розробку програмного забезпечення в 2–10 раз перевищують затрати на придбання та виготовлення апаратури.

1.4. Особливості проектування програмної частини

Особливості використання мікро-ЕОМ в прикладних задачах керування полягає в тому, що ОМЕОМ є частиною системи керування. Її вибір повинен здійснюватися з урахуванням конкретної сфери її використання. Вирішення цієї задачі не є тривіальним і вимагає від проектувальника достатньо високої кваліфікації як з питань теорії цифрових систем керування, системотехніки, так і з програмування. Неабияке значення має і досвід розробки ЦСАК на базі конкретних типів МК.

При проектуванні МК-систем постає задача оптимального (за певними критеріями) розподілу функцій між апаратними засобами та програмним забезпеченням. Її вирішення пов'язане з необхідністю аналізу багатьох факторів, першочерговими з яких, поряд із забезпеченням заданої продуктивності мікро-ЕОМ, є структура системи, характер подання вхідної інформації (аналогова, бінарна, цифрова) і особливості цифрових алгоритмів обробки сиґналів.

Використання спеціалізованих ВІС спрощує розробку та забезпечує високу швидкодію системи в цілому, однак приводить до зростання вартості, номенклатури комплектуючих елементів та споживаної потужності. Збільшення питомої ваги програмного забезпечення дозволяє зменшити число компонентів системи та вартість апаратних засобів, але це веде до зниження швидкодії та збільшення затрат і термінів розробки та налагодження прикладних програм. Часто при цьому дещо зростає і число ВІС

зовнішньої пам'яті МК-системи.

У більшості випадків розробникам САК на базі ОМЕОМ слід віддавати перевагу програмній реалізації алгоритмів систем. При цьому зростає гнучкість таких ЦСАК, тобто можливість модифікації основних елементів алгоритму шляхом перепрограмування. Тому час життя такого виробу значно зростає в порівнянні з МК-системами з апаратною реалізацією алгоритмів обробки інформації, в яких можливість зміни вже існуючої фіксації елементів алгоритму в апаратурі контролера практично відсутня.

На початкових стадіях розробки програмного забезпечення МК-систем часто практикувалась робота «тандемом» професійного програміста та непрограмуючого професіонала, тобто фахівця в конкретній предметній галузі знань. Однак багаторічна практика такої співпраці показала, що час на постановку і формалізацію прикладної задачі та розробку алгоритму її вирішення складає до 70% всього часу, необхідного для одержання закінченого мікроконтролерного виробу. При цьому до 60% помилок прикладних програм ОМЕОМ викликані не помилками в машинних кодах, не логічними помилками в програмі, а помилками формалізації прикладної задачі [22]. Трудозатрати власне на програмування, тобто на одержання машинних кодів, складають біля 10% від загальних трудозатрат розробки програм. Це ще раз підтверджує думку про необхідність освоєння алгоритмічних та програмних методів проектування МК-систем розробниками САК конкретної галузі виробництва.

Крім того, задача розробки програмного забезпечення спрощується в зв'язку з різким здешевленням ВІС пам'яті, що дозволяє в більшості випадків не економити пам'ять при розробці прикладних програм для системних засобів підтримки проектування. Наявність розвинутих систем автоматизованого проектування (САПР) керуючих мікро-ЕОМ дозволяє усунути помилки при розробці та налагодженні прикладних програм ЦСАК технологічним устакуванням.

1.4.1. Основні правила запису програм на мові асемблера

Вихідний текст програм на мові асемблера має певний формат. Кожна команда (і псевдокоманда) являє собою рядок чотириланкової конфігурації:

МІТКА–ОПЕРАЦІЯ–ОПЕРАНД(И)–КОМЕНТАР.

Ланки (поля) можуть відділятися одна від другої довільним числом пробілів.

МІТКА. В полі мітки розміщується символічне ім'я комірки пам'яті, в якій зберігається відмічена команда або операнд. Мітка являє собою буквенно-цифрову комбінацію, що починається з літери. Використовуються лише літери латинського алфавіту. Асемблер МК51 допускає використання в мітках символу підкреслювання (_). Довжина мітки не повинна перевищувати шість символів для МК48 і 31 - для МК51. Мітка завжди завершується двокрапкою (:).

Псевдокоманди асемблера не перетворюються в двійкові коди, а тому не можуть мати міток. Виняток складають псевдокоманди резервування пам'яті і визначення даних (DS, DB, DW) у псевдокомандах, які здійснюють визначення символічних імен; в поле мітки записується певне символічне ім'я, після якого двокрапка не ставиться.

Як символічні імена і мітки не можуть бути використані імена мнемокоманд, псевдокоманд та операторів асемблера, а також мнемонічні позначення регістрів і інших внутрішніх блоків МК.

ОПЕРАЦІЯ. В поле операції записується мнемонічне позначення команди МК або псевдокоманди асемблера, яке є скороченням (абревіатурою) повного англійського найменування виконуваної дії. Наприклад: MOV - move - перемістити, JMP - jump - перейти, DB - define byte - визначити байт.

Для МК48 і МК51 використовується строго визначений і обмежений набір мнемонічних кодів. Будь-який інший символ, який розміщений в полі операції, сприймається асемблером як помилковий.

ОПЕРАНДИ. В цьому полі визначаються операнди (або операнд), що беруть участь в операції. Команди асемблера можуть бути без-, одно- або двоопераднними. Операнди розділяються комою (,).

Операнд може бути заданий безпосередньо або у вигляді його адреси (прямої або посередньої). Безпосередній (прямий) операнд зображається числом (MOV A, #15) або символічним іменем (ADDC A, #OPER2) з обов'язковим зазначенням префікса безпосереднього операнда (#). Пряма адреса операнда може бути задана мнемонічним позначенням (IN A, P1), числом (INC 40), символічним ім'ям (MOV A, MEMORY). Вказівкою на посередню адресацію служить префікс (@). В командах передачі керування операндом може виступати число (LCALL 0135H), мітка (JMP LABEL), посередня адреса (JMP @ A), або вираз (JMP ⊗ - 2, де ⊗ - біжучий вміст лічильника команд).

Символічні імена і мітки, які використовуються як операнди, повинні бути строго визначені, а числа представлені з вказівкою (зазначенням) системи числення, для чого використовується суфікс (літера, яка стоїть після числа): B - для двійкової; Q - для вісімкової; D - для десяткової; H - для шістнадцяткової. Число без суфікса по замовчуванню вважається десятковим.

ОБРОБКА ВИРАЗІВ У ПРОЦЕСІ ТРАНСЛЯЦІЇ. Асемблери МК48 і МК51 допускають використання виразів у полі операндів, значення яких обчислюються в процесі трансляції.

Вирази являють собою сукупність символічних імен і чисел, що зв'язані операторами асемблера. Оператори асемблера забезпечують виконання арифметичних («+» - додавання, «-» - віднімання, «*» - множення, «/» - ціле ділення, MOD - ділення по модулю) та логічних (OR - АБО, AND - І, XOR - виключне АБО, NOT - заперечення) операцій у форматі 2-байтових слів.

Наприклад, запис ADD A, #(NOT(13)+1) еквівалентний запису ADD A, #0F3H і забезпечує додавання вмісту акумулятора з числом −13, яке подане в доповняльному коді.

Широко використовуються також оператори LOW і HIGH, які дозволяють виділити молодший і старший байти 2-байтового операнда.

КОМЕНТАР. Поле коментаря може бути використане програмістом для текстового або символьного пояснення логічної організації прикладної програми. Поле коментаря повністю ігнорується асемблером, а тому в ньому допускається використовувати будь-які символи. Згідно правил мови асемблера поле коментаря починається після крапки з комою (;).

ПСЕВДОКОМАНДИ АСЕМБЛЕРА. Асемблююча програма транслює вихідну програму в об'єктні коди. Хоча транслююча програма бере на себе багато рутинних задач програміста, таких як присвоєння дійсних адрес, перетворення чисел, присвоєння дійсних значень символьним змінним і таке інше, програміст все ж таки повинен вказати їй деякі параметри: початкова адреса прикладної програми, формати даних і т.п. Всю цю інформацію програміст вставляє у вихідний текст своєї прикладної програми у вигляді псевдокоманд (директив) асемблера, які лише керують процесом трансляції і не перетворюються в коди об'єктної програми.

Псевдокоманда ORG 10H задає асемблеру адресу комірки пам'яті (10H), в якій повинна бути розміщена наступна за нею команда прикладної програми.

Псевдокомандою EQU можна будь-якому символічному імені, яке використовується в програмі, поставити у відповідність певний операнд.

Наприклад, запис

```
PET EQU 13
```

приводить до того, що в процесі асемблювання всюди, де зустрінеться символічне ім'я PET, воно буде замінюватися числом 13.

Символічні імена операндів, які перевизначаються в процесі виконання програми, визначаються командою SET:

```
ALFA  SET   3
      ...
ALFA  SET   ALFA+1
```

Асемблер МК51 дозволяє визначити символічне ім'я адрес внутрішніх (псевдокоманда DATA), зовнішніх (XDATA) даних або адресу біта (псевдокоманда BIT). Наприклад, директива

```
ERROR_FLAG BIT 25H.3
```

визначає символічне ім'я ERROR_FLAG як третій біт комірки ОЗП з адресою 25H.

Псевдокоманда DB забезпечує занесення в ПП константи, що є байтом.

Псевдокомандою END програміст дає асемблеру вказівку про закінчення трансляції.

Внаслідок трансляції повинна бути отримана карта пам'яті програми, де кожній комірці пам'яті поставлений у відповідність код, що в ній зберігається.

У відповідності до формату команд для представлення їх об'єктних кодів виділяється одна, дві або три комірки пам'яті програм. В першій комірці завжди розміщується код операції, у другій (а для МК51 і у третій) - безпосередній операнд, адреса прямоадресованого операнда, адреса переходу всередині сторінки пам'яті програм (для команд передачі керування МК48) або зміщення (для команд передачі керування МК51). Для команд LCALL і LJMP у другому та третьому байтах об'єктного коду вказується адреса передачі керування (у другому - старша частина, в третьому - молодша).

1.4.2. Ввід, редагування, трансляція і налагодження прикладних програм в крос-системах розробки

Написанням тексту програми завершується перший етап розробки прикладного програмного забезпечення - «від постановки задачі до вихідної програми» і починається наступний - «від вихідної програми до об'єктного модуля».

Для простих програм об'єктний код можна одержати вручну (ручна трансляція). Однак для більш складних програм необхідні спеціальні засоби автоматизації підготовки програм. Звичайно такі засоби використовують великі ємності пам'яті і широкий набір периферійних пристроїв, тому вони не можуть бути резидентними, а використовуються лише в крос-режимі на універсальних міні і мікро-ЕОМ (СМ-1800, СМ-1810, СМ-4, СМ-1841, IBM PC/XT, IBM PC/AT).

До мінімального складу програмного забезпечення крос-засобів входять: системна програма для вводу вихідного тексту прикладної програми, його редагування і запису на зовнішній носій інформації - так званий редактор текстів, або символьний редактор (найбільш розповсюджені назви CREDIT, EDITER, ED); програма-транслятор, що забезпечує перетворення вихідного тексту прикладної програми в об'єктний модуль (ASM48, ASM51).

Більш потужні крос-засоби передбачають наявність редактора зовнішніх зв'язків (LINK), що дозволяє включати в програму модулі, які розроблені незалежно один від одного, і програму, яка забезпечує налагодження переміщуваних програмних модулів на абсолютні адреси (LOCATE).

Для вводу вихідного тексту прикладної програми необхідно викликати редактора текстів, вказавши йому тип носія, на якому буде створено вихідний файл. Найчастіше в ролі носія використовується накопичувач на гнучкому магнітному диску. Нижче подається фрагмент діалогу з мікро-ЕОМ при підготовці програм в середовищі ДОС 1800 на мікро-ЕОМ СМ-1800, який забезпечує створення вихідного файла PROBL1 на гнучкому магнітному диску, розміщеному на дисководі №1.

Виклик редактора:

```
- CREDIT :F1: PROBL1. ASM<<CR>>
ISIS - II CRT-BASED EDITER VX.Y
NEW FILE
МАКРОФАЙЛ
————
```

ввід тексту програми;

запис створеного файла на дискету:

;ОСТАННІЙ РЯДОК ПРОГРАМИ <<CR>>

3 <<HOME>>

* EX<<CR>>

EDITER:F1:PROBL1.ASM

Крос-система автоматично видає на екран монітора виділені повідомлення. В подвійних кутових дужках вказані імена функціональних клавіш, що натискаються оператором.

Для *трансляції* вихідного тексту програми необхідно викликати транслятор, вказавши йому файл з вихідним текстом, місце розміщення об'єктного коду, а також умови формування і виводу лістингу.

Наприклад, діалог

- ASM48 :F1:PROBL1. ASM<<CR>>

===DOS-MACRO ASSEMBLER===

забезпечує формування об'єктного коду в файлі PROBL1.HEX і лістингу в файлі PROBL1.LST на гнучкому магнітному диску. Після закінчення трансляції при відсутності синтаксичних помилок видається повідомлення

ASSEMBLY COMPLETE NO ERRORS

або повідомлення

ASSEMBLY COMPLETE NNN ERRORS(LLL)

з вказівкою числа помилок (NNN) і номера останнього помилкового рядка (LLL) при наявності синтаксичних помилок.

Всі виявлені помилки виправляються у вихідному тексті прикладної програми (це відноситься і до помилок, які були виявлені на етапі налагодження). Для цього необхідно знову викликати редактора тексту та здійснити редагування вихідного тексту програми, після чого виконати повторну трансляцію.

Якщо вихідний текст прикладної програми не мав зовнішніх посилань і містив директиву ORG, то після успішного завершення трансляції етап розробки програмного забезпечення «від вихідної програми до об'єктного модуля» можна вважати завершеним.

1.4.3. Налагодження прикладного програмного забезпечення мікроконтролерів

Після одержання об'єктного коду прикладної програми неминуче наступає етап налагодження, тобто встановлення факту її працездатності, а також виявлення (локалізація) і усунення

помилок. Без цього етапу розробки жодне програмне забезпечення взагалі не має права на існування. Налагодження програмного забезпечення є окремою складною задачею, яка майже не піддається формалізації і вимагає для свого виконання високого професіоналізму і глибоких знань розробника.

Звичайно налагодження прикладного програмного забезпечення здійснюється в декілька етапів. Прості (синтаксичні) помилки виявляються вже на етапі трансляції. Далі необхідно виконати:

автономне налагоджування кожної процедури в статичному режимі, яке дозволяє перевірити правильність здійснюваних обчислень, правильність послідовності переходів всередині процедури (відсутність «зациклювання») і т.п.;

комплексне налагоджування програмного забезпечення в статичному режимі, яке дозволяє перевірити правильність алгоритму керування (за послідовністю формування керуючих дій);

комплексне налагоджування в динамічному режимі без підключення об'єкта для визначення реального часу виконання програми і її окремих фраґментів.

Слід врахувати, що автономне налагодження окремих модулів значно простіше і ефективніше за налагоджування усієї прикладної програми, тому переходити до етапу комплексного налагоджування доцільно, лише вичерпавши всі засоби автономного налагоджування.

Вище перелічені етапи налагоджування здійснюються звичайно з використанням крос-системи.

В склад крос-системи входять програми-налагоджувачі (узагальнене ім'я DEBUG), які інтерпретують (моделюють) виконання програм, написаних для МК. Такі програмні імітатори дозволяють ефективно налагоджувати обчислювальні процедури, а також алгоритм функціонування контролера.

Розробнику надано доступ до будь-якого ресурсу МК, є можливість покомандного та пофраґментного виконання програм і зупинок при потребі, а також підрахунку числа тактів виконання тих чи інших фраґментів програми, ініціювання переривання, дизасемблювання вмісту ПП і таке інше.

Крос-налагоджувачі дозволяють промоделювати практично всі можливі варіанти роботи програми і таким чином переконатися в її працездатності. На цьому етапі можлива перевірка працездатності програми при нештатних ситуаціях в умовах

надходження некоректних вхідних дій (для застосування з підвищеними вимогами до безпеки).

Найбільш потужні імітатори дозволяють моделювати і середовище функціонування МК, тобто різного роду об'єкти та давачі, які підключені до нього. При цьому з'являється можливість проводити комплексне налагоджування програмного забезпечення, не побоюючись, що можливі помилки в програмі, алгоритмі або некоректні дії оператора приведуть до виходу з ладу технічних засобів розроблюваної системи. Головним недоліком крос-системи є неможливість прогону програми в реальному масштабі часу, тобто з швидкістю виконання програми в самому МК, а також неможливість комплексування апаратних і програмних засобів розроблюваної системи. В силу цих причин достовірність прикладних програм, які налагоджені в крос-режимі, є недостатньо високою.

Окремі фраґменти програмного забезпечення, які вимагають налагоджування в реальному часі, можуть бути проведені на налагоджувальному модулі. Налагоджувальний модуль являє собою невелику, як правило одноплатну, мікро-ЕОМ (правильніше, мікроконтролер), побудовану на однотипному МК. Однак при налагодженні доводиться враховувати обмеження, які пов'язані з тим, що частина ресурсів налагоджувального модуля (простір адрес пам'яті і пам'яті даних, деякі лінії портів та рівні переривань) не може бути використана прикладним програмним забезпеченням (ПЗ), тому що вимушено використовується резидентною операційною системою (ОС). Резидентна ОС, або монітор, - це програма, яка забезпечує взаємодію оператора з налагоджувальним модулем і надає йому ряд команд, що полегшують налагодження програмного забезпечення. До налагоджувального модуля може бути підключене додаткове обладнання, яке необхідне розроблювальній системі (зовнішня пам'ять, порти, таймери).

1.4.4. Налагодження програмного забезпечення ЦСАК з допомогою налагоджувачів-симуляторів

Використання персональних комп'ютерів привело до появи нових програмних продуктів, які значно полегшили розробку і освоєння сучасної елементної бази мікропроцесорних систем - однокристальних мікро-ЕОМ. Такі програми призначені для

логічного налагодження програм, написаних мовою асемблера і, як правило, апаратних засобів не підтримують. Одною з таких програм є повноекранний налагоджувач-симулятор для програм, написаних мовою асемблера ОМЕОМ КР1816ВЕ51 (або КР1816ВЕ31, КМ1816ВЕ751)[32].

Налагоджувач працює на персональних ЕОМ типу IBM PC XT/AT та сумісних з ними ЕОМ і вимагає для роботи не менш 256 кілобайтів оперативної пам'яті.

Вся інформація про стан ОМЕОМ, логічні рівні сиґналів в будь-якій зручній формі (двійковій, десятковій, шістнадцятковій) всіх внутрішніх реґістрів, внутрішньої та зовнішньої пам'яті відображаються на екрані комп'ютера. Налагоджувану програму можна безпосередньо вводити в кодах асемблера у налагоджувач-симулятор, і вона буде транслюватися і відображатися у заданій формі. Таким чином, такий програмний продукт дозволяє надзвичайно просто, зручно і наочно проводити логічне налагодження програм, оцінювати їх швидкодію, при необхідності змінювати розміщення програм і даних в пам'яті ОМЕОМ, симулювати підключення зовнішніх пристроїв.

Основні можливості, які дозволяє реалізувати налагоджувач-симулятор:

1. Завантажити для налагодження шістнадцяткові файли (з розширенням HEX), які вироблені різними кросзасобами (трансляторами з мови асемблера), а також файли двійкових кодів, зчитаних, наприклад, із ПЗП, або безпосередньо записати програму на асемблері з одночасною її трансляцією в необхідну область пам'яті ПЗП.

2. Проглянути на екрані комп'ютера текст завантаженої програми, включаючи адреси і коди команд, область імітованого ОЗП даних, область зовнішньої пам'яті, пам'яті програм, вміст всіх реґістрів і портів ОМЕОМ.

3. Виконати завантажену програму по кроках з переглядом результатів її виконання після кожного кроку або в неперервному режимі з зупинкою по точках переривання при досягненні заданих адрес.

4. Внести зміни в завантажену програму в кодах асемблера або машинних кодах, внести зміни в стани реґістрів, прапорців і пам'яті в командному режимі і в режимі повноекранного редагування.

5. Вивести на друк або дискові носії чи зберегти в пам'яті комп'ютера тексти програм і вміст пам'яті ОМЕОМ, завантажити пам'ять з дискового носія, отримати трасування програми.

6. Визначити час виконання завантаженої програми і її частин по внутрішньому лічильнику.

Таким чином, дана програма-симулятор надає широкі можливості для розробки і налагодження програм мовою асемблера ОМЕОМ. Спілкування з програмою можливе двома способами - в режимі повноекранного налагоджувача або в командному режимі. Для оперативного виконання команд симулятора необхідно користуватися поряд з командами функціональних клавіш F1-F10 клавішами курсорів [Home....PgDn].

Функціональні клавіші налагоджувача-симулятора

F1 - виконує біжучу інструкцію завантаженої програми. Біжуча інструкція виділена у вікні дизасембльованого тексту світлим прямокутником. Після виконання інструкції на екрані відразу видно її результати.

F2 - виконує програму до наступної за біжучою інструкції. Ця клавіша дозволяє виконати підпрограму або цикл як одну інструкцію, що зручно, тому що не потрібно проглядати вже налагоджені підпрограми.

F3 - дозволяє подати всю числову інформацію на екрані (вміст регістрів та пям'яті) в десятковій, а при повторному натисканні - в двійковій формі. Після запуску налагоджувача вся інформація подана шістнадцятковим кодом.

F4 - перемикає вікно пам'яті з внутрішньої (INT RAM) на зовнішню (EXT RAM) і назад.

F5 - встановлює точки переривання.

F6 - перемикає форму подання інформації у вікні в двійкову, потім в десяткову форму і назад в шістнаддяткову.

F7 - листає вікно пам'яті даних вгору на один рядок.

F8 - листає вікно пам'яті даних вниз на один рядок.

F9 - листає вікно пам'яті програм вгору на один рядок.

F10 - листає вікно пам'яті програм вниз на один рядок.

Для швидкого листання можна скористатися клавішами:
[Home] - листає вікно пам'яті даних вгору на одну сторінку,
[End] - листає вікно пам'яті даних вниз на одну сторінку,
[PgUp] - листає вікно пам'яті програм вгору на одну сторінку,
[PgDn] - листає вікно пам'яті програм вниз на одну сторінку.

Команди налагоджувача

Для довідок про команди налагоджувача необхідно ввести команду «H» або натиснути комбінацію клавіш «Ctrl-H».

Синтаксис команд наступний:

- <параметр>,
- [необов'зковий параметр].

Всі числові значення повинні мати шістнадцятковий формат, при цьому не потрібно вводити букву «h».

1. Завантажити файл у пам'ять.

L [<тип пам'яті><початкова адреса>,] <файл, спец.> [/A]

<Тип пам'яті> може бути I,E,P. Тоді файл завантажиться у внутрішню (Int), зовнішню (Ext) або програмну (Pgm) пам'ять відповідно.

<Початкова адреса> і <тип пам'яті> вказується лише при завантаженні двійкового коду.

Приклад. L I 0AF, A:\PGM\PROG1 - завантажити двійковий файл у внутрішню пам'ять, починаючи з адреси 0AF.

2. Зберегти область пам'яті в файлі.

S <тип пам'яті><початкова адреса>-<кінцева адреса>,<файл, спец>

Приклад: S P 20-642,C:\PGMLIB\MYFILE

3. Роздрукувати дизасембльований текст програми, починаючи з початкової адреси.

PRTD<початкова адреса>,<кількість команд>[,<файл, спец.>]

По замовчуванню вивід здійснюється на принтер.

4. Занести число в регістр.

R<номер регістра>=<число>

Заносить число в регістр (R0...R7) біжучого банку. Число повинно бути байтом.

Приклад: R4=FA

Аналогічно можна занести число в будь-який регістр спеціального призначення. До них відносяться такі регістри: A, B, TH0, TH1, TL0, TL1, DPH, DPL, DPTR, SP, IP, IE, TMOD,

TCON, SCON, SBUF, PC. Для регістрів DPTR і PC числа повинні бути двобайтовими.

Приклади: B=20 DPTR=FA00 PC=A011

5. Встановити чи скинути прапорець в PSW. Імена прапорців C, AC, F0, S1, S0, OV, P. Якщо число дорівнює нулю, то прапорець скидається, інакше - встановлюється.

<ім'я прапорця>=<число>.

Приклад: S1=0

6. Занести число в порт. Номер порту може бути від 0 до 3, число - байтом.

P0<номер порту>=<число>.

Приклад: P02=1A

7. Встановити адресу дезасембльованого тексту у вікні.

D<адреса>

Приклад: D 0240

8. Занести число в пам'ять.

<тип пам'яті><адреса>[-<кінцева адреса>]=<число>.

Якщо вказана кінцева адреса, то цим числом заповнюється область пам'яті.

Приклад: I 22=55 P 0-40=FF P 0C=23

9. Встановити чи скинути біт у пам'яті.

<тип пам'яті><адреса>.<номер біта>=<число>.

<номер біта> може бути 7...0, старший біт 7.

Приклад: I 20.6=1

10. Встановити чи скинути біт в регістрі спеціального призначення.

<ім'я регістра>.<номер біта>=<число>

До цих регістрів належать A, B, P00-P03, IP, IE, TMOD, TCON, SCON.

Приклад: TMOD.3=0

11. Встановити початкову адресу пам'яті у вікні.

M <тип пам'яті><початкова адреса>.

Приклад: M I 20 ME 0FF MP 0

12. Виконати програму.

G [<початкова адреса>[,<кінцева адреса>]]

Програма виконується від <початкової адреси> до <кінцевої адреси>. Якщо <початкова адреса> не вказана, то виконання починається із біжучої команди (виділена на екрані білим прямокутником). <Кінцеву адресу> можна не вказувати, якщо використовуються точки переривання.

Програму, що виконується, можна зупинити натисканням будь-якої клавіші. Можна вказати в команді G лише кінцеву адресу, але при цьому необхідно вказати кому. Команда G без параметрів еквівалентна комбінації клавіш <Alt-F10>.

Приклад: G 100-FF0 G,2200

13. Скинути лічильник часу виконання програм.

RSTC

14. Імітувати високий або низький рівень на входах INT0 або INT1

INT<0/1>=<число>

Приклад: INT1=0

15. Скинути процесор в початковий стан.

RST

16. Перезапустити налагоджувач.

N

17. Вийти в DOS.

QUIT

Перехід у режим повноекранного налагоджувача (редагування) здійснюється натисканням клавіші <Enter> без вводу команди. Після цього можна переміщувати курсор по екрану з допомогою клавіш переміщення і змінювати вміст регістрів, пам'яті і прапорців набором відповідних чисел на клавіатурі. Можна змінювати також початкову адресу дезасемблюваного тексту (біжучої інструкції) і початкові адреси вікон пам'яті (змінюючи числа в перших рядках вікон). Повноекранне редагування можна здійснювати не лише при шістнадцятковому, а й при десятковому та двійковому поданні інформації на екрані. Під час редагування залишаються доступними всі команди, які вводяться за допомогою функціональних клавіш. Для швидкого

переміщення по екрану можна користуватися клавішами [Tab] і [Shift-Tab]. Для того, щоб повернутися в командний рядок, слід натиснути клавішу [Enter] знову.

Для переходу в режим асемблера (тобто в режим вводу команд відлагоджуваної програми в кодах асемблера - мнемонічних позначеннях) необхідно в режимі повноекранного редагування перемістити курсор в поле біжучої інструкції завантаженої програми. Тепер можна набирати мнемонічний код команди (наприклад, MOV A, #0F4) і натиснути [Enter]. Якщо мнемонічний код правильний, то відповідні їй двійкові коди процесора заносяться в пам'ять програм, а вікно встановлюється на наступну адресу. Якщо мнемонічний код команди неправильний, то вона не заноситься в пам'ять, про що повідомляється звуковим сиґналом.

При асемблюванні підтримуються імена реґістрів спеціального призначення, а також бітові функції MK51.

Якщо виникає неоднозначність введених команд, слід числові значення доповнювати спереду нулем.

Вихід з режиму асемблера здійснюється зміщенням курсора із поля біжучої інструкції або натисканням клавіші [Q] (не змішувати з командою [Quit]).

Встановлення точок переривання проводиться незалежно від режиму роботи налагоджувача і може бути встановлене як в командному режимі, так і в режимі повноекранного редагування.

Меню точок переривання викликається натисканням клавіші F5. Одночасно можна встановити до 8 точок переривання.

Переривання (зупинка) програми здійснюється при досягненні вказаної в колонці «PC» адреси при виконанні умови «Counter=Occur».

Значення «Counter» - це лічильник, який визначає, скільки разів програма повинна пройти через вказану адресу, щоб сталася зупинка. «Occur» - показує, скільки разів програма проходила через вказану адресу.

Після закінчення редагування біжучі значення точок переривання можна зберегти на диску, натиснувши клавішу F2. При цьому записується номер набору точок переривання (0-9). Інформація про набір точок переривання налагодження записується у файл FD51.BRK. Відновити стан переривань налагодження можна, натиснувши клавішу F1, також вказавши її номер.

Для повернення в основне меню необхідно натиснути F5. Визначивши точки переривання, можна запускати програму командою G без параметрів. При зупинці програми по точці

переривання на екран комп'ютера видається повідомлення з вказівкою номера точки переривання.

Найбільш повне і комплексне налагодження прикладного програмного забезпечення разом з апаратними засобами може бути проведене на інструментальній мікро-ЕОМ з так званим внутрішньосхемним емулятором (ВСЕ). Прототип розробленої системи через сакет для встановлення МК з'єднується плоским багатожильним кабелем з ВСЕ, який в свою чергу забезпечує доступ до всіх технічних засобів інструментальної мікро-ЕОМ. При цьому майже всі ресурси МК залишаються в розпорядженні прикладного програмного забезпечення. Під керуванням мікро-ЕОМ ВСЕ дозволяє проганяти прикладну програму або її окремі фраґменти в реальному темпі, зупиняти виконання програми за багатьма ознаками, проводити трасування зовнішніх сиґналів МК і системи під час виконання програми. Достовірність програмного забезпечення, налагодженого на інструментальній мікро-ЕОМ за допомогою ВСЕ, є високою, хоч і не рівною одиниці.

У будь-якому випадку для доведення прикладного програмного забезпечення контролера необхідні комплексні і всесторонні випробування розробленої системи в реальному оточенні і в усяких можливих режимах.

1.5. Розрахунок швидкодії системи керування

Система керування повинна функціонувати в реальному режимі часу, тобто без нагромадження затримок [31]. Іншими словами, обробка всіх сиґналів і видача їх на пульт повинна закінчуватися до подання наступного сиґналу переривання. Виконання кожної операції ОМЕОМ займає певну кількість тактів, тривалість яких пропорційна частоті тактового ґенератора. Таким чином, враховуючи, що САК виконує послідовно всі підпрограми обробки, час виконання програми дорівнює:

$$t = (\sum_{i=1}^{n} k_i m_i) t_0.$$

де k_i – кількість тактових інтервалів на і-ту операцію,
m_i – число таких і-тих елементарних операцій,
n – кількість використовуваних операцій,
t_0 – тривалість тактового інтервалу.

Необхідно, щоб час t був менший за період слідування

сиґналів переривання від внутрішнього таймера.

Якщо значення t завелике, то необхідно або підвищити частоту тактового ґенератора, або оптимізувати алгоритм обробки сиґналів (зменшити кількість операцій або використати більш швидкодіючі команди, наприклад, при виконанні операцій множення).

Якщо вказані шляхи не приводять до бажаного результату або вони не можуть бути реалізованими (частота вже максимально допустима, алгоритм оптимізований), то потрібно переглянути структурну схему МПСК, вибрати інший тип мікропроцесора.

Розділ 2. ПРОЕКТУВАННЯ СИСТЕМ ОБРОБКИ АНАЛОГОВИХ СИҐНАЛІВ НА ОСНОВІ ОДНОКРИСТАЛЬНОЇ мікро-ЕОМ КМ1813ВЕ1А

2.1. Основні відомості про ОМЕОМ КМ1813ВЕ1А

Особливості ОМЕОМ КМ1813ВЕ1А наступні:
- наявність вмонтованих АЦП і ЦАП;
- можливість побудови функціонально завершених САК;
- порівняно проста система команд і структура процесора;
- простота програмування і побудови САК на його основі;
- можливість програмного множення;
- можливість цифрового і послідовного вводу і виводу;
- порівняно малий обсяг пам'яті ОЗП і ПЗП;
- можливість ультрафіолетового стирання пам'яті команд.

2.1.1. Призначення. Мікро-ЕОМ (структурна схема показана на рис.2.1) КМ1813ВЕ1А (далі ВЕ1) призначена для заміни (реалізації) аналогових вузлів систем автоматичного керування і обробки сиґналів в смузі частот 0-20 кГц.

2.1.2. Можливості. Дозволяє реалізувати типові функціональні вузли: фільтри нижніх частот (ФНЧ), смугові фільтри (ФС), порогові детектори, обмежувачі, формувачі нелінійних функцій, багаточастотні ґенератори, ґенератори спеціальних функцій, спектроаналізатори, модулятори і демодулятори і т.п. Перелік реалізованих функцій і вузлів є відкритим, тобто архітектура і система команд процесора є настільки гнучкими, що дозволяють реалізувати на ньому все нові і нові пристрої.

2.1.3. Програмна сумісність. Мікро-ЕОМ ВЕ1 програмно зверху сумісна з процесором І2920 фірми INTEL, крім того, в ній введені порти послідовного вводу/виводу даних і програмно реалізовані цикли.

2.2. Структурна схема

В структурній схемі ВЕ1 можна виділити три основні частини: І - аналогову частину; ІІ - пам'ять команди; ІІІ - арифметико-логічний пристрій (або пристрій цифрової обробки).

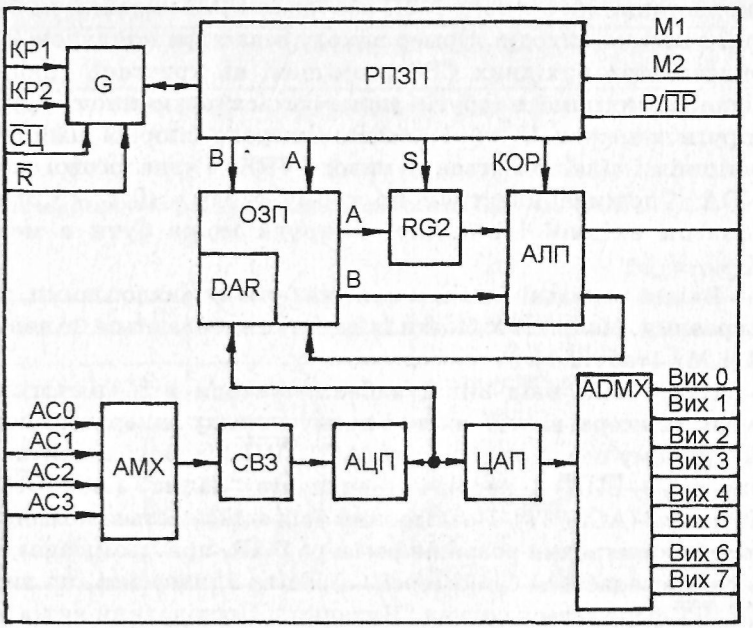


Рис. 2.1. Структурна схема однокристальної мікро-ЕОМ КМ1813ВЕ1А

2.2.1. Аналогова частина. Являє собою інтерфейс для вводу/виводу даних і має в собі схеми для виконання багатоканального аналого-цифрового і цифроаналогового перетворення (АЦП і ЦАП): вхідний аналоговий мультиплексор (АМХ) на 4 входи, вхідну схему вибірки-зберігання (СВЗ), компаратор, 9-розрядний ЦАП, вихідний аналоговий демультиплексор (ADMX) на 8 виходів, схему вибірки-зберігання і підсилювач на кожний із 8 виходів, схему керування аналоговою частиною.

Аналого-цифрове перетворення здійснюється методом порозрядного зрівноважування. Рівень перетворюваної напруги запам'ятовується на конденсаторі СВЗ, загальному для всіх входів. Ємність конденсатора вибирається від 100 до 500 пФ для перетворення з точністю 9 розрядів: старший розряд знаковий і 8 розрядів для подання модуля. Для формування 9-розрядного коду відліку вхідного сиґналу (при частоті кварцового резонатора 6,66 МГц) необхідно 25 команд: 9 – для зважування розрядів і 16 – для встановлення АЦП. Результат перетворення запам'ятовується в реґістрі даних DAR (digital analog register), який адресується як комірка з адресою 40 ОЗП. З DAR результат перетворення може бути переписаний в будь-яку комірку ОЗП,

використаний як операнд АЛП або через ЦАП виданий на будь-який з восьми виходів. Номер виходу задається командою OUT. Конденсатори вихідних СВЗ розміщені на кристалі. Діапазон вхідної і вихідної напруги визначається величиною опорної напруги в межах $U_{оп}$=(+1...+2)В. Джерело опорної напруги є зовнішнім і підключається до виводу VREF і «аналогової землі» GNDA. Споживаний струм по цьому входу - (0,1 - 0,2) мА. Діапазон вхідної і вихідної напруги може бути в межах $+U_{оп}$...$-U_{оп}$.

Вхідні і вихідні сиґнали можуть бути як аналоговими, так і цифровими. Можливі варіанти їх подання вибираються по виводах М1 і М2 (табл.2.1).

Цифровий ввід-вивід забезпечується від аналогового мультиплексора, виводи якого в цьому випадку використовуються для прийому послідовного коду (АС0/$\overline{\text{ППК}}$), видачі послідовного коду (АС3/$\overline{\text{ВПК}}$) і керуючих сиґналів "Запис" (АС1/$\overline{\text{ЗП}}$) та "Читання" (АС2/$\overline{\text{ЧТ}}$). Послідовний ввід здійснюється командами вводу і формування розрядів реґістра DAR, при цьому знаковий розряд передається без інверсії, а решта з інверсією, на виході АС2/$\overline{\text{ЧТ}}$ формується сигнал "Читання". Послідовний вивід коду в інверсному вигляді починається із старшого розряду. На виході АС1/$\overline{\text{ЗП}}$ формується сиґнал "Запис".

Таблиця 2.1

Входи		Режими вводу/виводу	
М1	М2	Ввід	Вивід
5В	5В	Аналоговий АС0-АС3	Аналоговий(вих.0-7)
5В	–5В	Цифровий АС0/ППК	Цифровий АС3/$\overline{\text{ВПК}}$ Цифровий (вих.4-7) Аналоговий (вих.0-3)
–5В	+5В	Аналоговий АС0...АС3	Аналоговий (вих.4-7) Цифровий (вих.0-3)
–5В	+5В	Аналоговий АС0...АС3	Цифровий (вих.0-7)

2.2.2. Пам'ять команд - РПЗП з ультрафіолетовим стиранням обсягом 192 х 24 розряди.

Командне слово РПЗП довжиною 24 розряди розділено на п'ять полів, одне із яких керує роботою аналогової частини, решта 4 - цифрової.

23 21	20 15	14 9	8 5	4 0
L2...L0	B5...B0	A5...A0	S3...S0	ADF ADK 10 2...0
Цифрова операція	Адреса B	Адреса A і константа	Код зсуву	Аналогова операція

Цифрові операції задаються полем L2...L0 (табл. 2.2).

Таблиця 2.2

L2	L1	L0	Код	Суть операції
1	1	1	LDA	пересилка даних
1	1	0	ADD	арифметичне додавання
1	0	1	SUB	віднімання
1	0	0	ABA	додавання з модулем
0	1	1	ABS	модуль
0	1	0	LIM	обмеження
0	0	1	AND	логічне І
0	0	0	XOR	виключне АБО

Операція LIM формує максимальне додатне число 011...1B, якщо A≥0; або максимальне від'ємне число 100...0B, якщо A<0.

В полях B5...B0 і A5...A0 задаються адреси операндів B і A ОЗП. Якщо A5, A4 = 11B, то в полі A3...A0 вказується константа в доповнюючому двійковому коді в діапазоні (-1...+0,875) з кроком 0,125. Кодування адрес наступні: B5...B4 = (0...0) = M00
= (10011) = M39
= (10100) = DAR.

В полі S3...S0 вказується, на скільки розрядів можна зсунути вправо (на 1...13 розрядів) або вліво (на 1...2 розряди) операнд A в регістрі RG2.

В полях ADF1, ADF0 і ADK2...ADK0 задаються аналогові операції (табл.2.3).

В робочому режимі в мікро-ЕОМ реалізовано конвеєрне вироблення команд: із РПЗП послідовно зчитуються 4 команди, які записуються в 96-розрядний буферний регістр. Зчитування однієї команди виконується за 4 такти роботи ґенератора тактових імпульсів. Програмний лічильник повертається в початковий стан після виконання 191-ї команди або команди EOP. Одночасно зі зчитуванням чотирьох команд процесор виконує попередні 4 команди.

Таблиця 2.3

F1	F0	K2	K1	K0	Код	Суть операції
0	0	0	К	К	IN_K	Відлік сигналів із входів AC0...AC3(K)
0	0	1	0	0	NOP	Холоста операція
0	0	1	0	1	EOP	Кінець програми
0	0	1	1	0	CVTS	Аналого-цифрове перетворення знака відліку вхідного сигналу
0	0	1	1	1	CNDS	Аналіз знакового біта DAR
0	1	К	К	К	OUT_K	Вивід сигналів на k-тий вихід
1	0	n	n	n	CVT_N	Аналогово-цифрове перетворення n-го розряду відліку вхідного сигналу
1	1	n	n	n	CND_N	Аналіз окремих бітів DAR

2.2.3. Арифметико-логічний пристрій. АЛП виконує обчислення в доповняльному коді з 25-розрядною точністю над промасштабованим операндом А і операндом В, які надходять із ОЗП. Результат обчислення записується в ОЗП за адресою операнда В. Старший розряд - знаковий, а діапазон змінної знаходиться в межах $-1 \le x \le +1$. Мінімальне значення будь-якої змінної складає $2^{-24} = 6 \cdot 10^{-8}$.

АЛП має логічні схеми для зсуву і фіксації переповнення, коли результат виходить за межі розрядної сітки (25 розрядів). Для роботи з переповненням в АЛП передбачено розширення розрядної сітки до 28 розрядів. При роботі без переповнення 4 старших розряди містять значення знакового розряду. При переповненні в 26 і 27 розряди записується результат зсуву вліво, а 28 розряд зберігає інформацію про перенос. Таким чином, ознакою переповнення служать різні значення в 4-знакових розрядах АЛП.

Для АЛП можна програмно задати 2 режими обробки переповнення: з обмеженням і без. В режимі з обмеженням результат замінюється найближчим максимально допустимим значенням із урахуванням знаку (+1 або -1), а на вивід переповнення (ПП) видається сигнал низького рівня. В режимі без обмеження переповнення в АЛП 25 молодших розрядів результату записуються в ОЗП за адресою операнда В.

Регістр RG2 призначений для масштабування сигналів в доповняльному коді з виходу А ОЗП (лівий зсув - 2 розряди, правий - до 13). При зсуві вліво праві вільні розряди замінюються нулями, а при зсуві вправо вільні ліві розряди замінюються знаковим розрядом (у доповняльному коді). Зсув еквівалентний множенню операнда А на 2^s, s = (-13...2), див. табл.2.4.

Таблиця 2.4

S3	S2	S1	S0	Код	Значення
0	0	0	0	R01	2^{-1}
0	0	0	1	R02	2^{-2}
.....	.....	.....	.....	.....	.....
1	1	0	0	R13	2^{-13}
1	1	0	1	L02	2^{2}
1	1	1	0	L01	2^{1}
1	1	1	1	R00	$2^{0}=1$

2.2.4. Оперативний запам'ятовуючий пристрій виконує функцію пам'яті даних і являє собою двоадресний ОЗП статичного типу з довільною вибіркою обсягом 40 х 25 бітів. ОЗП адресується шестирозрядними адресами (000000...100111) з використанням тільки прямої адресації. Розширене поле адрес ОЗП (до 64 слів) використовується для адресації DAR (101000) і формування 16 констант (110000...111111) - табл. 2.5.

ОЗП має два регістри А і В. З регістра А зчитується інформація, яка через RG2 поступає на один із входів АЛП як операнд А. Через регістр В інформація поступає на другий вхід АЛП як операнд В. Результат обчислення в АЛП записується в ОЗП тільки за адресою операнда В.

Регістр DAR займає дев'ять старших розрядів 25-розрядного слова. Виходи регістра підключені до ЦАП. Будь-який розряд регістра може аналізуватися і встановлюватися у відповідності із значенням розряду переповнення. При зчитуванні DAR всі розряди, крім 9 старших, приймають значення 1.

Таблиця 2.5

Константа						Код	Значення
A5	A4	A3	A2	A1	A0		
1	1	0	0	0	0	KP0	0
1	1	0	0	0	1	KP1	0,125
....	....	....	....	....	....	....	.....
1	1	0	1	1	1	KP7	0,875
1	1	1	0	0	0	KM8	-1
1	1	1	0	0	1	KM7	-0,875
....	....	....	....	....	....	.....	.....
1	1	1	1	1	1	KM1	-0,125

Практично кількість значень констант може бути значно розширено за рахунок їх зсуву в RG2 при передачі на вхід АЛП.

2.2.5. Схема синхронізації керує роботою всіх вузлів ВЕ1. Вивід Р/$\overline{ПР}$ (робота /програмування) задає режим роботи і повинен бути підключений до корпусу. Його логічне значення залежить від підключення джерела -5 В:

в робочому режимі (-5 В підключено) - 1,

програмування (-5 В від'єднано) - 0.

Виводи КР1, КР2/ТІ призначені для підключення кварцового резонатора (КР) або для подачі зовнішніх тактових імпульсів (ТІ). На вивід $\overline{СЦ}$ (синхронізація циклу) видається імпульс низького рівня, який означає початок циклу вибірки команди із РПЗП. Двонаправлений вихід $\overline{R}$/$\overline{КПР}$ служить для видачі сигналу низького рівня, який означає зчитування із РПЗП команди ЕОР і переходу на початок програми, а також для прийому зовнішнього сигналу скидання $\overline{R}$, який обнулює програмний лічильник, тобто ініціює виконання 0-ї команди. Сигнал $\overline{R}$ необхідно стробувати імпульсами $\overline{СЦ}$, які дозволяють синхронізувати роботу ВЕ1 з іншими пристроями (рис. 2.2).

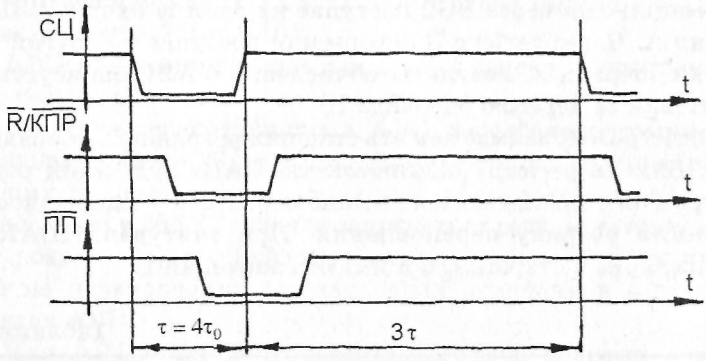


Рис. 2.2. Сигнали управління ВЕ1

2.3. Програмування і верифікація програм ОМЕОМ ВЕ1

Схема включення ОМЕОМ в режимі програмування показана на рис.2.3. В цьому режимі 24-розрядне командне слово розбивається на шість 4-розрядних слів, тобто РПЗП має організацію (192 х 6) х 4.

Запис і зчитування РПЗП здійснюється через двонаправлені виводи D0...D3, які керуються сигналом ЗП/$\overline{ЧТ}$. Високий рівень на цьому вході відповідає режиму запису, низький - режиму читання. Лічильник адрес змінює свій стан по спаду сигналу на

вході $\overline{CI}$, а при низькому рівні сигналу на вході $\overline{R}$ лічильник адрес встановлюється в нульовий стан.

Для запису даних в РПЗП по входах D3...D0 необхідно подати на вхід «Запис програми» (ЗППР - вивід 23) імпульси амплітудою 25 В тривалістю 50 мкс [27], а при зчитуванні - імпульси амплітудою 5 В. Для правильного програмування ОМЕОМ сигнали на входи $\overline{CI}$, $\overline{R}$, ЗП/$\overline{ЧТ}$ рекомендується подавати через вентилі з відкритим колектором. Для правильного зчитування даних з РПЗП на входи D3...D0 необхідно підключити зовнішні резистори. При запису в РПЗП дані подають в прямому коді, а при зчитуванні - в інверсному.

Розподіл бітів шини даних для програмування BE1 (табл.2.6).

Таблиця 2.6

	D3	D2	D1	D0
B0	ADF0	ADK2	ADK1	ADK0
B1	A2	B1	A1	ADF1
B2	A4	B3	A3	B2
B3	A0	B5	A5	B4
B4	S2	S1	S0	B0
B5	L2	L1	L0	S3

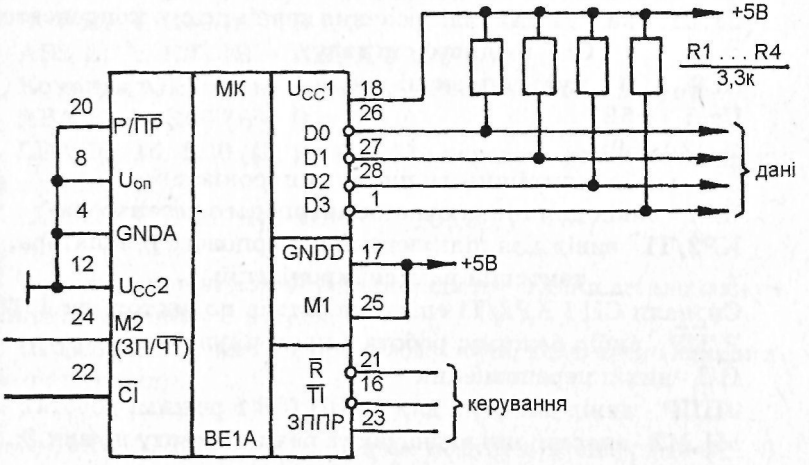


Рис. 2.3. Схема включення BE1 в режимі "Програмування"

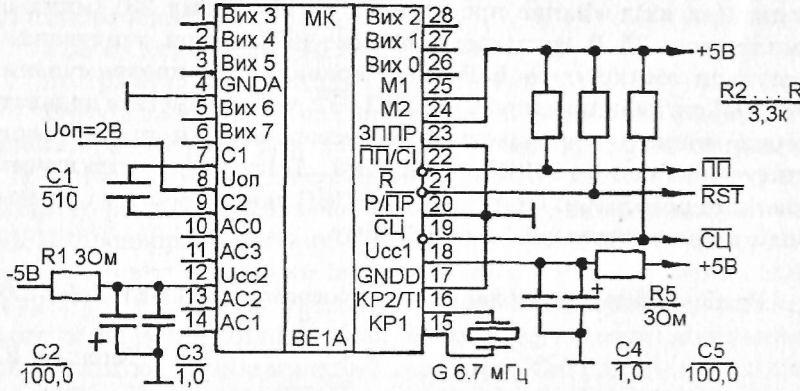


Рис. 2.4. Схема включення ВЕ1 в режимі "Робота"

Пояснення до рис.2.4

Вих. 0...7 - аналогові виходи;
GNDA - аналогова земля;
Uоп - вивід для подачі опорної напруги;
С1,С2, - виводи для підключення зовнішнього конденсатора СВЗ вхідного сиґналу;
AC0..AC3 - мультиплексні входи;
Ucc1 - +5В;
Ucc2 - -5В;
СЦ - вихід внутрішнього циклу синхронізації;
КР1 - вивід для підключення кварцового резонатора;
КР2/ТІ - вивід для підключення кварцового резонатора, тактовий вхід синхронізації;
Сиґнали СЦ і КР2/ТІ співвідносяться по частоті як 1/16;
Р/ПР - вибір режиму: робота / програмування;
ПП - вихід переповнення;
ЗППР - вивід джерела для РПЗП (0В в режимі робота);
М1,М2 - виводи, які визначають режим виводу на вих.0...7;
R - скидання в початковий стан.

При стиранні РПЗП (УФТ випромінюванням до 15 хв.) всі комірки встановлюються в 1.

2.5. Система команд КМ1813ВЕ1А

2.5.1. Безумовні команди.

Їх список приведений в табл.2.7.

Таблиця 2.7

Код	Позначення	Операція
000	XOR B,A,S	$B \oplus A \cdot 2^S \rightarrow B$ – виключне АБО
001	ABA B,A,S	$\lvert B \cdot 2^S \rvert + B \rightarrow B$ – додавання з модулем
010	LIM B,A,S	$+1 \rightarrow B$, якщо $A \geq 0$, $-1 \rightarrow B$, якщо $A \leq 0$ – обмеження
011	ADD B,A,S	$A \cdot 2^S + B \rightarrow B$ – додавання
100	AND B,A,S	$B \,\&\, (A \cdot 2^S) \rightarrow B$ – логічне І
101	SUB B,A,S	$B - (A \cdot 2^S) \rightarrow B$ – віднімання
110	ABS B,A,S	$\lvert A \cdot 2^S \rvert \rightarrow B$ – абсолютна величина
111	LDA B,A,S	$A \cdot 2^S \rightarrow B$ – пересилка

Опис безумовних команд наступний:

ADD B, A, R00 (B = B + A)

ADD B, KP4, R00 (B = B + 0,5).

ADD Y01, Y02, RO2 (Y01 = Y01 + Y01·2^{-2} - додавання оператора Y02·0,25 з оператором Y01).

SUB B, A, R02 (B = B-A·2^{-2} = B-A·0,25)

ABS B, A, R03 (B = ABS(A·0,125)).

Команда ABS встановлює прапорець переносу в нуль.

ABA B, A, R00 (B = B + A).

LDA 10, 15, R00 ((10) = (15)·2^0 - передача даних із комірки 15 в комірку 10)

LDA DAR, Y06, R01 (DAR = Y06·0,5).

Всі арифметичні команди і команди пересилки встановлюють прапорець переносу С в нуль.

Переповнення виникає при лівому зсуві, коли число виходить за розрядну сітку.

XOR B, A, R00 (додаються за модулем 2 операнди А і В, результат поміщається в В).

AND B, A, R00 (логічне І операндів А і В, результат поміщається в В).

Команда LIM встановлює мінімальний або максимальний результат за додатною або від'ємною шкалою в залежності від знакового розряду операнда A. Прапорець переносу при цьому C = 0.

2.5.2. Умовні команди

Команди безумовної арифметики стають умовними при вказуванні (табл.2.8) в полі аналогової команди керуючого слова умовної команди CND (s,n), де s,n - аналізовані біти DAR. Всі умовні операції так само, як і безумовні, можуть виконуватися із зсувом операнда A або без зсуву. Умовна команда аналізує вибраний біт DAR і використовує його значення для визначення того, як виконувати команду:

1 в аналізованому розряді DAR - виконання команди,
0 - NOP.

Приклад. ADD B, A, R00, CNDn - виконується аналіз n-ного розряду DAR. Якщо $DAR_n = 1$, то виконується додавання двох операндів, а при DAR = 0 - команда не виконується. Результат заноситься в B. Це еквівалентно наступному оператору:

if (DAR_n <> 0) then B = B + A else NOP.

Команда LDA B, A, R00, CND5 аналізує 5-й розряд DAR і пересилка відбувається в залежності від його значення.

Команда SUB B, A, R, CND_n використовується для умовного віднімання, аналізованим бітом є перенос із попереднього результату АЛП ($CY_{поп}$). В цьому випадку вибраний біт DAR встановлюється рівним переносу із біжучої команди $CY_{біж}$, наприклад:

SUB B, A, R00, CND5.

Тут в залежності від переносу із попереднього результату АЛП ($CY_{поп}$) виконується команда додавання або віднімання, при цьому у відповідності із значенням $CY_{біж}$ в 5-й розряд DAR запишеться 0 або 1.

Таблиця 2.8

Код	Позначення	Аналогова команда	Операція
000	XOR B,A,S	CND_N	$B \oplus A \cdot 2^S \rightarrow B$ – виключне АБО, переповнення дозволено
001	ABA B,A,S	CND_N	$\lvert A \cdot 2^S \rvert + B \rightarrow B$ – додавання з модулем, переповнення заборонено
011	ADD B,A,S	CND_N	$A \cdot 2^S + B \rightarrow B$ – додавання, якщо $DAR_n = 1$ $B \rightarrow B$ - пересилка, якщо $DAR_n = 0$
101	SUB B,A,S	CND_N	$B - (A \cdot 2^S) \rightarrow B$ – віднімання, якщо $CY_{попер.} = 1$, $CY_{біж} \rightarrow DAR_n$ $B + (A \cdot 2^S) \rightarrow B$ – додавання, якщо $CY_{попер.} = 0$, $CY_{біж} \rightarrow DAR_n$
111	LDA B,A,S	CND_N	$A \cdot 2^S \rightarrow B$ - пересилка, якщо $DAR_n = 1$ $B \rightarrow B$ - пересилка, якщо $DAR_n = 0$

Умовне виконання команд ABA, XOR, EOP використовується для керування переповненням (табл.2.9).

Таблиця 2.9

Код	Позначення	Аналогова команда	Операція
001	ABA B, A, S	CND_N , S	$\lvert A \cdot 2^S \rvert + B \rightarrow B$ - переповнення заборонено
000	XOR B,A,S	CND_N , S	$A \cdot 2^S \oplus B \rightarrow B$ – переповнення дозволено
	інші	EOP	переповнення дозволено

2.5.3. Команди переходів

Команда RNZ - повернення на початок програми. Ознака виконання команди RNZ:

XOR 44, 44, R00...

Команда JNZ - перехід на 32 команди вперед. Ознака виконання команди JNZ:

AND 44, 44, R00...

тут 44 - фізично відсутня адреса.
Умовою переходу є аналізований біт:
$DAR_n = 1$ - здійснюється перехід,
$DAR_n = 0$ - виконується програма далі.

Команди RNZ і EOP повинні знаходитись першими з 4-х вибраних: тобто адреси їх діляться на 4. Потім виконуються 3 команди і здійснюється перехід за умовою.

Команда JNZ повинна бути третьою з 4-х вибраних. Після неї виконується 5 команд і потім здійснюється перехід за умовою (на 32 команди вперед).

Невиконання цих умов може привести до неправильного виконання програми. Крім того, якщо команда JNZ розміщена за адресами 32n, n = 0, 6, то перехід не буде відбуватися незалежно від значення CND.

Перевірка DAR може здійснюватись як до виконання цих команд, так і під час їх виконання.

2.5.4. Аналогові команди приведені в табл.2.10.

Таблиця 2.10

Код	Позначення	Операція
00	IN_K, ADK=0...3	підключення входу К
00	NOP, ADK=4	немає операції
00	EOP, ADK=5	повернення в РПЗП в комірку 0
00	CVTS, ADK=6	перетворення знакового розряду
00	CNDS, ADK=7	умовна операція за знаковим розрядом
10	CVT_N, ADK=0...7	аналого-цифрове перетворення n-го розряду
01	OUT_K, ADK=0...7	вивід через канал К
11	CND_N, ADK=0...7	умовна операція за n-м розрядом

Команда INk забезпечує вибірку одного із 4-х аналогових входів. Команду викликають послідовно декілька разів для забезпечення заряду конденсатора СВЗ до необхідної точності.

Команда CVT_N здійснює перетворення вхідного відліку в цифровий еквівалент. Розряди DAR записуються, починаючи із старшого: 8, 7, ...1, 0. Перетворення здійснюється програмно методом послідовних наближень. Команда CVT встановлює

вибраний біт DAR рівним сигналу від компаратора. При цьому для встановлення вихідної напруги ЦАП між кожною парою CVT команд даються дві команди NOP. Час встановлення ЦАП з точністю 8 розрядів складає 1,2 мкс.

Послідовність команд для вводу к-го сигналу подана в табл.2.11.

Таблиця 2.11

N п/п	Аналогова команда
1	IN_K
2	IN_K
...	.... заряд конденсатора СВЗ
8	IN_K
9	NOP
10	CVTS - зважування знакового розряду
11	NOP
12	CVT7 - зважування 7-го розряду
13	NOP
...	.....
23	CVT1 - зважування 1-го розряду
24	NOP
25	CVT0 - зважування 0-го розряду

Команда OUTk забезпечує передачу обчисленого значення із DAR на один із аналогових виходів. Операція не повинна з'являтися в командах, які здійснюють запис в DAR. Для підвищення точності парна команда OUT повинна з'являтися тільки після встановлення вихідного сигналу ЦАП, тобто повинна бути відділена кількома командами NOP від команди запису в DAR (табл.2.12). Час встановлення СВЗ на виході довший за командний цикл, тому команду OUT виконують декілька разів підряд.

Таблиця 2.12

N п/п	Команда АЛП	B	A	Зсув	Аналогова команда
1	LDA	DAR	Y	R00	NOP
2	.....	.....	....	.....	NOP
...	.....	.....	....	.....	....
4	LDA	DAR	Y	R00	NOP
5	.....	.....	....	.....	OUT_K
...	.....	.....	....	.....	....
10	LDA	DAR	Y	R00	OUT_K

Вимоги до числа команд, а також довжина програми аналогового вводу/виводу для різних точностей наведені в табл.2.13.

Таблиця 2.13

Аналогова команда	Час, мкс	Число команд		
		Тактова частота, МГц		
		1	5	10
IN_K	2,4	1	6	8
NOP між перетвореннями	0,56	1	1	2
NOP після IN_K		1	1	1
NOP після LDA DAR Y	2,4	1	3	6
OUT_K	3,6	1	5	9

2.5.5. Послідовний цифровий ввід/вивід

При значеннях М1=+5В і М2=–5В виводи АС0...АС3 переходять в режим послідовного вводу/виводу; Вих0...Вих7 – в ТТЛ-режим.

Послідовний ввід. По команді CVT_n в n-біт DAR з виводу АС0 інверсно вводяться дані, одночасно за час команди CVT_n на вивід АС2 подається синхроімпульс (нульовий рівень). Порядок слідування CVT_n – від старшого до молодшого біта (n = s, 7, 6,...1,0).

Послідовний вивід. По команді OUT_n із n-ного біта DAR на вивід АС3 інверсно виводяться дані, одночасно протягом команди OUT_n на вивід АС1 подається синхроімпульс. Порядок слідування OUT_n – будь-який.

2.6. Програмування функцій типових вузлів пристроїв керування на ВЕ1

Реалізація системи автоматичного реґулювання методами ЦОС зводиться до виконання послідовності обчислювальних і логічних операторів, які визначаються заданим співвідношенням вхід/вихід. Цей алгоритм реалізується у вигляді програми, яка відповідає найпростішим операціям (додавання, віднімання, пересилка, зсув і т.д.) і включає часто використовувані фраґменти, які відповідають більш складним операціям.

Далі розглянуті найбільш важливі із цих фраґментів і приклади їх реалізації.

2.6.1. Множення на константу і змінну

В системі команд мікро-ЕОМ ВЕ1 множення виконується програмно за алгоритмом Бута з використовуванням операцій зсуву. В даному алгоритмі бажано мінімізувати число одиниць у представленні константи, тобто для множення використовується знакорозрядна система числення [1].

Нехай **змінна x множиться на константу** c=0,4726562, тобто виконується операція y=c·x. В двійковому і знакорозрядному представленні с має вигляд c=0,01111001=0,1000-1001.

Останній код дає $c = 2^{-1} - 2^{-5} + 2^{-8}$.

Тоді маємо $y = x \cdot 2^{-1} - x \cdot 2^{-5} + x \cdot 2^{-8}$, що реалізується програмно:

```
ADD Y X R01
SUB Y X R05
ADD Y X R08
```

Попередньо в комірку Y необхідно занести нульові значення XOR Y Y R00 - (сума за модулем 2 числа самим з собою дає нульове значення).

Множення на змінну. Операція y=x·z залежить від необхідної точності результату. Вона виконується з використанням команд умовного додавання шляхом послідовного аналізу розрядів множника. Якщо даний розряд 1, то виконується додавання зсунутого відповідного значення, при 0 – додавання відсутнє.

Програма множення для 8-розрядної точності (комірка Y попередньо очищена):

```
LDA DAR X R00 ; завантаження множника в DAR
ADD Y R01 CND7 ; перемноження кодів
ADD Y Z R02 CND6
- - - - - - - - - - - - - - - - - - -
ADD Y Z R08 CND0
SUB Z Z L01
ADD Y Z R00 CNDS ; визначення знака результату.
```

2.6.2. Ділення на константу

Ділення на константу відбувається шляхом множення змінної на величину, зворотну константі.

2.6.3. Ділення змінних

Операція y=x/z. Якщо обидві змінні можуть приймати додатні і від'ємні значення, то спочатку з допомогою команди XOR визначають знак частки. Подальші операції виконуються над абсолютними значеннями величин. Операція ділення полягає в послідовному умовному відніманні дільника від діленого, а частка отримується в регістрі DAR. Виконання операцій починається із безумовного віднімання. А тому що результат ділення - частка - завжди повинна залишатися меншою від одиниці, то дільник попередньо масштабується так, щоб найменше його значення завжди перевищувало значення діленого. Тоді результат першого віднімання буде завжди від'ємним.

Програма:

```
LDA T Z R13 ; в комірку T заноситься знак частки
XOR T Z R13
ABS X X R00
ABS Z Z R00 ; виділені абсолютні значення змінних
SUB X Z R00 ; безумовне віднімання x
SUB X Z R01 CND7 ; отримання значущих цифр частки в
                   DAR
SUB X Z R02 CND6
- - - - - - - - - - - -
SUB X Z R08 CND0
XOR DAR T R13 ; відновлення знаку частки в DAR
```

Результат має точність 8 розрядів плюс знак. Якщо необхідна вища точність, то DAR запам'ятовується, потім обнулюється, відновлюється значення переносу, і далі умовне віднімання продовжується з накопиченням в DAR молодших розрядів частки. Відновлення переносу, який дорівнює доповненню знака часткового залишку, виконується шляхом додавання і наступного віднімання дільника (відповідно зсунутого) із залишку частки.

2.6.4. Інтеґратор

Інтеґратор описується рівнянням $y = b_0 x_k + a_1 y_{k-1}$. Передаточна функція такого кола буде $T(z) = b_0/(1-a_1 z^{-1})$ і відповідає інтеґратору або фільтру нижніх частот (рис. 2.5). Приймемо, що $a_1 = 1-2^{-5}$. Тоді b_0 як нормуючий коефіцієнт повинен бути 2^{-5}, щоб у системі не виникало переповнень при $|x_k| < 1$.

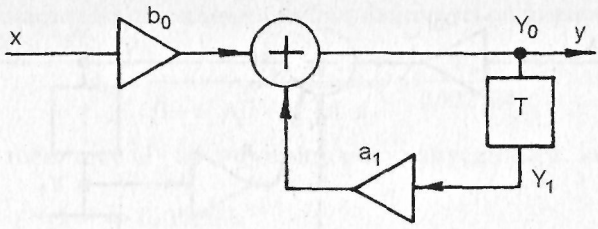


Рис. 2.5. Структурна схема інтеґратора

Програма, яка реалізує дану передаточну характеристику, наступна:

LDA Y1 Y0 R00 ; реалізація затримки
LDA Y0 Y1 R00 ; $y_0=y_1$
SUB Y0 Y1 R05 ; $y_0 = (1-2^{-5})\cdot y_1 = a_1\cdot y_1$
ADD Y0 X R05 ; $y_0 = a_1\cdot y_1+b_0\cdot x$.

Постійна часу такого інтеґратора $t=-T/\ln(a_1)$ і складає приблизно 31T.

На відміну від аналогового інтеґратора цифрова система не вносить втрат, не вимагає узгодження з навантаженням, має стабільні характеристики, постійна часу може бути встановлена в сотні тисяч інтервалів дискретизації. Початкове обнулення інтеґратора здійснюється за один такт.

Якщо коефіцієнти різницевого рівняння вибрати змінними, то можна реалізувати цифрове усереднення вхідних відліків за наступним алгоритмом:

$$y_n = 1/n\cdot x_n + [(n-1)/n]\cdot y_{n-1}.$$

2.6.5. Цифровий резонатор

Проектування цифрового резонатора розглянемо на прикладі. Задамо необхідні значення:

частота дискретизації $f_д = 13{,}02$кГц;
середня частота $f_0 = 1$кГц $\pm$ 0,5%;
добротність $75 \le Q \le 100$;
максимальне підсилення $G_m = 1{,}0 \pm 10\%$.

Структурна схема, яка реалізує цифровий резонатор показана на рис.2.6.

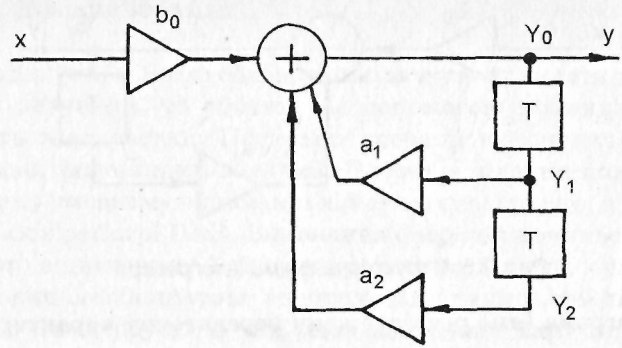


Рис. 2.6. Структурна схема цифрового резонатора

Коефіцієнти виражаються наступними значеннями:
$a_2 = -e^{-2\tau T}$; $a_1 = 2 \cdot e^{-2\tau T} \cdot \cos\omega_0 T$,
де $\tau = \omega_0 / 2Q$ - коефіцієнт затухання,
$\omega_0 = 2 \cdot \pi \cdot f_0 = 6280$ рад/с - резонансна частота,
$T = 1/f_д = 76{,}8$мс - інтервал дискретизації.

З урахуванням допусків на характеристики резонатора отримуємо:

$$\omega T = 0{,}48255 \pm 0{,}0024;\ 0{,}002412 \le \tau T \le 0{,}003217.$$

Виражаємо число a_2 в двійковому коді, маємо:

$$0{,}111111001011 \le -a_2 \le 0{,}111111101100.$$

Представимо його як:

$$-a_2 = 0{,}111111101 = 2^0 - 2^{-7} + 2^{-9} = 0{,}99414.$$

Для a_1 відповідно маємо:

$$1{,}1100010011 \le a_1 \le 1{,}1100001110.$$

Найбільше зручне для реалізації значення

$$a_1 = 1{,}110001 = 2^1 - 2^{-2} + 2^{-6} = 1{,}7656.$$

Для вибраних значень коефіцієтів $f_0 = 1001{,}8$ Гц, $Q = 82$.

Максимальне підсилення кола визначається виразом:

$$G_m = \frac{b_0}{(1+a_2)\sqrt{1+a_1^2/4\cdot a_2}} = \frac{b_0}{0{,}002724}$$

Для того, щоб G_m не перевищувало допустимого, вибираємо

$b_0 = 2^{-8}\text{-}2^{-10} = 0{,}00293.$

Програма, яка реалізує резонатор:

```
LDA Y2 Y1 R00 ; реалізація затримок
LDA Y1 Y0 R00
LDA Y0 Y1 L01
SUB Y0 Y1 R02
ADD Y0 Y1 R06 ; y0 =a1·y1

SUB Y0 Y2 R00
ADD Y0 Y2 R07
SUB Y0 Y2 R09 ; y0 = a1·y1 + a2·y2

ADD Y0 X R08
SUB Y0 X R10 ; y0 = a1·y1 + a2·y2 + b0·x.
```

Послідовність операцій необхідно вибирати таким чином, щоб команди ADD і SUB чергувалися. При цьому зменшується можливість переповнення проміжних результатів.

2.6.6. Обмежувачі сигналів

Командою LIM реалізується ідеальний пороговий логічний елемент: навіть дуже маленький сигнал дає повний додатний або від'ємний вихідний сигнал. Реалізація обмежувача за таким алгоритмом:

```
LIM Y00 X R00;
```

$$y = \begin{cases} +1, x \geq 0; \\ -1, x < 0. \end{cases}$$

Характеристика обмежувача має вигляд (рис.2.7)

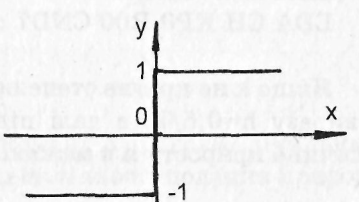


Рис. 2.7. Характеристика обмежувача

Якщо необхідні обмежувачі з порогом, то вони реалізуються зсувом вхідної величини:

```
LDA Y00 X L02 ; y = 4·x
ADD Y00 Y00 L02 ; y = y+4·y = 4·x+16·x = 20·x
```

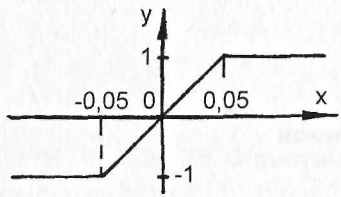


Рис. 2.8. Характеристика обмежувача з порогом

В такому обмежувачі сиґнал $x < 0,05$ не викликає переповнення (рис.2.8)

Ще один тип обмежувача описується виразом $y = |x-a| - |x+a|$, де a – поріг обмеження. Характеристика його відповідає попередній, але забезпечує більший діапазон значень сиґналів.

Програма такого обмежувача:

```
LDA C Y R00 ; c=x
SUB C A R00 ; c=x-a
ABS Y C R00 ; y=|x-a|
ADD C A L01 ; c=x+a
ABS C C R00 ; c=|x+a|
SUB Y C R00 ; y=|x-a| -|x+a|.
```

2.6.7. Таймери і лічильники

При реалізації лічильників з коефіцієнтом перерахунку $k=2^n$, приріст вибирається 2^{-n}. Такий приріст може бути здійснений шляхом зсуву константи КР4 вправо на n розрядів. Наприклад, для k=128, знаходимо що n=7; тоді програма має вигляд:

```
ADD CH KP4 R07
LDA DAR CH R00
LDA CH KP0 R00 CND7 ; обнулення лічильника.
```

Якщо k не кратне степеню числа 2, то приріст визначається із виразу h=0,5/k, а далі підбирається зручна для реалізації величина приросту h в межах:

$$(2-k^{-1})/\ 4\cdot k \le\ h\ \le (2+k^{-1})/\ 4\cdot k.$$

Значення приростів для лічильників з k=2...10 і фрагменти їх програмної реалізації подані в таблиці 2.14.

Таблиця 2.14

K	h	Зручне h для реалізації	Двійкове значення h	Фрагмент програми
2	0,2500	$0,25 \cdot 20$	0,01000000	ADD CH KP4 R01
3	0,1660	$0,375 \cdot 2^{-1}$	0,00110000	ADD CH KP3 R01
4	0,1250	$0,5 \cdot 2^{-2}$	0,00100000	ADD CH KP4 R02
5	0,1000	$0,875 \cdot 2^{-3}$	0,00011100	ADD CH KP7 R03
6	0,0833	$0,750 \cdot 2^{-3}$	0,00011000	ADD CH KP6 R03
7	0,0714	$0,625 \cdot 2^{-3}$	0,00010100	ADD CH KP5 R03
8	0,0625	$0,5 \cdot 2^{-3}$	0,00010000	ADD CH KP4 R08
9	0,0555	$0,5 \cdot 2^{-8}+0,875 \cdot 2^{-4}$	0,00001111	ADD CH KP7 R04
10	0,05	$0,875 \cdot 2^{-4}$	0,00001110	ADD CH KP7 R04

Можлива реалізація лічильника через віднімання, в цьому випадку k визначається значенням початкової константи, яка заноситься в лічильник.

Лічильник на 17 через віднімання:

```
SUB CH KP4 R04
LDA DAR CH R00
LDA CH KP4 R00 CNDS
```

Лічильник на 9 через додавання:

```
ADD CH KP4 R08
ADD CH KP7 R04
LDA DAR CH R00
LDA CH KP0 R00 CND7
```

Лічильник на 32768 через додавання:

```
ADD CH KP1 R13
LDA DAR CH R00
LDA CH KP0 R00 CND7.
```

В залежності від k кількість команд в програмі лічильника буде змінюватися. Це залежить від складності подання приросту h.

Програмування таймерів здійснюється за тими ж правилами, що і лічильників, але при цьому враховується час виконання одного кроку всієї програми обробки. Цей час розраховується за формулою:

$$T = 4 \cdot N / f_т,$$

де N - число команд в програмі, $f_т$ - тактова частота.

Для К1813ВЕ1А при $f_т$ = 6,67МГц T_{min} = 115,2мкс при N = 192.

Якщо необхідна реалізація таймера з іншим часом, то попередньо розраховується кількість прогонів програми, а потім здійснюється підбір кроку таймера аналогічно, як це робилось при реалізації лічильників (табл.2.15).

Таблиця 2.15

Фрагмент підпрограми	Інтервал таймера
ADD TM KP1 R11	943мс
ADD TM KP1 R12	1с 887мс
ADD TM KP1 R13	3с 774мс
LDA X KP1 R13 ADD TM X R02	7с 549мс
LDA X KP1 R13 ADD TM X R03	15с 099мс
LDA X KP1 R13 ADD TM X R03	30с 198мс
LDA X KP1 R13 ADD TM X R04	1хв 10с 780мс

2.6.8. Ґенератори

2.6.8.1. Ґенератор пилоподібної напруги реалізується програмою на основі співвідношення $y_{k+1}=k_1+y_k$; при умові, що при досягненні у заданої величини А0 ґенератор повертається в початковий стан. Для цього використовується цифрова команда АВА і аналогова CND_n. При досягненні переповнення (у=+1) змінній присвоюється мінімальне значення (-1). Таким чином

програма ґенератора пилоподібної напруги (рис.2.9) реалізується однією командою:

ABA Y0 KP1 R00 CND_n.

Тут n - будь-яке.

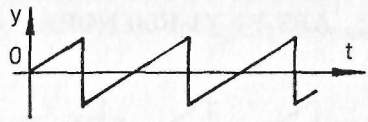


Рис. 2.9. Форма пилоподібної напруги

Для ґенерації пилоподібних сиґналів іншого виду (рис.2.10) програма ускладнюється.

Для сиґналу вигляду б) маємо програму:

SUB Y0 KP1 R00 NOP ; відняти від Y0 константу KP1;

LDA DAR Y0 R00 NOP ; переслати Y0 в DAR для аналізу знака результату;

ADD Y0 KP4 L01 CNDS ; додати до Y0 1, якщо Y0 < 0.

Змінюючи операцію додавання, початкові умови, можна реалізувати ґенератори з формою пилоподібної напруги а - г.

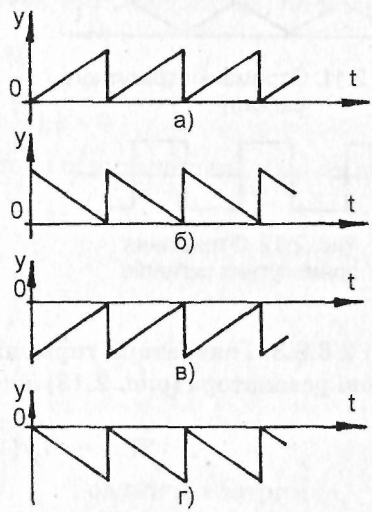


Рис. 2.10. Варіанти форм пилоподібної напруги

Частота ґенератора

$f = K_1 f_д / K_2$,

де K_1 - приріст у за час виконання програми,

K_2 - коефіцієнт, що визначає амплітуду (K_2=1).

Якщо величина K_1 є функцією часу, то частота ґенератора буде змінюватися у відповідності з законом зміни K_1.

2.6.8.2. Ґенератори трикутних і прямокутних імпульсів

Двопівперіодне випрямлення пилоподібної напруги дає трикутний сиґнал (рис.2.11).

Таке випрямлення реалізується однією командою ABS за програмою:

```
ABA Y0 KP1 R00 CND7 ; ґенерація пилоподібного сиґналу;
LDA Y1 Y0 R00 NOP
ABS Y1 Y1 R00 NOP   ; ґенерування трикутного сиґналу.
```

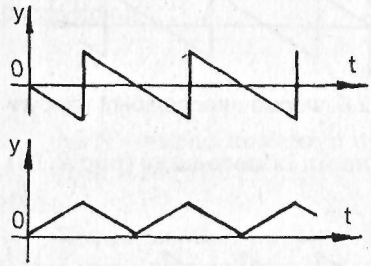


Рис. 2.11. Отримання трикутного сиґналу

Із пилоподібної напруги при допомозі команди LIM можна отримати прямокутні коливання (рис.2.12).

Відповідна програма:

```
ABA Y0 KP1 R00 CND3
LDA Y1 Y0 R00 NOP
LIM Y1 Y1 R00 NOP.
```

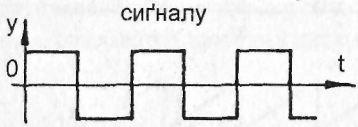


Рис. 2.12. Отримання прямокутних сиґналів

2.6.8.3. Ґенератори гармонічних коливань реалізуються на основі резонатора (рис. 2.13) з передаточною функцією:

$$T(z) = b_0/(1-a_1z^{-1}-a_2z^{-2}).$$

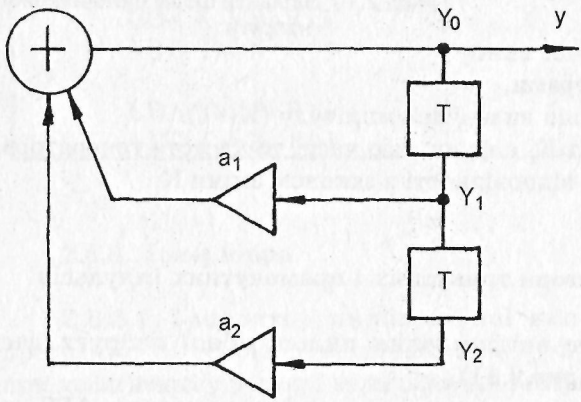


Рис. 2.13. Ґенератор гармонічних коливань на основі резонатора

При $a_2=-1$ добротність резонатора стає нескінченною і резонатор переходить в режим ґенератора гармонічних коливань. Амплітуда і початкова фаза реакції резонатора на одиничний скачок залежить від початкових умов, а частота залежить від a_1:

$$a_1 = 2\cos(2\pi f_г/f_д).$$

Тому реалізація ґенератора зводиться до програмування різницевого рівняння:

$$y_k = a_1 y_{k-1} - y_{k-2} + b_0 x_k .$$

Оскільки в процесорі існують шуми округлення і можуть виникати ефекти переповнення, періодично необхідно відновлювати початкові умови ґенерації, наприклад, по таймеру. Значення

$$x_k = \begin{cases} 0, k \neq 0; \\ 1, k = 0. \end{cases}$$

Через певний час змінні ґенератора обнулюються і на вхід знову подається одиничний скачок.

Розподіл пам'яті:

Y00 - y_k
Y01 - y_{k-1}
Y02 - y_{k-2}
Y03 - таймер.
Нехай $a_1 = 0{,}125$; $b_0 = 1$, тоді:

```
LDA Y02 Y01 R00          ; одинична затримка,
LDA Y01 Y00 R00
ADD Y00 Y01 R01          ; множення на 0,125,
SUB Y00 Y02 R00          ; множення на -1,
SUB Y03 KP1 R06          ; програма таймера,
LDA DAR Y03 R00
ADD Y03 KP4 L01 CNDS
LDA Y01 KP0 R00 CNDS     ; обнулення Y01,
LDA Y01 KP4 L01 CNDS     ; одинична дія на резонатор.
```

Якщо період таймера вибраний таким чином, що в момент заміни змінних на початкові умови вони відрізняються на малу величину, то відновлення ґенерації відбудеться без розриву фази.

2.6.8.4. Ґенератор лінійно-змінної частоти використовується в САК для періодичної зміни параметрів системи. Частота вихідного синусоїдального сиґналу цього ґенератора змінюється за лінійним законом (рис.2.14).

Його можна представити як послідовне з'єднання ґенератора, що формує пилоподібну напругу F2 в межах від S2 до S2 + M/4, і ґенератора, що формує синусоїдальний сиґнал, частота якого відповідає значенню F2.

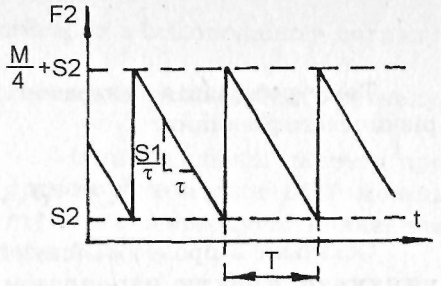


Рис. 2.14. Закон зміни частоти

Пилоподібна форма сиґналу F2 найпростіше формується неперервним декрементуванням значення реґістра. При цьому формується лінійно-спадаюча напруга, зміна якої S1 повинна бути незначною за час формування одного періоду синусоїдального сиґналу τ. Для прикладу вибрано період зміни частоти ґенератора пилоподібної напруги від мінімального значення до максимального значення $T^{-1} = 2$ Гц, при мінімальній частоті вихідного синусоїдального сиґналу $\tau^{-1} = 13$ кГц.

Програма ґенератора пилоподібної напруги використовує дві константи KMx і KPx. При реалізації зовнішньої зміни частоти ґенератора необхідно задати цю зміну у вигляді аналогового сиґналу, виконати аналого-цифрове перетворення і занести відповідні значення в S1 і S2. Варіант програми, що використовує додавання і зсуви констант, наведено нижче. Величина S2 задає рівень, при якому значення F2 замінюється на амплітуду пилоподібної напруги.

```
SUB F1 S1 R00          ; декремент реґістра;
LDA DAR F1 R00         ; завантаження DAR;
ADD F1 M R00 CNDS      ; умовне додавання амплітуди;
LDA F2 F1 R02          ; формування нахилу;
ADD F2 S2 R00          ; додавання кроку.
```

Реалізуємо синусоїдальний ґенератор лінійно-змінної частоти із такими параметрами (рис.2.14):

$\tau = 70{,}8$ мкс (13кГц);

M = 1B;

T = 0,5c.

Тоді маємо $S1 = M/(T/\tau) = 15.36 \cdot 10^{-6}$В/крок.

Перетворимо S1 у двійкову форму

$S1 = 15{,}36 \cdot 10^{-6} = 2^{-12} + 2^{-15} + 2^{-20} = (2^{-1} + 2^{-3} + 2^{-8}) \cdot 2^{-12} =$

$= (0.10100001) \cdot 2^{-12}$.

Видно, що S1 найзручніше скласти із двох 4-розрядних слів: $S1 = [0.101 + 0.001 \cdot 2^{-5}] \cdot 2^{-12}$.

Використовуючи константи і зсуви $(KP5 + KP1 \cdot 2^{-5}) \cdot 2^{-12}$, маємо

```
LDA S1 KP5 R00
ADD S1 KP1 R05
LDA S1 S1 R12
```

Похибку меншу за 1% дає величина KP1·R05. Тому два кроки попередньої програми можуть бути замінені однією командою

```
LDA S1 KP5 R12.
```

Формування синусоїдального сигналу здійснюється на основі кусочно-лінійної апроксимації (рис.2.15).

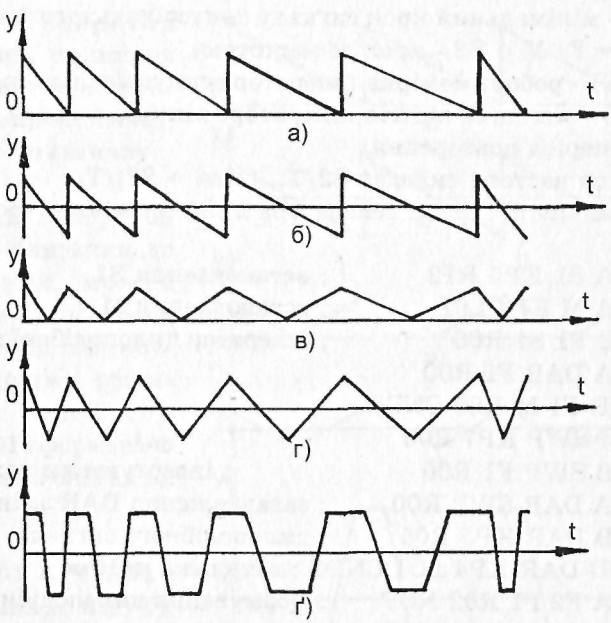


Рис. 2.15. Формування синусоїдального сигналу ґенератора лінійно-змінної частоти

Програма формування синусоїдального сиґналу.

```
SUB OSC1 F2 R00          ; формування пилоподібного сиґналу
LDA DAR OSC1 R00           (рис.2.15а)
ADD OSC1 M R00 CNDS
LDA OSC OSC1 R00
SUB OSC M R01            ; центрування сиґналу (рис.2.15б)
ABS OSC OSC L01          ; підсилення в два рази і
                           випрямлення (рис.2.15в)
SUB OSC M R01            ; центрування відносно нуля
                           (рис.2.15г)
ADD OSC OSC L01          ; множення на три з обмеженням
                           (рис.2.15ґ)
```

Загальна програма. В цій програмі:

F1 - початкова зміна кроку ґенератора синусоїдального сиґналу;

M - амплітуда, менша 1 В;

S2 - мінімальний крок сиґналу синусоїдального ґенератора;

F2 = F1·M + S2 - крок за частотою;

OSC1 - робоча комірка ґенератора синусоїдального сиґналу;

OSC - вихідний сиґнал ґенератора лінійно-змінної частоти;

T - період повторення.

Зміна частоти складає S2/T...(M/4 + S2)/T.

Загальна програма ґенератора лінійно-змінної частоти має вигляд:

```
LDA S1 KP5 R12           ; встановлення S1
LDA M KP4 L01            ; встановлення M
SUB F1 S1 R00            ; ґенератор пилоподібної напруги
LDA DAR F1 R00
ADD F1 M R00 CNDS
LIM SWP KP7 R00
SUB SWP F1 R00                  ; інвертування нахилу
LDA DAR SWP R00          ; завантаження DAR змінною
SUB DAR KP5 R05            пилоподібного сиґналу
ADD DAR KP4 LO1 CNDS ; затримка на10мс
LDA F2 F1 R02 NOP        ; формування пилоподібного
SUB F2 F1 R06 NOP          сиґналу
SUB F2 F2 R12 NOP
SUB F2 F1 R08 NOP
NOP
NOP
```

```
LDA S2 KP3 R02 OUTO  ; встановлення S2
ADD S2 KP3 R06 OUTO
ADD S2 KP1 R09 OUTO
ADD F2 S2 R00 OUTO   ; додавання кроку
OUTO
SUB OSC1 F2 R00 OUTO  ; ґенератор синусоїдального сиґналу
LDA DAR OSC1 R00
ADD OSC1 M R00 CNDS
LDA OSC OSC1 R00
SUB OSC M R01
ABS OSC OSC L01
SUB OSC M R01
LDA DAR F1 R00
LDA OSC1 KP0 R00 CNDS; встановлення ґенератора в
                      нульовий стан
ADD OSC OSC L01       ; формування вихідного сиґналу -
                        OSC.
```

2.6.8.5. Ґенератор гармонічних коливань на основі кусочно-лінійної апроксимації гармонічного сиґналу

Стійкий ґенератор коливань, близьких до гармонічних, можна отримати шляхом перетворення пилоподібної напруги в трапецеїдальну (рис.2.16). Якщо в Y01 сформовано пилоподібний сиґнал а), то

```
SUB Y01 KP4 R01 ; перетворення до виду б),
ABS Y01 Y01 L01 ; перетворення до виду в),
SUB Y01 KP4 R00 ; перетворення до виду г),
```

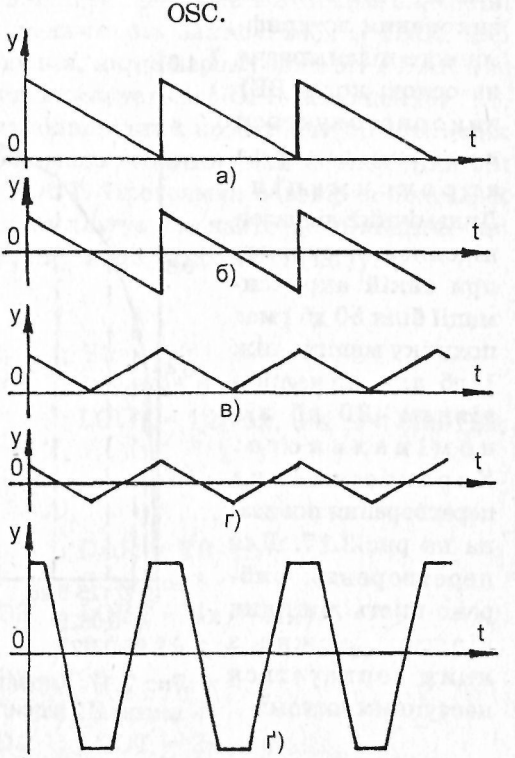


Рис. 2.16. Ґенератор гармонічних коливань на основі кусочно-лінійної апроксимації

ADD Y01 Y01 L01 ; перетворення до виду ґ).

В отриманому сиґналі відсутні парні гармоніки, а відношення амплітуди n-ої непарної гармоніки до основної дорівнює:

$$P_n = \sin(n\pi/3)/(n^2\pi/3).$$

Керувати частотою такого ґенератора можна зміною приросту K_1 в ґенераторі пилоподібної напруги. При цьому зміна частоти відбувається без розриву фази.

2.7. Логарифмічний підсилювач має функцію передачі, яка часто використовується в САК: підсилення малих рівнів сиґналу здійснюється з більшим коефіцієнтом передачі, ніж великих, тобто реалізується стиснення діапазону зміни сиґналу. Для програмного виконання логарифмічного підсилювача на основі кодів ВЕ1 використовується кусочно-лінійна апроксимація. Динамічний діапазон підсилення складає при такій апроксимації біля 50 дб і має похибку меншу, ніж 1 дб для сиґналів з рівнем -30 дб від номінального. Характеристика перетворення показана на рис.2.17. Для перетворення вибрано шість лінійних ділянок, кожна з яких описується наступним чином:

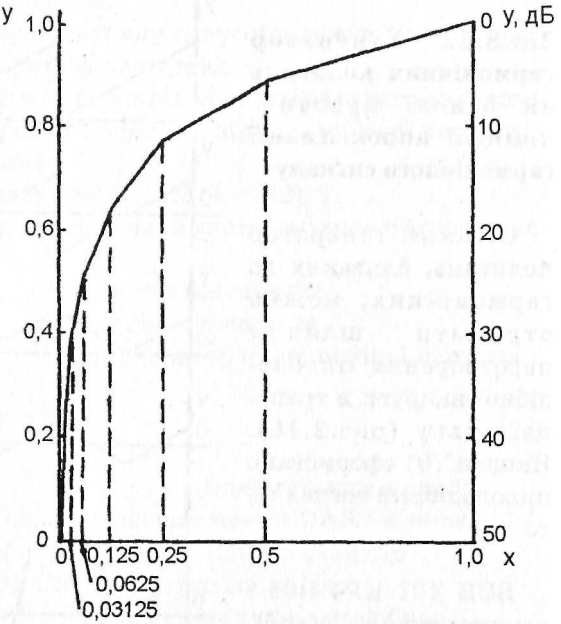


Рис. 2.17. Характеристика логарифмічного підсилювача

$$y = \begin{cases} 0.219x + 0.781, 0.5 \le x < 1; \\ 0.5x + 0.641, 0.25 \le x < 0.5; \\ x + 0.516, 0.125 \le x < 0.25; \\ 2x + 0.391, 0.0625 \le x < 0.125; \\ 4x + 0.27, 0.03125 \le x < 0.0625; \\ 12.75x, 0 \le x < 0.03125. \end{cases}$$

Вхідний і вихідний сигнали такого підсилювача додатні і повинні бути менші від номінального значення 1В.

Вихід 1В відповідає 0 дб;

0.8В - -10 дб;

0.6В - -20 дб і т.д.

Перша лінійна ділянка підсилювача відповідає вхідному сигналу меншому, ніж 1/32В. Тому всі вхідні сигнали меншої амплітуди будуть відповідно підсилюватися. Початкове значення сигналу за абсолютною величиною записується в DAR. Всі наступні операції виконуються, якщо перевірений біт в DAR має одиничне значення, інакше виконується NOP. Якщо цей біт "1", виконується множення на відповідний коефіцієнт і обчислення вихідного результату. Обчислена величина для кожної ділянки пересилається в комірку LOUT. Програмна перевірка біта DAR виконується від менших амплітуд сигналів до більших, що відповідає заданому діапазону зміни вхідного сигналу.

Програма:

```
ABS Y0 Y0 R00              ; Y0 = |Y0|
LDA LOUT Y0 L02            ; Ділянка 6
ADD LOUT Y0 L02            ; LOUT = 12.75x, 0 ≤ x < 0.03125;
ADD LOUT Y0 L02
ADD LOUT Y0 R01
ADD LOUT Y0 R02
LDA DAR Y0 R00             ; DAR = Y0
LDA LOUT Y0 L02 CND3       ; Ділянка 5
ADD LOUT KP2 R00 CND3      ; LOUT = 4x + 0.27,
                             0.03125 ≤ x < 0.0625;
ADD LOUT KP5 R05 CND3
LDA LOUT Y0 L01 CND4       ; Ділянка 4
ADD LOUT KP3 R00 CND4      ; LOUT = 2x + 0.391,
                             0.0625 ≤ x < 0.125;
```

```
ADD LOUT KP2 R04 CND4
LDA LOUT Y0 R00 CND5     ; Ділянка 3
ADD LOUT KP4 R00 CND5    ; LOUT = x + 0.516,
                                  0.125 ≤ x < 0.25;
ADD LOUT KP2 R04 CND5
LDA LOUT Y0 R01 CND6     ; Ділянка 2
ADD LOUT KP5 R00 CND6    ; LOUT = 0.5x + 0.641,
                           0.25 ≤ x < 0.5;
ADD LOUT KP2 R04 CND6
LDA LOUT Y0 R03 CND7     ; Ділянка 1
ADD LOUT Y0 R04 CND7     ; LOUT = 0.219x + 0.781,
                           0.5 ≤ x < 1;
ADD LOUT Y0 R05 CND7
ADD LOUT KP6 R00 CND7
ADD LOUT KP4 R04 CND7
LDA DAR LOUT R00         ; Пересилка результату в DAR.
```

2.8. Програмна реалізація алгоритму швидкого перетворення Фур'є

Дискретне перетворення Фур'є записується виразом:

$$X(k) = \sum_{n=0}^{N-1} x_n \cdot e^{-j\cdot 2\pi kn/N}; k = 0, N-1.$$

Основою «швидких» алгоритмів перетворення Фур'є є операція «метелик», графічне зображення якої має вигляд (рис.2.18).

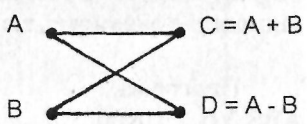


Рис. 2.18. Базова операція ШПФ "метелик"

Програмна реалізація такої операції має вигляд:

```
LDA Y01 A R00    ; зберігання значення комірки A
SUB A B R00      ; A = A - B
ADD B Y01 R00    ; B = A + B
```

Обмеженням в даній мікро-ЕОМ є число команд для обробки і вводу значень. Тому з використанням аналогових входів і виходів можлива реалізація лише «скануючих» алгоритмів ШПФ (що обумовлено довжиною вводу - 25 команд на відлік). При використанні цифрового вводу/виводу ввід здійснюється тільки

за 10 команд. Це дозволяє реалізувати на ВЕ1 блочне ШПФ розміру N=8...16, 32. Приклад програмної реалізації такого ШПФ для N=8 за алгоритмом Винограда [11] наведений в [36].

2.9. Програмна реалізація тригонометричних функцій sin(x), cos(x) на ВЕ1 раціонально виконується на основі розкладу за поліномами Чебишева - при заданій точності обчислень вони мають найменший порядок. Похибка апроксимації $\Delta m = f(n, b)$ подана на рис.2.19. Допустима загальна похибка визначається подальшим використанням отриманого результату. Якщо результат використовується для обчислень всередині мікро-ЕОМ, то точність може бути 24 розряди - $6 \cdot 10^{-8}$. Якщо результат виводиться через ЦАП, то 9 розрядів відповідає точності $4 \cdot 10^{-3}$. Тому використання ЦАП для виводу результату визначає межу, вище якої степінь полінома збільшувати нераціонально.

Програмування тригонометричних функцій зводиться до:

1. Визначення степеня полінома за заданою точністю апроксимації;
2. Перетворення коефіцієнтів полінома із десяткової системи в двійкову;
3. Складання програми в кодах мікро-ЕОМ.

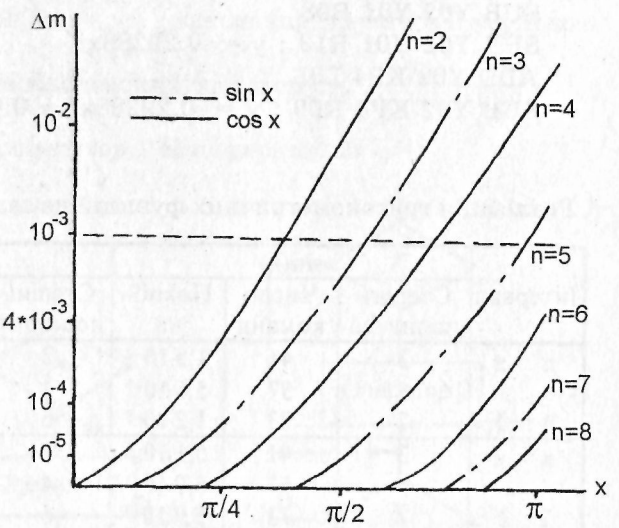


Рис. 2.19. Похибка апроксимації

Приклад.

Програма для обчислення cos(x) на інтервалі $-\pi/4 \ldots \pi/4$ з точністю $\Delta m = 10^{-3}$. За графіком знаходимо степінь полінома (він дорівнює 2):

$$y = -0.29288x^2 + 0.99807.$$

Перетворення коефіцієнтів полінома дає:

0.29288 = 0.0100101100001B,
0.99807 = 0.1111111110000B.

Програма:

```
SUB Y01 Y01 R00 ; обнулення комірки Y1
LDA DAR Y02 R00
ADD Y01 Y02 R01 CND7
.....
ADD Y01 Y02 R08 CND0
ABS Y01 Y01 R00 ; y = x²
SUB Y02 Y01 R02
SUB Y02 Y01 R05
SUB Y02 Y01 R07
SUB Y02 Y01 R08
SUB Y02 Y01 R13 ; y = -0.29288x²
ADD Y02 KP4 L01
SUB Y02 KP4 R09 ; y = -0.29288x² + 0.99807.
```

Таблиця 2.16

Реалізація тригонометричних функцій на заданому інтервалі

Інтервал	sin(x)			cos(x)		
	Степінь полінома	Число команд	Похибка	Степінь полінома	Число команд	Похибка
$-\frac{\pi}{4}...\frac{\pi}{4}$	3	41	$1{,}5\cdot10^{-4}$	2	22	$1{,}9\cdot10^{-3}$
	5	57	$5{,}6\cdot10^{-7}$	4	38	$9{,}9\cdot10^{-6}$
	7	73	$1{,}2\cdot10^{-9}$	6	53	$2{,}7\cdot10^{-8}$
$-\frac{\pi}{2}...\frac{\pi}{2}$	3	41	$5{,}1\cdot10^{-3}$	2	22	$2{,}8\cdot10^{-2}$
	5	57	$6{,}7\cdot10^{-5}$	4	38	$5{,}9\cdot10^{-4}$
	7	73	$5{,}9\cdot10^{-7}$	6	53	$6{,}7\cdot10^{-6}$
	9	89	$3{,}0\cdot10^{-9}$	8	68	$4{,}6\cdot10^{-8}$
$-\pi,...,\pi$	3	41	$1{,}1\cdot10^{-1}$	2	22	$3{,}0\cdot10^{-1}$
	5	57	$6{,}8\cdot10^{-3}$	4	38	$2{,}9\cdot10^{-2}$
	7	73	$2{,}5\cdot10^{-4}$	6	53	$1{,}4\cdot10^{-3}$
	9	89	$5{,}8\cdot10^{-6}$	8	68	$4{,}0\cdot10^{-5}$
	11	115	$9{,}0\cdot10^{-8}$	10	83	$7{,}8\cdot10^{-7}$

2.10. Реалізація умовних переходів

В даній мікро-ЕОМ відсутні звичайні операції переходів, крім того, реалізована лише лінійна структура програми, коли почергово виконуються всі команди програми, тобто програма є завжди постійної довжини. Тому для реалізації умовних переходів необхідно модифікувати програму з розгалуженнями так, щоб отримати лінійну програму.

Варіант 1 (рис.2.20 і 2.21).

```
LDA DAR A R00          ; занесення умов в DAR
Оператор 2 - CND_n     ; виконання оператора 1 за умовою
LDA Y00 KP0 R00        ; обнулення робочої комірки
SUB Y00 DAR R00        ; інвертування умови
LDA DAR Y00 R00        ; запис інвертованої умови в DAR
Оператор 1 CND_n       ; виконання оператора 2 за умовою.
```

Цей варіант еквівалентний оператору:

if умова then оператор 1 else оператор 2.

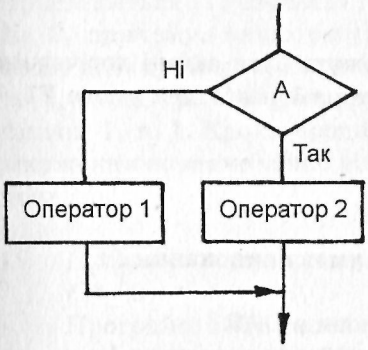


Рис. 2.20. Задання умов

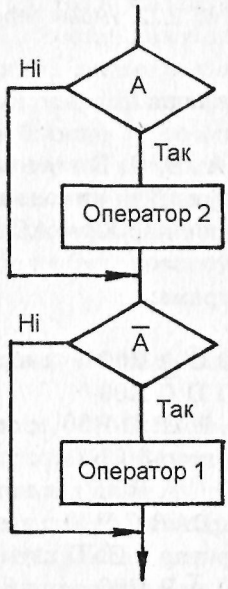


Рис. 2.21. Реалізація умовних переходів

Варіант 2. Цей варіант для складних алгоритмів обробки з великою кількістю умов (рис.2.22).

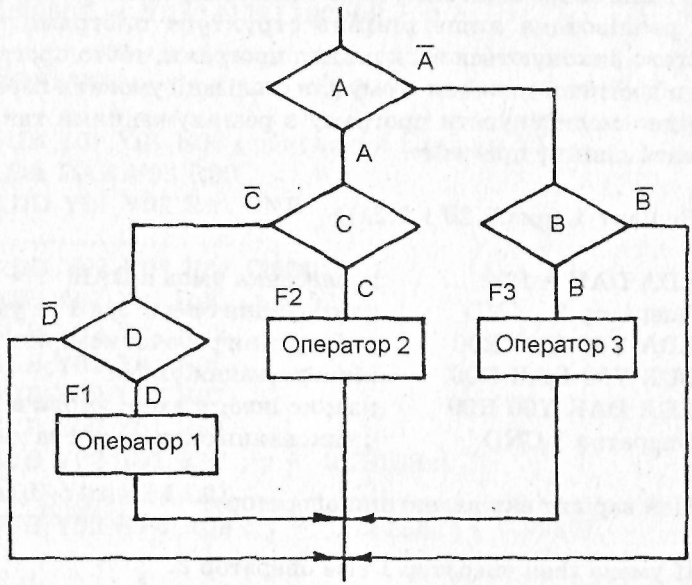


Рис. 2.22. Умовні переходи з великою кількістю умов

Приклад.

Тут A, B, C, D - умови, які можуть бути задані логічними значеннями. Для виконання оператора 1 необхідна умова F1 = $A\overline{C}D$, відповідно F2 = AC і F3 = $\overline{A}B$.

Програма:

```
AND C̄ A R00      ; визначення умови виконання F1
AND D C̄ R00
LDA DAR D R00    ; засилка умови в DAR
Оператор 1 CND_п ; умовне виконання оператора 1
AND C A R00      ; визначення умови виконання F2
LDA DAR C R00    ; засилка умови в DAR
Оператор 2 CND_п ; умовне виконання оператора 2
AND Ā B R00      ; визначення умови виконання F3
LDA DAR Ā R00    ; засилка умови в DAR
Оператор 3 CND_п ; умовне виконання оператора 3.
```

2.11. Програма перетворення десяткового числа в двійковий код

Перетворення десяткового числа в двійкове є частковим випадком переводу чисел із однієї системи числення в іншу.

Найбільш зручною і часто застосовуваною є наступна процедура перетворення цілих чисел. Десяткове число ділять на два. Залишок від ділення може дорівнювати 0 або 1. Цей залишок присвоюється молодшому значущому розряду. Результат ділення на першому кроці ділимо ще раз на два. Залишок (0 або 1) використовується як значення наступного розряду. Кроки цієї процедури повторюються, поки частка від ділення не стане рівною нулеві. Тоді залишок від останнього ділення використовується як значення старшого значущого розряду.

Викладена процедура застосовується для перетворення цілих (або цілої частини) десяткових чисел в двійкові. Для дробів (або дробових частин чисел) застосовують окрему подібну процедуру. Тоді результат перетворення отримують шляхом запису двійкових еквівалентів цих частин відповідно зліва і справа від двійкової коми.

Перетворення десяткового дробу в двійковий здійснюється шляхом множення дробу на основу системи числення (на 2). Якщо результат множення менший 1, то старшому значущому розряду після коми присвоюється 0, якщо більше 1, то присвоюється 1. Результат попередньої операції знову множимо на 2, причому, якщо результат був більше 1, то множиться лише його дробова частина. Аналогічно, якщо результат менший за 1, то наступному розряду присвоюється 0, якщо рівний або більше 1, то 1. Кроки процедури повторюються, поки результат множення не стане точно рівним 1, або не буде досягнута задана точність.

Програма перетворення десяткового числа в двійкове (D_ B).

Програма побудована за описаним алгоритмом. Вибрана точність перетворення - 25 двійкових розрядів. За запрошенням вводиться десяткове число. Якщо число від'ємне, то друкується знак «-» мінус, а далі здійснюється перетворення модуля числа, окремо цілої частини і дробової. Результат перетворення записується у текстові змінні s_1 та s_2 і виводиться на екран.

Програма перетворення десяткового числа в двійкове знакорозрядне з мінімальним числом одиниць (D_ BM).

Програма використовує описаний алгоритм переводу десяткового числа в двійкове. За запрошенням вводиться десяткове число і його допустимі межі $d \pm \Delta d$ для даного перетворення. Потім послідовно для чисел від $d-\Delta d$ до $d+\Delta d$ з кроком, який дорівнює молодшому значущому розряду, здійснюється перевід десяткового числа в двійкове. Всі групи із $r \geq 3$ послідовних одиниць замінюються на послідовність із $r+1$ знака виду 10...-1, чим досягається зменшення числа одиниць в двійковому поданні. Число одиниць підраховується для кожного десяткового числа.

На екран виводяться всі локальні мінімуми (тобто числа з мінімальним числом одиниць у двійковому поданні). Серед них буде одне (або декілька) значень із мінімальним числом одиниць двійкового подання числа у вказаному діапазоні.

Для перетворення коефіцієнтів різницевих рівнянь в двійкову систему числення або в двійкову знакорозрядну систему числення з мінімальним числом ненульових значень можна скористатися програмами D_B і D_BM із Додатка 1.

2.12. Реалізація типових коригуючих ланок на BE1

З допомогою даної мікро-ЕОМ можлива реалізація різницевих рівнянь високого порядку - в залежності від складності представлення коефіцієнтів степенями числа 2 до 38-го порядку включно. Тому дана мікро-ЕОМ дозволяє реалізувати достатньо складні і швидкодіючі алгоритми керування поліграфічним обладнанням.

Дана реалізація передбачає множення чисел зі знаком і їх накопичення. Для такого множення розроблена наступна програма.

Виконання 4-квадрантного перемножувача. Виконання такого перемножувача на основі BE1 для додатних і від'ємних величин сигналів базується на множенні модулів за допомогою додавань і зсувів і відповідного присвоєння знака результатові.

Нехай число X є множимою величиною (знак і модуль). Запишемо його у вигляді суми двох компонентів:

$$X = -1 \cdot s + s\sum_{i=0}^{n} \bar{b}_i 2^{-i} + \bar{s}\sum_{i=0}^{n} b_i 2^{-i} + s \cdot 2^{-n}$$

$$= s[-1 + \sum_{i=0}^{n} \bar{b}_i 2^{-i} + 2^{-n}] + \bar{s}\sum_{i=0}^{n} b_i 2^{-i},$$

де s - знаковий біт: 0 - для додатних, 1 - для від'ємних чисел; b_i - розрядний біт, що дорівнює 1 або 0.

Число X може бути записано також як X= - s + x, де x - представляє модуль, включаючи знаковий біт.

$$x = \sum_{i=0}^{n} \bar{b}_i 2^{-i} + 2^{-n}, \text{ для } x < 0;\ s = 1;$$

$$x = \sum_{i=0}^{n} b_i 2^{-i}, \text{ для } x \text{ і } 0;\ s = 0;$$

Результат множення Z=X·Y може бути подано так:

якщо X=(s,x)=-s+x;

Y=(t,y)=-t+y;

де s - знаковий біт X; t - знаковий біт Y; x - модуль X; y - модуль Y; Z=(-s+x)·(-t+y)=st+xy-sy-tx.

Тепер BE1 можна використати для одержання результату Z (множення модулів і однополярних складових сиґналів), виконуючи лише зсуви і умовні додавання. Умовою додавання буде біт DAR. Якщо знаковий біт іґнорується в множимому X, то результат подамо так:

Z=x(-t+y)=xt+xy.

Цьому виразу не вистачає величини

st-sy=s(-Y),

яка може бути додана до сформованої раніше величини Z:

Z=Z+s(-Y),

виконуючи умовне додавання -Y на основі величини s.

Програма:

```
LDA DAR X R00
ADD Z Y R01 CND7     ;формування величини
ADD Z Y R02 CND6     ;Z=x(-t+y);
 .....

ADD Z Y R08 CND0
SUB Y Y L01          ;формування -Y
ADD Z Y R00 CNDS     ;умовне додавання -Y,
                     ;якщо X від'ємне.
SUB Y Y L01          ;відновлення початкового знака,
                     ;якщо Y - від'ємне
```

Приклад. Реалізація коригуючої ланки.

На основі даної програми реалізуємо адаптивну інтеґрувальну ланку, що описується функцією

$$y_n = y_{n-1} + b_i \varepsilon_{n-1},$$

де y_n - вихідний сиґнал ланки, ε_n - сиґнал похибки, b_1 - коефіцієнт пропорційності. Вважаємо, що ці сиґнали знаходяться в комірках пам'яті Y, E, B. Тоді маємо

```
LDA DAR B R00
ADD Y E R01 CND7;     формування добутку
ADD Y E R02 CND6              ε_{n-1}·b_i
.....

ADD Y E R08 CND0
SUB E E L01
ADD Y E R00 CNDS
```

В даному прикладі не застосовуються додаткові комірки пам'яті, а значення b_i може бути скориговане в процесі роботи.

Аналогічно можуть бути побудовані практично будь-які коригувальні ланки (ізодромна, пропорційна, інтеґродиференціювальна і т.п.).

2.13. Спектральний аналізатор

Ідеальний сканувальний спектральний аналізатор може бути виконаний шляхом сканування центральної частоти вузькосмугового фільтра, щоб визначати енергію вхідного сиґналу на будь-якій частоті [39]. Але практично неможливо спроектувати перенастроюваний аналоговий фільтр, який може перекрити діапазон частот з співвідношенням від 10 до 1, особливо близько до нульової частоти. Навіть цифрове виконання такого фільтра дуже складне і конструктивно неефективне, коли необхідне таке перенастроювання частот. Тому легше реалізувати еквівалент сканувального фільтра шляхом пропускання сиґналу через фіксовано настроєний вузькосмуговий фільтр. Це здійснюється супергетеродинною системою (рис. 2.23), часові діаграми якої показані на рис.2.24.

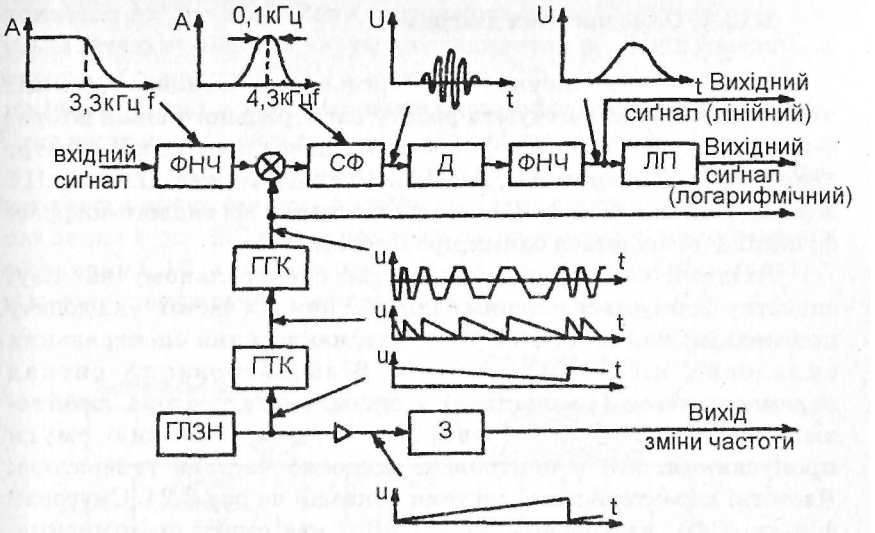


Рис. 2.23. Спектральний аналізатор

де ФНЧ - фільтр нижніх частот, СФ - смуговий фільтр, Д - детектор, ЛП - логарифмічний підсилювач, ГГК - ґенератор гармонічних коливань, ГТК - ґенератор трикутних коливань, ГЛЗН - ґенератор лінійно-змінної напруги, З - затримка

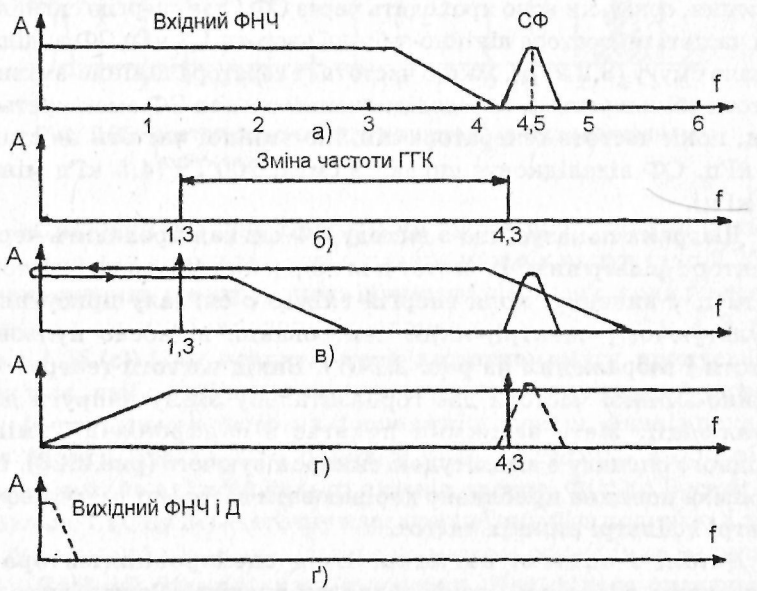


Рис. 2.24. Частотні характеристики сиґналів спектрального аналізатора

2.13.1. Опис часових діаграм

Діаграми на рисунку 2.23 показують основні функції підсистем, які забезпечують роботу спектрального аналізатора. В цифровому виконанні повинні включатися попередній фільтр, схема вибірки-зберігання, АЦП та відповідний вихідний ЦАП і згладжувальний фільтр. В схемних рішеннях всі аналого-цифрові функції здійснюються одним процесором BE1.

Вхідний сиґнал, який підлягає спектральному аналізу, спочатку формується вхідним фільтром нижніх частот (і на додачу попереднім) для того, щоб уникнути накладання спектральних складових після змішування. Відфільтрований сиґнал перемножується (змішується) з сиґналом ґенератора лінійно-змінної частоти, щоб утворити верхню і нижню смуги пропускання, які є центровані відносно частоти ґенератора. Частотні характеристики системи показані на рис.2.24. Смуговий фільтр (СФ), настроєний на 4,5 кГц, має смугу пропускання, яка показана на рис.2.24(а). Ґенератор лінійно змінної частоти змінює частоту в межах від 1,3 кГц до 4,3 кГц, як показано на рис.2.24(б). Нижня і верхня границі смуг після змішування для частот ґенератора лінійно-змінної частоти від 1,3 кГц до 4,3 кГц відповідно показані на рис.2.24(в) і (г). Нас цікавить лише верхнє значення, оскільки воно проходить через СФ і дає енерґію сиґналу. При частоті ґенератора лінійно-змінної частоти 1,3 кГц СФ виділяє верхню смугу (3,2 кГц). Якщо частота ґенератора лінійно-змінної частоти збільшується, то відслідкований сиґнал СФ зменшується доти, поки частота ґенератора лінійно-змінної частоти не стане 4,3 кГц. СФ відслідковує сиґнал в смузі 200 Гц (4,5 кГц мінус 4,3 кГц).

Діаграма показує, що з виходу СФ сиґнал проходить через детектор і фільтр нижніх частот для виділення огинаючої частоти 4,5 кГц, у випадку, коли енерґія вхідного сиґналу присутня в результуючому спектрі, який центрований відносно нульової частоти і зображений на рис. 2.24(ґ). Вихід частоти ґенератора лінійно-змінної частоти дає горизонтальну зміну напруги для візуалізації. Мета затримки полягає в синхронізації зміни вихідного сиґналу з амплітудою синхронізуючого (рис.2.23). Ця затримка повинна приблизно дорівнювати затримці в смуговому фільтрі і фільтрі нижніх частот.

Вхідні і вихідні сиґнали. Вхід спектроаналізатора є аналоговим сиґналом, який підлягає аналізу. Решта виходів

показано на рис.2.23. Вони включають частоту виходу, яка дає горизонтальну вісь для керування індикатором, вихід ґенератора керованої напруги (ГКН) і амплітуду СФ, що відповідає (як в лінійному, так і логарифмічному масштабах) вихідному сиґналу і дає вертикальну складову керування індикатором.

Розширення смуги частот згідно з рис.2.24, для включення миттєвого значення при 13 кГц, показує спотворення спектра, що видно з рис.2.25. З цього графіка стає зрозумілим введення обмеження 13 кГц для ширини смуги пропускання фільтра відносно центральної частоти.

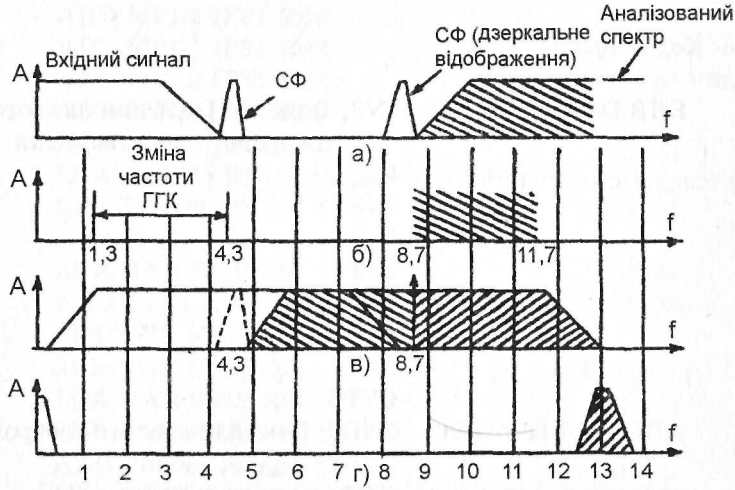


Рис. 2.25. Частотні залежності в аналізаторі з розширенням смуги вхідних сиґналів

Смуговий фільтр. Положення смуги пропускання фільтра визначається шириною смуги фільтра нижніх частот (рис.2.25(а)) і спотворенням спектра в нижній частині діапазону, коли ґенератор лінійно-змінної частоти настроєний на частоту 4,3 кГц (рис.2.25.(с)). СФ повинен мати достатню смугу пропускання для того, щоб запобігти спотворенню сиґналу як вихідної частоти, так і модулюючих частот, які присутні в спектрі. Аналіз показує, що фільтр Беселя 3-го порядку буде достатнім, якщо вірно спроектовано вхідний фільтр нижніх частот. Фільтр Беселя має ідеально гладку передаточну характеристику (без коливань), тому результуючий вихід аналізатора не матиме сплесків та коливань.

Вхідний фільтр нижніх частот. Цей фільтр визначає не тільки центрований відносно нуля спектр, але також спотворення

в нижній ділянці діапазону ґенератора лінійно-змінної частоти. Виявлено, що фільтр четвертого порядку з двома нулями забезпечує необхідне (не менше, ніж 48 дБ) затухання поза смугою пропускання смугового фільтра.

Вихідний фільтр нижніх частот. Цей фільтр призначений для усунення гармонічних складових з виходу ЦАП (відповідно спотворених компонент сиґналу) перед тим, як цифровий сиґнал перетворюється в аналоговий і виводиться на вихід.

2.13.2. Програма спектрального аналізу

```
N ко-  Код операції
манди
0      SUB DAR DAR R00   IN3; Очистка DAR для аналого-
                               цифрового перетворення
1                        IN3
2                        IN3
3                        IN3
4                        IN3
5                        IN3
6                        NOP
7                        NOP
8                        CVTS
9      ADD DAR KM2 R00   CND6; Команди аналого-цифрового
                               перетворення
10                       NOP
11                       NOP
12                       CVT7
13                       NOP
14     LDA TEMP IF11 R00 NOP
15     LDA IF11 IF10 R00 CVT6
16     SUB IF10 IF10 R02 NOP
17     SUB IF10 IF10 R06 NOP
18     ADD IF10 IF10 R09 CVT5
19     SUB IF10 TEMP R02 NOP
20     ADD IF10 TEMP R06 NOP
21     ADD IF10 TEMP R09 CVT4
22     ADD IF10 TEMP R10 NOP
23     SUB IF10 TEMP R12 NOP
24     LDA TEMP IF31 R00 CVT3
25     LDA IF31 IF30 R00 NOP
26     LDA IF30 TEMP R09 NOP
```

```
27    SUB IF30 TEMP R01 CVT2
28    SUB IF30 TEMP R03 NOP
29    ADD IF30 IF30 R04 NOP
30    ADD IF30 IF31 R03 CVT1
31    SUB IF30 IF31 R05 NOP
32    ADD IF30 IF31 R07 NOP
33    ADD IF30 IF31 R10 CVT0
34    ADD IF10 DAR R03; Запис вхідного відліку
35    ADD IF30 IF10 R00
36    LDA MPL2 IF30 R00
37    ADD MPL2 IF31 R00
38    ADD MPL2 IF31 R03
39    ADD MPL2 IF31 R08
40    ADD MPL2 IF31 R09
41    ADD MPL2 TEMP R00
42    LDA S1 KP5 R12; Ґенератор лінійно-змінної частоти
43    LDA M KP4 L01
44    SUB F1 S1 R00
45    LDA DAR F1 R00
46    ADD F1 M R00 CNDS
47    LIM SWP KP7 R00
48    SUB SWP F1 R00
49    LDA DAR SWP R00
50    SUB DAR KP5 R05
51    ADD DAR KP4 L01 CNDS
52    LDA F2 F1 R02 NOP
53    SUB F2 F1 R06 NOP
54    SUB F2 F2 R12 NOP
55    SUB F2 F1 R08 NOP
56                        NOP
57                        NOP
58    LDA S2 KP3 R02 OUT0
59    ADD S2 KP3 R06 OUT0
60    ADD S2 KP1 R09 OUT0
61    ADD F2 S2 R00 OUT0
62                          OUT0
63    SUB OSC1 F2 R00 OUT0; Початкове встановлення
64    LDA DAR OSC1 R00        ґенераторів
65    ADD OSC1 M R00 CNDS
66    LDA OSC OSC1 R00
67    SUB OSC M R01
68    ABS OSC OSC L01
```

```
69    SUB OSC M R01
70    LDA DAR F1 R00
71    LDA OSC1 KP0 R00 CNDS
72    ADD OSC OSC LO1
73    SUB MPL1 MPL1 R00; Перемноження вхідних відліків
                        поданих зі знаком
74    LDA DAR OSC R00
75    ADD MPL1 MPL2 R01 CND7
76    ADD MPL1 MPL2 R02 CND6
77    ADD MPL1 MPL2 R03 CND5
78    ADD MPL1 MPL2 R04 CND4
79    ADD MPL1 MPL2 R05 CND3
80    ADD MPL1 MPL2 R06 CND2
81    ADD MPL1 MPL2 R07 CND1
82    ADD MPL1 MPL2 R08 CND0
83    ADD MPL1 MPL2 L01
84    ADD MPL1 MPL2 R00 CNDS
85    LDA TEMP BP11 R00 NOP; Програмування смугового
                            фільтра
86    LDA BP11 BP10 R00 NOP;
87    LDA BP10 BP11 R05 NOP
88    SUB BP10 BP11 R00 NOP
89    ADD BP10 BP10 R03 NOP
90    SUB BP10 BP10 R08 NOP
91    SUB BP10 TEMP R00 OUT2
92    ADD BP10 TEMP R05 OUT2
93    SUB BP10 TEMP R07 OUT2
94    LDA TEMP BP31 R00 OUT2
95    LDA BP31 BP30 R00 OUT2
96    LDA BP30 TEMP R06 OUT2
97    SUB BP30 TEMP R00
98    LDA DAR V0 R00
99    SUB BP30 BP30 R07 NOP
100   SUB BP30 BP31 R00 NOP
101   SUB BP30 BP31 R03 NOP
102   SUB BP30 BP31 R05 NOP
103   ADD BP30 BP31 R07 NOP
104   LDA TEMP BP51 R00 NOP
105   LDA BP51 BP50 R00 OUT4
106   LDA BP50 TEMP R05 OUT4
107   ADD BP50 TEMP R10 OUT4
108   SUB BP50 TEMP R00 OUT4
```

```
109   ADD BP50 BP50 R08 OUT4
110   SUB BP50 BP51 R00 OUT4
111   SUB BP50 BP51 R03
112   LDA DAR LOUT R00
113   ADD BP50 BP51 R06 NOP
114   SUB BP50 BP51 R09 NOP
115   ADD BP10 MPL1 R06 NOP
116   ADD BP30 BP10 R04 NOP
117   ADD BP50 BP30 R04 NOP
118   LDA Y2 Y1 R00 NOP; Програмування ФНЧ
119   LDA Y1 Y0 R00 OUT6
120   SUB Y0 Y1 R05 OUT6
121   SUB Y0 Y1 R09 OUT6
122   SUB Y0 Y1 R10 OUT6
123   SUB Y0 Y2 R00 OUT6
124   ADD Y0 Y1 R00 OUT6
125   ADD Y0 Y2 R05
126   ADD Y0 Y2 R09
127   ADD Y2 Y2 R12
128   ADD Y2 Y2 R13
129   ABA Y0 BP50 R09
130   LDA Y0 Y0 R00
131   ABS Y0 Y0 R00; Програмування логарифмічного
                    підсилювача
132   LDA LOUT Y0 L02;
133   ADD LOUT Y0 L02
134   ADD LOUT Y0 L02
135   ADD LOUT Y0 R01
136   ADD LOUT Y0 R02
137   LDA DAR Y0 R00
138   LDA LOUT Y0 L02 CND3; ділянка 5
139   ADD LOUT KP2 R00 CND3
140   ADD LOUT KP5 R05 CND3
141   LDA LOUT Y0 L01 CND4 ; ділянка 4
142   ADD LOUT KP3 R00 CND4
143   ADD LOUT KP2 R04 CND4
144   LDA LOUT Y0 R00 CND5 ; ділянка 3
145   ADD LOUT KP4 R00 CND5
146   ADD LOUT KP2 R04 CND5
147   LDA LOUT Y0 R01 CND6 ; ділянка 2
148   ADD LOUT KP5 R00 CND6
```

```
149     ADD LOUT KP2 R04 CND6
150     LDA LOUT Y0 R03 CND7 ; ділянка 1
151     ADD LOUT Y0 R04 CND7
152     ADD LOUT Y0 R05 CND7
153     ADD LOUT KP6 R00 CND7
154     ADD LOUT KP4 R04 CND7
155     NOP
.....
191     NOP
        END
```

Розподіл пам'яті ОЗП

Символ	Комірка
TEMP	0
IF11	1
IF10	2
IF31	3
IF30	4
MPL2	5
S1	6
M	7
F1	8
SWP	9
F2	10
S2	11
OSC1	12
OSC	13
MPL1	14
BP11	15
BP10	16
BP31	17
BP30	18
Y0	19
BP51	20
BP50	21
LOUT	22
Y2	23
Y1	24

Розділ 3. ПРОЕКТУВАННЯ СИСТЕМ НА ОСНОВІ ОДНОКРИСТАЛЬНИХ мікро-ЕОМ СІМЕЙСТВА МК-51

3.1. Загальні відомості про ОМЕОМ сімейства МК51

Восьмирозрядні високопродуктивні однокристальні ОМЕОМ сімейства МК51 виконані за високоякісною n-МОН технологією (серія 1816) і КМОН технологією (серія 1830). Найбільш повно сімейство МК51 описане в довіднику [18].

Використання ОМЕОМ сімейства МК51 в порівнянні з МК48 забезпечує збільшення обсягу пам'яті команд і пам'яті даних. Нові можливості вводу/виводу і периферійних пристроїв розширюють діапазон застосування і знижують загальні затрати системи. В залежності від умов використання швидкодія системи збільшується від двох з половиною до десяти разів.

Сімейство МК51 включає п'ять модифікацій ОМЕОМ (що мають ідентичні основні характеристики), основна різниця між якими полягає в реалізації пам'яті програм і потужності споживання.

ОМЕОМ КР1816ВЕ51 і КР1830ВЕ51 включають маскованопрограмований в процесі виготовлення кристалу ПЗП пам'яті програм ємністю 4096 байтів і розраховані на застосування в масовій продукції. За рахунок використання зовнішніх мікросхем пам'яті загальний обсяг пам'яті програм може бути розширений до 64 Кбайтів.

ОМЕОМ КМ1816ВЕ751 містить ППЗП ємністю 4096 байтів з стиранням ультрафіолетовим випромінюванням і зручна на етапі розробки системи при налагоджуванні програм, а також при виготовленні невеликими партіями чи при створенні систем, що вимагають у процесі експлуатації періодичного підстроювання. За рахунок використання зовнішніх мікросхем пам'яті загальний обсяг пам'яті програм може бути розширений до 64 Кбайтів.

ОМЕОМ КР1816ВЕ31 і КР1830ВЕ31 не включають вмонтованої пам'яті програм, але можуть використовувати до 64 Кбайтів зовнішньої постійної чи перепрограмовуваної пам'яті програм і ефективно використовуватися в системах, що вимагають значно більшого за обсягом (ніж 4 Кбайти на кристалі) ПЗП пам'яті програм.

Кожна з перерахованих вище мікросхем є відповідно аналогом BIC 8051, 80C51, 8031, 80C31 сімейства MCS-51 фірми Intel (США).

Порівняльні дані мікросхем наведені в табл. 3.1.

Кожна ОМЕОМ розлянутого сімейства включає вмонтований ОЗП пам'яті даних ємністю 128 байтів із можливістю розширення загального обсягу оперативної пам'яті даних до 64 Кбайтів за рахунок використання зовнішніх мікросхем ЗППВ.

Таблиця 3.1

Мікросхеми	Аналог	Обсяг внутрішньої пам'яті програм, байтів	Тип пам'яті програм	Обсяг внутрішньої пам'яті даних, байтів	Максимальна частота слідування тактових сигналів, МГц	Струм споживання, мА
КР1816ВЕ31	8031АН	-	зовн.	128	12.0	150.0
КР1816ВЕ51	8051АН	4К	ПЗП	128	12.0	150.0
КМ1816ВЕ751	8751Н	4К	ППЗП	128	12.0	220.0
КР1830ВЕ31	80С31ВН	-	зовн.	128	12.0	18.0
КР1830ВЕ51	80С51ВН	4К	ПЗП	128	12.0	18.0

Загальний обсяг пам'яті ОМЕОМ сімейства МК51 може досягати 128 Кбайтів: 64 Кбайти пам'яті програм і 64 Кбайти пам'яті даних.

При розробці на базі ОМЕОМ більш складних систем можуть бути використані стандартні ІС з байтовою організацією. Перелік інтерфейсних мікросхем для реалізації апаратної підтримки ЦСАК подано в додатку 3.

Надалі позначення «МК51» буде загальним для всіх моделей сімейства, за винятком випадків, які будуть обумовлені особливо.

ОМЕОМ включають всі вузли, необхідні для автономної роботи:

1) центральний восьмирозрядний процесор;

2) пам'ять програм обсягом 4 Кбайти (тільки КМ1816ВЕ751, КР1816ВЕ51 і КР1830ВЕ51);

3) пам'ять даних обсягом 128 байтів;

4) чотири восьмирозрядних програмовуваних канали вводу/виводу;

5) два 16-бітових багаторежимних таймери/лічильники;

6) систему переривань з п'ятьма векторами і двома рівнями;

7) послідовний інтерфейс;

8) тактовий ґенератор.

Система команд ОМЕОМ включає 111 базових команд з форматом 1, 2 чи 3 байти.

ОМЕОМ має:

- 32 РЗП;
- 128 визначених користувачем програмно-керованих прапорців;
- набір регістрів спеціальних функцій.

РЗП і визначені користувачем програмно-керовані прапорці розташовані в адресному просторі внутрішнього ОЗП даних. Регістри спеціальних функцій (SFR, SPECIAL FUNCTION REGISTERS) з вказівками їхніх адрес наведені в таблиці 3.2.

Таблиця 3.2

Позначення	Назва	Адреса
* ACC	Акумулятор	0E0H
* B	Регістр B	0F0H
* PSW	Регістр стану програми	0D0H
SP	Вказівник стека	81H
DPTR	Вказівник даних. 2 байти:	
DPL	Молодший байт	82H
DPH	Старший байт	83H
* P0	Порт 0	80H
* P1	Порт 1	90H
* P2	Порт 2	0A0H
* P3	Порт 3	0B0H
* IP	Регістр пріоритетів переривань	0B8H
* IE	Регістр дозволу переривань	0A8H
TMOD	Регістр режимів таймера/лічильника	89H
* TCON	Регістр керування таймера/лічильника	88H
TH0	Таймер/лічильник 0. Старший байт	8CH
TL0	Таймер/лічильник 0. Молодший байт	8AH
TH1	Таймер/лічильник 1. Старший байт	8DH
TL1	Таймер/лічильник 1. Молодший байт	8BH
* SCON	Керування послідовним портом	98H
SBUF	Буфер послідовного порту	99H
PCON	Керування споживанням	87H

*- регістри, які допускають побітову адресацію.

Нижче коротко описуються функції регістрів, наведених в таблиці 3.2.

Акумулятор. ACC - регістр акумулятора. Команди,

призначені для роботи з акумулятором, використовують мнемоніку «А», наприклад, MOV A, P2. Мнемоніка «ACC» використовується, наприклад, при побітовій адресації акумулятора. Так, символічне ім'я п'ятого біта акумулятора при використанні асемблеру ASM51 буде наступним: ACC.5.

Регістр В. Використовується під час операцій множення і ділення. Для інших інструкцій регістр В може розглядатися як додатковий надоперативний регістр.

Регістр стану програми. Регістр PSW включає інформацію про стан програми.

Вказівник стека SP. 8-бітовий регістр, вміст якого інкрементується перед записом даних в стек при виконанні команд PUSH і CALL. При початковому скиданні вказівник стека встановлюється в 07H, а область стека в ОЗП даних починається з адреси 08H. При необхідності шляхом перевизначення вказівника стека область стека може бути розташована в будь-якому місці внутрішнього ОЗП даних мікро-ЕОМ.

Вказівник даних. Вказівник даних (DPTR) складається з старшого байта (DPH) і молодшого байта (DPL). Включає 16-бітову адресу при звертанні до зовнішньої пам'яті. Може використовуватися як 16-бітовий регістр чи як два незалежних восьмибітових регістри.

Порт 0 – Порт 3. Регістрами спеціальних функцій P0, P1, P2, P3 є регістри-«фіксатори» відповідно портів P0, P1, P2, P3.

Буфер послідовного порту. SBUF містить два окремі регістри: буфер передавача і буфер приймача. Коли дані записуються в SBUF, вони поступають в буфер передавача, причому запис байта в SBUF автоматично ініціалізує його передачу через послідовний порт. Коли дані зчитуються з SBUF, вони вибираються з буфера приймача.

Регістри таймера. Регістрові пари (TH0, TL0) і (TH1, TL1) утворюють 16-бітові лічильні регістри відповідно таймера/лічильника 0 і таймера/лічильника 1.

Регістри керування. Регістри спеціальних функцій IP, IE, TMOD, TCON, SCON і PCON включають біти керування і біти станів системи переривань, таймерів/лічильників і послідовного порту.

ОМЕОМ при функціонуванні забезпечує:

- мінімальний час виконання команд додавання - 1 мкс;
- апаратне множення і ділення з мінімальним часом виконання команд множення/ділення - 4 мкс.

В ОМЕОМ передбачена можливість задання частоти

внутрішнього ґенератора за допомогою кварцового резонатора, LC-ланки чи зовнішнього ґенератора.

Важливою і відмінною рисою архітектури сімейства МК51 є те, що АЛП може разом з виконанням операцій над 8-розрядними типами даних маніпулювати однорозрядними даними. Окремі програмно-доступні біти можуть бути встановлені, скинуті чи замінені їх доповненням, можуть пересилатися, перевірятися і використовуватися в логічних обчисленнях. Ця особливість робить мікро-ЕОМ сімейства МК51 особливо зручними для побудови контролерів. Алгоритми роботи останніх за своєю суттю передбачають наявність вхідних і вихідних булівих змінних, які складно реалізувати за допомогою стандартних мікропроцесорів. Всі ці властивості в цілому називаються булівим процесором сімейства МК51. Завдяки такому потужному АЛП набір інструкцій мікро-ЕОМ сімейства МК51 однаково добре підходить для реалізації як керування в реальному маштабі часу, так і для алгоритмів з великим обсягом даних.

3.2. Структурна схема МК51

ОМЕОМ, структурна схема якої представлена на рис. 3.1а і рис.3.1б, складається з таких основних функціональних вузлів: блоку керування, арифметико-логічного пристрою (АЛП), блоку таймерів/лічильників, блоку послідовного інтерфейсу і переривань, програмного лічильника, пам'яті даних і пам'яті програм. Двосторонній обмін інформацією здійснюється за допомогою внутрішньої 8-розрядної магістралі даних. Умовне графічне зображення мікросхеми показане на рис.3.1в, призначення виводів зведені в таблицю 3.3.

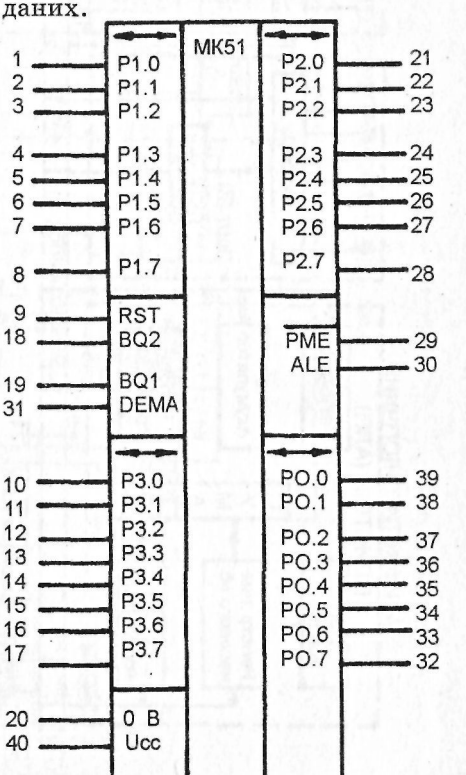


Рис. 3.1в. Умовне графічне позначення

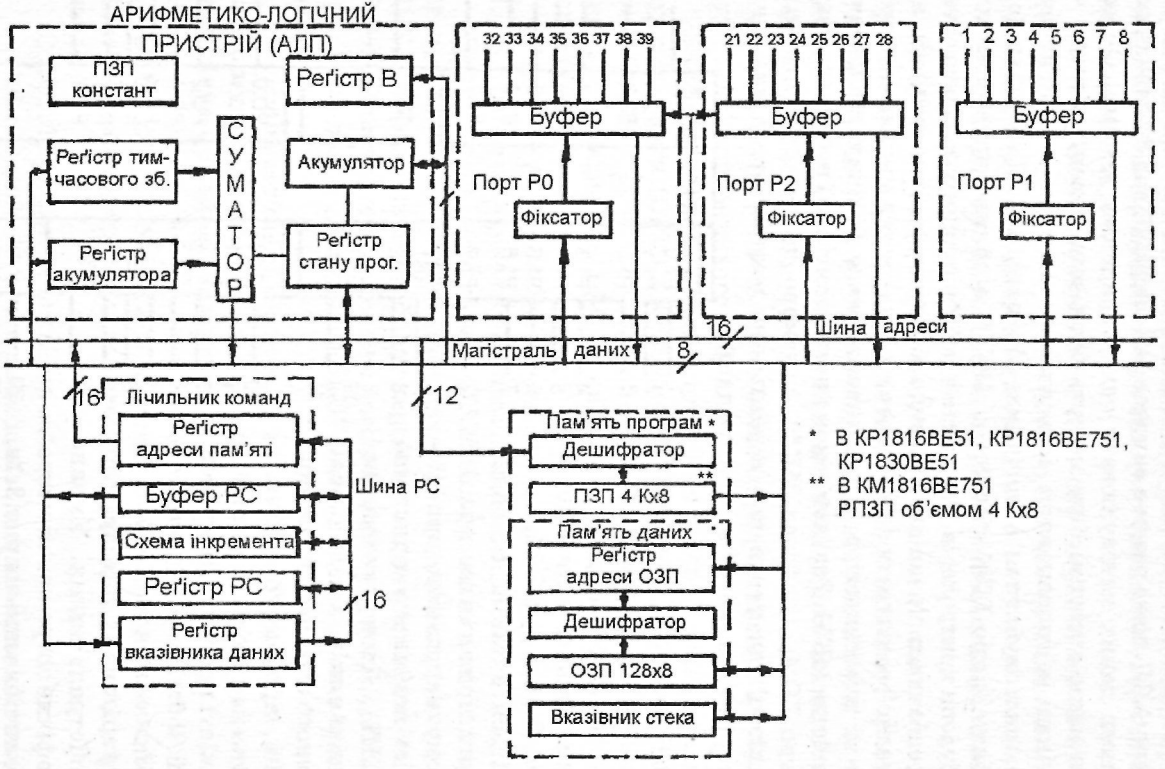


Рис. 3.1а. Структурна схема ОМЕОМ

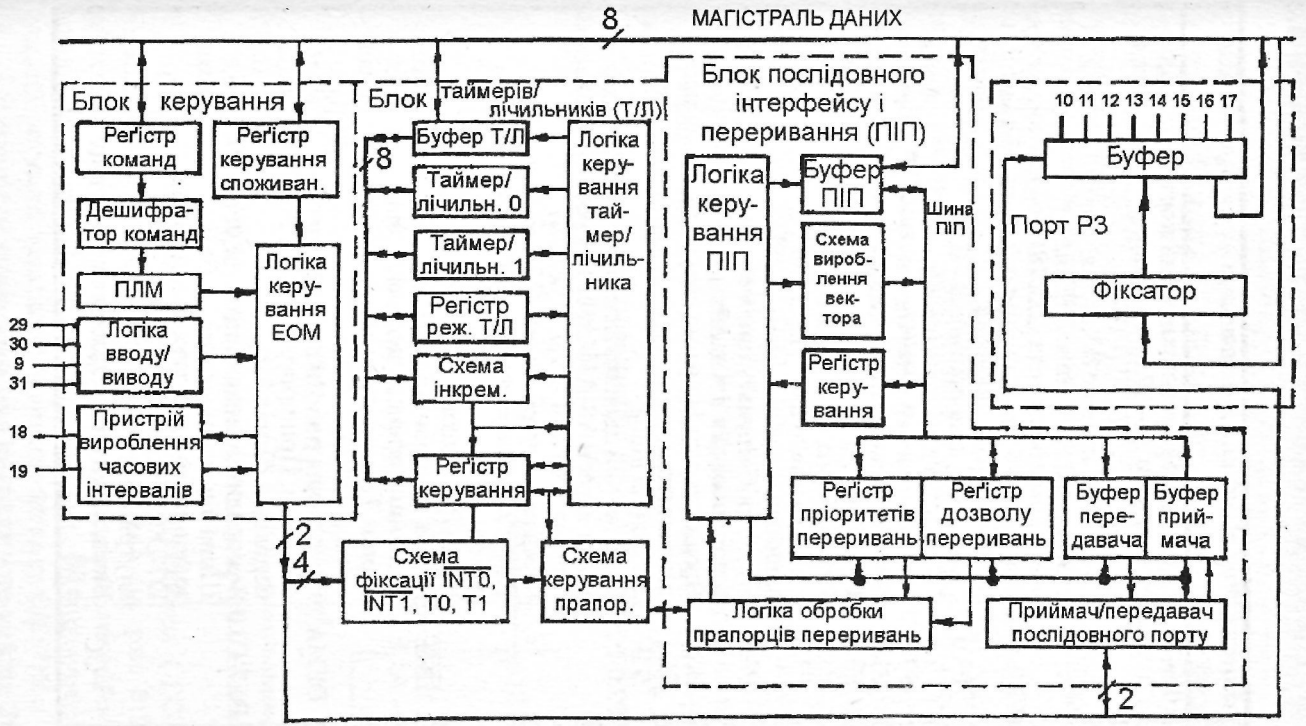


Рис. 3.1б. Структурна схема ОМЕОМ

Таблиця 3.3

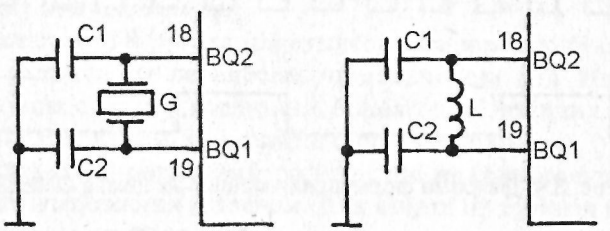
N виводу	Позначення	Призначення	Тип
1-8	P1.0-P1.7	8-розрядний двонаправлений порт P1. Вхід адреси A0-A7 при перевірці внутрішнього ПЗП (РПЗП).	вхід/ вихід
9	RST	Сигнал загального скидання. Вивід резервного живлення ОЗП від зовнішнього джерела (для 1816)	вхід
10-17	P3.0-P3.7	8-розрядний двонаправлений порт P3 з додатковими функціями:	вхід/ вихід
	P3.0	Послідовні дані приймача - RxD	вхід
	P3.1	Послідовні дані передавача - TxD	вихід
	P3.2	Вхід зовнішнього переривання 0 - $\overline{INT0}$	вхід
	P3.3	Вхід зовнішнього переривання 1 - $\overline{INT1}$	вхід
	P3.4	Вхід таймера/лічильника 0: - T0	вхід
	P3.5	Вхід таймера/лічильника 1: - T1	вхід
	P3.6	Вхід стробімпульсу при запису в зовнішню пам'ять даних: - $\overline{WR}$	вихід
	P3.7	Вхід стробімпульсу при читанні з зовнішньої пам'яті даних: - $\overline{RD}$	вихід
18	BQ2	Входи для підключення кварцового	вихід
19	BQ1	резонатора.	вхід
20	0 В	Загальний вивід	
21-28	P2.0-P2.7	8-розрядний двонаправлений порт P2. Вихід адреси A8-A15 в режимі роботи з зовнішньою пам'яттю. В режимі перевірки внутрішнього ПЗП виводи P2.0-P2.6 використовуються як вхід адреси A8-A14. Вивід P2.7 - дозвіл читання ПЗП: - $\overline{E}$	вхід/ вихід
29	$\overline{PME}$	Дозвіл програмної пам'яті	вихід
30	ALE	Вихідний сигнал дозволу фіксації адреси. При програмуванні РПЗП сигнал: - $\overline{PROG}$	вхід/ вихід
31	DEMA	Блокування роботи з внутрішньою пам'яттю. При програмуванні РПЗП подається сигнал U_{PR}	вхід/ вихід
32-39	P0.7-P0.0	8-розрядний двонаправлений порт P0. Шина адреси/даних при роботі з зовнішньою пам'яттю. Вихід даних D7-D0 в режимі перевірки внутрішнього ПЗП (РПЗП).	вхід/ вихід
40	U_{CC}	Вивід живлення від джерела напруги +5В	

3.2.1. Блок керування

Блок керування призначений для формування синхронізуючих і керуючих сиґналів, що забезпечують координацію сумісної роботи блоків ОМЕОМ у всіх допустимих режимах її роботи.

В склад блоку керування входять: пристрій формування часових інтервалів, логіка вводу-виводу, реґістр команд, реґістр керування споживанням, дешифратор команд, ПЛМ і логіка керування мікро-ЕОМ.

Пристрій формування часових інтервалів призначений для формування і видачі внутрішніх синхросиґналів фаз, тактів і циклів. Кількість машинних циклів визначає тривалість виконання команд. Практично всі команди ОМЕОМ виконуються за один чи два машинні цикли, крім команд множення MUL AB і ділення DIV AB, тривалість виконання яких складає 4 машинні цикли. Машинний цикл має фіксовану тривалість і включає 6 станів S1 - S6, кожен з яких за тривалістю відповідає такту і, в свою чергу, складається з двох часових інтервалів, що визначаються фазами P1 і P2. Тривалість фази рівна періоду слідування зовнішнього сиґналу BQ, що є первинним сиґналом синхронізації ОМЕОМ при підключенні до її виводів 18 (BQ2) і 19 (BQ1) кварцового резонатора чи LC-ланки або зовнішнім джерелом тактових сиґналів.



C1, C2 - ємність 30пФ ± 10пФ

Cpp ≈ 10пФ - ємність виводу

Рис. 3.2. Підключення кварцового резонатора і LC-ланки

Схема підключення кварцового резонатора і LC-ланки до виводів ОМЕОМ BQ2 і BQ1 показана на рис.3.2. Схема підключення зовнішнього джерела тактових сиґналів наведена на рис.3.3. Джерело тактових сиґналів має забезпечувати наступні характеристики зовнішнього синхросиґналу ОМЕОМ:

- тривалість низького рівня сиґналу - не менше 20 нс;
- тривалість високого рівня сиґналу - не менше 20 нс;

- часи фронтів наростання і спаду сиґналу - не більше 20нс.

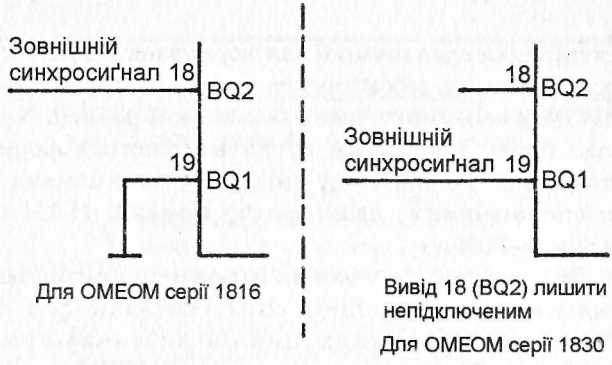


Рис. 3.3. Підключення зовнішнього джерела тактових сиґналів

Рисунок 3.4 ілюструє формування машинних циклів OMEOM. Всі машинні цикли OMEOM однакові, складаються з 12 періодів сиґналу BQ, починаються фазою S1P1 і закінчуються фазою S6P2. Двічі за один машинний цикл формується сиґнал ALE. На рис.3.5 показана діаграма зовнішнього синхросиґналу OMEOM.

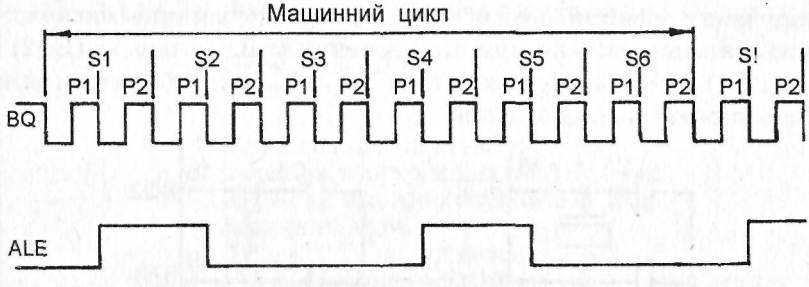


Рис. 3.4. Діаграма формування машинних циклів OMEOM

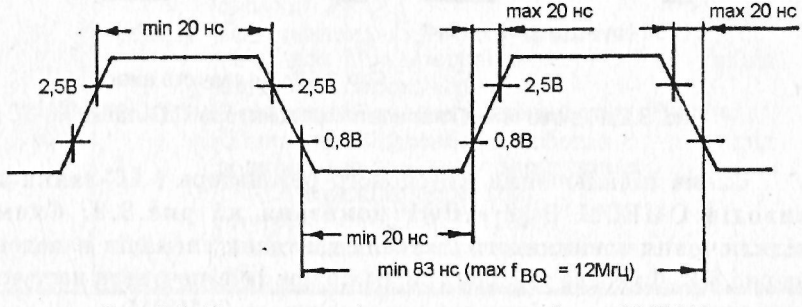


Рис. 3.5. Діаграма зовнішнього синхросиґналу OMEOM

Логіка вводу/виводу призначена для прийому і видачі

сигналів, що забезпечують обмін інформацією ОМЕОМ з зовнішніми пристроями через порти вводу/виводу Р0 - Р3.

Регістр команд призначений для запису і зберігання 8-розрядного коду операції виконуваної команди, який за допомогою дешифратора команд перетворюється в 24-розрядний код для ПЛМ, за допомогою якої виробляється набір мікрооперацій у відповідності з мікропрограмою виконання команди. Регістр команд програмно недоступний.

3.2.2. Арифметико-логічний пристрій

АЛП є паралельним 8-розрядним пристроєм, що забезпечує виконання арифметичних і логічних операцій, а також операції логічного зсуву, обнулення, встановлення і т.п.

АЛП складається з регістра акумулятора, регістра тимчасового зберігання, ПЗП констант, суматора, додаткового регістра (регістра В) акумулятора, регістра стану програми.

Регістр акумулятора і регістр тимчасового зберігання - 8-розрядні регістри, призначені для прийому і зберігання операндів на час виконання операцій над ними. Регістр тимчасового зберігання програмно не доступний.

ПЗП констант забезпечує вироблення коригуючого коду при двійково-десятковому поданні даних, коду маски при бітових операціях і коду констант.

Паралельний 8-розрядний суматор є схемою комбінаційного типу з послідовним переносом, призначеною для виконання арифметичних операцій додавання, віднімання і логічних операцій додавання, множення, нерівнозначності і тотожності.

Регістр В - 8-розрядний регістр, що використовується під час операцій множення і ділення. Для інших інструкцій він може розглядатися як додатковий надоперативний регістр.

Акумулятор є 8-розрядним регістром, що призначений для прийому і зберігання результату, отриманого при виконанні арифметико-логічних операцій чи операцій пересилки.

Регістр стану програми (PSW) призначений для зберігання інформації про стан АЛП при виконанні програми. Позначення розрядів регістра PSW і призначення розрядів наведені відповідно в таблицях 3.4, 3.5.

Таблиця 3.4

Біти	7	6	5	4	3	2	1	0
Позначення	C	AC	F0	RS1	RS0	OV	-	P

Таблиця 3.5

Біти	Назва	Призначення бітів	Доступ до біта
7	C	Прапорець переносу. Міняється під час виконання деяких арифметичних і логічних інструкцій.	Апаратно або програмно
6	AC	Прапорець додаткового переносу. Апаратно встановлюється/скидається під час виконання інструкцій додавання або віднімання для вказівки переносу або позики в біті 3 при утворенні молодшого напівбайта результату (D0-D3).	Апаратно або програмно
5	F0	Прапорець 0. Прапорець стану, який визначається користувачем.	Програмно
4	RS1	Вказівник банку робочих регістрів	Програмно
3	RS0	Вказівник банку робочих регістрів	Програмно
	RS1 RS0		
	0 0	Банк 0 з адресами (00H-07H)	
	0 1	Банк 1 з адресами (08H-0FH)	
	1 0	Банк 2 з адресами (10H-17H)	
	1 1	Банк 3 з адресами (18H-1FH)	
2	OV	Прапорець переповнення. Апаратно встановлюється/скидається під час виконання арифметичних інструкцій для вказування стану переповнення.	Апаратно або програмно
1		Резервний. Містить тригер, доступний по запису ("0" і "1") і читання, який можна використовувати.	
0	P	Біт парності. Апаратно скидається/встановлюється в кожному циклі інструкцій для вказування парної/непарної кількості розрядів акумулятора, які знаходяться в стані "1".	Апаратно або програмно

Прапорець переносу C може встановлюватися і скидатися як апаратними, так і програмними засобами. Прапорець C може бути програмно прочитаний. Апаратними засобами C встановлюється, якщо в старшому біті результату виникає перенос чи позика. При виконанні операцій множення і ділення C скидається. Крім того, прапорець C виконує функції «булівого акумулятора» в командах, що працюють з бітами.

Прапорець додаткового переносу AC програмно доступний за записом («0» і «1») і зчитуванням.

Прапорці F0, RS1, RS0 програмно доступні за записом («0» і «1») зчитуванням.

Прапорець переповнення OV програмно доступний за записом («0» і «1»). Встановлюється апаратно, якщо результат операції додавання/віднімання не вкладається в семи бітах і старший (восьмий) біт результату не може інтерпретуватися як знаковий. При виконанні операції ділення прапорець OV апаратно скидається, а у випадку ділення на нуль встановлюється. При множенні прапорець OV апаратно встановлюється, якщо результат більший 255.

Прапорець P (паритету) є доповненням вмісту акумулятора до парності. В 9-розрядному слові, що складається з 8 розрядів акумулятора і біта P, завжди міститься парне число одиничних бітів. У випадку, якщо в акумуляторі всі розряди встановлені в «0», прапорець P прийме нульове значення. Програмно доступний тільки за зчитуванням.

3.2.3. Блок таймерів/лічильників

Таймери/лічильники (Т/Л) призначені для підрахунку зовнішніх подій, для одержання програмно керованих часових затримок і виконання часозадаючих функцій ОМЕОМ.

В склад блоку Т/Л входять:

1) два 16-розрядні реґістри Т/Л0 і Т/Л1;
2) восьмирозрядний реґістр режимів Т/Л (TMOD);
3) восьмирозрядний реґістр керування (TCON);
4) схема фіксації $\overline{\text{INT0}}$, $\overline{\text{INT1}}$, T0, T1;
5) схема керування прапорцями;
6) логіка керування Т/Л.

Два 16-розрядних реґістри Т/Л0 і Т/Л1 виконують функцію зберігання вмісту рахунку. Кожний із них складається з пари восьмирозрядних реґістрів, відповідно TH0, TL0 і TH1, TL1. Причому реґістри TH0, TH1 - старші, а реґістри TL0, TL1 - молодші 8 розрядів. Кожний із восьмирозрядних реґістрів має свою адресу і може бути використаний як РЗП, якщо Т/Л не використовується (біт TR0 для Т/Л0 і біт TR1 для Т/Л1 в реґістрі керування TCON дорівнює «0»).

Код величини початкового рахунку заноситься в реґістри таймерів/лічильників програмно. В процесі рахунку вміст реґістрів Т/Л інкрементується. Ознакою закінчення рахунку, як правило,

є переповнення реґістра Т/Л, тобто перехід його вмісту із стану «всі одиниці» в стан «всі нулі». Всі реґістри TH0, TH1, TL0, TL1 доступні за зчитуванням, і, при необхідності, контроль досягнення необхідної величини рахунку може виконуватися програмно.

Реґістр режимів Т/Л (TMOD) призначений для прийому і зберігання коду, який визначає:

- один із 4-х можливих режимів роботи кожного Т/Л;
- роботу в ролі таймерів або лічильників;
- керування Т/Л від зовнішнього виводу.

Позначення розрядів реґістра TMOD показано в таблиці 3.6. Призначення розрядів реґістра TMOD подано в таблиці 3.7.

Таблиця 3.6

Біти	7	6	5	4	3	2	1	0
Позначення	GATE1	Л/$\overline{Т1}$	M1.1	M0.1	GATE0	Л/$\overline{Т0}$	M1.0	M0.0

Таблиця 3.7

Біти	Назва	Призначення бітів	Примітка
0-1 4-5	M0-M1	Визначають один з 4-х режимів роботи, окремо для Т/Л1 і Т/Л0 M1 M0 Режим 0 0 0 0 1 1 1 0 2 1 1 3	Всі біти встановлюються програмно; біти 0-3 визначають режим роботи Т/Л0, біти 4-7 визначають режим роботи Т/Л1,
2,6	Л/$\overline{Т0}$ Л/$\overline{Т1}$	Визначають роботу в ролі: Л/$\overline{Т0}$, Л/$\overline{Т1}$ = 0 – таймера Л/$\overline{Т0}$, Л/$\overline{Т1}$ = 1 – лічильника	
3,7	GATE	Дозволяє керувати таймером від зовнішнього виводу ($\overline{INT0}$ - для Т/Л0, $\overline{INT1}$ - для Т/Л1). GATE = 0 - керування заборонено GATE = 1 - керування дозволено	

Під час роботи в ролі таймера вміст реґістра Т/Л інкрементується в кожному машинному циклі, тобто Т/Л є лічильником машинних циклів ОМЕОМ. Оскільки машинний цикл складається із 12 періодів частоти синхронізації ОМЕОМ f_{BQ}, то частота рахунку в даному випадку дорівнює $f_{BQ}/12$.

Під час роботи Т/Л в ролі лічильника зовнішніх подій вміст регістра Т/Л інкрементується у відповідь на перехід з «1» в «0» сигналу на лічильному вході ОМЕОМ (вивід Т0 для Т/Л0 і вивід Т1 для Т/Л1). Лічильні входи апаратно перевіряються у фазі S5P2 кожного машинного циклу. Коли перевірки показують високий рівень на лічильному вході в кожному машинному циклі і низький рівень в другому машинному циклі, регістр Т/Л інкрементується. Нове (інкрементоване) значення заноситься в регістр Т/Л в фазі S3P1 машинного циклу, який безпосередньо слідує за тим, в якому був виявлений перехід із «1» в «0» на лічильному вході ОМЕОМ. Для розпізнавання такого переходу необхідно два машинних цикли (24 періоди частоти синхронізації ОМЕОМ f_{BQ}), то максимальна частота рахунку Т/Л в режимі лічильника дорівнює $f_{BQ}/24$.

Щоб рівень сигналу на лічильному вході був гарантовано зафіксований, він повинен залишатися незмінним протягом як мінімум одного машинного циклу.

Регістр керування (TCON) призначений для прийому і зберігання коду керуючого слова. Позначення розрядів регістра TCON наведено в таблиці 3.8. Призначення розрядів регістра TCON приведено в таблиці 3.9.

Таблиця 3.8

Біти	7	6	5	4	3	2	1	0
Позначення	TF1	TR1	TF0	TR0	IE1	IT0	IE0	IT0

Прапорці переповнення TF0 і TF1 встановлюються апаратно при переповненні відповідних Т/Л (перехід Т/Л із стану «всі одиниці» в стан «всі нулі»). Якщо при цьому переривання від відповідного Т/Л дозволено, то встановлення прапорця TF викличе переривання. Прапорці TF0 і TF1 скидаються апаратно при передачі керування програмі обробки відповідного переривання.

Прапорці TF0 і TF1 програмно доступні і можуть бути встановлені/скинені програмою. Використовуючи цей механізм, переривання по TF0 і TF1 можуть бути викликані (встановленням TF) і відмінені (скиданням TF) програмою.

Прапорці IE0 і IE1 встановлюються апаратно від зовнішніх переривань (відповідно входи ОМЕОМ $\overline{INT0}$ і $\overline{INT1}$) чи програмно та ініціюють виклик програми обробки відповідного переривання. Скидання цих прапорців виконується апаратно при обслуговуванні переривання тільки в тому випадку, коли переривання було викликано по фронту сигналу. Якщо переривання було викликане

рівнем сигналу на вході $\overline{INT0}$ ($\overline{INT1}$), то скидання прапорця IE повинна виконувати програма обслуговування переривання, впливаючи на джерело переривання для зняття ним запиту.

Таблиця 3.9

Біти	Назва	Призначення бітів	Примітка
6 4	TR1 TR0	Біти виключення Т/С, окремо для Т/С 0 і Т/С 1. TR=0 – виключений, TR=1 – включений.	Біти встановлюються і скидуються програмно. Доступні до читання.
7 5	TF1 TF0	Прапорці переповнення Т/С.	Біти скидаються і встановлюються апаратно і програмно. Доступні до читання.
2 0	IT1 IT0	Біти, які визначають вид переривання по входам $\overline{INT0}$, $\overline{INT1}$: IT=0 – переривання по рівню (низькому) IT=1 – переривання по фронту (перехід з "1" в "0")	Біти встановлюються і скидаються програмно. Доступні до читання.
3 1	IE1 IE0	Прапорці запиту зовнішніх переривань по входах $\overline{INT0}$, $\overline{INT1}$	Біти скидаються і встановлюються апаратно і програмно. Доступні до читання.
			Біти 4,5 відносяться до Т/Л 0; біти 6, 7 – до Т/Л 1. Біти 0, 1 визначають зовнішні переривання $\overline{INT0}$, біти 2, 3 – по входу $\overline{INT1}$.

Схема інкременту призначена:

- для збільшення на 1 в кожному машинному циклі вмісту регістрів Т/Л0, Т/Л1, для яких встановлений режим таймера і рахунок дозволений;

- для збільшення на 1 вмісту регістрів Т/Л0, Т/Л1, для яких встановлений режим лічильника, рахунок дозволений і на відповідному вході ОМЕОМ (Т0 для Т/Л0 і Т1 для Т/Л1) зафіксований лічильний імпульс.

Буфер передавача призначений для прийому з шини блока послідовного інтерфейса і переривань (ППП) паралельної інформації і видачі її у вигляді послідовного потоку символів на передавач послідовного порту.

Буфер приймача призначений для прийому даних у вигляді послідовного потоку символів з послідовного порту, перетворення їх в паралельний вигляд, зберігання і видачі в паралельному вигляді на внутрішню шину ППП.

Буфер приймача і буфер передавача при програмному доступі мають однакове ім'я (SBUF) і адресу (99H). Якщо команда використовує SBUF як регістр призначення, то звертання відбувається до буфера передавача.

У всіх чотирьох режимах роботи послідовного порту передача ініціюється будь-якою командою, яка використовує SBUF як регістр призначення. Прийом в режимі 0 ініціюється умовою RI=0 і REN=1. В усіх решта режимах прийом ініціюється приходом старт-біта, якщо REN=1.

Приймач/передавач послідовного порту призначений для приймання послідовного потоку символів із входу послідовного порту, виділення даних і видачі їх у буфер приймача, а також для приймання послідовних даних з буфера передавача, перетворення їх у послідовний потік символів і видачі його на вихід послідовного порту.

Регістр пріоритету переривань (IP) призначений для встановлення рівня пріоритету переривань для кожного з п'яти джерел переривань. Позначення розрядів регістра IP наведене в таблиці 3.10, а їх призначення вказано нижче:

PX0 - встановлення рівня пріоритету переривань від зовнішнього джерела $\overline{\text{INT0}}$;

PT0 - встановлення рівня пріоритету переривань від Т/Л0.

PX1 - встановлення рівня пріоритету переривань від зовнішнього джерела $\overline{\text{INT1}}$;

PT1 - встановлення рівня пріоритету переривань від Т/Л1.

PS - встановлення рівня пріоритету переривань від послідовного порту.

X - резервний розряд.

Таблиця 3.10

Біти	7	6	5	4	3	2	1	0
Позначення	X	X	X	PS	PT1	PX1	PT0	PX0

Наявність в розряді IP «1» встановлює для відповідного джерела високий рівень пріорітету, а наявність в розряді IP «0» - низький рівень пріорітету. При зчитуванні резервних розрядів відповідні лінії маґістралі даних не визначені. Користувач не повинен записувати «1» в резервні розряди, тому що вони зарезервовані під подальше розширення сімейства МК51.

Регістр дозволу переривань (IE) призначений для дозволу чи заборони переривань від відповідних джерел. Позначення розрядів регістра IE показане в таблиці 3.11, а їх призначення вказане нижче:

EA - керування всіма джерелами переривань одночасно. Якщо EA=0, то переривання заборонені. Якщо EA=1, то переривання можуть бути дозволені індивідуальними дозволами EX0, ET0, EX1, ET1, ES;

X - *резервний розряд*;

ES - керування перериванням від послідовного порту. ES=1- дозвіл. ES=0 – заборона;

ET1 - керування перериванням від Т/Л1. ET1=1 - дозвіл, ET1=0 – заборона;

EX1 - керування перериванням від зовнішнього джерела $\overline{\text{INT1}}$. EX1=1 - дозвіл. EX1=0 – заборона;

ET0 - керування перериванням від Т/Л0. ET0=1 - дозвіл, ET0=0 – заборона;

EX0 - керування перериванням від зовнішнього джерела $\overline{\text{INT0}}$. EX0=1 - дозвіл, EX0=0 - заборона.

При зчитуванні резервних розрядів відповідні лінії магістралі не визначені. Користувач не повинен записувати «1» в резервні розряди, тому що вони зарезервовані під подальше розширення сімейства МК51.

Таблиця 3.11

Біти	7	6	5	4	3	2	1	0
Позначення	EA	X	X	ES	ET1	EX1	ET0	EX0

Логіка обробки прапорців переривань здійснює пріоритетний вибір запиту переривання, скидання його прапорця і ініціює формування апаратно реалізованої команди переходу на підпрограму обслуговування переривання.

Схема формування вектора переривання виробляє двобайтові адреси підпрограм обслуговування переривання в залежності від джерела переривань, які наведені в таблиці 3.12.

Таблиця 3.12

Джерело переривання	Вектор переривання
Зовнішнє переривання $\overline{\text{INT0}}$	0003H
Таймер/лічильник Т/Л 0	000BH
Зовнішнє переривання $\overline{\text{INT1}}$	0013H
Таймер/лічильник Т/Л 1	001BH
Послідовний порт	0023H

3.2.4. Послідовний інтерфейс

Через універсальний прийомопередавач (УАПП) здійснюється прийом і передача інформації, яка представлена послідовним кодом (молодшим бітом вперед), у повному дуплексному режимі обміну. В склад УАПП, який часто називають послідовним портом, входить приймальний та передаючий регістри зсуву, а також спеціальний буферний регістр (SBUF) прийомопередавача. Запис байта в буфер приводить до перепису байта в зсуваючий регістр передавача і ініціює початок передачі байта. Якщо до моменту закінчення прийому байта попередній байт не був зчитаний з SBUF, то він буде втрачений.

Послідовний порт МК51 може працювати в чотирьох різних режимах.

Режим 0. В цьому режимі інформація передається і приймається через зовнішній вивід приймача (RxD). Приймається або передається 8 бітів даних. Через зовнішній вивід входу приймача (TxD) видаються імпульси зсуву, які супроводжують кожний біт. Частота передачі біта інформації рівна 1/12 частоти резонатора.

Режим 1. В цьому режимі передається через TxD або приймається з RxD 10 бітів інформації: старт-біт (0), 8 бітів даних та стоп-біт (1). Швидкість прийому/передачі - величина змінна і задається таймером.

Режим 2. В цьому режимі через TxD передається або з RxD приймається 11 бітів інформації: старт-біт, 8 бітів даних, програмований дев'ятий біт і стоп-біт. При передачі дев'ятий біт даних може приймати значення 0 або 1, або, наприклад, для підвищення достовірності передачі шляхом контролю по парності в ньому може бути поміщене значення ознаки паритету з слова стану програми (PSW.0). Частота прийому/передачі визначається програмою і може бути рівна 1/32 або 1/64 частоти резонатора в залежності від керуючого біта SMOD.

Режим 3. Режим 3 співпадає з режимом 2 у всіх деталях, за винятком частоти прийому/передачі, яка є величиною змінною і задається таймером.

3.2.4.1. Регістр керування/статусу УАПП

Керування режимом роботи УАПП здійснюється через спеціальний регістр з символічним ім'ям SCON. Цей регістр містить не лише керуючі біти, які визначають режим роботи

послідовного порту, але і дев'ятий біт даних, що приймається або передається (RB8 і TB8) та біти переривання прийомопередавача (R1 і T1).

Функціональне призначення біт регістра керування/статусу УАПП приведене в табл. 3.13.

Таблиця 3.13

Символ	Позиція	Назва і призначення
SM0 SM1	SCON.7 SCON.6	Біти керування режимом роботи УАПП. Встановлюються/скидаються програмно. (Див. примітку)
SM2	SCON.5	Біт керування режимом УАПП. Встановлюється програмно для заборони прийому повідомлення, в якому дев'ятий біт має значення 0.
REN	SCON.4	Біт дозволу прийому. Встановлюється/ скидається програмно для дозволу/ заборони прийому послідовних даних.
TB8	SCON.3	Передача біта 8. Встановлюється/скидається програмною для завдання дев'ятого біта передачі в режимі УАПП-9 біт.
RB8	SCON.2	Передача біта 8. Встановлюється/скидається апаратно для фіксації дев'ятого біта прийому в режимі УАПП-9 біт.
TI	SCON.1	Прапорець переповнення передавача. Встановлюється апаратно при закінченні передачі байта. Скидається програмно після обслуговування переривання.
RI	SCON.0	Прапорець переповнення приймача. Встановлюється апаратно при прийомі байта. Скидається програмно після обслуговування переривання.

Примітка.

SM0	SM1	Режим роботи УАПП
0	0	Зсуваючий регістр розширення вводу/виводу.
0	1	УАПП-8 біт. Змінна швидкість передачі.
1	0	УАПП-9 біт. Фіксована швидкість передачі.
1	1	УАПП-9 біт. Змінна швидкість передачі.

Прикладна програма шляхом завантаження в старші біти регістра SCON 2-бітного коду визначає режим роботи УАПП. У всіх чотирьох режимах робота передавача УАПП ініціюється будь-якою командою, в якій буферний регістр SBUF визначений як

одержувач байта. Прийом в УАПП в режимі 0 здійснюється за умови, що RI=0 і REN=1. В режимах 1, 2, 3 прийом починається з приходом старт-біта, якщо REN=1.

В біті TB8 програмно встановлюється значення дев'ятого біта даних, який буде переданий в режимі 2 або 3. В біті RB8 фіксується в режимах 2 і 3 дев'ятий біт даних, який приймається. В режимі 1, якщо SM2=0, в біт RB8 заноситься стоп-біт. В режимі 0 біт RB8 не використовується.

Прапорець переривання передавача TI встановлюється апаратно в кінці періоду передачі біта даних в режимі 0 і на початку періоду передачі стоп-біта в режимах 1, 2 і 3. Відповідна підпрограма обслуговування переривання повинна скидати біт TI.

Прапорець переривання приймача RI встановлюється апаратно в кінці періоду прийому восьмого біта даних в режимі 0 і в середині прийому стоп-біта в режимах 1, 2 і 3. Підпрограма обслуговування переривання повинна скидати біт RI.

3.2.4.2. Робота УАПП в мультиконтролерних системах

В системах децентралізованого керування, які використовуються для керування і регулювання в топологічно розподілених об'єктах (наприклад, папероробних машинах, друкарсько-обробних лініях та лініях інших автоматизованих виробництв), постає завдання обміну інформацією між системою мікроконтролерів, об'єднаних в локальну обчислювально-керуючу мережу. Як правило, локальні мережі на основі МК51 мають магістральну архітектуру з розподілювальним моноканалом (коаксіальний кабель, вита пара, оптичне волокно), по якому здійснюється обмін інформації між МК.

В регістрі спеціальних функцій SCON мікроконтролера є керуючий біт SM2, а в режимах 2 і 3 УАПП дозволяє відносно простими засобами реалізувати мікроконтролерний обмін інформацією в локальних керуючих мережах.

Механізм мікроконтролерного обміну інформацією через послідовний порт МК51 побудований так, що в режимах 2 і 3 програмований дев'ятий біт даних при прийомі фіксується в біт RB8. УАПП може бути запрограмованим таким чином, що при одержанні стоп-біта переривання від приймача буде можливим лише за умови RB8=1. Це виконується встановленням керуючого біта SM2 в регістрі SCON.

Пояснимо процес міжконтролерного обміну інформацією на

прикладі. Нехай ведучому МК необхідно передати блок даних певному (або декільком) МК. З цією метою ведучий МК в протокольному режимі «широкомовної» передачі (всім веденим МК) видає в моноканал байт-ідентифікатор абонента (код адреси МК одержувача), який відрізняється від байтів даних лише тим, що в його дев'ятому біті міститься 1. Програма реалізації протоколу мережного обміну інформацією повинна бути побудована таким чином, щоб при отриманні байта ідентифікатора (RB8=1) у всіх ведених МК відбулися переривання прикладних програм і виклик підпрограми порівняння байта-ідентифікатора з кодом власної мережної адреси. Адресований МК скидає свій керуючий біт SM2 і готується до прийому блоку даних. Решта ведених МК, адреса яких не співпала з кодом байта-ідентифікатора, залишають незмінним стан SM2=1 і передають керування основній програмі. При SM2=1 інформаційні байти, що передаються по моноканалу і поступають в УАПП ведених МК, переривання не викликають, тобто ігноруються.

В режимі 1 УАПП автономного МК керуючий біт SM2 використовується для контролю істинності стоп-біта (при SM2=1 переривання не відбудеться до тих пір, поки не буде отримане істинне (одиничне) значення стоп-біта). В режимі 0 біт SM2 не використовується і повинен бути скинутим.

3.2.4.3. Швидкість прийому/передачі

Швидкість прийому/передачі, тобто частота роботи УАПП в кожному режимі роботи, визначається різними способами.

В режимі 0 частота передачі залежить лише від резонансної частоти кварцового резонатора $f_0=f_{рез}/12$. За один машинний цикл послідовний порт передає один біт інформації.

В режимах 1, 2 і 3 швидкість прийому/передачі залежить від значення керуючого біта SMOD в регістрі спеціальних функцій PCON (табл.3.14).

Врежимі 2 частота передачі визначається виразом $f_2=(2^{SMOD}/64)f_{рез}$. Тобто, при SMOD = 1 рівна $f_{рез}/64$, а при SMOD = 1 рівна $f_{рез}/32$.

У режимах 1 і 3 у формуванні частоти передачі, крім керуючого біта SMOD, бере участь таймер 1. При цьому частота передачі залежить від частоти переповнення таймера (OVT1) і визначається таким чином $f_{1,3}=(2^{SMOD}/32)f_{OVT1}$. Переривання від таймера 1 в цьому випадку повинно бути заблокованим. Сам Т/Л1 може працювати і як таймер, і як лічильник подій у будь-

якому з трьох режимів. Однак найбільш зручно використати режим таймера з автоперезавантаженням (старша тетрада TMOD=0010). При цьому частота передачі визначається виразом $f_{1,3}=(2^{SMOD}/32)(f_{рез}/12)(256-(TH1))$, де (TH) – десятковий код вмісту TH1.

В [18,22] поданий опис способів налагодження Т/Л1 для одержання типових частот передачі даних через УАПП.

Таблиця 3.14

Реґістр керування потужністю PCON

Символ	Позиція	Назва і функція
SMOD	PCON.7	Подвоєна швидкість передачі. Якщо біт встановлений в 1, то швидкість передачі вдвоє більша, ніж при SMOD=0.
	PCON.6	Не використовується.
	PCON.5	Не використовується.
	PCON.4	Не використовується.
GF1	PCON.3	Прапорці, які спеціалізує (встановлює) користувач
GF0	PCON.2	Прапорці загального призначення.
PD	PCON.1	Біт пониженої потужності. При встановленні біта в 1 МК переходить в режим пониженої споживаної потужності.
IDL	PCON.0	Біт холостого ходу. Якщо біт встановлений в 1, то МК переходить в режим холостого ходу.

3.2.5. Лічильник команд

Лічильник команд (PC) призначений для формування поточної 16-розрядної адреси програмної пам'яті і 8/16-розрядної адреси зовнішньої пам'яті даних.

В склад лічильника команд входять 16-розрядні буфер PC, реґістр вказівника даних DPTR, реґістр PC, схема інкремента, реґістр адреси пам'яті.

Буфер PC здійснює зв'язок між 16-розрядною шиною PC і восьмирозрядною маґістраллю даних, забезпечуючи запис, зберігання і комутацію інформації.

Реґістр вказівника даних (DPTR) призначений для зберігання 16-розрядної адреси зовнішньої пам'яті даних. Він складається з двох 8-розрядних реґістрів DPH і DPL, що входять в блок реґістрів спеціальних функцій. Ці реґістри програмно доступні і

можуть використовуватися як два незалежних РЗП, якщо немає необхідності в зберіганні 16-розрядної адреси зовнішньої пам'яті даних.

В регістрі PC зберігається біжуча 16-розрядна адреса пам'яті програм.

Схема інкремента збільшує біжуче значення 16-розрядної адреси пам'яті програм на одиницю.

Регістр адреси пам'яті призначений для запису і зберігання виконуваної 16-розрядної адреси пам'яті програм чи 8/16-розрядної адреси зовнішньої пам'яті даних, а також для передачі даних на порт P0 при виконанні команд MOVX @Ri, A і MOVX @DPTR, A, що забезпечують запис даних через порт P0 у зовнішні пристрої.

3.2.6. Порти вводу/виводу

Порти P0, P1, P2, P3 є двонаправленими портами вводу-виводу і призначені для обміну інформацією ОМЕОМ з зовнішніми пристроями, утворюючи 32 лінії вводу-виводу. Кожен з портів містить фіксатор, який являє собою 8-розрядний регістр, що має байтову і бітову адресацію для встановлення (скидання) розрядів за допомогою програмного забезпечення.

Фізичні адреси фіксаторів P0, P1, P2, P3 складають для:

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

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

P2 - A0H, при бітовій адресації A0H-A7H;

P3 - B0H, при бітовій адресації B0H-B7H.

Крім роботи як звичайних портів вводу/виводу, лінії портів P0-P3 можуть виконувати ряд додаткових функцій, описаних нижче.

Через порт P0:

- виводиться молодший байт адреси A0-A7 при роботі з зовнішньою пам'яттю програм і зовнішньою пам'яттю даних;

- видається з ОМЕОМ і приймається в ОМЕОМ байт даних при роботі з зовнішньою пам'яттю (при цьому обмін байтом даних і вивід молодшого байта адреси зовнішньої пам'яті мультиплексовані в часі);

- задаються дані при програмуванні внутрішнього ППЗП і читається вміст внутрішньої пам'яті програм.

Через порт P1:

- задається молодший байт адреси при програмуванні внутрішнього ППЗП і при читанні внутрішньої пам'яті програм.

Через порт Р2:

- виводиться старший байт адреси А8-А15 при роботі з зовнішньою пам'яттю програм і зовнішньою пам'яттю даних (для зовнішньої пам'яті даних - тільки при використанні команд MOVX A, @DPTR і MOVX @DPTR, A, які формують 16-розрядну адресу);

- задається старший байт (розряди А8-А14) адреси при програмуванні внутрішнього ППЗП і при читанні внутрішньої пам'яті програм.

Кожна лінія порту Р3 виконує індивідуальну альтернативну функцію:

Р3.0 - RxD, вхід послідовного порту, призначений для вводу послідовних даних в приймач послідовного порту;

Р3.1 - TxD, вихід послідовного порту, призначений для виводу послідовних даних з передавача послідовного порту;

Р3.2 - $\overline{\text{INT0}}$, використовується як вхід 0 зовнішнього запиту переривання;

Р3.3 - $\overline{\text{INT1}}$, використовується як вхід 1 зовнішнього запиту переривання;

Р3.4 - Т0, використовується як вхід лічильника зовнішніх подій Т/Л0;

Р3.5 - Т1, використовується як вхід лічильника зовнішніх подій Т/Л1;

Р3.6 - $\overline{\text{WR}}$, строб запису в зовнішню пам'яті даних, вихідний сигнал, що супроводжує вивід даних через порт Р0 при використанні команд MOVX @Ri, A і MOVX @DPTR, A.

Р3.7 - $\overline{\text{RD}}$, строб зчитування з зовнішньої пам'яті даних, вихідний сигнал, що супроводжує ввід даних через порт Р0 при використанні команд MOVX A, @Ri і MOVX @DPTR.

Альтернативна функція будь-якої з ліній порту Р3 реалізується лише в тому випадку, коли у відповідному цій лінії розряді фіксатора міститься «1». В протилежному випадку на лінії порту Р3 буде присутній «0».

На рис.3.6-3.9 показані функціональні схеми одного виводу в кожному з портів для серії 1816. Для серії 1830 конфігурація портів Р1-Р3 відрізняється лише вихідним підсилювальним каскадом.

Електричні параметри портів Р0-Р3 наведені в таблиці 3.15 (статичні параметри) та таблиці 3.16 (динамічні параметри).

На рис.3.6-3.9 фіксатор показаний у вигляді D-тригера, тактованого внутрішнім сигналом «Запис у фіксатор», який виробляється при записуванні даних в порт. Вихід тригера Q може бути підключений на внутрішню шину ОМЕОМ через буфер

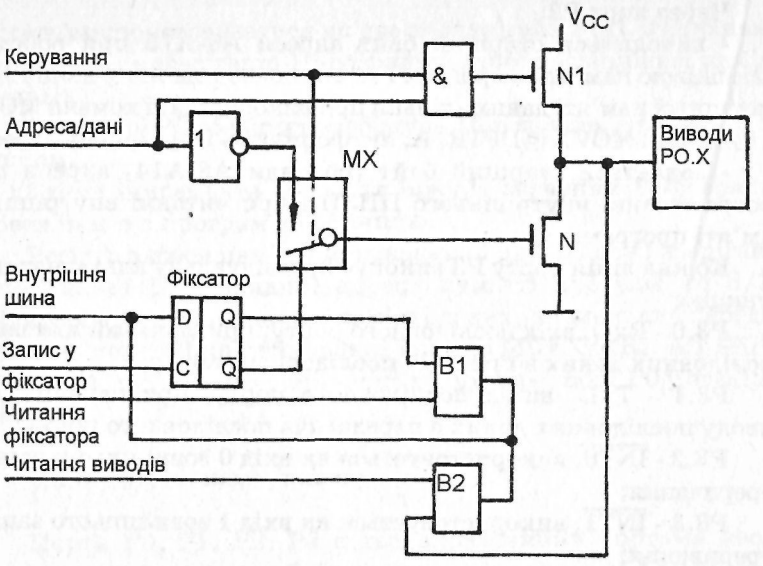


Рис. 3.6. Розряд порту 0

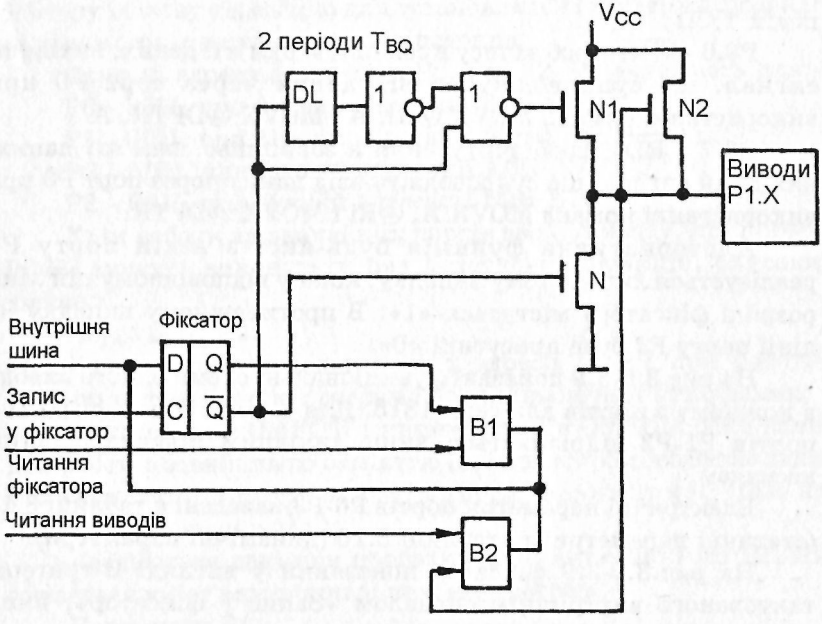


Рис. 3.7. Розряд порту 1

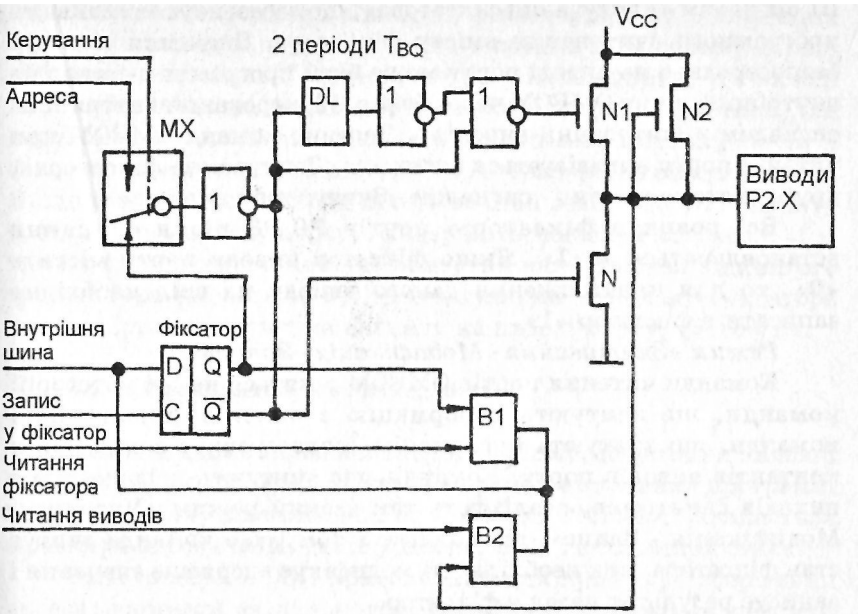


Рис. 3.8. Розряд порту 2

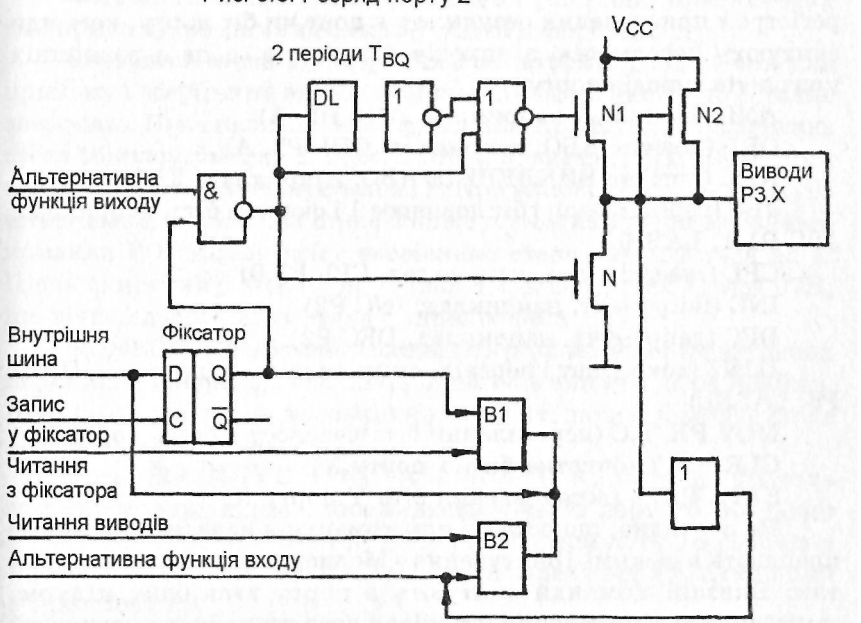


Рис. 3.9. Розряд порту 3

В1 сиґналом «Зчитування фіксатора», що забезпечує можливість програмного зчитування вмісту фіксатора. Значення сиґналу безпосередньо на виводі порту може бути програмно зчитано на внутрішню шину ОМЕОМ через буфер В2, керований внутрішнім сиґналом «Зчитування виводів». Частина команд ОМЕОМ при читанні порту активізується сиґналом «Зчитування фіксатора», друга частина команд - сиґналом «Зчитування виводів».

Всі розряди фіксаторів портів Р0-Р3 після скидання встановлюються в «1». Якщо фіксатор виводу порту містить «0», то для налагодження даного виводу на ввід необхідно записати в фіксатор «1».

Режим «Зчитування - Модифікація - Запис»

Команди читання портів ОМЕОМ діляться на дві категорії: команди, що зчитують інформацію з виходів фіксаторів, і команди, що зчитують інформацію безпосередньо з зовнішніх контактів виводів порту. Команди, що зчитують інформацію з виходів фіксаторів, реалізують так званий режим «Читання - Модифікація - Запис», що полягає в тому, що команда зчитує стан фіксатора, при необхідності модифікує одержане значення і записує результат назад у фіксатор.

Нижче наведені команди, що працюють в режимі «Читання - Модифікація - Запис». У всіх випадках, коли операндом і реґістром призначення результату є порт чи біт порту, команди зчитують інформацію з виходів фіксаторів, а не з зовнішніх контактів виводів порту.

ANL (Логічне І, наприклад, ANL P1, A)

ORL (логічне АБО, наприклад, ORL P2, A)

XRL (логічне ВИКЛЮЧНЕ АБО, наприклад, XRL P3, A)

JBC (перехід, якщо біт дорівнює 1 і очистка біта, наприклад, JBC P1.1, LABEL)

CPL (інверсія біта, наприклад, CPL P3.0)

INC (інкремент, наприклад, INC P2)

DEC (декремент, наприклад, DEC P2)

DJNZ (декремент і перехід, якщо не нуль, наприклад, DJNZ P3, LABEL)

MOV PX.Y,C (пересилання біта переносу в біт Y порту X)

CLR PX.Y (очистка біта Y порту X)

SETB PX.Y (встановлення біта Y порту X)

Не очевидно, що останні три команди в наведеному списку працюють в режимі «Зчитування - Модифікація - Запис», але це так. Вказані команди зчитують з порту весь байт цілком, модифікують адресований біт, після чого записують одержаний новий байт назад у фіксатор порту.

Читання інформації з виходів фіксаторів, а не з зовнішніх контактів виводів порту дозволяє виключити можливу в ряді випадків неправильну інтерпретацію рівня напруги на виводі порту. Наприклад, вивід порту може використовуватися для керування базою n-p-n транзистора. В цьому випадку, коли у фіксатор виводу порту записується «1», транзистор відкривається. Якщо після цього ОМЕОМ прочитає стан зовнішнього контакту розглянутого виводу порту, то одержить значення логічного «0», так як на контакті в цей час присутня напруга бази відкритого транзистора (не більше 0,7В). Зчитування ж виходу фіксатора покаже істинне значення сигналу на виводі порту, тобто «1».

3.2.7. Резидентна пам'ять даних

Пам'ять даних призначена для прийому, зберігання і видачі інформації, що використовується в процесі виконання програми. Пам'ять даних, розташована на кристалі ОМЕОМ, складається з регістра адреси ОЗП, дешифратора, ОЗП і вказівника стека.

Регістр адреси ОЗП призначений для прийому і зберігання адреси, вибраної за допомогою дешифратора комірки пам'яті, яка може містити як біт, так і байт інформації.

ОЗП містить 128 восьмирозрядних регістрів, призначених для прийому, зберігання і видачі різної інформації.

Вказівник стека є 8-розрядним регістром, призначеним для прийому і зберігання адреси комірки стека, до якої було останнє звертання. При виконанні команд LCALL, ACALL вміст вказівника стека збільшується на 2. При виконанні команд RET, RETI вміст вказівника стека зменшується на 2. При виконанні команди PUSH direct вміст вказівника стека збільшується на 1. При виконанні команди POP direct вміст вказівника стека зменшується на 1. Після скидання у вказівнику стека встановлюється адреса 07H, що відповідає початку стека з адресою 08H.

В ОМЕОМ передбачена можливість розширення пам'яті даних шляхом підключення зовнішніх пристроїв ємністю до 64 Кбайтів. При цьому звертання до зовнішньої пам'яті даних можливе лише за допомогою команд MOVX.

Команди MOVX @Ri, A і MOVX A, @Ri формують восьмирозрядну адресу, яка видається через порт P0. Команди MOVX @DPTR, A і MOVX @A, DPTR формують 16-розрядну адресу, молодший байт якої видається через порт P0, а старший - через порт P2.

Байт адреси, що видається через порт P0, повинен бути зафіксований у зовнішньому регістрі по спаду сигналу ALE, тому

що надалі лінії порту P0 використовуються як шина даних, через яку байт даних приймається з пам'яті при зчитуванні чи видається в пам'ять даних при запису. При цьому зчитування стробується сигналом ОМЕОМ $\overline{RD}$, а запис - сигналом ОМЕОМ $\overline{WR}$. При роботі з внутрішньою пам'яттю даних сигнали $\overline{RD}$ і $\overline{WR}$ не формуються.

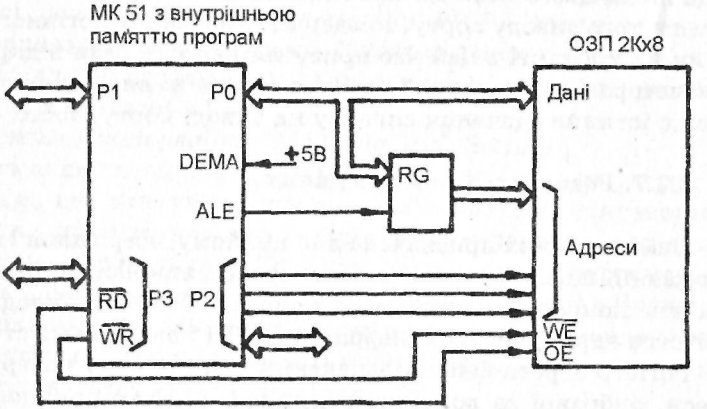
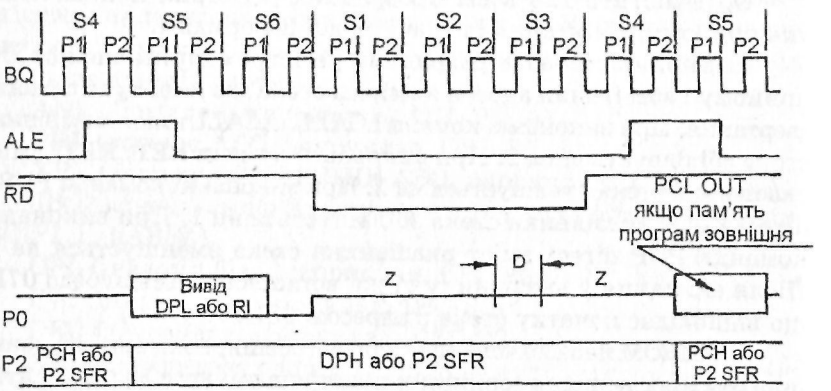
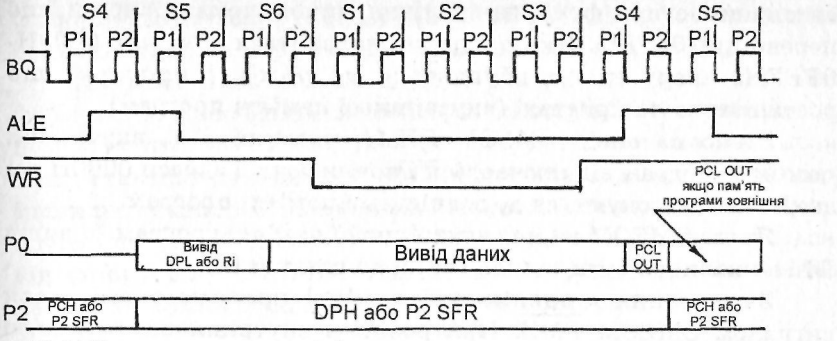


Рис. 3.10.Сторінкова організація зовнішньої пам'яті даних



PCH OUT - видача молодшого байта лічильника команд PC
PCH - старший байт лічильника команд PC
DPL, DPH - відповідно молодший і старший байти регістрів вказівника даних DPTR, який використовується як регістр адрес в командах MOVX A, @DPTR і MOVX @DPTR, A
P2 SFR - фіксатори порту P2
Ri - регістри R0 і R1, які використовуються як регістри адреси в командах MOVX A, @Ri і MOVX @Ri, A
Z - високоімпедансний стан
D - період, протягом якого дані з P0 вводяться в ОМЕОМ

Рис. 3.11. Цикл читання з зовнішньої пам'яті даних



PCH OUT - видача молодшого байта лічильника команд PC
PCH - старший байт лічильника команд PC
DPL, DPH - відповідно молодший і старший байти реґістрів вказівника даних DPTR, який використовується як реґістр адрес в командах MOVX A, @DPTR і MOVX @DPTR, A
P2 SFR - фіксатори порту P2
Ri - реґістри R0 і R1, які використовуються як реґістри адреси в командах MOVX A, @Ri і MOVX @Ri, A

Рис. 3.12. Цикл запису в зовнішню пам'ять даних

На рис. 3.10 показана сторінкова організація зовнішньої пам'яті даних. Ця схема дозволяє працювати з пам'яттю даних ємністю 2 Кбайти, використовуючи команди типу MOVX @Ri. Порт P0 при цьому працює як мультиплексована шина адреса/дані, а три лінії порту P2 адресують сторінки зовнішнього ОЗП. Останні 5 ліній порту P2 можуть використовуватися як лінії вводу/виводу.

На рис. 3.11 і 3.12 відповідно приведені діаграми циклів зчитування і запису при роботі ОМЕОМ з зовнішньою пам'яттю даних.

3.2.8. Резидентна пам'ять програм

Пам'ять програм призначена для зберігання програм і має окремий від пам'яті даних адресний простір обсягом 64 Кбайти, причому для мікросхем КР1816ВЕ51 і для КР1830ВЕ51 частина пам'яті програм з адресами 0000H-0FFFH розташована на кристалі ОМЕОМ. Пам'ять програм, що розташована на кристалі, складається з 12-розрядного дешифратора і ПЗП ємністю 4Кх8 бітів для мікросхем КР1816ВЕ51, КР1830ВЕ51 або ППЗП з ультрафіолетовим стиранням ємністю 4Кх8 біт для КМ1816ВЕ751. Запис програм в ПЗП відбувається під час виготовлення кристалів.

Якщо на вивід ОМЕОМ DEMA подана напруга живлення U_{CC}, то звертання до зовнішньої пам'яті програм відбувається

автоматично при формуванні лічильником команд адреси, що перевищує 0FFFH. Якщо адреса знаходиться в межах 0000H-0FFFH, звертання відбувається до пам'яті програм, що розташована на кристалі (внутрішньої пам'яті програм).

Якщо на вивід ОМЕОМ DEMA поданий «0», внутрішня пам'ять програм відключається і починаючи з адреси 0000H всі звертання виконуються до зовнішньої пам'яті програм.

Якщо ОМЕОМ не має внутрішньої пам'яті програм, її вивід DEMA повинен бути підключений до шини 0 В.

Зчитування з зовнішньої пам'яті програм стробується сигналом ОМЕОМ $\overline{\text{PME}}$. При роботі з внутрішньою пам'яттю програм сигнал PME не формується. ОМЕОМ не мають інструкцій і апаратних засобів для програмного запису в пам'ять програм.

При звертаннях до зовнішньої пам'яті програм завжди формується 16-розрядна адреса, молодший байт якої видається через порт P0, а старший - через порт P2. При цьому байт адреси, що видається через порт P0, повинен бути зафіксований у зовнішньому регістрі по спаду сигналу ALE, тому що надалі лінії порту P0 використовуються в ролі шини даних, по якій байт із зовнішньої пам'яті програм вводиться в ОМЕОМ.

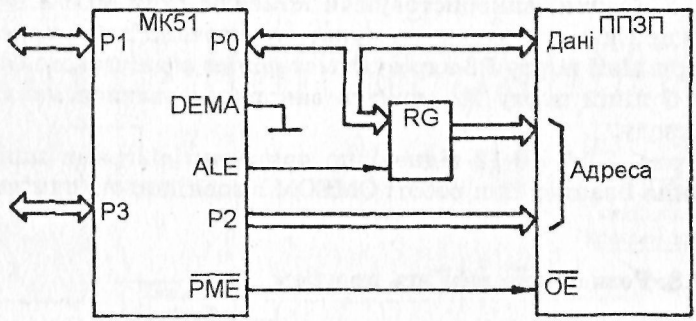
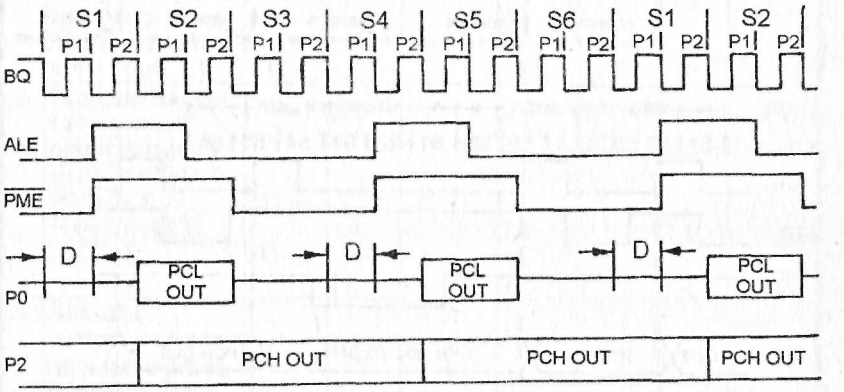


Рис. 3.13. Схема включення МК51 з зовнішнім ППЗП програм

На рис. 3.13 показана функціональна схема включення ОМЕОМ МК51 з зовнішнім ППЗП програм. Порт P0 працює як мультиплексована шина адреса/дані: видає молодший байт лічильника команд, а потім переходить у високоімпедансний стан і очікує приходу байта з ППЗП програм. Коли молодший байт адреси знаходиться на виходах порту P0, сигнал ALE записує його в адресний регістр RG. Старший байт адреси знаходиться на виходах порту P2 протягом всього часу звертання до ППЗП.

Сиґнал $\overline{\text{PME}}$ дозволяє вибірку байта з ППЗП, після чого вибраний байт поступає на порт P0 МК51 і вводиться в ОМЕОМ.

На рис. 3.14 і 3.15 подані діаграми, що показують формування відповідних сиґналів при роботі ОМЕОМ з зовнішньою пам'яттю програм. Як видно з діаграм, при роботі з зовнішньою пам'яттю програм сиґнал $\overline{\text{PME}}$ формується двічі в кожному машинному циклі незалежно від кількості байтів у команді. Якщо другий вибраний байт у поточній команді не використовується, він іґнорується ОМЕОМ. Надалі при переході до виконання наступної команди байт буде введений повторно.

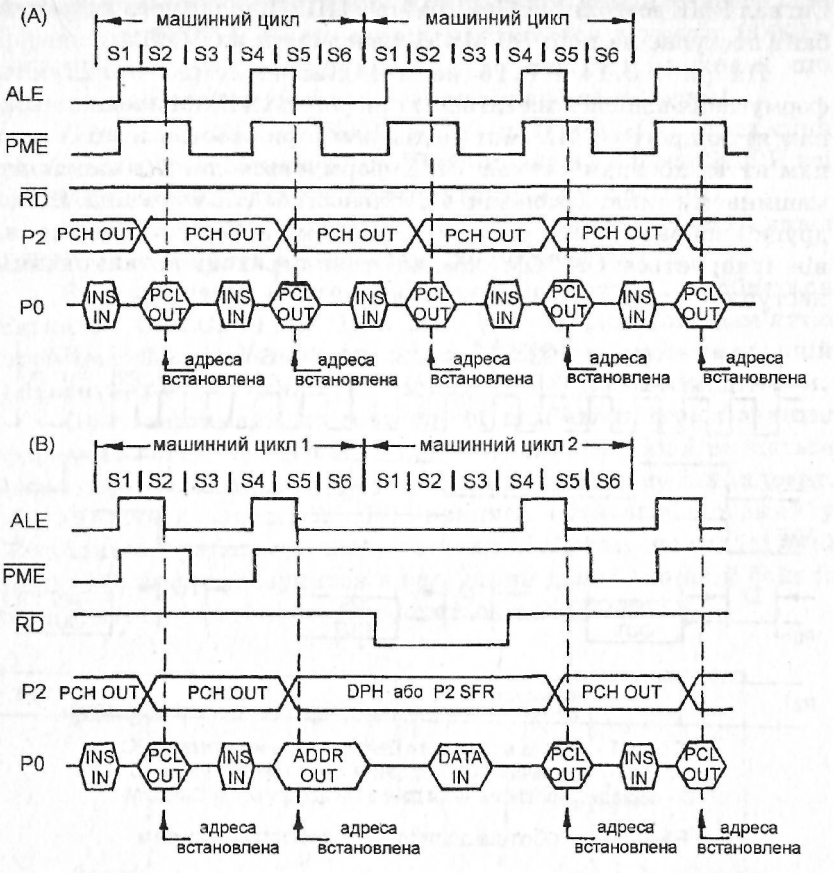


PCL OUT - видача молодшого байта лічильника команд PC
PCH OUT - видача старшого байта лічильника команд PC
D - періоди, протягом яких дані з P0 вводяться в ОМЕОМ

Рис. 3.14. Робота з зовнішньою пам'яттю програм

Якщо виконується команда MOVX (рис.3.15, діаграма B), два сиґнали $\overline{\text{PME}}$ не формуються, тому що порт P0 звільнюється для адресації і обміну даними з зовнішньою пам'яттю даних.

Коли ОМЕОМ працює з внутрішньою пам'яттю програм, сиґнал $\overline{\text{PME}}$ не формується і адреса на портах P0 і P2 не видається. Тим не менше, сиґнал ALE буде формуватися двічі в кожному машинному циклі завжди, за винятком випадку команди MOVX (в цьому випадку один сиґнал ALE пропускається). Таким чином, якщо не використовуються команди MOVX, сиґнал ALE може бути задіяний в ролі вихідного синхросиґналу.



(A) - без виконання команди MOVX
(B) - з виконанням команди MOVX
PCL OUT - видача молодшого байта лічильника команд PC
PCH OUT - видача старшого байта лічильника команд PC
DPH - старший байт регістра вказівника даних DPTR, який використовується як регістр посередньої адреси в командах MOVX A, @DPTR і MOVX @DPTR, A
P2 SFR - фіксатори порту P2
INS IN - ввід байта інструкції з пам'яті програм
ADDR OUT - видача молодшого байта адреси зовнішньої пам'яті даних з регістрів R0, R1 або регістра DPL
Задній фронт ALE стробує адресу на порті P0. В цей момент адреса гарантовано встановлена

Рис. 3.15. Цикли роботи з зовнішньою пам'яттю програм

3.3. Функціонування МК51

Статичні параметри мікросхем наведені в таблиці 3.15.

Таблиця 3.15

N п/п	Параметри	Буквенні позн.	Значення параметрів					
			КР1816ВЕ31 КР1816ВЕ51		КР1816ВЕ751		КР1830ВЕ31 КР1830ВЕ51	
			min	max	min	max	min	max
1	2	3	4	5	6	7	8	9
1	Напруга живлення, В	U_{CC}	4.5	5.5	4.75	5.25	4.5	5.5
2	Напруга програмуючого живлення високого рівня, В	U_{PR}	-	-	6 20.5	21.5	-	-
3	Вихідна напруга високого рівня сигналів: P1(0-7), P2(0-7), P3(0-7),B P0(0-7), ALE, PME, B	U_{OH} U_{OH1}	2.4 2.4	- -	2.4 2.4	- -	2.4 2.4	- -
4	Вихідна напруга низького рівня сигналів: P1(0-7), P2(0-7), P3(0-7),B P0(0-7), ALE, PME, B	U_{OL} U_{OL1}	- -	0.45 0.45	- -	0.45 0.45	- -	0.45 0.45
5	Вихідний струм високого рівня сигналів: P1(0-7), P2(0-7), P3(0-7), P0(0-7), ALE, PME, мА	I_{OH} I_{OH1}	-0.08 -0.4	- -	-0.08 -0.4	- -	-0.08 -0.4	- -
6	Вихідний струм низького рівня сигналів: P1(0-7), P2(0-7), P3(0-7), P0(0-7), ALE, PME, мА	I_{OL} I_{OL1}	- -	1.6 3.2	- -	1.6 2.4	- -	1.6 3.2
7	Вхідна напруга високого рівня сигналів, крім RST і BQ2*, В	U_{IN}	2.0	-	2.0	-	2.0	-

1	2	3	4	5	6	7	8	9
8	Вхідна напруга високого рівня сигналів RST і BQ2*, В	U_{IN1}	2.5	U_{CC} +0.5	2.5	U_{CC} +0.5	3.8	U_{CC} +0.5
9	Вхідна напруга низького рівня, В	U_{IL}	-0.5	0.8	-0.5	0.8	-0.5	0.8
10	Вхідний струм високого рівня сигналів: RST**, мкА DEMA, мкА	I_{IN} I_{INL1}	- -	| -500.0 | -	- -	| -500.0 | | -500.0 |	- -	| -500.0 | -
11	Вхідний струм низького рівня сигналів: P1(0-7), P2(0-7), P3(0-7), мкА BQ2*, мА DEMA, мА	I_{IL} I_{IL1} I_{IL2}	- - -	| -800.0 | | -2.5 | -	- - -	| -500.0 | | -2.5 | | -15.0 |	- - -	| -50.0 | - -
12	Струм тікання сигналів, мкА	I_L	-10.0	10.0	-10.0	10.0	-10.0	10.0
13	Вихідний струм в стані "вимкнено", мкА	I_{OZ}	-10.0	-10.0	10.0	-10.0	10.0	-10.0
14	Струм споживання, мА	I_{CC}	-	150.0	-	220.0	-	18.0
15	Струм споживання в режимі резервного живлення, мА	I_{CCS}	-	10.0	-	-	-	-
16	Струм споживання в режимі зберігання вмісту регістрів спецфункцій (в режимі холостого ходу), мА	I_{CCS1}	-	-	-	-	-	4.2
17	Струм споживання в режимі зберігання вмісту ОЗП (в режимі мікроспоживання), мкА	I_{CCS2}	-	-	-	-	-	50.0

продовження таблиці 3.15

1	2	3	4	5	6	7	8	9
18	Струм споживання по виводу U_{PR} при програмуванні, мА	I_{PR}	-	-	-	30.0	-	-
19	Внутрішній опір в колі "скидання", кОм	R_{RST}	-	-	-	-	40.0	125.0
20	Ємність входу/виходу, пФ	$C_{I/O}$	-	20.0	-	20.0	-	20.0
21	Вхідна ємність, пФ	C_I	-	10.0	-	10.0	-	10.0

* - для серії КР 1830 - BQ1.

** - в серії КР1830 значення приведено для сиґналів P1 (0–7), P2(0–7), P3(0–7).

Динамічні параметри мікросхем наведені в таблиці 3.16.

Таблиця 3.16

N п/п	Параметри	Буквенні позн.	Значення параметрів КР1816ВЕ51	
			не більше	не менше
1	2	3	4	5
Динамічні параметри				
1	Період слідування імпульсів тактових сиґналів BQ1 (BQ2), нс	$T_{BQ} = t$	83.3	286
2	Час фронту наростання сиґналу BQ1 (BQ2), нс	t_{BQ}, LH	-	20
3	Час фронту спаду сиґналу BQ1 (BQ2), нс	t_{BQ}, HL	-	20
4	Мінімальний час циклу, нс	tcy min	12t	-
5	Тривалість сиґналу ALE, нс	tw(ALE)	2t - 40	-
6	Тривалість сиґналу $\overline{PME}$, нс	tw($\overline{PME}$)	3t - 60	-
7	Тривалість сиґналу $\overline{RD}$, нс	tw($\overline{RD}$)	6t - 100	-
8	Тривалість сиґналу $\overline{WR}$, нс	tw($\overline{WR}$)	6t - 100	-
9	Час затримки сиґналу ALE відносно сиґналів адреси A(7-0), A(8-15), нс	td(A, ZH/ZL - ALE, HL)	t - 30	-

1	2	3	4	5
10	Час затримки сигналів адреси A(7-0) відносно сигналу ALE, нс	td(ALE, HL - A, HZ/LZ)	t - 35	-
11	Час затримки сигналу $\overline{PME}$ відносно сигналу ALE, нс	td(ALE, HL - $\overline{PME}$, HL)	t - 25	-
12	Час затримки сигналів адреси A(7-0) відносно сигналу $\overline{PME}$, нс	td($\overline{PME}$, LH - A, ZH/ZL)	t - 8	-
13	Час затримки сигналу $\overline{PME}$ відносно сигналів адреси A(7-0), A(8-15), нс	td(A, HZ/LZ - $\overline{PME}$, HL)	0	-
14	Час затримки сигналу $\overline{RD}$ відносно сигналу ALE, нс	td(ALE, HL - $\overline{RD}$, HL)	3t - 50	3t + 50
15	Час затримки сигналу $\overline{WR}$ відносно сигналу ALE, нс	td(ALE, HL - $\overline{WR}$, HL)	3t - 50	3t + 50
16	Час затримки сигналу $\overline{RD}$ відносно сигналів адреси A(7-0), A(8-15), нс	td(A, ZH/ZL - $\overline{RD}$, HL)	4t - 130	-
17	Час затримки сигналу $\overline{WR}$ відносно сигналів адреси A(7-0), A(8-15), нс	td(A, ZH/ZL - $\overline{WR}$, HL)	4t - 130	-
18	Час затримки сигналу $\overline{WR}$ відносно сигналів даних, нс	td(D, LH/HL - $\overline{WR}$, HL) td(D, LH/HL - WR, LH)	t - 70 7t - 150	- -
19	Час затримки сигналів даних D(7-0) відносно сигналу $\overline{WR}$, нс	td($\overline{WR}$, LH - D, HZ/LZ)	t - 50	-
20	Час затримки сигналів даних D(7-0) відносно сигналу $\overline{RD}$, нс	td($\overline{RD}$, HL - A, HZ/LZ)	-	0
21	Час затримки сигналу ALE відносно сигналу $\overline{WR}$, нс	td($\overline{WR}$, LH - ALE, LH)	t - 50	t + 50
22	Час затримки сигналу ALE відносно сигналу $\overline{RD}$, нс	td($\overline{RD}$, LH - ALE, LH),	t - 50	t + 50
23	Час зберігання сигналів INS відносно сигналу $\overline{PME}$, нс	tv($\overline{PME}$, LH - INS, HZ/LZ)	0	t - 20
24	Час зберігання сигналів даних D(0-7) відносно сигналу $\overline{RD}$, нс	tv($\overline{RD}$, LH - D, HZ/LZ)	-	2t - 70

1	2	3	4	5
25	Час встановлення сигналу INS відносно сигналу ALE, нс	tsu(ALE, HL - INS, ZH/ZL)	-	4t - 150
26	Час встановлення сигналу INS відносно сигналу $\overline{PME}$, нс	tsu($\overline{PME}$, HL - INS, ZH/ZL)	-	3t - 150
27	Час встановлення сигналу INS відносно сигналів адреси A(7-0), нс	tsu(A, ZH/ZL - INS, ZH/ZL)	-	5t - 150
28	Час встановлення сигналів даних D(0-7) відносно сигналу RD, нс	tsu($\overline{RD}$, HL - D,ZH/ZL)	-	5t - 165
29	Час встановлення сигналів даних D(0-7) відносно сигналу ALE, нс	tsu(ALE, HL - D, ZH/ZL)	-	8t - 150
30	Час встановлення сигналів даних D(0-7) відносно сигналів адреси A(7-0), нс	tsu(A, ZH/ZL - D, ZH/ZL)	-	9t - 165

Динамічні параметри при програмуванні і перевірці РПЗП при t=(25±5)С

N п/п	Назва сигналу	Буквенне позначення	1816ВЕ751	1816ВЕ51	1830ВЕ51
1	2	3	4	5	6
31	Час затримки сигналів даних D(0-7) відносно сигналів адреси A(7-0), нс	td(A, ZH/ZL - D, ZH/ZL)	не більше 48t	-	-
32	Час затримки сигналів даних D(0-7) відносно сигналу $\overline{E}$, нс	td($\overline{E}$, HL - D, ZH/ZL) td($\overline{E}$, LH - D, HZ/LZ)	не більше 48t	-	-
33	Тривалість сигналу $\overline{PR}$ низького рівня, мс	tw($\overline{PR}$, L)	45-55	-	-
34	Час встановлення адреси відносно сигналу $\overline{PR}$, нс	tsu(A, ZH/ZL - $\overline{PR}$, HL)	не більше 48t	-	-
35	Час зберігання адресного сигналу відносно сигналу $\overline{PR}$, нс	tv($\overline{PR}$, LH - A, HZ/LZ)	не менше 48t	-	-

1	2	3	4	5	6
36	Час встановлення даних D(0-7) відносно сигналу $\overline{PR}$, нс	tsu(D, ZH/ZL - $\overline{PR}$, HL)	не менше 48t	-	-
37	Час зберігання сигналу даних D(0-7) відносно сигналу $\overline{PR}$, нс	tv($\overline{PR}$, HL - D, HZ/LZ)	не менше 48t	-	-
38	Час встановлення сигналу U_{PR} відносно $\overline{PR}$, мкс	tsu(U_{PR}, LH - $\overline{PR}$, HL)	не менше 48t	-	-
39	Час зберігання сигналу U_{PR} відносно сигналу $\overline{PR}$, мкс	tv($\overline{PR}$, HL - U_{PR}, HL)	не менше 48t	-	-
40	Час встановлення сигналу U_{PR} відносно сигналу $\overline{E}$, нс	tsu($\overline{E}$, HL - U_{PR}, LH)	не менше 48t	-	-
41	Частота слідування імпульсів BQ, МГц	f_{BQ}	4-6	4-6	4-6
42	Час затримки сигналів даних D(7-0) відносно сигналів адреси AP1(0-7), AP2(0-3) при перевірці внутрішнього ПЗП	td(AP1, LH/HL - D, ZH/ZL) td(AP2, LH/HL - D, ZH/ZL)	-	не більше 48t	не більше 48t
43	Час затримки сигналів даних D(7-0) відносно сигналу дозволу P2(7) при перевірці внутрішнього ПЗП	td(P2, HL - D, ZH/ZL) td(P2, LH - D, HZ/LZ)	-	не більше 48t	не більше 48t

Пояснення до таблиці 3.16.

1. Для ВІС КР1830ВЕ51(ВЕ31) $t=T_{BQ1}$. Для ВІС КР1816ВЕ51 (ВЕ31) і КМ1816ВЕ751 $t=T_{BQ2}$.

2. Часові діаграми подані на рис. 3.17-3.19.

3. Символ "LH"("HL") означає перехід сигналу з стану низького (високого) рівня в протилежний рівень. Символ "ZH"("ZL") означає перехід сигналу з високоімпедансного стану в стан високого (низького) рівня. Символ "HZ"("LZ") означає перехід сигналу високого (низького) рівня в високоімпедансний стан.

Граничні значення електричних режимів експлуатації наведені в таблиці 3.17.

Таблиця 3.17

N п/п	Параметри	Буквенні позн.	Значення параметрів					
			КР1816ВЕ31 КР1816ВЕ51		КР1816ВЕ751		КР1830ВЕ31 КР1830ВЕ51	
			min	max	min	max	min	max
1	2	3	4	5	6	7	8	9
1.	Напруга живлення, В	U_{CC}	-0.51	7.0	-0.51	7.0	-0.51	7.0
2.	Напруга програмуючого живлення високого рівня, В	U_{PR}	-	-	-	23.0	-	-
3.	Вихідна напруга на виводах, В	U_I	0.51	7.0	0.51	7.0	0.51	7.0
4.	Вихідний струм низького рівня для виводів P0, ALE, $\overline{PME}$, мА інших виводів, мА	I_{OL1} I_{OL}	- -	5.0 3.0	- -	3.5 2.0	- -	5.0 3.0
5.	Вихідний струм високого рівня для виводів P0, ALE, $\overline{PME}$, мА інших виводів, мА	I_{OH1} I_{OH}	-0.8 -0.2	- -	-0.4 -0.08	- -	-0.8 -0.2	- -
6.	Ємність навантаження для виводів P0, ALE, $\overline{PME}$, мА інших виводів, мА	C_{L1} C_L	- -	500.0 100.0	- -	500.0 100.0	- -	500.0 100.0
7.	Кількість циклів перепрограмування	$N_{цикл}$				25		

3.3.1. Режими роботи

ОМЕОМ можуть працювати в таких режимах:

- тільки з зовнішньою пам'яттю програм (всі модифікації ОМЕОМ);

- тільки з внутрішньою пам'яттю програм (тільки КР1816ВЕ51, КМ1816ВЕ751 і КР1830ВЕ51);

- з внутрішньою і зовнішньою пам'яттю даних (всі модифікації ОМЕОМ);

- в режимі програмування внутрішньої пам'яті програм (тільки КМ1816ВЕ751);

- в режимі перевірки внутрішньої пам'яті програм (тільки КР1816ВЕ51, КМ1816ВЕ751 і КР1830ВЕ51).

Режим роботи встановлюється комбінацією вхідних і вихідних сигналів.

Ініціалізація (скидання) мікросхем здійснюється сигналом RST (активний високий рівень напруги) при умові подачі на мікросхему зовнішнього сигналу синхронізації чи при підключеному кварці. Вхід RST є входом внутрішнього тригера Шмітта.

Для того, щоб скидання мікросхеми здійснилося гарантовано, тривалість сигналу високого рівня на вході RST повинно бути не менше двох машинних циклів ОМЕОМ (24 періоди частоти синхронізації f_{BQ}). При надходженні зовнішнього сигналу скидання на вхід RST і виконанні вказаної умови ОМЕОМ формує сигнал скидання.

Зовнішній сигнал скидання є асинхронним по відношенню до внутрішньої синхронізації ОМЕОМ. Стан виводу RST перевіряється в фазі S5P2 кожного машинного циклу. Після подачі сигналу високого рівня на вхід RST ОМЕОМ продовжує роботу протягом часу від 19 до 31 періоду частоти f_{BQ} (формується ALE, $\overline{PME}$ і т.п.), після чого ALE і $\overline{PME}$ встановлюються в «1» і знаходяться в цьому стані доти, поки на вході RST присутній активний (одиничний) сигнал скидання. Після подачі на вхід RST рівня «0» проходить від 1 до 2 машинних циклів до початку формування сигналів ALE і $\overline{PME}$.

При подачі сигналу скидання на вхід RST внутрішній алгоритм скидання ОМЕОМ здійснює наступні дії:

- встановлює лічильник команд PC і всі регістри спеціальних функцій, крім фіксаторів портів P0-P3, вказівника стека SP і регістра SBUF, в нуль;

- вказівник стека приймає значення рівне 07H;

- забороняє всі джерела переривань, роботу таймерів-лічильників послідовного порту;

- вибирає банк регістрів 0 ОЗП, підготовлює порти P0-P3 для приймання даних і визначає виводи ALE і $\overline{PME}$ як входи для зовнішньої синхронізації;

- в регістрах спеціальних функцій PCON, IP і IE резервні біти приймають випадкові значення, а всі решта біти скидаються

в нуль;

- в регістрах SBUF встановлюються випадкові значення;
- встановлює фіксатори портів P0-P3 в «1».

Узагальнені дані за станами регістрів після скидання адреси вказані в таблиці 3.18.

Таблиця 3.18

Регістр	Інформація
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0-P3	0FFH
IP	XXX00000B
IE	0XX00000B
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
SCON	00H
SBUF	Не визначена
PCON	0XXX0000B для серії 1830 0XXXXXXB для серії 1816.

Сигнал скидання на вході RST не впливає на внутрішній ПЗП даних. Після включення живлення вміст комірок внутрішнього ОЗП даних приймає випадкові значення.

На рис.3.16 показана схема підключення ОМЕОМ для реалізації автоматичного скидання при включенні живлення.

Для n-МОН ОМЕОМ автоматичне скидання при включенні живлення U_{CC} може бути реалізоване підключенням входу RST до U_{CC} через конденсатор ємністю 10 мкФ і до шини 0 В через резистор 8,2 кОм. Для КМОН ОМЕОМ цей резистор не потрібний, але його наявність не принесе шкоди. КМОН ОМЕОМ містять внутрішній резистор, включений між RST і виводом 0 В. Якщо використовувати лише внутрішній резистор, ємність конденсатора може бути зменшена до 1 мкФ.

Для того, щоб при включенні живлення скидання було гарантовано виконано, вивід RST має утримуватися в стані

високого рівня протягом часу, достатнього для запуску тактового ґенератора ОМЕОМ ще мінімум два машинних цикли. Час запуску тактового ґенератора ОМЕОМ залежить від його частоти роботи і для 10 МГц кварцового резонатора складає в середньому 1 мс, а для 1 МГц кварцового резонатора - 10 мс.

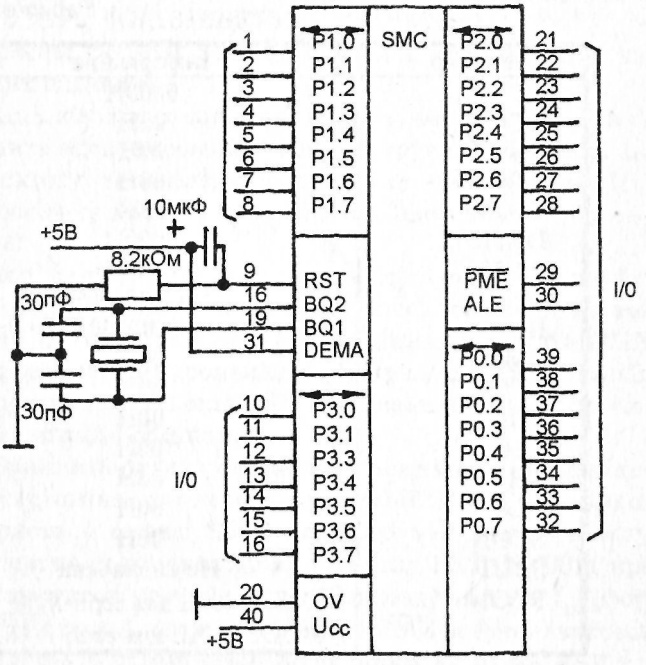


Рис. 3.16. Схема підключення ОМЕОМ для реалізації автоматичного скидання після включення живлення

Представлена на рис. 3.16 ланка скидання при швидкому зменшенні напруги живлення викликає появу на вході RST від'ємної напруги, яка не є небезпечною для мікросхем завдяки наявності в ОМЕОМ внутрішньої схеми захисту.

Виводи портів знаходяться у випадковому стані до моменту запуску тактового ґенератора ОМЕОМ і тільки після цього внутрішній сиґнал скидання записує «1» в фіксатори портів, налагоджуючи їх на вивід.

Вмикання живлення без забезпечення ґарантованого скидання може привести до того, що ОМЕОМ почне виконання програми з деякої випадкової адреси. Це пояснюється тим, що лічильник команд PC не буде скинутий в 0000H.

При роботі з зовнішньою пам'яттю програм видача молодших

розрядів адреси (A0-A7) здійснюється через порт P0 (P0.0, ...,P0.7). При цьому адреса фіксується за сигналом ALE, а команди приймаються за сигналом $\overline{PME}$. Старші розряди адреси A8-A15 видаються через P2 (P2.0, ..., P2.7).

Часові діаграми роботи ОМЕОМ з зовнішньою пам'яттю програм представлені на рис. 3.17.

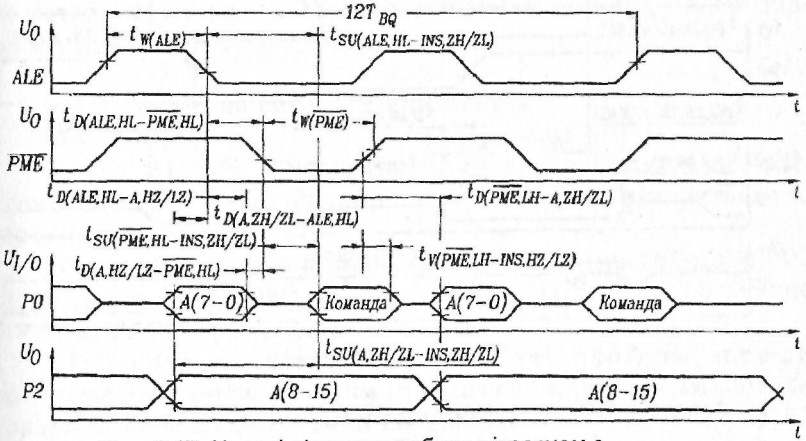


Рис. 3.17. Часові діаграми роботи мікросхем з зовнішньою пам'яттю програм

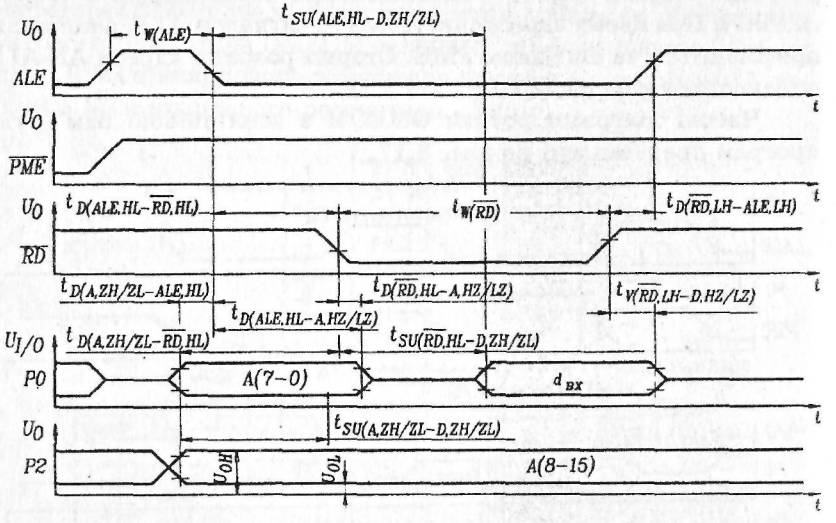
Режим роботи з внутрішньою пам'яттю програм встановлюється заданням високого рівня напруги на виводі DEMA. Виконання програми, що зберігається в пам'яті, починається з команди, розташованої за адресою 00H, тому що лічильник команд PC за сигналом «скидання» обнулюється.

В цьому режимі порти P0 і P2 можна використовувати як порти вводу-виводу, бо адреси/дані пам'яті програм передаються по внутрішній магістралі ОМЕОМ.

Очевидно, що даний режим роботи можливий тільки для ОМЕОМ, що мають внутрішню пам'ять програм: КМ1816ВЕ751, КР1816ВЕ51, КР1830ВЕ51.

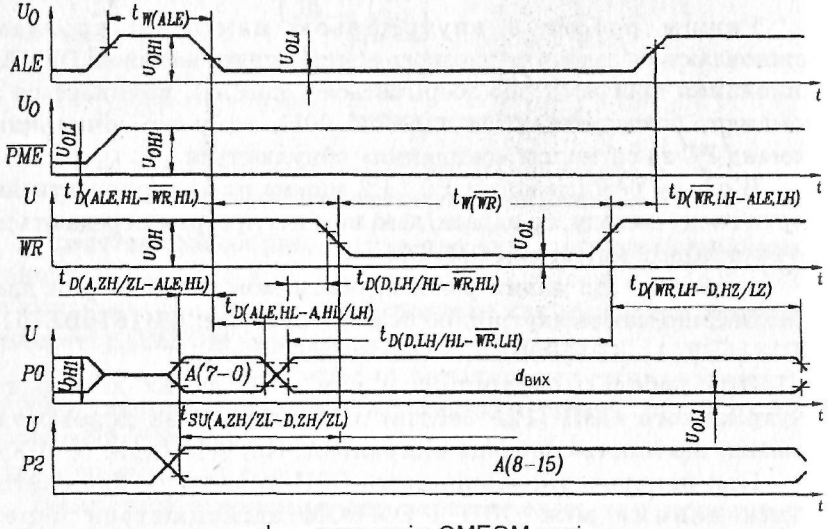
При роботі з внутрішньою пам'яттю даних доступ до внутрішнього ПЗП (128 байтів) здійснюється за допомогою команд, що мають операнди типу direct, Rn, @Ri (крім MOVX).

При підключенні зовнішнього ОЗП ємністю до 256 байтів обмін даними між ОЗП і ОМЕОМ здійснюється через двонаправлений порт P0 за допомогою команд MOVX @Ri, A і MOVX A, @Ri. Для роботи з зовнішнім ОЗП ємністю вище 256 байтів (до 64 Кбайтів) використовуються команди MOVX A, @DPTR і MOVX @DPTR, A. При цьому видача молодших розрядів



$d_{вх}$ - зчитувані дані

Рис. 3.18. Часові діаграми роботи мікросхем при читанні даних з зовнішньої пам'яті



$d_{вих}$ - виведені з ОМЕОМ дані

Рис. 3.19. Часові діаграми роботи мікросхем при запису даних в зовнішню пам'ять

адреси (A0, ..., A7) і обмін даними здійснюється через порт P0 (P0.0, ..., P0.7). Старші розряди адреси A8, ..., A15 видаються через порт P2 (P2.0, ..., P2.7). При цьому адреса A7, ..., A0 фіксується по спаду сиґналу ALE, а прийом і видача даних стробуються сиґналами $\overline{RD}$ і $\overline{WR}$.

Часові діаграми роботи ОМЕОМ з зовнішньою пам'яттю даних представлені на рис. 3.18 (при зчитуванні даних) і 3.19 (при запису даних).

3.3.2. Передача сиґналів через порти

Порти P1 - P3 мають ідентичні характеристики. Дані, записані в них, статично фіксуються і не змінюються до перезапису.

В режимі роботи з зовнішньою пам'яттю програм порт P2 служить для видачі сиґналів старших розрядів адреси і статично фіксує їх до моменту зміни.

Звертання до зовнішньої пам'яті програм завжди здійснюються з використанням 16-розрядної адреси. Коли ОМЕОМ працює з зовнішньою пам'яттю програм, всі 8 бітів порту P2 задіяні на видачу старшого байта адреси (старший байт лічильника команд PC) і не можуть бути використані як лінії вводу/виводу загального призначення.

Звертання до зовнішньої пам'яті даних можуть виконуватися як з використанням 16-розрядної адреси (команди MOVX @DPTR), так і з використанням 8-розрядної адреси (команди MOVX @Ri). При використанні 16-розрядної адреси старший байт адреси видається через порт P2, на лініях якого байт адреси утримується протягом всього часу циклу запису чи зчитування.

Якщо через лінію порту P2 видається розряд адреси, що включає «1», то напруга логічної «1» формується потужним транзистором (транзистор N1 на рис. 3.6 - 3.9), який при цьому буде відкритим протягом всього часу видачі адреси.

Видача адреси через порт P2 не впливає на вміст фіксаторів порту P2. Якщо порт P2 не задіяний на видачу адреси, то на його виводах виставляється вміст фіксаторів (реґістр P2 в області SFR).

Для переводу будь-якої лінії портів P1-P3 на приймання вхідної інформації необхідно у відповідний фіксатор порту записати «1» за допомогою команди видачі даних. Сиґнал RST встановлює всі порти на прийом вхідної інформації.

Порт P0 - восьмирозрядний двонаправлений порт з трьома

станами. Інформація, яка видається портом P0 за допомогою команд видачі, супроводжується строб-імпульсом $\overline{WR}$.

При запису інформації в порт P0 за допомогою команд прийому виробляється строб-імпульс $\overline{RD}$.

Крім операцій вводу/виводу інформації, передбачена можливість виконання логічних операцій І, АБО і виключне АБО безпосередньо на фіксаторах портів P0-P3.

В режимі роботи з зовнішньою пам'яттю програм порт P0 служить для видачі молодших розрядів адреси пам'яті програм і прийому кодів команд. В режимі роботи з зовнішнім ОЗП даних порт P0 служить для видачі адреси зовнішнього ОЗП даних і прийому/видачі даних при обміні з зовнішнім ОЗП.

Будь-яку лінію вводу/виводу можна перевірити за допомогою команд умовного переходу.

Крім того, виводи P3.2 і P3.3 можна використовувати для зовнішнього апаратного переривання, виводи P3.4 і P3.5 - як входи лічильників зовнішніх подій для таймерів/лічильників, а вивід P3.1 можна використовувати як вихід тактового сиґналу в синхронному режимі роботи послідовного інтерфейса ОМЕОМ. Виводи P3.6 і P3.7 порту P3 служать для видачі сиґналів дозволу відповідно запису ($\overline{WR}$) і зчитування ($\overline{RD}$) байта зовнішнього запам'ятовуючого пристрою даних через порт P0.

Під час роботи з послідовним портом ОМЕОМ лінії P3.0 і P3.1 використовуються, відповідно, як вхід і вихід послідовного каналу.

Система команд ОМЕОМ дозволяє зчитувати інформацію з фіксатора порту або безпосередньо з виходу, в залежності від коду інструкції.

В інструкціях, в яких порт служить операндом-джерелом, інформація зчитується безпосередньо з виводів порту, наприклад, ADD A, P1: вміст акумулятора додається з інформацією на виводах порту P1 і результат заноситься в акумулятор.

У всіх випадках, коли операндом і реґістром призначення є порт або біт порту, команди зчитують інформацію з виводів фіксаторів, а не з зовнішніх контактів виводів порту. Наприклад, ORL P2, A.

3.3.3. Структура переривань

Механізм переривань в ОМЕОМ дозволяє автоматично реаґувати на зовнішні і внутрішні події (переповнення таймерів/лічильників; закінчення послідовного обміну). Алгоритм обробки

переривань при виявленні запиту переривання представлений на рис. 3.20. На рис. 3.21 зображені всі можливі джерела переривань.

Кожне із зовнішніх переривань $\overline{INT0}$, $\overline{INT1}$ може бути активізоване по рівню («0») або по фронту (перехід із «1» в «0») сигналів на виводах ОМЕОМ Р3.2, Р3.3, що визначається станом бітів IT0 і IT1 регістра TCON. При поступленні запиту зовнішнього переривання $\overline{INTx}$ встановлюється прапорець IEx регістра TCON. Встановлення прапорців IEx в регістрі TCON викликає відповідне переривання. Очищення прапорця IEx виконується наступним чином: при перериванні по фронту IEx скидається апаратно (автоматично внутрішніми засобами ОМЕОМ) при звертанні до відповідної підпрограми обробки переривання; при перериванні по рівню прапорець очищається при знятті запиту зовнішнього переривання, тобто в IEx відслідковується стан виводу $\overline{INTx}$.

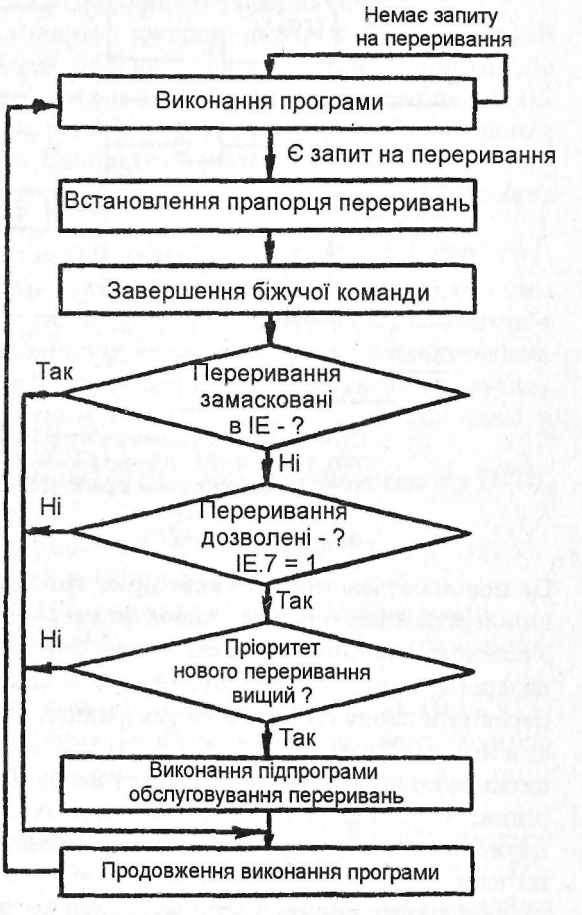
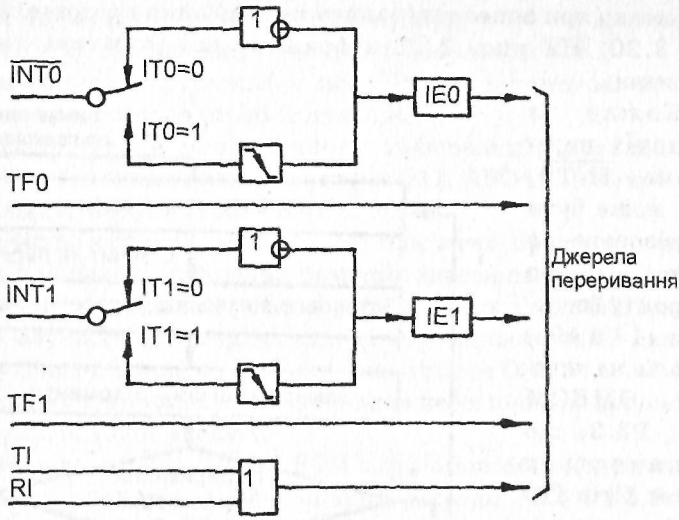


Рис. 3.20. Алгоритм обробки переривань

Щоб зовнішнє переривання по рівню було розпізнане, необхідно, щоб низький рівень на виводі $\overline{INTx}$ утримувався протягом не менше 12 періодів сигналу тактової частоти ОМЕОМ.



IT0, IT1 - біти керування регістра TCON
TF0, TF1, IE0, IE1 - прапорці-ознаки в регістрі TCON
TI, RI - прапорці-ознаки в регістрі SCON

Рис. 3.21. Можливі джерела переривань

Це пояснюється тим, що перевірка виводів ОМЕОМ $\overline{\text{INT0}}$, $\overline{\text{INT1}}$ виконується внутрішніми апаратними засобами ОМЕОМ один раз в кожному машинному циклі. У випадку зовнішнього переривання по фронту прапорець IEx буде встановлений, якщо дві послідовні перевірки входу $\overline{\text{INTx}}$ покажуть в одному машинному циклі «1», а в наступному «0». Тому, якщо зовнішні переривання активізуються по переходу із стану високого рівня в стан низького рівня, то мінімум одному машинному циклу низького рівня повинен передувати мінімум один машинний цикл високого рівня на виводі $\overline{\text{INTx}}$. Якщо зовнішні переривання активізуються за рівнем, запит повинен утримуватися до початку обслуговуючої підпрограми і зніматися до завершення цієї підпрограми для запобігання повторного обслуговування.

Переривання від таймерів/лічильників викликаються встановленням прапорців TF0 і TF1 регістра TCON, при переповненні відповідних регістрів таймерів/лічильників (за винятком режиму 3, див. розділ 3.3.3.). Очищення прапорців TF0 і TF1 виконується внутрішньою апаратурою ОМЕОМ при переході до підпрограми обслуговування переривання.

Переривання від послідовного порту викликається встановленням прапорця переривання приймача RI або прапорця

переривань передавача TI в регістрі SCON. На відміну від усіх решти прапорців, RI і TI скидаються тільки програмним шляхом, звичайно в межах підпрограм обробки переривань, де визначається, якому з прапорців RI чи TI відповідає переривання.

Кожне з перерахованих джерел переривань може бути індивідуально дозволене або заборонене встановленням або скиданням відповідного біта в регістрі дозволу переривань IE. Регістр IE вміщує також біт EA, скидання якого в «0» забороняє одразу всі переривання. Необхідною умовою переривання є його дозвіл в регістрі IE. Формат і опис регістра дозволу переривань наведені в розділі 3.2.4.

Всі біти, які викликають переривання (IE0, IE1, TF0, TF1, RI, TI), можуть бути програмно встановлені або скинені з тим же результатом, що і у випадку їх апаратного встановлення або скидання. Тобто переривання можуть програмно ліквідовуватися. Крім того, переривання по $\overline{\text{INT0}}$, $\overline{\text{INT1}}$ можуть викликатися програмним встановленням P3.2=0 і P3.3=0, як показано в наведеному нижче прикладі:

```
MAIN: MOV IE, #00000101B; дозвіл переривання від INT0,
                                                  INT1.
      MOV IP, #04H; присвоєння INT1 старшого пріоритету.
      SETB EA ; загальний дозвіл переривання.
      MOV P3; #11110011B ; імітація зовнішніх переривань.
SUBR: ORG013H ; перехід до підпрограми обслуговування
                INT1.
```

У розглянутому прикладі запити переривань $\overline{\text{INT0}}$ і $\overline{\text{INT1}}$, що мають різний пріоритет, поступають одночасно (однією командою). При цьому обслуговується переривання з вищим пріоритетом.

У випадку, коли переривання по $\overline{\text{INTx}}$ (x=0,1) викликається рівнем сигналу на відповідному вході OMEOM, прапорець IEx (x=0,1) при переході до підпрограми обробки переривання автоматично скидається, а потім, якщо відповідний вивід OMEOM P3.2 або P3.3 все ще знаходиться у стані логічного «0», знову встановлюється. Тому, у випадку, коли переривання по входах $\overline{\text{INT0}}$, $\overline{\text{INT1}}$ викликається рівнем, програмне встановлення в «1» прапорців IE0, IE1 викличе переривання, після чого відповідний прапорець IEx (x=0,1) буде автоматично скинутий при переході до підпрограми обробки переривань.

Прапорці IE0, IE1, TF0, TF1, R1, T1 встановлюються незалежно від того, дозволено чи ні відповідне переривання в регістрі IE.

Структура пріоритетів переривань є двоступеневою. Кожному джерелу переривання може бути індивідуально присвоєний один із двох рівнів пріоритету: високий або низький. Виконується це встановленням (високий рівень пріоритету) або скиданням (низький рівень пріоритету) відповідного біта в регістрі пріоритетів переривань IP (структура регістра IP описана в розділі 3.2.3). Програма обробки переривання з низьким рівнем пріоритету може бути перервана другим запитом переривання з низьким рівнем пріоритету. Програма обробки переривання з високим рівнем пріоритету не може бути перервана ніяким іншим запитом переривання ні від одного з джерел. Якщо два запити з різними рівнями пріоритетів прийняті одночасно, спочатку буде виконано готування запиту з високим рівнем пріоритету. Якщо одночасно прийняті запити з однаковим рівнем пріоритету, обробка їх буде виконуватися у порядку, який задається послідовністю внутрішнього опиту прапорців переривань. Таким чином, в межах одного пріоритетного рівня існує ще одна структура пріоритетів:

Джерело	Пріоритет всередині рівня
1. IE0	(вищий)
2. TF0	
3. IE1	
4. TF1	
5. RI+TI	(нижчий)

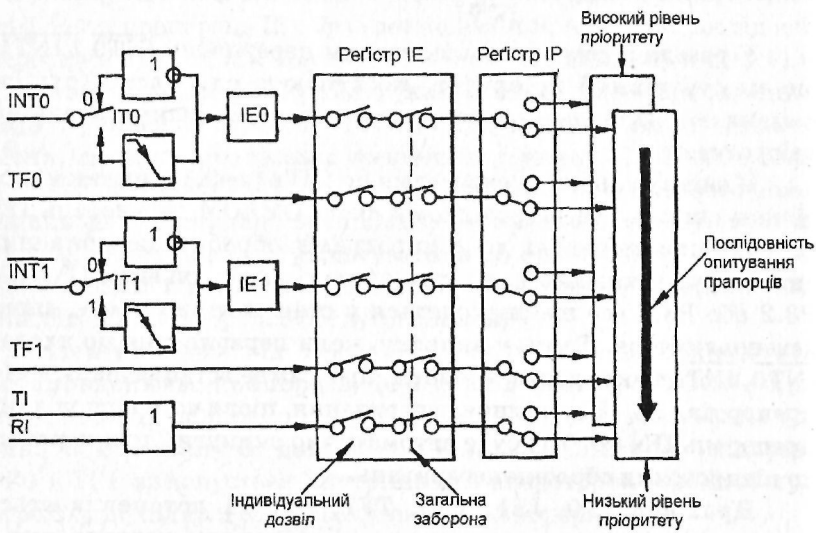


Рис. 3.22. Система переривань ОМЕОМ

Необхідно особливо підкреслити, що структура «Пріоритет всередині рівня» працює тільки в тих випадках, коли визначається послідовність обслуговування запитів на переривання, які прийняті одночасно і при цьому мають однаковий рівень пріоритету.

Загальна схема системи переривань ОМЕОМ наведена на рис. 3.22.

3.3.4. Адресний простір пам'яті

Всі ОМЕОМ сімейства МК51 мають кілька адресних просторів, функціонально і логічно розподілених за рахунок різниці у механізмах адресації і сиґналах керування записом і зчитуванням:

- пам'ять програм;
- внутрішня пам'ять даних;
- зовнішня пам'ять даних.

Структура адресного простору ОМЕОМ показана на рис. 3.23. Зліва подані адреси відповідних областей пам'яті.

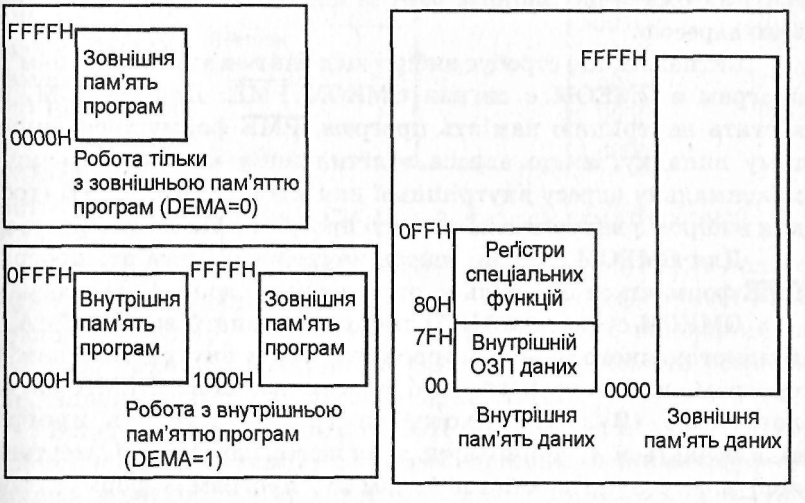


Рис. 3.23. Простір пам'яті ОМЕОМ

Пам'ять програм має 16-бітову адресну шину, її елементи адресуються з використанням лічильника команд (PC) або інструкцій, які формують 16-розрядні адреси.

Пам'ять програм доступна тільки за зчитуванням. ОМЕОМ не мають команд і керуючих сиґналів, призначених для запису у

пам'ять програм. Пам'ять програм має байтову організацію і загальний об'єм до 64 Кбайтів. Ряд ОМЕОМ (КР1816ВЕ51, КМ1816ВЕ751, КР1830ВЕ51) містять розташовану на кристалі внутрішню пам'ять програм ємністю 4 Кбайтів, яка може бути розширена до 64 Кбайтів за рахунок підключення мікросхем зовнішньої пам'яті програм. Внутрішня пам'ять програм КР1816ВЕ51 і КР1830ВЕ51 являє собою ПЗП, що формується при виготовленні ОМЕОМ. Внутрішня пам'ять програм КМ1816ВЕ751 є ППЗП з ультрафіолетовим стиранням.

Таким чином, для ОМЕОМ КР1816ВЕ51, КМ1816ВЕ751 і КР1830ВЕ51 внутрішня і зовнішня пам'ять програм розділені у співвідношенні 4К/60К.

ОМЕОМ КР1816ВЕ31 і КР1830ВЕ31 не мають внутрішньої пам'яті програм і можуть працювати тільки з зовнішньою пам'яттю ємністю до 64 Кбайтів.

З точки зору програміста є лише один вид пам'яті програм ємністю 64 Кбайти. Той факт, що в ряді ОМЕОМ він утворюється комбінацією масивів, які знаходяться на кристалі і поза ним, у співвідношенні 4К/60К, для програміста невідчутний, тому що АЛП автоматично вибирає байт із відповідного масиву згідно з його адресою.

Сиґналом, що стробує вибір і ввід байта з зовнішньої пам'яті програм в ОМЕОМ є сиґнал ОМЕОМ $\overline{\text{PME}}$. Для ОМЕОМ, що містять внутрішню пам'ять програм, $\overline{\text{PME}}$ формується лише в тому випадку, якщо адреса в лічильнику команд перевищує максимальну адресу внутрішньої пам'яті програм 0FFFH (тобто для вибірок з внутрішньої пам'яті програм $\overline{\text{PME}}$ не формується).

Для ОМЕОМ, які не мають внутрішньої пам'яті програм, $\overline{\text{PME}}$ формується при будь-якому звертанні до пам'яті програм.

ОМЕОМ сімейства МК51 мають зовнішній вивід DEMA, за допомогою якого можна заборонити роботу внутрішньої пам'яті програм, для чого необхідно подати на вивід DEMA сиґнал логічного «0». При цьому внутрішня пам'ять програм відключається і, починаючи з нульової адреси, всі звертання відбуваються до зовнішньої пам'яті програм з формуванням сиґналу $\overline{\text{PME}}$. У випадку, якщо DEMA=1, працюють і внутрішня, і зовнішня пам'ять програм. Для ОМЕОМ, які не мають внутрішньої пам'яті програм, для нормальної роботи завжди необхідно задавати DEMA=0.

Отже, доступ до зовнішньої пам'яті програм здійснюється в двох випадках:

1) при дії сиґналу DEMA=0 незалежно від адреси звертання,

2) в будь-якому випадку, якщо програмний лічильник (PC) містить число, більше ніж 0FFFH.

Якщо центральний процесор здійснює доступ до зовнішньої пам'яті програм, сигнал $\overline{PME}$ формується двічі за час кожного машинного циклу (виняток складає команда MOVX) незалежно від того, необхідний чи ні вибраний байт для поточної команди. При вибірці з зовнішньої пам'яті програм завжди використовується 16-бітова адреса, молодший байт якої видається через порт P0, а старший байт - через порт P2 ОМЕОМ. Байт з зовнішньої пам'яті програм вводиться в ОМЕОМ через порт P0, який у цьому випадку використовується як шина адрес/даних в режимі мультиплексування.

На рис. 3.24 показані молодші адреси пам'яті програм, які, як правило, відводяться під обробку переривань і початок роботи ОМЕОМ (скидання).

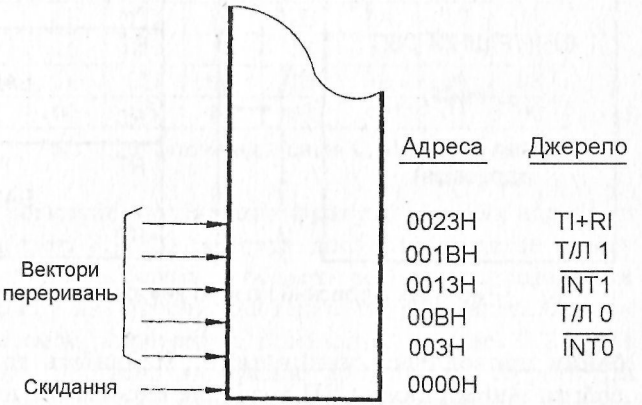


Рис. 3.24. Молодші адреси пам'яті програм

Внутрішня пам'ять даних ОМЕОМ складається з двох областей: 128 байтів оперативної пам'яті (ОЗП) з адресами 00-7FH і області регістрів спеціальних функцій, що займає адреси 80H-FFH. Розподіл простору внутрішньої пам'яті даних показаний на рис. 3.25. Фізично внутрішній ОЗП даних і область регістрів спеціальних функцій є окремими пристроями.

Всі комірки внутрішнього ОЗП даних можуть адресуватися з використанням прямої і непрямої адресації (режими адресації описані в розділі 3.3.5). Крім того, внутрішній ОЗП даних має такі особливості.

Молодші 32 байти внутрішнього ОЗП даних згруповані в 4 банки по 8 регістрів в кожному (БАНК0-БАНК3 на рис. 3.25). Команди програми можуть звертатися до регістрів, використовуючи їх імена R0-R7. Два біти PSW (вказівники банку робочих регістрів RS0 і RS1) визначають, з регістрами якого

Рис. 3.25. Адресний простір внутрішньої пам'яті даних

банку проводяться маніпуляції. Наявність такого механізму роботи з комірками ОЗП дозволяє економити пам'ять програм, тому що команди, які працюють з регістрами R0-R7, коротші від команд, що використовують пряму адресацію.

Наступні після банків регістрів внутрішнього ОЗП даних 16 байтів (адреси 20H-2FH) утворюють область комірок, до яких можлива побітова адресація. Набір команд ОМЕОМ сімейства МК51 містить значну кількість інструкцій, що дозволяють працювати з окремими бітами, використовуючи при цьому пряму адресацію. 128 бітів, що складають розглянуту область внутрішнього ОЗП даних, мають адреси 00H-7FH і призначені для роботи з такими інструкціями. Бітова адресація ОЗП показана на рис. 3.26, де в квадратах, що символізують біти, вказані їх адреси.

Звертання до внутрішнього ОЗП даних завжди здійснюється з використанням 8-розрядної адреси. При вмиканні живлення вміст ОЗП буде мати випадкове значення.

Область регістрів спеціальних функцій містить фіксатори портів, регістри таймерів/лічильників, регістри керування і т.п.

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

Рис. 3.26. Бітова адресація ОЗП

Повний список регістрів спеціальних функцій з їхніми адресами наведений в таблиці 3.2. Ці регістри допускають лише пряму адресацію. Дванадцять байтів в області регістрів спеціальних функцій допускають як байтову, так і побітову адресацію. Список регістрів з побітовою адресацією показаний на рис. 3.27. Як видно з рисунка, побітову адресацію допускають ті регістри спеціальних функцій, чия адреса закінчується 000B. Біти в розглянутій області регістрів спеціальних функцій мають адреси 08H-F7H.

Регістр	Адреса регістра	Адреси бітів D7...D0
P0	80H	87...80H
TCON	88H	8F...88H
P1	90H	97...90H
SCON	98H	9F...98H
P2	A0H	A7...A0H
IE	A8H	AF,AC...A8H
P3	B0H	B7...B0H
IP	B8H	BC...B8H
PSW	D0H	D7...D0H
ACC	E0H	E7...E0H
B	F0H	F7...F0H

Рис. 3.27. Адреси бітів регістрів спеціальних функцій

Зовнішня пам'ять даних формується додатковими мікросхемами пам'яті, під'єднаними до ОМЕОМ і може мати ємність до 64 Кбайтів. Простори внутрішньої і зовнішньої пам'яті даних не перетинаються, тому що доступ до них здійснюється за допомогою різних команд. Для роботи з зовнішньою пам'яттю даних існують спеціальні команди MOVX, які не впливають на внутрішню пам'ять даних ОМЕОМ. Таким чином, в системі можуть одночасно бути присутніми внутрішня пам'ять даних з адресами 00H-FFH і зовнішня пам'ять даних з адресами 0000H-FFFFH. Звертання до комірок зовнішньої пам'яті даних здійснюється лише з використанням непрямої адресації по регістрах R0 і R1 активного банку регістрів внутрішнього ОЗП (команди типу MOV @Ri) чи по регістру спеціальних функцій DPTR (команди типу MOV @DPTR). Відповідно в першому випадку буде формуватися 8-розрядна, а в другому випадку 16-розрядна адреса внутрішньої пам'яті даних.

При звертаннях до зовнішньої пам'яті даних адреса виводиться через порт P0 (молодший байт) і порт P2 (старший байт) ОМЕОМ. Обмін байтом даних (запис і зчитування) проводиться через порт P0 ОМЕОМ, тому що порт P0 використовується як шина адреси/даних в режимі *мультиплексування*.

Зчитування даних з зовнішньої пам'яті даних ОМЕОМ проводиться за допомогою вихідного сигналу ОМЕОМ $\overline{WR}$.

Кожний тип зовнішньої пам'яті (пам'ять програм, пам'ять даних) може бути доданий незалежно від першого і кожний використовує ті ж адреси і шини даних, але різні сигнали керування.

3.3.5. Способи адресації операндів

Існують наступні способи адресації операндів-джерел:

- регістрова адресація;
- пряма адресація;
- непрямо-регістрова адресація;
- безпосередня адресація;
- непряма адресація по сумі базового і індексного регістра.

Перші три способи використовуються також для адресації операнда призначення. Вказані п'ять способів адресації, які використовуються в різноманітних сполученнях, забезпечують 21 режим адресації.

Багато команд вміщують поля: «приймач» і «джерело». Вони

визначають тип даних, метод адресації і операнди, які приймають участь.

Для команд, які не виконують операції перезапису, операнд призначення є операндом-джерелом.

Велика кількість команд містить операнди, розташовані у внутрішньому ОЗП даних ОМЕОМ. Вибір адресного простору пам'яті програм або зовнішньої пам'яті даних як другого операнда визначається командною мнемонікою (якщо тільки другий операнд не є безпосередньою величиною).

Область адрес внутрішнього ОЗП даних визначається способом адресації і величиною адреси. Наприклад, звертання до регістрів спеціального призначення може бути виконано лише за допомогою прямої адресації.

Регістрова адресація використовується для звертання до восьми робочих регістрів вибраного банку робочих регістрів (ці ж регістри можуть бути вибрані за допомогою прямої адресації і непрямо-регістрової адресації як звичайні комірки внутрішнього ОЗП).

Регістрова адресація використовується також для звертання до регістрів A, B, AB (подвійного регістра), DPTR і до прапорця переносу C. Використання регістрової адресації дозволяє отримати двобайтовий еквівалент трибайтових команд прямої адресації.

Пряма байтова адресація використовується для звертання до комірок внутрішньої пам'яті (ОЗП) даних (0-127) і до регістрів спеціального призначення.

Пряма побітова адресація використовується для звертання до окремо адресованих 128 бітів, розташованих в комірках з адресами 20H-2FH і до окремо адресованих бітів регістрів спеціального призначення.

Старший біт байта коду прямої адреси вибирає одну з двох груп окремо аресованих бітів, розташованих в ОЗП або регістрах спеціального призначення. Прямо адресовані біти з адресами 0-127 (00H-7FH) розташовані у блоці із 16 комірок внутрішнього ОЗП, який має адреси 20H-2FH. Вказані комірки послідовно пронумеровані від молодшого біта до старшого біта старшого байта. Окремо адресовані біти в регістрах спеціального призначення пронумеровані наступним чином: п'ять старших розрядів адреси співпадають з п'ятьма старшими розрядами адреси самого регістра, а три молодших - визначають місце розташування окремого біта всередині регістра.

Непрямо-регістрова адресація використовується для звертання до комірок внутрішнього ОЗП даних. В якості регістрів-

вказівників використовуються регістри R0, R1 вибраного банку регістрів.

У командах PUSH і POP використовується вміст вказівника стека (SP).

Непрямо-регістрова адресація використовується також для звертання до зовнішньої пам'яті даних. У цьому випадку за допомогою регістрів-вказівників R0 і R1 (вибраного банку робочих регістрів) вибирається комірка із блоку в 256 байтів зовнішньої пам'яті даних. Номер блоку попередньо задається вмістом порту P2.

16-розрядний вказівник даних (DPTR) може бути використаний для звертання до будь-якої комірки адресного простору зовнішньої пам'яті даних об'ємом до 64 Кбайтів.

Безпосередня адресація дозволяє вибирати із адресного простору пам'яті програм константи, які явно вказані у команді.

Непрямо-регістрова адресація по сумі: базовий регістр плюс індексний регістр (вміст акумулятора A) спрощує перегляд таблиць, записаних в пам'яті програм.

Будь-який байт із таблиці може бути вибраний за адресою, визначених сумою вмісту DPTR або PC і вмісту A.

3.4. Система команд

Система команд МК51 має 111 базових команд, які поділяються на 5 груп: передачі даних (28), арифметичні (24), логічні (25), передачі керування (22) і операцій з бітами (12).

В склад системи входять такі потужні команди як множення, ділення і операції з бітами.

Більшість команд (94) мають формат один або два байти і виконуються за один або два машинні цикли (при тактовий частоті 12МГц один машинний цикл займає час 1мкс). Перший байт команди будь-якого типу і формату завжди містить код операції. Другий і третій байти містять адреси операндів, або безпосередні операнди.

Можливості МК51 значно ширші за інші типи мікропроцесорів (МП) завдяки наявності операцій з 4 типами операндів: біти, 4-бітові цифри, байти і 16-бітові слова. Існує можливість програмного доступу до 128 бітів в резидентній пам'яті даних (РПД) і, крім того, в регістрах спеціальних функцій. Адресація бітів завжди 8-розрядна і тільки пряма.

Чотирибітові операнди використовуються лише при операторах обміну в командах SWAP і XCHD.

Восьмибітовим операндом може бути комірка пам'яті програми або даних (резидентної або зовнішньої), константа, регістри спеціальних функцій, а також порти вводу/виводу.

Порти і регістри спеціальних функцій можуть адресуватись також через адресні регістри R0, R1, DPTR, PC.

Двобайтові операнди - це константи і прямі адреси, для представлення яких використовуються другий і третій байти команди.

Команди передачі даних (28). Ці команди працюють з операндом і по шляхах, показаних на рис.3.28. Слід зазначити, що велика кількість передач даних може здійснюватися без використання акумулятора.

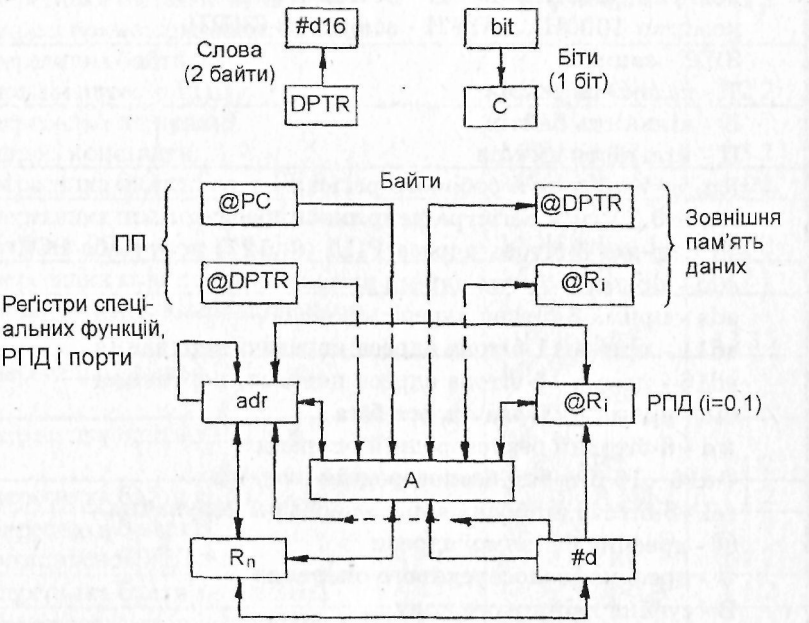


Рис. 3.28. Граф шляхів передачі даних в МК51

Команди арифметичних операцій (24) здійснюють операції додавання, віднімання, десяткової корекції, інкремента/декремента, множення і ділення байтів. Існує команда інкремента DPTR.

Опис кожної наведеної машинної команди ОМЕОМ типу МК51 складається із речення мови асемблера, коду, довжини

команди в байтах, часу її виконання, алгоритму і прикладу.

При описі операцій використовуються такі умовні скорочення в списку команд:

ак. - акумулятор
рег. - регістр
т/л - таймер/лічильник
РПД - резидентна пам'ять даних, комірки 20H...7FH
РСФ - регістр спеціальних функцій або
РСП - регістр спеціального призначення
ССП - слово стану процесора
ПП - пам'ять програм,
комірки 0000H...0FFFH - резидентна (РПП)
комірки 1000H...FFFFH - зовнішня (ЗПП)
ЗПР - запит
Т - кількість тактів
Б - кількість байтів
Ц - кількість циклів
Rn, n=0...7 - ім'я робочого регістра
Ri, i=0,1 - ім'я регістра непрямої адреси
ad - пряма 8-бітова адреса РПД (0..127) порту або РСФ
add - пряма 8-бітова адреса призначення
ads - пряма 8-бітова адреса джерела
ad11 - пряма 11-бітова адреса передачі керування
ad16 - пряма 16-бітова адреса передачі керування
bit - пряма 8-бітова адреса біта
#d - 8-бітовий безпосередній операнд
#d16 - 16-бітовий безпосередній операнд
rel - 8-бітова відносна адреса передачі керування
@ - префікс непрямої адреси
# - префікс безпосереднього операнда
B - суфікс двійкового коду
H - суфікс шістнадцяткового коду

Докладний опис машинних команд приведено в Додатку 2.

3.4.1. Команди передачі даних

В системі команд МК51 є 28 команд передачі даних [18].

		ТБЦ
пересилка байта з рег. в ак.	MOV A,R_n	1 1 1
пересилка прямоадресованого байта з РПД	MOV A,ad	3 2 1
пересилка байта з РПД в ак.	MOV A,@R_i	1 1 1
завантаження константи в ак.	MOV A,#d	2 2 1
пересилка байта з ак. в рег.	MOV R_n,A	1 1 1
пересилка прямоадресованого байта в регістр	MOV R_n,ad	3 2 2
завантаження константи в регістр	MOV R_n,#d	2 2 1
пересилка байта по прямій адресі з ак.	MOV ad,A	3 2 1
пересилка по прямій адресі регістра	MOV ad,R_n	3 2 2
пересилка байта по прямій адресі прямоадресованого байта	MOV add,ads	9 3 2
пересилка байта по прямій адресі з РПД	MOV ad,@R_i	3 2 2
пересилка по прямій адресі константи	MOV ad,#d	7 3 2
пересилка байта з ак. в РПД	MOV @R_i,A	1 1 1
пересилка прямоадресованого байта в РПД	MOV @R_i,ad	3 2 2
пересилка константи в РПД	MOV @R_i,#d	2 2 1
завантаження вказівника даних	MOV DPTR,#d16	13 3 2
пересилка байта з ПП в ак.	MOVC A,@A+DPTR	1 1 2
пересилка байта з ПП в ак.	MOVC A,@A+PC	1 1 2
пересилка байта із ЗПД в ак.	MOVX A,@R_i	1 1 2
пересилка байта із розширеної ЗПД в ак.	MOVX A,@DPTR	1 1 2
пересилка байта з ак. в ЗПД	MOVX @R_i,A	1 1 2
пересилка з ак. в розширену ЗПД	MOVX @DPTR,A	1 1 2
завантаження в стек	PUSH ad	3 2 2
вилучення з стека	POP ad	3 2 2
обмін байтів рег. і ак.	XCH A,R_n	1 1 1
обмін ак. з прямоадресованим байтом	XCH A,ad	3 2 1
обмін ак. з РПД	XCH A,@R_i	1 1 1
обмін молодших тетрад ак. і байта з РПД	XCHD A,@R_i	1 1 1

Виконання команд передачі даних має наступні особливості. Таблиці кодів і зашиті в ПЗП програми можуть бути вибрані за допомогою команд передачі даних з використанням непрямої адресації. Байт константи може бути переданий в акумулятор із комірки пам'яті програм (Prg), що адресується сумою базового регістра (PC або DPTR) і індексного регістра (A) - команди MOVC A,@A+DPTR і MOVC A,@A+PC. Цим можна здійснити, наприклад, зручний алгоритм перетворення коду ASCII в семисегментний код.

Будь-яка комірка 256-байтового блоку зовнішнього (Ext) ПЗП даних може бути вибрана з використанням регістрової адресації через адресні регістри R0 та R1 вибраного банку даних командою типу MOVX @Ri.

Комірка всередині адресного простору 64 Кбайти зовнішнього (Ext) ОЗП також може бути вибрана за допомогою регістрової адресації через регістр-вказівник даних DPTR командою MOVX @DPTR.

Команди передачі між прямоадресованими регістрами дозволяють заносити значення із порту в комірку внутрішнього ОЗП (Int) без використання робочих регістрів або акумулятора командами MOV.

Вміст акумулятора може бути замінений вмістом робочого регістра (вибраного банку) і вмістом адресованих за допомогою регістрової адресації комірок внутрішнього ОЗП, а також вмістом прямоадресованих комірок внутрішнього ОЗП і вмістом регістрів спеціального призначення командами XCH і XCHD.

3.4.2. Арифметичні команди

В системі команд MK51 є 24 арифметичні команди [18]. Частина з них впливає на значення прапорців (регістр PSW), решта не змінює їх.

В наборі команд OMEOM MK51 є такі арифметичні операції: додавання - ADD, додавання з врахуванням прапорця переносу - ADDC, віднімання з позикою - SUBB, інкрементування (збільшення на одиницю) - INC, декрементування (зменшення на одиницю) DEC, десяткова корекція - DA, множення MUL та ділення DIV.

		Т Б Ц
додавання байтів з рег. і ак.	ADD A,Rn	1 1 1
додавання байта з ак. і прямоадресованого байта	ADD A,ad	3 2 1
додавання байтів з РПД і ак.	ADD A,@Ri	1 1 1
додавання константи і байта з ак.	ADD A,#d	2 2 1
додавання рег., ак. і переносу	ADDC A,Rn	1 1 1
додавання прямоадресованого байта, ак. і переносу	ADDC A,ad	3 2 1
додавання байта з РПД, ак. і переносу	ADDC A,@Ri	1 1 1
додавання константи, байта з ак. і переносу	ADDC A,#d	2 2 1
десяткова корекція ак.	DA A	1 1 1
віднімання з ак. рег. і зайому	SUBB A,Rn	1 1 1
віднімання прямоадресованого байта і зайому з ак.	SUBB A,ad	3 2 1
віднімання байта РПД і зайому з ак.	SUBB A,@Ri	1 1 1
віднімання константи і зайому з ак.	SUBB A,d	2 2 1
інкремент (збільшення на 1) ак.	INC A	1 1 1
інкремент значення рег.	INC Rn	1 1 1
інкремент прямоадресованого байта	INC ad	3 2 1
інкремент байта в РПД	INC @Ri	1 1 1
інкремент вказівника даних	INC DPTR	1 1 2
декремент (зменшення на 1) ак.	DEC A	1 1 1
декремент значення рег.	DEC Rn	1 1 1
декремент прямоадресованого байта	DEC ad	3 2 1
декремент байта в РПД	DEC @Ri	1 1 1
множення байта з ак. на байт рег. В	MUL AB	1 1 4
ділення байта з ак. на байт з рег.В	DIV AB	1 1 4

В АЛП проводяться дії над цілими числами без знака. В двоопераднних операціях з даними числами: додавання (ADD), додавання з переносом (ADDC) і віднімання з позикою (SUBB) акумулятор є першим операндом і приймає результат операції. Другим операндом може бути робочий регістр вибраного банку робочих регістрів, регістр внутрішньої пам'яті даних з регістровою і прямою адресацією або байт безпосередніх даних. Ці операції впливають на прапорці: переповнення OV - PSW.2, переносу C - PSW.7, проміжного переносу AC - PSW.6 і прапорця парності P - PSW.0.

При використанні розряду переносу можна підвищити

точність при опереціях додавання (ADDC) та віднімання (SUBB).

Виконання операцій додавання та віднімання може бути здійснено за допомогою програмного керування прапорцем переповнення (OV) регістра PSW. Прапорець проміжного переносу (AC) забезпечує виконання арифметичних операцій в двійково-десятковому коді.

Операції інкрементування та декрементування на прапорці не впливають.

Операції порівняння не впливають ні на операнд призначення, ні на операнд джерела, але впливають на прапорець переносу.

Існують три арифметичні операції, які виконуються тільки в акумуляторі: дві команди перевірки вмісту акумулятора (JZ, JNZ) і команда десяткової корекції DA при додаванні двійково-десяткових кодів.

При виконанні операції множення MUL вміст акумулятора A множиться на вміст регістра B, а результат розміщується наступним чином: старший байт – в акумуляторі, а молодший – в регістрі B.

У випадку виконання операції ділення DIV ціле від ділення розміщується в акумуляторі A, а залишок від ділення - в регістрі B.

3.4.3. Команди логічних операцій

В системі команд MK51 є 25 логічних операцій, виконуваних над даними у вигляді байтів.

Система команд OMEOM MK51 дозволяє реалізувати логічні операції І, АБО і виключне АБО на регістрі-акумуляторі (A) і байті-джерелі. Другим операндом (байтом-джерелом) при цьому може бути робочий регістр у вибраному банку робочих регістрів, регістр внутрішнього ОЗП, адресований за допомогою регістрів, прямоадресовані комірки внутрішнього ОЗП та регістри спеціального призначення, безпосередня величина. Вказані логічні операції можуть бути реалізовані на будь-якому прямоадресованому регістрі внутрішнього ОЗП або регістрі спеціального призначення з використанням другого операнда вмісту акумулятора A або безпосередніх даних.

Існують логічні операції. які виконуються лише на акумуляторі: скидання CLR і інвертування CPL всіх восьми розрядів акумулятора A; циклічний зсув вліво RL та вправо RR; циклічний зсув вліво RLC та вправо RRC з врахуванням прапорця переносу, обмін місцями молодшої і старшої тетрад всередині акумулятора SWAP.

		ТБЦ
логічне множення байтів рег. і ак.	ANL A,Rn	1 1 1
логічне множення байта з ак. і прямоадресованого байта	ANL A,ad	3 2 1
логічне множення байта з РПД і ак.	ANL A,@Ri	1 1 1
логічне множення константи і ак.	ANL A,#d	2 2 1
логічне множення прямоадресованого байта і ак.	ANL ad,A	3 2 1
логічне множення прямоадресованого байта і константи	ANL ad,#d	7 3 2
логічне додавання байта рег. і ак.	ORL A,Rn	1 1 1
логічне додавання ак. і прямоадресованого байта	ORL A, ad	3 2 1
логічне додавання байта з РПД і ак.	ORL A,@Ri	1 1 1
логічне додавання константи і ак.	ORL A,#d	2 2 1
логічне додавання прямоадресованого байта і ак.	ORL ad,A	3 2 1
логічне додавання прямоадресованого байта і константи	ORL ad,#d	7 3 2
виключне АБО рег. і ак.	XRL A,Rn	1 1 1
виключне АБО ак. і прямоадресованого байта	XRL A,ad	3 2 1
виключне АБО байта з РПД і ак.	XRL A,@Ri	1 1 1
виключне АБО константи і ак.	XRL A,#d	2 2 1
виключне АБО прямоадресованого байта і ак.	XRL ad,A	3 2 1
виключне АБО прямоадресованого байта і константи	XRL ad,#d	7 3 2
очистка ак.	CLR A	1 1 1
інвесія ак.	CPL A	1 1 1
зсув ак. вліво циклічно	RL A	1 1 1
зсув ак. вліво через перенос	RLC A	1 1 1
зсув ак. вправо циклічно	RR A	1 1 1
зсув ак. вправо через перенос	RRC A	1 1 1
обмін тетрад в ак.	SWAP A	1 1 1

3.4.4. Команди бітового процесора

За допомогою бітового процесора можна реалізувати найрізноманітніші функції логічних (бітових) систем керування-вмикання і вимикання за заданими критеріями і т.п., значно простіше, ніж в універсальних мікропроцесорах.

Бітовий процесор є частиною ОМЕОМ МК51 і його можна

розглядати як незалежний процесор бітової обробки. Бітовий процесор виконує свій набір команд, має свій побітно-адресований ОЗП і свій ввід/вивід.

		Т Б Ц
очистка переносу	CLR C	1 1 1
скидання біта	CLR bit	4 2 1
встановлення переносу	SETB C	1 1 1
встановлення біта	SETB bit	4 2 1
інверсія переносу	CPL C	1 1 1
інверсія біта	CPL bit	4 2 1
логічне множення біта і переносу	ANL C,bit	4 2 2
логічне І інверсії біта і переносу	NL C,/bit	4 2 2
логічне АБО біта і переносу	ORL C,bit	4 2 2
логічне АБО інверсії біта і переносу	ORL C,/bit	4 2 2
пересилка біта в перенос	MOV C,bit	4 2 1
пересилка переносу в біт	MOV bit,C	4 2 2

Команди, які оперують з бітами, забезпечують пряму адресацію 128 бітів (від 0 до 127) в шістнадцяти комірках внутрішнього ОЗП (комірки з адресами 20H - 2FH) і пряму побітну адресацію регістрів спеціального призначення, адреси яких кратні восьми.

Кожний з окремо адресованих бітів може бути встановлений в «1», скинутий в «0», проінвертований, перевірений. Можуть бути реалізовані такі переходи в програмі:

- якщо біт встановлений;
- якщо біт не встановлений;
- перехід, якщо біт встановлений, із скиданням цього біта;
- біт може бути переписаний в розряд (із розряду) переносу.

Між будь-яким прямоадресованим бітом і прапорцем переносу можуть бути проведені логічні операції І, АБО, де результат заноситься у прапорець переносу. Команди побітової обробки забезпечують реалізацію складних функцій комбінаційної логіки і оптимізацію програм користувача.

3.4.5. Команди передачі керування

В системі команд МК51 є 22 команди передачі керування.

Під командами передачі керування розуміють команди, що дозволяють змінити порядок виконання програми. Вказані команди можуть бути умовними і безумовними. Виконання їх має деякі особливості.

		Т Б Ц
довгий перехід в повному обсязі пам'яті програми	LJMP ad16	12 3 2
абсолютний перехід в сторінці в 2 Кбайти	AJMP ad11	6 2 2
короткий відносний перехід	SJMP rel	5 2 2
непрямий відносний перехід	JMP @A+DPTR	1 1 2
перехід, якщо в ак. нуль	JZ rel	5 2 2
перехід, якщо в ак. не нуль	JNZ rel	5 2 2
перехід, якщо перенос = 1	JC rel	5 2 2
перехід, якщо перенос = 0	JNC rel	5 2 2
перехід, якщо біт = 1	JB bit,rel	11 3 2
перехід, якщо біт = 0	JNB bit,rel	11 3 2
перехід, якщо біт встановлений, з наступним скиданням біта	JBC bit,rel	11 3 2
декремент рег. і перехід, якщо не нуль	DJNZ Rn,ad	5 2 2
декремент прямоадресованого байта і перехід, якщо не нуль	DJNZ ad,rel	8 3 2
порівняння значення в акумуляторі з прямоадресованим байтом і перехід, якщо не рівно	CJNE A,ad,rel	8 3 2
порівняння акумулятора з константою і перехід, якщо не рівно	CJNE A,#d,rel	10 3 2
порівняння регістра з константою і перехід, якщо не рівно	CJNE Rn,#d,rel	10 3 2
порівняння байта в РПД з константою і перехід, якщо не рівно	CJNE @Ri,#d,rel	10 3 2
довгий виклик підпрограми	LCALL ad16	12 3 2
абсолютний виклик підпрограми в межах сторінки в 2 Кбайти	ACALL ad11	6 2 2
повернення з підпрограми	RET	1 1 2
повернення з підпрограми обробки переривання	RETI	1 1 2
холоста команда	NOP	1 1 1

Адресний простір пам'яті програм ОМЕОМ не має організації по сторінках, що дозволяє вільно переміщати фраґменти програми всередині адресного простору, при цьому не потрібно змінювати номер сторінки.

Переміщення окремих фраґментів програми забезпечує можливість використання переміщуваних програмних модулів різними програмами.

Команди 11-розрядних переходів і викликів підпрограм забезпечують переходи всередині програмного модуля обсягом 2 Кбайти. В системі команд є команди умовних та безумовних переходів відносно початкової адреси наступної команди в межах від (PC)-128 до (PC)+127. Команди перевірки окремих розрядів дозволяють здійснити умовні переходи по стану «0» або «1» відповідних бітів. Команди перевірки значення акумулятора на «0» (не «0») дозволяють здійснити перехід по вмісту акумулятора A.

Команди 16-розрядних переходів і викликів підпрограм дозволяють здійснити перехід в будь-яку точку адресного простору обсягом 64 Кбайти.

Непрямі регістрові переходи в системі команд ОМЕОМ забезпечують розгалуження програми відносно базового регістра (вмісту DPTR або PC) із зміщенням, що знаходиться в акумуляторі A.

Розділ 4. ОБРОБКА СИҐНАЛІВ НА ОСНОВІ МК51

4.1. Використання команд передачі даних

Приклад 1. Передати вміст буфера послідовного адаптера в резидентну пам'ять даних за непрямою адресою в R0:

```
MOV @R0, SBUF; передача прийнятого по послідовному
               каналу байта в РПД.
```

Приклад 2. Записати в комірки РПД з адресами 39 і 40 число FA1BH:

```
LOAD: MOV 39H, #0FAH
MOV 40H, #1BH
```

Приклад 3. Завантажити у вказівник даних початкову адресу 4300H масиву даних, розташованого у зовнішній пам'яті даних:

```
MOV DPTR, #4300H; завантаження початкового значення
                  вказівника даних.
```

Приклад 4. Завантажити керуюче слово в регістр керування таймером:

```
MOV TCON, #00000101B; дозвіл зовнішніх переривань по
                     низькому рівню сиґналу.
```

Приклад 5. Скинути всі прапорці користувача (біти) в резидентній області пам'яті від 20H до 2FH:

```
    MOV R0, #20H; задана початкова адреса області пам'яті
    MOV R1, #0FH; завантажити лічильник (довжина області
                  пам'яті)
LOOP: MOV @R0, #0; скинути один байт (8 прапорців)
    INC R0 ; перехід до наступного байта
    DJNZ R1,LOOP; цикл, якщо не всі прапорці скинуті
```

Приклад 6. Передати керування по мітці L0, якщо лічильник 0 досягнув значення 128:

```
MOV A, TL0 ; передача вмісту лічильника в акумулятор
JB A.7, L0 ; перейти на L0, якщо A.7=1
 ...
L0 :  : продовження програми
```

Приклад 7. Запам'ятати у зовнішній пам'яті даних вміст регістрів банку 0. Початкова адреса ЗПД - 2000H:

```
MOV PSW,#00010000B; вибір банку регістрів 1
MOV R0, #8; лічильник ← 8
MOV DPTR, #2000H; визначення початкової адреси ЗПД
MOV R1, #0; визначення початкової адреси РПД
LOOP: MOV A, @R1 ; (A) ← (регістр)
MOVX @ DPTR,A ; передача із акумулятора в ЗПД
INC R1 ; перехід до наступного регістра
INC DPTR ; приріст вказівника адреси
DJNZ R0,LOOP ; R0=R0-1, якщо R0>0 то повторити цикл
```

Приклад 8. Звертання до пам'яті програм, де зберігається готова таблиця значень. Для цього використовується спеціальна команда MOVC. Наприклад: програма для вибору значень функції синуса із таблиці з точністю 0,4 % і дискретом 1^0. Початковий параметр для підпрограми є значення кута x, яке знаходиться в акумуляторі. Дана програма працює без вказівника даних DPTR. Інкремент акумулятора перед звертанням до таблиці необхідний через використання однобайтної команди повернення. Таблиця синусів займає в пам'яті 90 байтів.

```
; обчислення sin(x) по таблиці значень: вхід (A) ← (x),
x ∈ (0,89⁰);
; вихід (A) ← дробова частина значення синуса
SIN: INC A ; інкремент акумулятора
     MOVC A, @A+PC ; завантаження значення синуса із
                     таблиці
     RET ; повернення
     ; таблиця синусів
     SINUS : DB 00000000B ; SIN (0⁰)=0
             DB 00000100B ; SIN (1⁰)=0,017
             DB 00001001B ; SIN (2⁰)=0,035
             .....
             DB 11111111B ; SIN (89⁰)=0,999
```

Приклад 9. Послідовність дій для перегляду багатомірних таблиць складається із 3-х кроків: завантаження акумулятора індексом, відновлення адреси інструкції перегляду таблиць шляхом додавання зміщення до акумулятора, виконання інструкцій MOV A, @A+PC.

Далі дано приклад, коли в одномірній пам'яті програм зберігаються великі багатомірні таблиці, які вміщують комбінації

двійкових розрядів, нелінійні калібровочні параметри і т.п. Для вибірки даних із таблиць змінні, що представляють індекси елементів, повинні бути перетворені в адресу пам'яті для запису. Для матриці з розміром MxN і початковою адресою STRT з індексами I і J адреса елемента (i,j) визначається за формулою:

Адреса = [STRT + (N*I) + J].

Підпрограма MATRX1 може здійснити доступ до запису масиву із менше, ніж 255 елементів в прикладі 11*21 =231 елемент. Запис в таблицю здійснюється директивою DATA BYTE (DB) і зберігається як сама програма.

Підпрограма MATRX2 забезпечує доступ до таблиць необмежених розмірів шляхом комбінацій інструкції MUL, арифметичних операцій подвійної точності і інструкції MOVC. Обмеження існують лише по величині індексу в межах 0-255.

```
MATRX1; завантажує константу в А із двомірної таблиці
     I EQU R6 ; перша координата запису (0 - 10).
     J DATA 28H; друга координата запису (0 - 20).
MATRX1: MOV A,I
     MOV B, #21
     MUL AB
     ADD A,J ; додавання зміщення
     INC A
     MOVC A, @A+PC вибір комірки з індексами 0,0
     RET
BASE: DB1 1  ; запис 0, 0
     DB2 2
     .....; запис 0, 20
     DB 21     ; запис 1, 0
     DB 22
     .....; запис 1, 20
     DB 231    ; запис 10, 20
MATRX2: MOV A,I ; завантаження першої координати.
     MOV B, #N
     MUL AB
     ADD A, #LOW ; додати до 16 розряду базового індексу
     MOV DPL, A
     MOV A, B
     ADDC A, #HIGH
     MOV DPH, A; DPTR=(STRT)+(i*N)
     MOV A, J
     MOV A, @A+DPTR ; додати індекс і вибрати байт
     RET
```

Приклад 10. Операції зі стеком і організація переривань. Механізм доступу до стека МК51: перед завантаженням в стек вміст регістра-вказівника стека SP інкрементується, а після вивантажування даних із стека декрементується.

За сигналом системного скидання в SP заноситься початкове значення 07H. Для перевизначення початкового значення SP можна використати команду MOV SP,#d.

Необхідно враховувати, що оскільки у вказівник стека завантажується початкова адреса 07H, то перше завантажене значення розміститься в комірці 08H.

Таким чином, стек може бути розташований в будь-якому місці РПД. Стек використовується для організації звертання до підпрограм і при обробці переривань може бути використаний також для передачі параметрів підпрограм та для тимчасового зберігання значень регістрів спеціальних функцій. Підпрограма обробки переривань повинна зберігати в стеці стан тих регістрів, які буде сама використовувати, а перед поверненням в перервану підпрограму повинна відновити їх значення.

```
        Підпрограма обробки переривань.
        ORG 3 ; задання адреси вектора переривань
        SJMP SUBINO ; перехід на підпрограму обробки
        ORG 30H
SUBINO: PUSH PSW ; зберігання в стеці PSW
        PUSH A ; зберігання в стеці акумулятора A
        PUSH B ; зберігання в стеці доповнення-акумулятора B
        PUSH DPL ; зберігання в стеці DPTR
        PUSH DPH ; зберігання в стеці DPTR
        MOV PSW,#00001000B; вибір іншого банку регістрів (1)
        .
        . обробка переривань
        .
        POP DPH ; відновлення DPTR
        POP DPL ; відновлення DPTR
        POP B ; відновлення B
        POP A ; відновлення акумулятора
        POP PSW ; відновлення PSW і банку регістрів 0
        RETI ; повернення в основну програму
```

Якщо SP=1FH, розміщення регістрів в стеці після входу в підпрограму обробки буде таким, як показано на рис. 4.1.

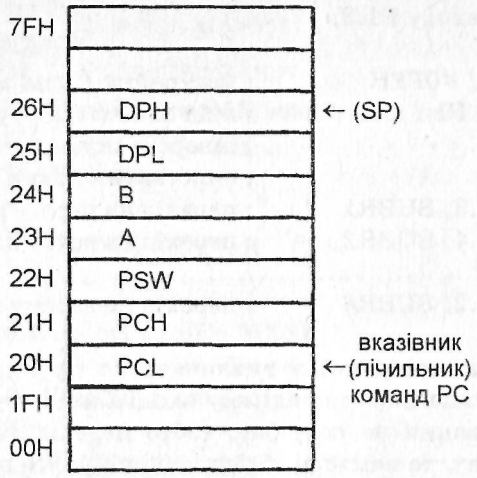


Рис. 4.1. Розподіл пам'яті при виклику підпрограми

Приклад 11. Програмне переривання.

Всі біти, які викликають переривання можуть бути програмно встановлені і скинуті (IE0, IE1, TF0, TF1, RI, TI) з тим же результатом, що і апаратно. Тобто переривання можуть викликатися програмно і відповідно програмно ліквідовуватися.

Переривання $\overline{\text{INT0}}$, $\overline{\text{INT1}}$ можуть викликатися програмно встановленням P3.2=0 і P3.3=0, як показано в прикладі с.135.

В прикладі запити $\overline{\text{INT1}}$ і $\overline{\text{INT0}}$, маючи різний пріоритет, викликаються одночасно, але обслуговується $\overline{\text{INT1}}$, якому призначено більш високий пріоритет по замовчуванню.

Прапорці IE0, IE1, TF0, TF1, RI, TI встановлюються незалежно від того, дозволено чи ні переривання в реґістрі IE.

При виникненні переривання і завершенні виконання біжучої команди апаратно реалізується команда LCALL. При цьому завантажується в стек PC адреса відповідної команди обробки переривань згідно з таблицею джерел переривань (2 байти). Підпрограма обслуговування переривань продовжується до виконання команди RETI. Після чого відновлюється стан логіки переривань і в PC завантажуються 2 байти із стека. Таким чином, RETI дозволяє знову виконувати переривання з будь-яким пріоритетом.

Приклад 12. Передати керування одній із восьми підпрограм при появі нульового рівня на відповідному вході порту 1. Вищий

пріоритет у входу P1.3.

```
    ORL P1, #0FFH          ; настройка P1 на ввід
L1 : MOV A, P1             ; ввід даних з порту
    CPL A                  ; інверсія акумулятора
    JZ L1                  ; очікування появи першого нуля
    JNB P1.3, SUBR1        ; перехід на першу підпрограму
    JNB P1.4, SUBR2        ; перехід на другу підпрограму
     .....
    JNB P1.2, SUBR8        ; перехід на восьму підпрограму
```

Порядок пріоритетів визначається порядком перевірки нульового рівня на відповідному вході і може бути будь-яким. Якщо очікування не потрібне, тобто перехід здійснюється по заданому коду, то немає необхідності очікувати перший нуль.

Приклад 13. Умовний перехід по N підпрограмах.

Використовується таблиця зміщень в ПЗП для вказівників початкових адрес підпрограм. Інструкція JMP @A+DPTR виконує перехід за адресою, яка визначається в процесі виконання програми. При цьому значення акумулятора і DPTR зберігаються.

Приклад програми, в якій байт пам'яті із ЗПД зчитується в акумулятор з однієї з 4-х областей пам'яті, що визначається змінною, зчитуваною MEMSEL. Адреса читаного байта визначається R0. При цьому розмір таблиці і довжина всіх програм не повинні перевищувати 256 байтів.

```
    MEMSEL EQU R3
BRANCH: MOV A, MEMSEL
    MOV DPTR, #TBL
    MOVC A,@A+DPTR
    JMP @A+DPTR
TBL: DB MSP0-TBL
    DB MSP1-TBL
    DB MSP2-TBL
    DB MSP3-TBL
    MSP0: MOV A, @R0       ; зчитування внутрішньої пам'яті
    RET
MSP1: MOVX A, @R0          ; зчитування 256 байтів зовнішньої
                             пам'яті
    RET
MSP2: MOV DPL, R0        ; зчитування 64К зовнішньої пам'яті
```

```
    MOV DPH, R1
    MOVX A, @DPTR
    RET
MSP3: MOV A, R1 ; зчитування 4К зовнішньої пам'яті
    ANL A, #07H
    ANL P1, #11111000B
    ORL P1, A
    MOVX A, @R0
    RET
```

4.2. Арифметичні операції в МК51

Приклад 14. Додати десяткові двійково-кодовані числа, розміщені в A і R5:

```
    ADD A, R5 ; двійкове додавання
    DA A ; десяткова корекція результату
```

Приклад 15. Віднімання байтів. Може бути виконано двома методами: перевести зменшуване як від'ємне в доповнюючий код і виконати операцію додавання; перевести зменшуване в зворотний код і провести інверсію суми.

Наприклад, із даних в A необхідно відняти дані реґістра R3. Віднімання виконуємо за алгоритмом

$A \leftarrow \overline{\overline{A} + R3}$.

```
    CPL A  ; інверсія акумулятора
    ADD A, R3 ; додавання байтів
    CPL A  ; отримання різниці
```

Приклад 16. Додати два двійкові багатобайтові числа. Обидва доданки знаходяться в РПД, починаючи з молодшого байта. Початкова адреса доданків знаходиться в R0 і R1, формат в R2. Результат розміщують на місці першого доданка.

```
    CLR C ; скидання переносу
LOOP: MOV A, @R0 ; завантаження в акумулятор
                 біжучого байта першого доданка
    ADDC A, @R1 ; додавання байтів з урахуванням переносу
    MOV @R0, A ; розміщення байта результату
    INC R0 ; зміщення вказівника
    INC R1 ; зміщення вказівника
    DJNZ R2,LOOP ; цикл, якщо не всі байти просумовані
```

Час сумування складає (1+7N) мкс, де N - довжина в байтах.

Приклад 17. Перемножити число в акумуляторі на число 2^x, де $x \le 8$ - значення в R6. Множення на 2 замінюється арифметичним зсувом акумулятора і R0:

```
    MOV R0, #0 ; скидання R0
    CLR C ; скидання переносу
L1 : RLC A ; зсув вліво 16-ти розрядного даного в A і R0
    XCH A, R0
    RLC A
    XCH A, R0
    DJNZ R6, L1 ; цикл
```

Приклад 18. Множення (MUL). Виконується для двох беззнакових чисел, що знаходяться в акумуляторі А і його доповненні В. Після виконання операції молодший байт добутку розміщується в А, старший – у В.

Множення числа будь-якого формату на константу 168. Число розміщене в РПД, адреса молодшого байта знаходиться в R0. Формат числа в байтах заданий в R2.

```
    MOV A, #00H ; скидання акумулятора A
LOOP: ADD A, @R0 ; завантаження множеного
    MOV B,#168D ; завантаження множника
    MUL AB ; перемноження
    MOV @R0, A ; запис молодшого байта часткового добутку
    INC R0 ; приріст адреси
    MOV A,B ; пересилка старшого байта часткового добутку в
                акумулятор A
    XCH A,@R0 ; попереднє формування чергового байта добутку
    JNZ R2,LOOP ; цикл, якщо не всі байти початкового числа
                перемножені на константу.
```

Отриманий результат знаходиться на місці початкового числа і займає в РПД на 1 байт більше. Час обчислення добутку складає (1+13N) мкс, де N - довжина числа в байтах.

Приклад 19. Ділення (DIV) - ділиться вміст акумулятора на значення в В. Після ділення в А знаходиться ціла частина частки, в В - остача. Команда може бути використана для швидкого перетворення двійкового числа в двійково-десятковий формат (BCD-числа).

Програма переводу двійкового числа в акумуляторі в BCD-код, який може бути трирозрядним (в десятковій системі числення). Старше число - сотень буде розміщено в R0, дві молодші цифри - в А і В.

```
MOV B,#100D ; (B)←100 для обчислення числа сотень в
              числі
DIV AB ; в А є число сотень (тобто старша цифра)
MOV R0,A ; пересилка в R0 старшої цифри
XCH A,B ; пересилка остачі від ділення вхідного числа в А
MOV B,#10D ; (B) ← 10 (визначається число десятків у
             числі)
DIV AB ; в А - число десятків, В - число одиниць
SWAP A ; розміщення числа десятків в старшій тетраді А
ADD A,B; підсумовування остачі (числа одиниць), акумулятор
         зберігає дві молодші цифри.
```

Час перетворення - 16 мкс.

Приклад 20. Підпрограма DISBCD - розпаковування двох запакованих двійково-десяткових чисел, отриманих в акумуляторі, і повертання їх двійково-десяткового добутку в акумулятор.

```
MOV B, #10H ; початкове значення ділиться на 16
DIV AB ; в регістрах А і В зберігаються виділені числа
MUL AB ; регістр А має добуток в 2-ому форматі (0-99) або
         (0-63H)
MOV B, #10D ; добуток ділиться на 10
DIV AB ; регістр А містить число десятків, регістр В - остачу
SWAP A ; цифри в запакованому форматі
ORL A, B
RET
```

Приклад 21. Порівняння двох чотирирозрядних чисел, що подані на входи порту Р2, на рівно, більше і менше.

Сигнал рівності виводиться на Р1.1, більше - Р1.2, менше - Р1.3, дозвіл порівняння подається на Р1.0.

```
    ANL P1, #11110001B    ; скидання всіх порівнянь
L0 : JNB P1.0, L0         ; очікування дозволу
    MOV A, P2
    ANL A, #F0H           ; виділення першого числа
    SWAP A
    MOV B, A              ; запам'ятовування першого числа
    CLR C                 ; скидання переносу
    MOV A, P2
    ANL A, #0FH           ; виділення другого числа
    SUBB A, B             ; обчислення A-B
    JZ LR                 ; перехід, якщо A=B
```

```
    JC LL ; перехід, якщо A<B
    SETB P1.2 ; результат A>B
    JMP EXIT
LR : SETB P1.1 ; результат A=B
    JMP EXIT
LL : SETB P1.3 ; результат A<B
    EXIT: ; продовження виконання програми
```

Аналогічно виконується порівняння чисел більшої розрядності. Якщо необхідно порівнювати багатобайтові числа, то порівняння слід починати із старших байтів.

4.3. Логічні операції в МК51

Приклад 22. Вивід керуючих сиґналів із МК.

Формування статичних сиґналів використовується для керування виконавчим механізмом за принципом "включений–виключений". При цьому відповідні лінії портів просто встановлюються в 0 або 1.

Якщо маємо групу виконавчих механізмів, то для їх керування необхідно сформувати керуюче слово (КС), кожний біт якого є відповідним керуючим сиґналом.

Для зміни КС достатньо виконувати логічні операції над тими бітами портів, які необхідно змінити.

Наприклад: ANL - для скидання тих бітів КС, які задані нулями

ORL - для встановлення бітів

XRL - інверсія бітів по часу

Приклад 23. Виявлення переповнень. При накопиченні великої кількості доданків може виникнути переповнення розрядної сітки. Для виявлення цього використовують модифікований доповняльний код, який відрізняється від звичайного доповняльного введенням додаткового знакового розряду. При додаванні k доданків таких розрядів повинно бути

$$r = \log_2 k$$

Ознакою переповнення служить неоднаковість знакових розрядів. Відзначимо, що використання r знакових розрядів звужує діапазон зміни сиґналів в 2^r разів.

Приклад 24. Скинути біти 0,2,4,6 порту 2:

```
ANL P2, #10101010B ; скидання бітів 0,2,4,6 порту 2
```

Приклад 25. Встановити біти 0...3 порту 1

```
ORL P1, #00001111B ; ( P1.0...P1.3) ← 1111
```

Можна також записати

```
ORL P1, #0FH
```

Приклад 26. Вибрати нульовий реґістровий банк:

```
ANL PSW, #11100111B ; скидання бітів RS0 і RS1
```

Приклад 27. Проінвертувати біти порту P1, які відповідають одиничним бітам в акумуляторі:

```
XRL P1, A ; сума за модулем два значень порту 1 і
            акумулятора
```

Приклад 28. Проінвертувати біти 7,6,5,4 порту 0:

```
XRL P0, #11110000B; сума за модулем два значень порту 0
                    і константи
```

Можна також записати

```
XRL P0, #0F0H
```

Приклад 29. Проінвертувати біти 0...3 в акумуляторі:

```
XRL A, #0FH ; сума за модулем два значень акумулятора і
              константи
```

Приклад 30. Настроїти біти 1,3,5,7 порту 1 на ввід:

```
ORL P1, #10101010B ; встановлення P1.1, P1.3, P1.5,
                     P1.7
```

Приклад 31. Маскування даних при вводі. Ввести в реґістр R3 інформацію з ліній 1,3,5,6,7 порту 1:

```
MOV A, P1  ; ввід байта з P1
ANL A, #11101010B ; маскування
MOV R3, A  ;  передача в R3
```

Приклад 32. Виконати логічний зсув вправо двобайтового числа, що розміщене в R5, A:

```
SHIFTR : CLR C ; скидання переносу
         CPL C ; встановлення переносу
         XCH A, R5 ; обмін байтами
         JNB A.7, L1 ; якщо R5.7=1, то скинути прапорець переносу
```

```
    CLR C
    RRC A ; зсув прапорця переносу
L1 : XCH A, R5 ; обмін
    RRC A ; зсув молодшого байта
```

Приклад 33. Виконати логічний зсув вліво двобайтового числа, що розміщене в R5 і A:

```
SHIFT L : RLC A ; зсув молодшого байта
    XCH A, R5 ; обмін A і R5
    RLC A ; зсув старшого байта
    XCH A, R5 ; обмін
```

Приклад 34. Керування групою бітів порту.

В РПД знаходиться масив розпакованих десяткових цифр. Необхідно передати їх зовнішньому пристрою у відповідності до протоколу (рис.4.2). Для передачі 4-х бітів даних використовуються молодші лінії порту 1. Лінії P1.4 і P1.5 використовуються як сиґнали квітування, тобто передачу сиґналів на вихід МК супроводжує стробуючий сиґнал на лінії P1.4. Зовнішній пристрій, який прийняв дані, сповіщає про це сиґналом на вході P1.5. Біти P1.6 і P1.7 не повинні змінювати своїх значень.

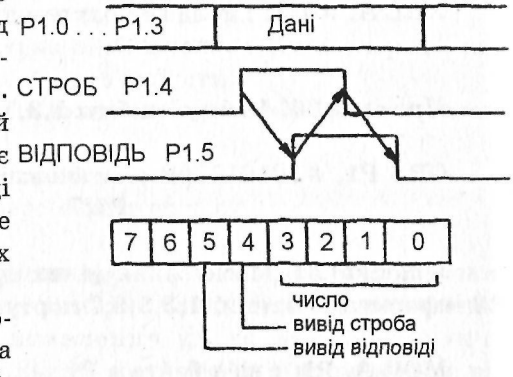


Рис. 4.2. Протокол виводу даних

Початкові дані програми: початкова адреса масиву знаходиться в (R0), довжина масиву знаходиться в (R1).

```
    ORL P1, #00100000B ; настроювання P1.5 на ввід
LOOP: MOV A, @R0 ; завантаження байта в акумулятор
    ANL P1, #11100000B ; скидання даних і стробу
    ORL P1, A ; видача даних
    ORL P1, #00010000B ; видача стробу
WAIT: JNB P1.5, WAIT ; очікування відповіді
    INC R0 ; пересування вказівника адреси
    JNZ R1, LOOP ; цикл, якщо не всі дані передані
```

4.4. Бітові операції. Бітовий процесор МК51

Приклад 35. Операція Виключне АБО використовується дуже часто для порівняння бітів, або їх скидання. В кодах бітового процесора такої операції немає, але легко здійснюється накладання логічної операції Виключне АБО на прапорець переносу:

```
JNB bit, LZ ; виключне АБО для прапорця переносу
CPL C ; інвертування прапорця

LZ : ; продовження програми
```

Приклад 36. Передати байт послідовним кодом через P1.0, залишаючи без змін решту бітів. Передачу вести, починаючи з молодшого біта:

```
    MOV R7, #8D ; ініціалізація лічильника циклів
LOOP: RRC A ; присвоєння переносу значення біта A.0
    MOV P1.0, C ; передача біта
    DJNZ R7, LOOP ; цикл, якщо не всі біти передані
```

Час виконання програми 41 мкс, час передачі - 5 мкс (швидкість передачі - 200 кбіт/с).

Приклад 37. Обчислити буліву функцію 3-х змінних $Y=(X\cdot\overline{V})+W\cdot(X+V)$. Змінні X, V, W поступають на лінії 2, 1, 0 порту 1, результат Y необхідно вивести на лінію 3 порту 1. Для зберігання проміжних значень використати біт F0.

```
    Y BIT P1.3 ; специфікація бітів порту 1
    X BIT P1.2
    V BIT P1.1
    W BIT P1.0
    MOV C, X ; ввід X
    ANL C, /V ; X·V̄
    MOV F0, C ; запам'ятовування результату в F0
    MOV C, X ; ввід X
    ORL C, V ; X+V
    ANL C, W ; W·(X+V)
    ORL C, F0 ; (W·(X+V))+(X·V̄)
    MOV Y, C ; вивід результату
```

Час виконання програми 14 мкс.

Приклад 38. Організувати послідовну передачу даних із акумулятора на 0 вивід порту 2. Пересилку вести манчестерським кодом (кожний біт кодується двома інтервалами: перший інтервал має інверсію біта, другий - його пряме значення):

```
MOV R0, #8D ; ініціалізація лічильника бітів
LOOP: RRC A ; (C) ← зсув молодшого біта із акумулятора в
                    перенос
CPL C ; інверсія біта
MOV P2.0,C ; передача інверсії біта
CPL C ; відновлення біта
NOP ;
NOP ; вирівнювання довжини інтервалів
NOP ;
MOV P2.0, C ; передача прямого значення біта
DJNZ R0, LOOP ; цикл, якщо лічильник не нульовий
```

Передача виконується молодшими бітами вперед. Тривалість одного інтервалу 6 машинних циклів (6 мкс), час передачі одного біта - 12 мкс, час передачі байта - 96 мкс (швидкість передачі 83кбіт/с, або 10,4кбайт/с).

Приклад 39. Покроковий режим роботи. Використовується особливість системи переривань МК51 - до виконання команди RETI забороняються всі переривання з нижчим пріоритетом, а після виконання RETI виконується хоча б одна команда основної програми. Переривання активізуються по рівню.

В основній програмі необхідно дописати таку послідовність команд:

```
SETB IE.0 ; дозвіл перериваня рівня 0
CLR TCON.0 ; переривання дозволені по нульовому рівню
```

Підпрограма обслуговування переривання має таку послідовність:

```
STEP : ; дії за підпрограмою
L1 : JNB P3.2, L1 ; очікування рівня 1
L2 : JB P3.2, L2 ; очікування рівня 0
RETI ; повернення і виконання однієї команди основної
```

програми, після чого знову відбувається повернення в підпрограму.

Покроковий режим зручний для відлагодження програм користувача.

Приклад 40. Звертання до повільних мікросхем зовнішньої пам'яті. Програмним шляхом можна додати необхідну тривалість імпульсів $\overline{WR}$ та $\overline{RD}$. Наприклад, якщо сигнал $\overline{RD}$ повинен тривати 50 мкс, то це здійснюється так:

```
CLR P3.7 ; RD =0
MOV R3, #24D ; ініціалізація лічильника (2мкс)
L0 : DJNZ R3, L0 ; цикл (24*2мкс)

SETB P3.7 ; RD=1
```

Приклад 41. Розширена бітова адресація. МК51 може адресувати прямо лише 144 біти в РПД. Розповсюдження на будь-який біт з деякою втратою ефективності відбувається наступним чином.

Виконуються логічні операції над змінними байтів, один з яких дозволяє джерелу служити безпосередньою маскою, а приймачем є біт з прямою адресою. Будь-який біт може бути встановлений, очищений або доповнений з допомогою команди, при якій маска накладається на всі біти, крім одного:

```
SETBR3: MOV B, R3 ; встановлення R3.7
    SETB B.7 ; встановлення B.7
    MOV R3, B ; передача в R3

JMPR3 : MOV A, R3 ; перехід, якщо R3.6 =1
    JB A.6, L1
L1 : ; - мітка для продовження програми
```

Приклад 42. Порівняння на рівність двох чотирирозрядних чисел, що подані на входи порту 2. Дозвіл порівняння поданий на вхід порту 1, розряд 0, сигнал рівності чисел виводиться на розряд 1 порту 1. З використанням команд бітового процесора умова порівняння двох чисел A і B еквівалентна обчисленню значення

$$F= A_1 \oplus B_1 + A_2 \oplus B_2 + A_3 \oplus B_3 + A_4 \oplus B_4,$$

де $\oplus$ – сума за модулем два;

$+$ – логічне АБО;

A_i, B_i, i=1,...,4 - відповідні розряди чисел. У випадку нерівності будь-яких розрядів значення F=1.

```
L0 : JNB P1.0, L0 ; очікування дозволу
     CLR P1.1 ; скидання рівності чисел
     MOV C, P2.7
     JNB P2.3, L1 ; перевірка рівності старших розрядів
     CPL C
L1 : JC L0 ; перехід, якщо нерівні
     MOV C, P2.6 ; перевірка рівності наступних розрядів
     JNB P2.2, L2
     CPL C
L2 : JC L0
     MOV C, P2.5
     JNB P2.1, L3
     CPL C
L3 : JC L0
     MOV C, P2.4
     JNB P2.0, L4
     CPL C
L4 : JC L0
     SETB P1.1 ; встановлення сиґналу рівності чисел
     JMP L0 ; перехід на початок програми
```

4.5. Різні операції для порівняння ефективності ОМЕОМ типу МК51 і МК48

Порівнюється програмна ємність пам'яті та час виконання програми.

Приклад 43. Встановити прапорець переносу

```
SETB C ; встановити прапорець C
```

МК51-1 байт/1мкс
МК48-2 байта/5мкс

Приклад 44. Встановити біт 5 порту 1:

```
SETB P1.5 ; встановлення P1.5
```

МК51 – 2 байти/1мкс
МК48 – 2 байти/5мкс

Приклад 45. Проінвертувати біт 4 порту 1:

```
CPL P2.4 ; інверсія P2.4
```

МК51 – 2 байти/1мкс
МК48 – 4 байти/15мкс

Приклад 46. Скинути прапорець в РПД:

```
FLAG BIT 20H.0 ; визначення прапорця
CPL FLAG ; скидання прапорця в РПД
```

МК51 – 2 байти/1мкс

МК48 – 6 байтів/15мкс

Приклад 47. Перейти, якщо програмний прапорець F0 - скинутий

```
JNB PSW.5, ZERO ; перейти до ZERO, якщо PSW.5=0
```

МК51 – 3 байти/2мкс

МК48 – 4 байти/10мкс

Приклад 48. Перевірити на 0 лінію 5 порту 1:

```
JNB P1.5, ZERO ; перейти до ZERO, якщо P1.5=0
```

МК51 – 3 байти/2мкс

МК48 – 4 байти/12,5мкс

Приклад 49. Очікування заданого коду на входах порту. Задано код: AAH, порт 1:

```
MOV A, #0AAH ; завантаження AAH
WAIT: CJNE A,P1,WAIT ; очікування приходу коду AAH
```

МК51 – 5 байтів/3мкс

МК48 – 5 байтів/15мкс

Приклад 50. Обчислити буліву функцію $Q=X\cdot(Z+Y)+\overline{W}$.

Змінні X,Y,Z,W подаються на вхідні лінії порту 1 (біти 0...3), функція Q видається на лінію 4 цього ж порту:

```
Q BIT P1.4 ; специфікація бітів порту 1
X BIT P1.0
Y BIT P1.1
Z BIT P1.2
W BIT P1.3
MOV C,Z ; ввід Z
ORL C,Y ; Z+Y
ANL C,X ; X·(Z+Y)
ORL C,/W ; (X·(Z+Y)+W)
MOV Q,C ; вивід Q
EXIT ; вихід із програми.
```

МК51 – 10 байтів/9мкс

МК48 – 21 байт/35мкс

Таким чином, ефективність бітового процесора МК51 дозволяє підвищити швидкість виконання бітових операцій у порівнянні з МК48 в середньому в 4...5 разів.

4.6. Взаємодія МК з об'єктом керування

Приклад 51. Очікувати появу нульового рівня на вході Т0:

WAIT : JB P3.4, WAIT ; перехід на WAIT, якщо на вході таймера/лічильника 0 одиничний рівень

Приклад 52. Опитування двійкового давача, наприклад, кінцевого вимикача.

Рис.4.3. Схема узгодження двійкового давача з МК51

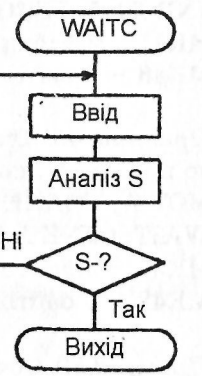


Рис. 4.4. Блок-схема опитування двійкового давача

Ключ під'єднаний до входу МК51: порт 1, розряд 3.

```
WAIT0: JNB P1.3, WAIT0 ; очікування розмикання
                         давача
WAITC: JB P1.3, WAITC ; очікування замикання давача
```

Приклад 53. Опитування групи двійкових давачів (аналогічно знаходженню заданого коду або комбінації давачів).

```
WTCODE: MOV A, #10D ; завантаження в А еталонного
                      коду 00001010B
WAIT : CJNE A, P1, WAIT ; якщо кодова комбінація не
                          співпала з заданою, то чекати
.....
EXIT:                   ; вивід
```

Приклад 54. Якщо код рівний 135, то необхідно передати управління на підпрограму з міткою LABEL A, інакше LABEL B.

```
MOV A, #135D ; завантаження контрольованого коду
CJNE A, P1, LABEL B ; порівняння і передача управління
LABEL A:
LABEL B:
```

Приклад 55. Очікування імпульсного сиґналу. МК51 повинен виявити не тільки факт появи, а й факт закінчення сиґналу.

```
WAITC: JB P1.3, WAITC ; очікування P1.3=0
WAIT0: JNB P1.3, WAIT0 ; очікування P1.3=1
```

Виявлений імпульсний сиґнал має вигляд ⌐⌙

Для сиґналу зворотного виду ⌙⌐ потрібно переставити WAIT0 і WAITC місцями.

Мінімальна тривалість сиґналу, який виявляє МК

Таблиця 4.1

Підключення давача до виводів	Мінімальна тривалість імпульсу, мкс	
	(від'ємного)	(додатного)
P1,P2,BUS/P0	10/2	12,5/2
T0,T1	5/2	5/2
ЗПР	10/2	5/2

Приклад 56. Передати керування на мітку TEST і встановити P3.7, якщо на T0 поступить 30 імпульсів.

```
     MOV  R1, #30D : завантаження числа імпульсів
L1   : JB  P3.4, L1  : очікування нуля
L0   : JNB  P3.4, L2  : очікування одиниці
     JMP  L0
L2   : DJNZ R1, L1  : повторити 30 разів
     JMP TEST
     .....
TEST : SETB P3.7 : встановлення біта
```

Приклад 57. Схема для фіксації короткого імпульсу: D-тригер встановлюється коротким імпульсом, а скидається програмно, після визначення наявності сиґналу на вході T0.

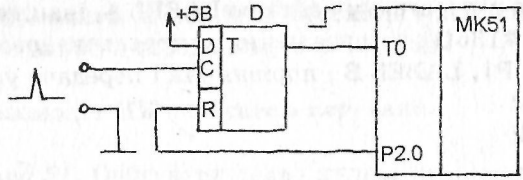


Рис. 4.5. Схема фіксації коротких імпульсів

Приклад 58. Усунення деренчання контактів.

Деренчання полягає в тому, що завдяки пружним властивостям контактів можливий відрив контактів, який приводить до перехідного процесу. При цьому сиґнал може бути прочитаний багаторазово як випадкова послідовність нулів і одиниць. Це можна усунути за допомогою тригера (рис.4.6).

Частіше з допомогою МК51 це робиться програмно так, як показано на рис.4.7 і рис.4.8.

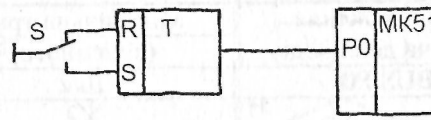


Рис. 4.6. Апаратне усунення деренчання контактів

Рис. 4.7. Метод 1: підрахунок заданого числа співпадінь значень сиґналів

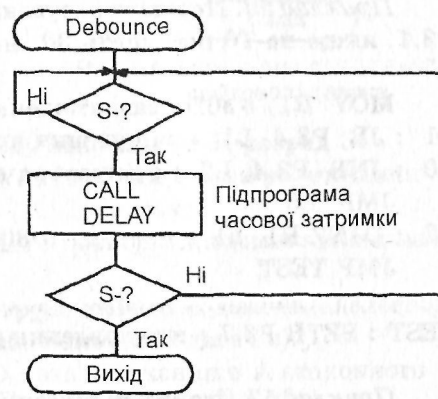


Рис. 4.8. Метод 2: часова затримка

Реалізація першого методу, кількість співпадінь задана в R3.

```
DBNC : MOV R3, #3 ; ініціалізація лічильника
DBNC1: JB P3.4, DBNC ; якщо контакт замкнутий, то
                       почати відлік опитувань сиґналу
                       спочатку
DJNZ R3, DBNC1 ; повторювати, поки значення в R3 не
                 стане рівним 0
JB P3.4, DBNC ; кінцева перевірка
```

Часова затримка підбирається експериментально (1-10мс) в залежності від типу давача.

Приклад 59. Формування імпульсних сиґналів робиться за принципом включено/виключено з необхідною часовою затримкою:

```
PULLS:ON: ANL P1,#11110111B; видача імпульсу в лінію 3
                            порту 1
   CALL DELAY ; часова затримка
   OFF: ORL P1,#00001000B; виключення
```

Приклад 60. Ґенерація меандра. Використовується затримка на половину часу періоду:

```
MEANDR:
XCOR: CPL P1.3 ; інвертування біта
      ACALL DLYX ; часова затримка
      SJMP XCOR
```

Приклад 61. Зчитування даних з таймера. Для усунення можливих збоїв при зчитуванні спочатку зчитується старший байт, потім – молодший, потім підтверджується, що старший байт за цей час не змінився.

```
RDTIME: MOV A, TH0
        MOV R0, TL0
        CJNE A, TH0, RDTIME
        MOV R1, A
        RET
```

Приклад 62. Підрахунок числа імпульсів, наприклад, книг, що рухаються по конвеєру, зафіксованих фотоелементом (рис.4.9).

```
MOV TMOD,#01000000B; настроювання лічильника 1
MOV TH1, #00H ; скидання лічильника книг
```

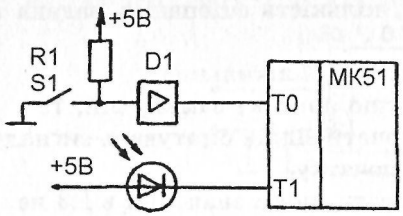


Рис. 4.9. Схема підрахунку числа книг

```
WAIT0: JB P3.4,WAIT0; очікування включення транспортера
SETB TCON.6 ; запуск лічильника 1
WAITC: JNB P3.4, WAITC ; очікування виключення транспортера
```

```
CLR TCON.6  ; зупинка лічильника 1
MOV A,TH1   ; A ← число книг
EXIT:       ; вихід
```

Максимальне число книг 255.

Приклад 63. Реалізація функцій часу на основі таймерів.

В МК51 на вхід таймера/лічильника можуть поступати сигнали з частотою 1 МГц (в режимі таймера), або сигнали від зовнішнього джерела (в режимі лічильника).

Якщо використовувати таймер/лічильник в режимі таймера повного формату (16 бітів), то отримаємо затримки в межах (1 - 65536 мкс).

Програма затримки на 50 мс в МК51. Біт IE.7 встановлений: організація переходу до мітки NEXT при переповненні Т/Л0:

```
ORG 0BH         ; адреса вектора переривань від Т/Л0
CLR TCON.4      ; зупинка Т/Л0
RETI            ; вихід із підпрограми обробки переривань

ORG 100H        ; початкова адреса програми
MOV TMOD, #01H ; настроювання Т/Л0
MOV TL0, #LOW(NOT(50000)-1) ; завантаження таймера
MOV TH0, #HIGH(NOT(50000)-1)
SETB TCON.4     ; старт Т/Л0
SETB IE.1       ; дозвіл переривань від Т/Л0
SETB PCON.0     ; перевід в режим холостого ходу

NEXT:           ; продовження програми.
```

Приклад 64. Програмним шляхом затримка реалізується наступними операторами:

```
DELAY : MOV R2, X ; (R2) ← X
COUNT : DJNZ R2, COUNT; декремент R2 і цикл, якщо не нуль
RET ; повернення з підпрограми
```

Приклад 65. Підпрограма затримки на 50 мс, використовуючи Т/Л0, похибка не перевищує 2 мкс.

```
DELAY : MOV TMOD, #00000001B ; настроювання Т/Л0
        MOV TH0, #HIGH(NOT(50000-16))
        MOV TL0, #LOW(NOT(50000-16))
        SETB TCON.4 ; запуск Т/Л0
    D : JNB TCON.5, D ; очікування
        ANL TCON, #NOT(30H) ; зупинка Т/Л0, скидання TF0
        RET ; повернення із підпрограми
```

Приклад 66. Підпрограма затримки на 1 секунду. Похибка не більша, ніж 123 мкс.

```
OS : MOV R3, #20D ; лічильник циклів
S1 : ACALL DELAY ; затримка на 50 мс
     DJNZ R3, S1 ; організація циклу
     RET ; повернення із підпрограми
```

Приклад 67. Вимірювання часових інтервалів.

При застосуванні таймера використовується вхід дозволу лічильника ($\overline{INT}$). Тоді вимірюваний сигнал подається на вхід $\overline{INT0}$, а вимірювання тривалості виконується в Т/Л0.

«Додатний» імпульс:

```
    MOV TMOD, #00001001B ; настроювання Т/Л0
    MOV TH0, #0 ; скидання таймера
    MOV TL0, #0 ;
    SETB TCON.4 ; запуск Т/Л0
WAIT0: JNB P3.2, WAIT0 ; очікування одиничного рівня
WAITC: JB P3.2, WAITC ; очікування нульового рівня
    CLR TCON.4 ; зупинка Т/Л0
RET; вихід із процедури
```

Керування програмі повинно передаватися за умови $\overline{INT0}$=0. Переривання від Т/Л0 і зовнішні від $\overline{INT0}$ - повинні бути заборонені. По завершенні програми в Т/Л0 буде знаходитися число, пропорційне тривалості імпульсу на вході $\overline{INT0}$. Максимальна тривалість імпульсу 65536 мкс, похибка 1 мкс.

Якщо необхідно виміряти більшу тривалість імпульсу, то можна програмним шляхом підраховувати число переповнень від таймера.

Приклад 68. Перетворення паралельного коду в послідовний в МК51 зводиться до передачі одного байта в буфер прийомопередавача:

```
MOV SBUF, A
```

Приклад 69. Зворотне перетворення послідовного коду в паралельний ініціюється стоповим бітом і виконується апаратно без участі програми. Основна програма повинна виконати команду

```
MOV A, SBUF.
```

Приклад 70. Перетворення однобайтових чисел в доповняльний код і навпаки. Одним байтом можна подати числа зі знаком в доповняльному коді в межах від -128 до +127. Алгоритм переводу чисел із прямого коду зі знаком в доповняльний і зворотного перетворення однаковий:

```
DOD-PR : JNB A.7, EXIT ; перевірка знакового розряду
    CPL A ; інверсія акумулятора
    ADD A, #1 ; додавання одиниці
    SETB A.7 ; встановлення знака
EXIT : ; вихід
```

Для прикладу знайдемо доповняльний код числа -4: для числа +4 маємо 00000100B. Інверсія цього коду 11111011B, додавання одиниці дає 11111100B, тобто доповняльний код числа -4.

При перетворенні чисел в доповняльний код знак встановлюється автоматично, при зворотному - лише апаратно.

Зворотний код не рекомендується використовувати в мікро-ЕОМ через неоднозначність подання числа 0, що приводить до додаткових труднощів при цифроаналоговому перетворенні.

Приклад 71. Перетворення 3-бітового двійкового коду в R2 у 8-бітовий унітарний. Перетворення здійснюється шляхом зсовування одиниці із прапорця переносу:

```
    INC R2 ; інкремент початкового коду
    MOV A, #80H ; встановлення старшого біта акумулятора
REVER : RL A ; зсув унітарного коду
    DJNZ R2, REVER ; декремент позиційного коду і цикл,
                   поки не нуль
```

Перетворення здійснюється за N+1 зсувів

Приклад 72. Перетворення чисел із одної системи числення в іншу методом «двох лічильників». При цьому методі із початкового коду віднімається, а до нового коду додається одиниця до обнулення початкового коду. Причому віднімання здійснюється в старій системі числення, а додавання - в новій.

Програма переводу двійкового числа в двійково-десяткове.

```
    MOV R5, A ; передача початкового числа в R5
    CLR A ; скидання акумулятора
REV : ADD A, #1 ; додавання одиниці
    DA A ; десяткова корекція
    DJNZ R5, REV ; декремент початкового коду і цикл, якщо
                   не нуль
```

Результат перетворення знаходиться в акумуляторі.

Приклад 73. Підрахунок числа імпульсів за заданий проміжок часу. Вирішується 4-ма методами:

1. Програмна реалізація часового інтервалу і програмний підрахунок числа імпульсів.

2. Програмна реалізація часового інтервалу і апаратний підрахунок числа імпульсів (на внутрішньому таймері/лічильнику).

3. Апаратна реалізація часового інтервалу і програмний підрахунок числа імпульсів.

4. Апаратна реалізація часового інтервалу і апаратна реалізація підрахунку імпульсів.

Реалізація на МК51: в таймері-лічильнику Т/Л1 здійснюється підрахунок числа імпульсів; в Т/Л0 заданий часовий інтервал. Давач імпульсів повинен бути під'єднаний до Т1.

```
    TIME: EQU NOT(10000)+1 ; визначення константи TIME
                             для відліку часу в 10мс
    MOV TMOD,#01010001B ; настроювання таймерів-
лічильників: 1-й - лічильник 16 бітів, 0-й - таймер
    CLR A ; скидання акумулятора
    MOV TH1,A ; скидання Т/Л1
    MOV TL1,A ;
    MOV TH0, #HIGH(TIME) ; завантаження в Т/Л0 константи
                           TIME
    MOV TL0, #LOW (TIME) ;
    ORL TCON, #50H ; запуск Т/Л1 і Т/Л0
WAIT: JBC TCON.5,EXIT ; перевірка переповнення Т/Л0
    SJMP WAIT ; цикл, якщо TF=0
```

```
EXIT: MOV B, TH1 ; (B) ¬ (A), число імпульсів за час 10 мс
      MOV A,TL1
```

Приклад 74. Організація опитування великої кількості давачів.

Опитування більш ніж 32 давачів ОК забезпечується скануванням давачів. Це дозволяє застосувати МК51 в системах з великим числом вхідних і вихідних змінних. На рис. 4.10 показана організація контролера з 64 давачами вхідних змінних, 12 вихідних сигналів та двома зовнішніми перериваннями.

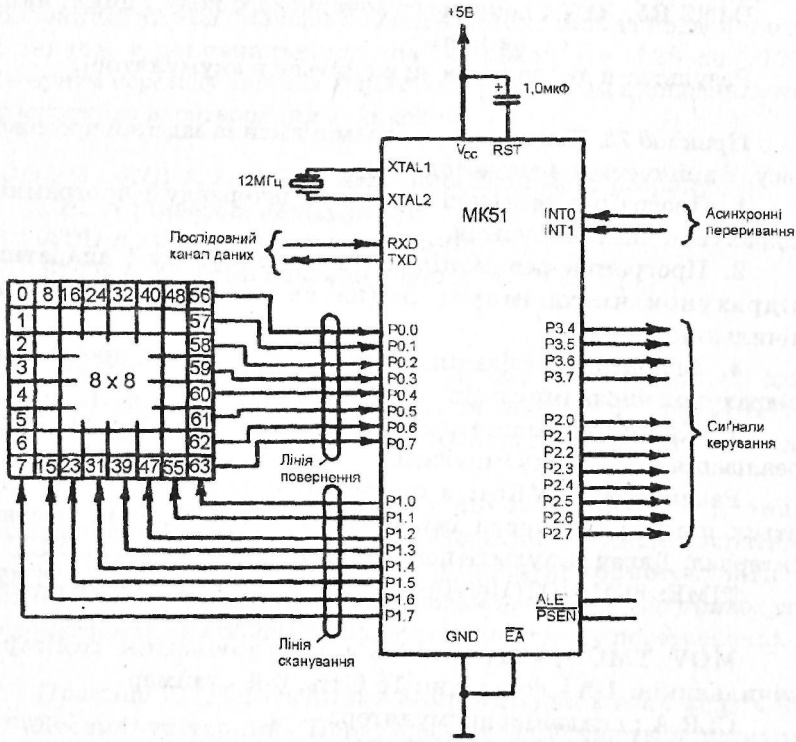


Рис. 4.10. Схема контролера з розширеною кількістю давачів

64 вхідні давачі організовані як матриця розмірністю 8x8. Сигнали на виходах порту 1 послідовно сканують стовпці матриці, а через порт 0 зчитується стан кожного давача в цьому стовпці. Зчитані дані запам'ятовуються у восьмибайтовому блоці ОЗП з бітовою адресацією. Завадостійкий та електрично ізольований від МК51 спосіб реалізації матриці давачів зображений на рис.4.11,

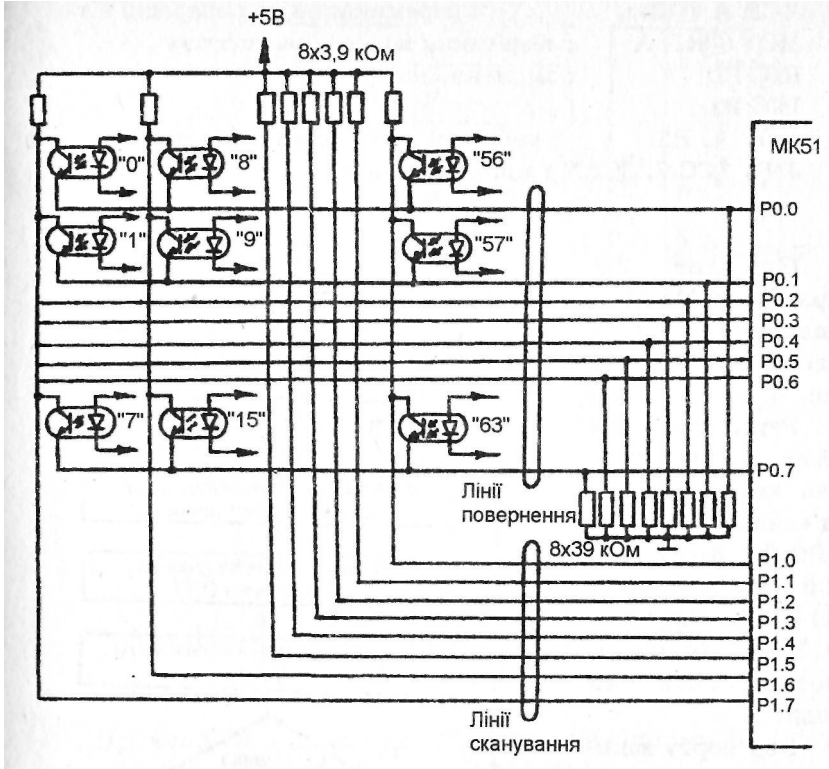


Рис. 4.11. Використання оптронів у схемі контролера

в якому на лініях сканування і повернення використовуються оптрони. Алгоритм сканування матриці давачів подано на рис. 4.12.

В програмі сканування використовуються дві області ОЗП: в одній зберігаються біжучі стани давачів (адреси 0-63), а в іншій – попередні стани (адреси 64-127). При необхідності програма може усувати деренчання контактів перемикачів за алгоритмами, описаними в прикладі 57, або шляхом порівняння кожного біту з його попереднім значенням.

```
    MOV R0, #20H        ; ініціалізація вказівників R0 і R1
    MOV R1, #28H
    MOV A, #80H         ; встановлення A.7
SCAN: MOV P1, A         ; збудження однієї лінії сканування
    RR A                ; зсув на наступну лінію сканування
    MOV R2, A           ; запам'ятовування біжучої позиції
    MOV A, P0           ; читання лінії повернення
```

```
XCH A @R0              ; перемикання на попередні біти
MOV @R1, A     ; зберігання попереднього стану
INC R0         ; зміщення вказівників
INC R1
MOV A, R2      ; вибір наступної лінії опитування
JNB ACC.7, SCAN ; цикл для зчитування всіх 8 стовпців
```

Приклад 75. Спряження МК51 із розширювачем вводу/виводу КР580ВР43 (рис. 4.13).

Реалізується ввід даних із розширювача вводу/виводу, під'єднаного до P2.3 - P2.0. Контакти P2.5 і P2.4 імітують $\overline{CS}$ і $\overline{PROG}$. Контакти P2.7, P2.6 використовуються як входи.

Код порту запитується із A.1 - A.0.

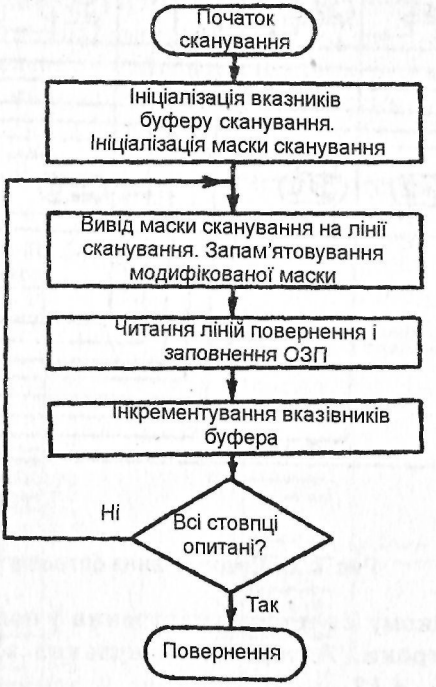


Рис. 4.12. Алгоритм сканування матриці давачів

```
PROG BIT P2.4                  ; опис контакту PROG
BP43: ORL A, #11010000B ; встановлення PROG і контактів
                                входу
MOV P2, A ; вивід коду порту і коду операції
CRL PROG
ORL P2, #00001111B ; встановлення молодших контактів
                                для вводу
MOV A, P2 ; зчитування даних порту
ORL P2, #00110000B ; встановлення PROG і CS
```

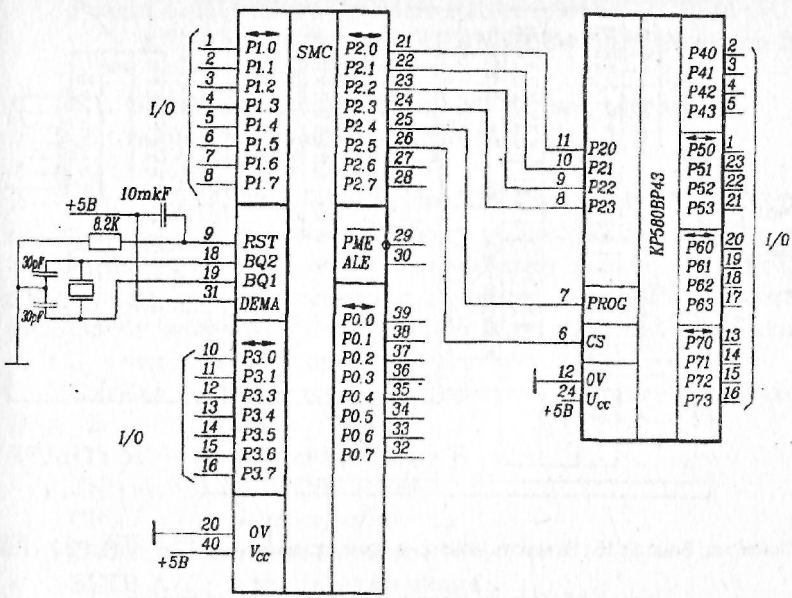


Рис. 4.13. Схема спряження МК51 із розширювачем вводу/виводу КР580ВР43

Приклад 76. Спряження МК 51 із зовнішніми ІС пам'яті програм і даних (рис. 4.14, рис. 4.15) [18].

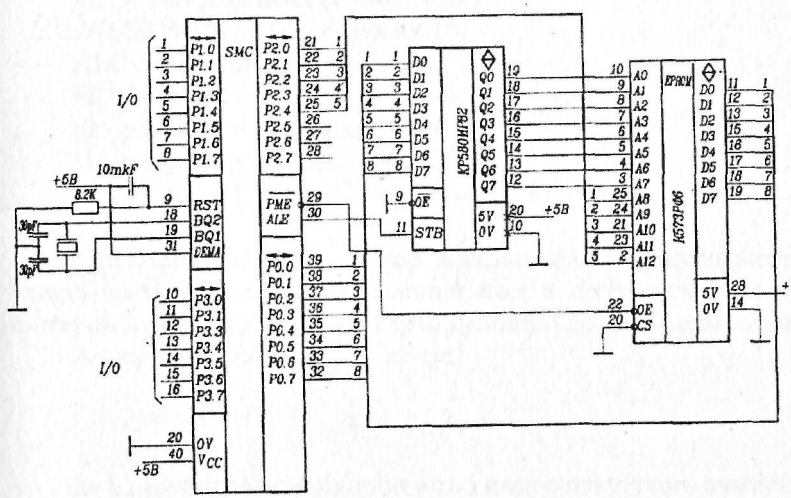
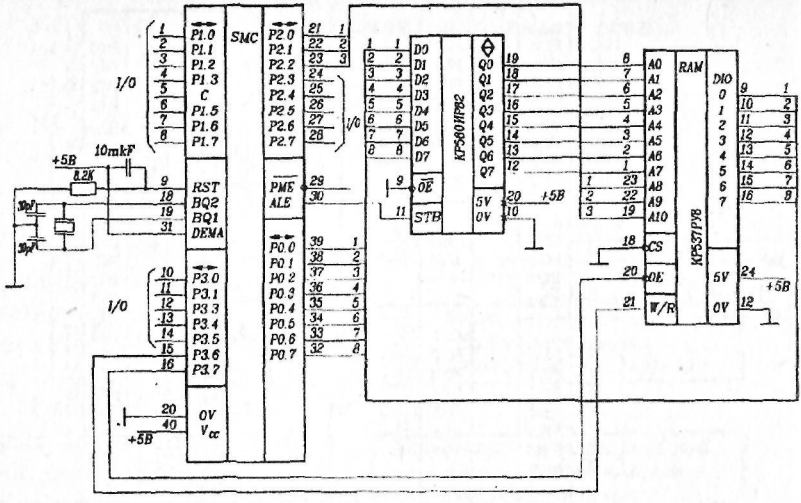


Рис. 4.14. Схема підключення зовнішньої пам'яті програм із застосуванням ІС К573РФ6



Примітка. Виводи 15, 16 мають альтернативне призначення: P3.6 - $\overline{WR}$, P3.7 - $\overline{RD}$

Рис. 4.15. Схема підключення зовнішньої пам'яті даних із застосуванням ІС КР537РУ8

Розділ 5. ПРОЕКТУВАННЯ ЦИФРОВИХ СИСТЕМ КЕРУВАННЯ НА ОДНОКРИСТАЛЬНИХ мікро-ЕОМ

5.1. Приклади використання МК51 для керування технологічними об'єктами

Приклад 77. Формування сиґналу арифметичного пристрою слідкуючої системи електроприводу. Задано: на входах порту Р1-завдання A_n, сиґнал зворотного зв'язку давача швидкості в двійковому коді B_n подається на входи порту Р2. Необхідно визначити сиґнал розузгодженості і видати його в Р3.0. Сиґнали A_n і B_n - числа із знаками.

Сиґнал розузгодженості формується наступним чином $D=A_n-B_n$.

```
START: MOV A, P2 ; зчитування Bn
       JNB A.7, L1 ; аналіз знака
       CPL A ; інверсія акумулятора
       INC A ; додавання одиниці
       SETB A.7 ; встановлення знака
  L1: XCH A, R0 ; запис доповняльного коду Bn в R0
       MOV A, P1 ; зчитування An
       JNB A.7, L2 ; аналіз знака
       CPL A ; інверсія акумулятора
       INC A ; додавання одиниці
       SETB A.7 ; встановлення знака
  L2: SUBB A, R0 ; формування Dn
       RLC A ; виділення знака
       MOV P3.0, C ; пересилання знака
       ;0 - розузгодженість додатна
       ;1 - розузгодженість від'ємна
       SJMP START ; перехід на початок програми
```

Приклад 78. Перетворення завадостійкого коду за методом «подвійної щітки» в двійковий код в слідкуючій системі синхронізації формних циліндрів рулонної друкарської машини.

Алгоритм перетворення коду:

$$a_i = a_{i-1}b_i + \overline{a_{i-1}}c_i,$$

де b_i, c_i - сиґнали з відстаючого і випереджуючого зчитуючих елементів. a_1 - сиґнал, що знімається лише з молодшого розряду зчитуючого елемента.

```
TDS:MOV C, P3.0 ; C ← a1
    ANL C, P1.O ; a1b1
    MOV P3.1, C ; зберігання часткового добутку
    MOV C, P2.0 ; C ← c1
    ANL C, /P3.0 ; ā1c1
    ORL C, P3.1 ; a1b1 +ā1c1
    MOV P3.1, C ; запис a2
    ANL C, P1.O ; a2b2
    MOV P3.1, C ; зберігання часткового добутку
    MOV C, P2.0 ; C ← c2
    ANL C, /P3.0 ; ā2c2
    ORL C, P3.1 ; a2b2 +ā2c2
    MOV P3.1, C ; запис a3
    ——————————
    ...                 ; результат формується у P3 (P3.1...P3.7).
```

Приклад 79. Перетворення завадостійкого коду Грея в двійковий код для цифрової системи реґулювання положенням подавача паперорізальної машини.

Перетворення здійснюється за алгоритмом:

$b_n = a_n$; $a_i = b_{i+1} \oplus b_i$,

$a_n,\ldots,a_1$ - двійковий код;

$b_n,\ldots,b_1$ - код Грея;

$\oplus$ - знак суми за модулем два.

```
TGB:MOV A, P1 ; зчитування коду Грея
    MOV R0, #8 ; ініціалізація лічильника
    L0:RLC A ; зсув акумулятора вліво
    JNB A.7, L1 ; виключне АБО
    CPL C ; інверсія переносу
L1:DJNZ R0, L0 ; цикл за розрядами
    MOV P2, A ; запис двійкового коду
```

Приклад 80. Реалізація задавача інтенсивності намотувального верстата. Завдання D_0 в двійково-десятковому коді подано на входи порту 2, імпульси зворотного зв'язку поступають в унітарному коді. Результат $D_3=D_0-D_i$ необхідно видати в двійковому коді на виходи порту 1.

Використовується підрахунок числа імпульсів за час 20 мс, максимальна кількість імпульсів - 255, підрахунок імпульсів і формування часового інтервалу здійснюється внутрішніми таймерами/лічильниками.

```
T: EQU NOT(20000)+1 ; завдання інтервалу часу 20мс
    MOV TMOD, #01100001B ; настроювання таймерів/лічильників: 1-й - 8-розрядний з автоматичним перезавантаженням, 0-й - 16-розрядний таймер
ST:CLR A ; скидання акумулятора
    MOV TH1, A ; скидання Т/Л1
    MOV TL1, A
    MOV TH0, #HIGH(T) ; завантаження в Т/Л0 константи Т
    MOV TL0, #LOW(T)
    MOV A, P2 ; перетворення 2-10 коду в двійковий
    ANL A, #00001111B ; виділення молодшої цифри
    MOV R0, A ; запам'ятовування
    MOV A, P2
    ANL A, #11110000B ; виділення старшої цифри
    SWAP A ; обмін тетрад в акумуляторі
    MOV B, #0AH ; занесення числа 10 в регістр В
    MUL AB
    ADD A, R0 ; формування двійкового числа з Р2
    ORL TCON, #50H ; запуск Т/Л1 і Т/Л0
WAIT:JBC TCON.5, EXIT ; перевірка переповнення Т/Л0
    SJMP WAIT ; цикл, якщо TF=0
EXIT:MOV B, TL1 ; число імпульсів за 20мс
    SUBB A, B ; обчислення коду задавача
    MOV P1, A
    SJMP ST ; перехід на початок програми
```

Приклад 81. Реалізація автоматичного електроінверсора фоторепродукційного апарата.

Добуток значень в R0 і R1 при настройці на фокус повинен бути рівний заданій величині f^2 (ZAD).

Прийнята точність переміщення визначається давачем положення і дорівнює 1/256 від ділянки переміщення.

```
    AEI :
    MOV R0, #0 ; занесення початкових значень
    MOV R1, #255
    SETB P1.2 ; керування включено (зсув вправо)
WAIT0: JNB P1.0, WAIT0 ; очікування 1 (очікування імпульсу)
WAITC: JB P1.0, WAITC ; очікування 0
    JNB P1.1, L1 ; знак руху
    DEC R1 ; знак руху від'ємний (вліво)
```

```
      INC R0
      SJMP L2
L1 : INC R1 ; знак руху додатний (вправо)
      DEC R0
L2 : MOV A,R0 ; підготовка множення
      MOV B, R1
      MUL AB
      SUBB A, #LOW(ZAD) ; порівняння молодшого байта із
                          завданням
      JNZ L3
      MOV A,B
      SUBB A, #HIGH(ZAD) ; порівняння старшого байта із
                           завданням
      JZ STOP
L3 : MOV P1.2,C ; видача керування
      JMP WAIT0
STOP : ... ; зупинка програми
```

При необхідності в програму легко можна внести порівняння добутку із заданою точністю.

Приклад 82. Схема і програма виявлення несправності електричного кола. До виходів порту 2 через потужні транзистори (ключі) під'єднано відповідні виконавчі механізми. Схема і програма індикують обрив у відповідному колі.

Нехай включені всі навантаження, крім одного, тобто на колекторах транзисторів VT1...VT6, крім одного, присутній низький потенціал. Через вимкнуте навантаження в базу VT7 протікає струм, що викликає на вході T0 низький потенціал. Наявність високого рівня на цьому вході служить ознакою несправності в даному колі.

```
      ORL P2, #0FH ; високий рівень на виходах керування
                     навантаженням
      CLR P2.0 ; вимкнути навантаження
      JB T0, ERROR ; рівень на T0 повинен бути низьким
      SETB P2.0 ; ввімкнути навантаження
      CLR P2.1
      JB T0, ERROR
      SETB P2.1
      ..........
      CLR P2.5
      JB T0, ERROR
```

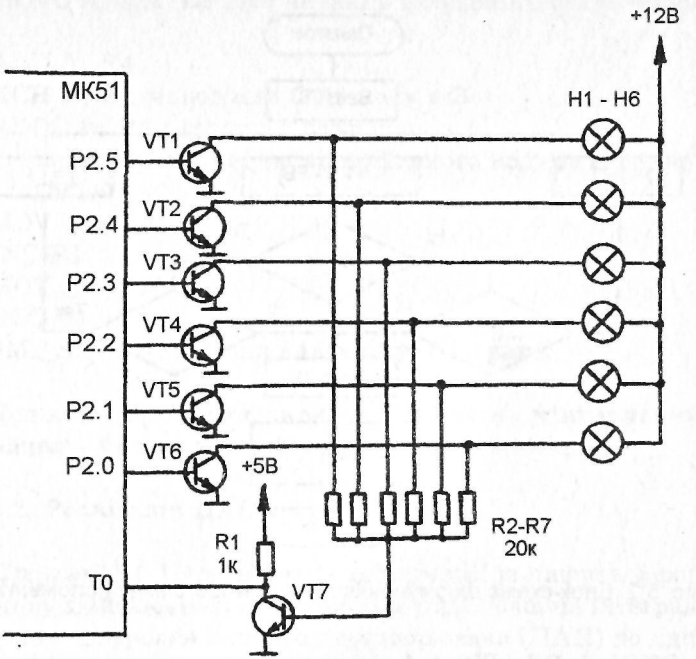


Рис. 5.1. Використання входу T0 для виявлення несправності

```
    SETB P2.5
    JB T0, EXIT ; перехід на вихід, якщо немає несправності
ERROR : ; підпрограма обробки несправності
EXIT ; вихід
```

Приклад 83. Формування двовідлікового коду положенням вала. Початкові умови: коди точного (ТВ) і грубого (ГВ) відліків - зчитуються із комірок ТВ і ГВ з точністю 8 розрядів; коди опорних точок a_j, $j=\overline{0,r-1}$, записані в комірки ПЗП з точністю 16 розрядів, початкова адреса масиву опорних точок знаходиться в DPTR; $r=2^g$ ($g=5...8$) - задане значення редукції; значення точності ГВ відповідає величині $2^8/r=256/r$.

Програма працює за таким алгоритмом (рис. 5.2) [33].

```
START: MOV R2, #FB ; запис коду ГВ
    MOV R3, #TB ; запис коду ТВ
ST : MOV A, R0 ; знаходження опорної точки в R0 ← j
    MOV C A, @A+DPTR ; читання старшого байта a_j
    MOV B, A ; запам'ятовування в B
```

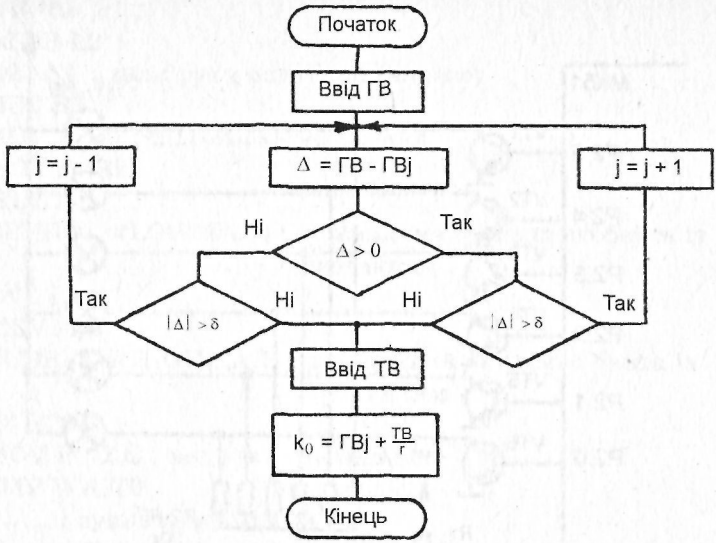


Рис. 5.2. Блок-схема формування двовідлікового коду положення вала

```
        MOV A, R2 ; ГВ → A
        SUBB A, B ; ГВ - (a0)
        JC L1 ; перехід, якщо Δ<0
        SUBB A, #δ  ; знаходження Δ-δ
        JC L2 ; перехід, якщо Δ-δ<0
        INC R0 ; переміщення вказівника до
        INC R0 ; наступного старшого байта опорної точки
        JMP ST
L1 :    ADD A, #δ ; знаходження різниці δ-Δ
        JNC L2 ; перехід, якщо δ>|Δ|
        DEC R0 ; переміщення вказівника до
        DEC R0 ; попередньої опорної точки (старшого байта)
        JMP ST
L2 :    MOV A, R3 ; ТВ → A
        MOV  B, #r  ; запис результату
        DIV AB ; ТВ/r
        MOV R5, A ; запам'ятовування результату
        MOV R4, B
        MOV A, R0 ; читання старшого байта опорної точки
        MOVC A, @A+DPTR
        INC R0
        MOV A, R0
```

```
MOVC A, @A+DPTR ; читання молодшого байта опорної
                  точки
ADD A, R4
XCH A, B ; молодший байт коду в B
ADDC A, R5 ; старший байт коду в A
; кінцеві пересилки, запис знайденого коду за адресою @R1
  і @R1+1
MOV @R1, A
INC R1
MOV @R1, B
DEC R1
JMP START ; перехід на початок програми
```

Довжина програми складає 53 байти, мінімальний час виконання - 47 мкс при тактовій частоті 12 МГц.

5.2. Реалізація ЦАП і АЦП

Приклад 84. Перетворення інформації із цифрової форми в аналогову здійснюється за допомогою підключення інтегральних мікросхем цифроаналогового перетворювача (ЦАП) до одного з портів МК51. Видача інформації при цьому здійснюється за допомогою однієї команди, наприклад: MOV P1, A.

Якщо необхідно ґенерувати якусь складну функцію, то краще записати її таблично і зчитувати з відповідним періодом в заданий порт.

Приклад 85. Аналогоцифровий перетворювач (АЦП) послідовного наближення (рис.5.3).

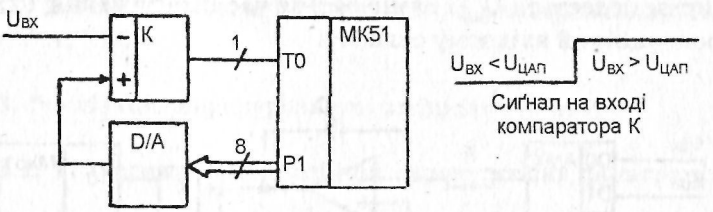


Рис. 5.3. Реалізація АЦП послідовного наближення

МК видає через порт P1 байт даних, який перетворюється в аналогову форму і порівнюється з вхідним сиґналом компаратором, потім аналізується результат порівняння. В залежності від значення сиґналу на вході T0 МК51 або залишає старший байт

в одиничному стані, або скидає в 0. Потім аналогічно перевіряються всі решта бітів.

R4 - регістр цифрового еквіваленту;

R3 - регістр біжучого значення 1 для вказівки зважуваного біта.

```
CONVRT: MOV R5,#08H ; завантаження лічильника циклів
        MOV R3, #01H
        MOV R4, #00H
LOOP:   MOV A, R3 ; зсув R3 вправо на 1 розряд
        RR A
        MOV R3, A
        ORL A, R4 ; логічне АБО зважуваного розряду і цифрового
                    еквіваленту
        MOV P1, A ; видача в P1
        JBC T0, ENDUGH ; якщо T0=1, то аналоговий еквівалент
```

виданого байта більший за $U_{вх}$ і зберігається в R4, якщо навпаки, то встановлений біт запам'ятовується в R4

```
        MOV R4, A
ENDUGH: DJNZ R5, LOOP ; декремент R5 і якщо не 0, то
                        перехід до аналізу наступного біта
```

Приклад 86. Реалізація АЦП за методом подвійного інтеґрування.

Метод подвійного інтеґрування є наступним. Спочатку інтеґрується вхідний опорний сиґнал $E_{оп}$. На виході інтеґратора встановлюється від'ємний потенціал. Потім інтеґрується вхідний аналоговий сиґнал за строго визначений час T1. Відлік T1 відраховується від моменту переходу сиґналу інтеґратора через 0. Потім подається $U_{оп}$ і вимірюється час інтеґрування T2, який є пропорційний вхідному сиґналу.

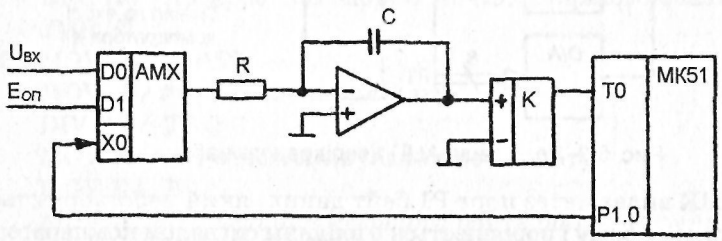


Рис. 5.4. Реалізація АЦП подвійного інтеґрування

Максимальний час перетворення для точності 16 розрядів 2x65.536 мс. Якщо необхідно точність перетворення 8 розрядів, то Т/Л0 переключається в режим 8-бітного таймера, а час перетворення складає 2x256 мкс.

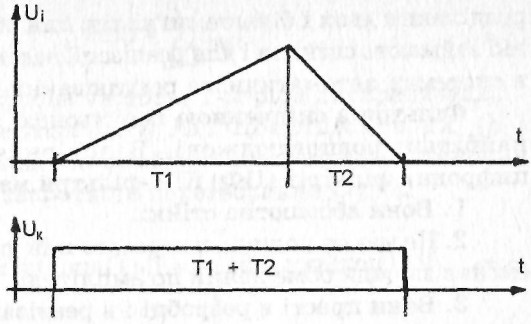


Рис. 5.5. Часові діаграми сиґналів в АЦП подвійного інтеґрування

```
    MOV TMOD,#01H ; настроювання Т/Л0 на режим 16 бітів
    MOV TH0, #HIGH (NOT(T1)+1); завантаження Т/Л0
    MOV TL0, #LOW (NOT(T1)+1) :
    SETB P1.1 ; настроювання P1.1 на ввід
    SETB P1.0 ; подача Еоп на інтеґратор
WAIT: JB P1.1, WAIT ; очікування появи на виході інтеґратора
                      від'ємного рівня сиґналу
    CLR P1.0 ; подача Uвх на інтеґратор
WAITT0: JNB P1.1, WAITT0 ; очікування моменту Т0
    SETB TCON.4 ; запуск Т/Л0
WAITT1: JNB TCON5, WAITT1 ; очікування моменту Т1
    SETB P1.0 : подача Еоп на інтеґратор і початок зворотного
                інтеґрування
WAITT2: JB P1.1, WAITT2 ; очікування моменту Т2
    CLR TCON.4 ; зупинка Т/Л0
    CLR TCON.5 ; скидання прапорця TF0
    MOV B,TH0 ; формування результату в реґістровій парі А, В
    MOV A, TL0
```

5.3. Реалізація нерекурсивних цифрових фільтрів

5.3.1. Теоретичні відомості про нерекурсивну фільтрацію

Фільтрацією називається процес зміни частотного спектра сиґналу в бажаному напрямі. Цей процес може привести до підсилення або послаблення частотних складових в деякому діапазоні частот, або виділення чи подавлення деякої конкретної частоти. Цифрова фільтрація знайшла широке застосування, наприклад, для подавлення шуму, маскуючого сиґнал, для

розділення двох і більше сиґналів, для обмеження смуги частот, які займають сиґнали і для реалізації заданих передаючих функцій в системах автоматичного реґулювання [1,19,23].

Фільтри з скінченною імпульсною характеристикою (КІХ) найбільш розповсюджені. В порівнянні з іншими типами цифрових фільтрів (ЦФ) КІХ-фільтри мають ряд переваг:

1. Вони абсолютно стійкі.

2. Не мають полюсів в передаточній функції, тому їх вихідний сиґнал завжди обмежений по амплітуді.

3. Вони прості в розробці і в реалізації.

4. Мають низьку чутливість до точності коефіцієнтів, що дозволяє обмежитись низькою розрядністю оброблюваних відліків.

5. Легко реалізують задану передаточну функцію.

Дискретизація і квантування вхідного аналогового сиґналу СК здійснюється за допомогою АЦП. При аналого-цифровому перетворенні виникають похибки квантування. Для їх зменшення необхідно вибирати розрядність квантування із умови забезпечення необхідного рівня шумів квантування. Якщо динамічний діапазон сигналу $D=U_{cmax} - U_{cmin}$, то при розрядності АЦП r розрядів інтервал квантування складає:

$$\Delta = D/2^r$$

Звідси, враховуючи, що при великих r закон розподілу помилок квантування рівномірний в діапазоні $\{-\Delta/2, \Delta/2\}$, знаходимо потужність шуму квантування

$$\sigma^2_{кв} = \Delta^2/12 = \frac{1}{3}\cdot(D/2^{r+1})^2$$

і співвідношення потужностей сигнал/шум на вході фільтра

$$s/n = P_c/\sigma^2_{кн},$$

де P_c – потужність вхідного сигналу.

Звідси по заданому s/n вибирається значення r.

Вихідний сиґнал КІХ-фільтра записується як лінійна згортка вхідного сиґналу {x} і вагової послідовності (імпульсної характеристики - ІХ) {h}:

$$y_n = \sum_{m=0}^{M-1} h_m x_{n-m}, \qquad (5.1)$$

$\{h_m\}$, $m = \overline{0,M-1}$ - ІХ фільтра; $\{x_n\}$, n=1,2,... - вхідні відліки.

Передаточна функція такого фільтра, враховуючи, що параметри фільтра (коефіцієнти ІХ) постійні, має вигляд:

$$H(z) = \sum_{m=0}^{M-1} h_m z^{-m}. \qquad (5.2)$$

де $Z = e^{j\omega T}$; $\omega = 2\pi f$ - цифрова частота, T-період дискретизації.

Знаходження коефіцієнтів ІХ. Коефіцієнти ІХ $\{h_m\}$ знаходяться різними методами. Найбільш простим є знаходження їх із рішення рівняння зворотного перетворення Фур'є:

$$h_m = \int_{-0,5}^{0,5} H(j2\pi fT)\exp(j2\pi mfT)df = 4\int_{0}^{0,5} H(f)\cos(2\pi mfT)df. \qquad (5.3)$$

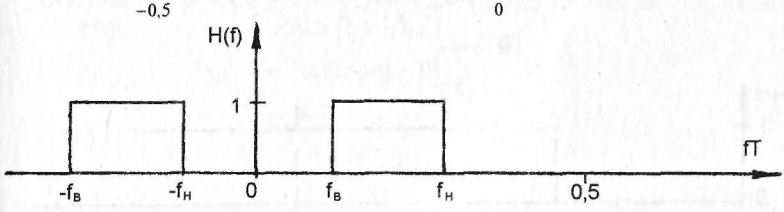


Рис. 5.6. АЧХ цифрового фільтра

де $T^{-1}=1$ – частота дискретизації; $f \in \{-0,5;\ 0,5\}$ – цифрова частота; $H(\cdot)$ – задана АЧХ (рис.5.6). Для фізичної реалізації згідно формули (5.1) послідовність $\{h_m\}$ зсувається на $(M-1)/2$ і стає симетричною або несиметричною відносно середини.

Відомо 4 види фільтрів з точно лінійною ФЧХ, які мають різний характер передаточної характеристики:

1) M – парне; $h_m = h_{M-m-1}$ – симетричні коефіцієнти;
2) M – непарне; $h_m = h_{M-m-1}$ – симетричні коефіцієнти;
3) M – парне; $h_m = -h_{M-m-1}$ – антисиметричні коефіцієнти;
4) M – непарне; $h_m = -h_{M-m-1}$ – антисиметричні коефіцієнти.

Симетрія коефіцієнтів ІХ визначається заданою АЧХ: симетричною або несиметричною відносно нульової частоти. На рис. 5.6...5.9 показані АЧХ 2 типу. Наведені нижче формули для коефіцієнтів ІХ відповідають симетричним коефіцієнтам з непарним M. АЧХ фільтра для непарного M може бути задана наступним чином:

1) для фільтра нижніх частот (ФНЧ) $f_н = 0$, $f_в = f_з$ (рис.5.7):

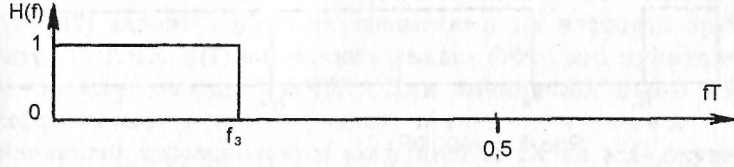


Рис. 5.7. АЧХ ЦФНЧ

$$H(f) = \begin{cases} 1, |f| < |f_B|, \\ \text{інакше } 0, \end{cases}$$

тоді

$$h_m = 2\sin(2\pi m f_3) / m\pi; \; h_0 = 4f_3; \qquad (5.4)$$

2) для фільтра верхніх частот (ФВЧ) $f_н = f_з$, $f_в = 0{,}5$ (рис. 5.8):

$$H(f) = \begin{cases} 1, |f_3| < |f| < 0{,}5, \\ 0 - \text{інакше}, \end{cases}$$

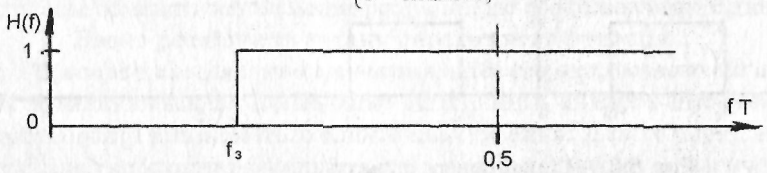


Рис. 5.8. АЧХ ЦФВЧ

тоді

$$h_m = 2\sin(2\pi m f_3) / m\pi; \; h_0 = 2 - 4f_3; \qquad (5.5)$$

3) для смугового фільтра (ФС)(рис. 5.6):

$$H(f) = \begin{cases} 1, |f_н| < |f| < |f_в|, \\ 0 - \text{інакше}, \end{cases}$$

тоді

$$h_m = [\sin(2\pi m f_в) - \sin(2\pi m f_н)] / (m\pi / 2); \qquad (5.6)$$

$$h_0 = 4(f_в - f_н);$$

4) для режекторного фільтра (ФР) (рис.5.9):

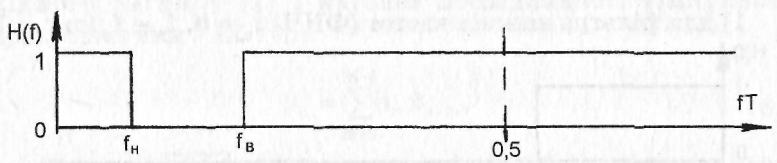


Рис. 5.9. АЧХ ФР

тоді

$$H(f) = \begin{cases} 0, |f_н| < |f| < |f_в|, \\ 1, \text{інакше}, \end{cases}$$

$$h_m = [\sin(2\pi m f_в) + \sin(2\pi m f_н)] / (m\pi / 2); \qquad (5.7)$$

$$h_0 = 2 + 4(f_в - f_н).$$

Відповідно АЧХ з несиметричними коефіцієнтами має вигляд (рис. 5.10):

$$H(f) = \begin{cases} 1, 0 < f < f_з \\ -1, 0 > f > -f_з, \\ 0 - \text{інакше}, \end{cases}$$

Аналогічно можуть бути знайдені коефіцієнти ІХ для парного М (а також симетричних і несиметричних видів ІХ).

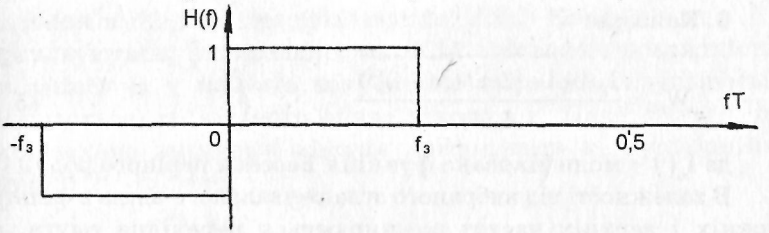


Рис. 5.10. АЧХ ФНЧ для асиметричних коефіцієнтів ІХ

Фазочастотна характеристика КІХ-фільтра лінійна і має вигляд для непарних і парних М:

$$\Theta(f) = -(M-1)\pi f;$$

$$\Theta(f) = -M\pi f.$$

Накладання згладжувальних вікон

В зв'язку з тим, що у формулах (5.4)...(5.7) початкові АЧХ H(f) задані з різкими границями на частотах зрізу, в отриманій АЧХ H(f) виникають явища Гібса, які приводять до спотворення (викидів) АЧХ. Для зменшення цього явища використовуються згладжувальні вікна, які в часовій області еквівалентні перемноженню коефіцієнтів ІХ на згладжувальні коефіцієнти $\{W_m\}$:

$$h'_m = h_m \cdot W_m.$$

Найбільш часто вживаються наступні види вікон (розширені вікна $W_{(M-1)/2} \neq 0$ – для непарного M):

1. Ханна:

$$W_m = 0{,}5 + 0{,}5\cos(2\pi m / M); \qquad (5.8)$$

2. Хемінга:

$$W_m = 0{,}54 + 0{,}46\cos(2\pi m / M); \qquad (5.9)$$

3. Бартлета (трикутне):

$$W_m = 1 - 2m / M; \qquad (5.10)$$

4. Блекмана:

$$W_m = 0{,}42 + 0{,}5\cos(2\pi m / M) + 0{,}08\cos(4\pi m / M); \qquad (5.11)$$

5. Ланцоша:

$$W_0 = 1;\ W_m = \sin(2\pi m / M) / (2\pi m / M); \qquad (5.12)$$

6. Кайзера:

$$W_m = \frac{I_0(\beta\sqrt{1 - [2m / (M-1)]^2})}{I_0(\beta)}, \qquad (5.13)$$

де $I_0(\cdot)$ – модифікована функція Бесселя першого роду.

В залежності від вибраного згладжувального вікна в фільтрах нижніх і верхніх частот розширюється перехідна смуга, а в режекторних та смугових фільтрах – головний пелюсток, але одночасно збільшується затухання в смугах непропускання і зменшуються пульсації в смугах пропускання (див. табл. 5.1).

Таблиця 5.1

Параметри згладжувальних вікон

Вікно	Ширина головного пелюстка ФС	Можливий рівень бокових пелюстків, дБ	Рівень пульсацій в смузі пропускання, дБ
Ханна	$8\pi f_з/M$	–45	0,26
Хемінга	$8\pi f_з/M$	–42,7	0,09
Блекмана	$12\pi f_з/M$	–75	1,11
Кайзера	$(4...12)f_з/M$	–30...–100 росте із збільшенням β	0,1...1,0

Вікно Бартлета використовується у випадку обмежених обчислювальних можливостей, наприклад, при побудові адаптивних фільтрів, коли необхідно швидко розрахувати нову згладжену вагову послідовність (h_m).

Особливістю вікна Кайзера є те, що пульсації в смугах пропускання можна змінювати неперервно від дуже малої величини, яка відповідає вікну Блекмана, до великих значень, у випадку вікна Бартлета (трикутного), змінюючи лише параметр β. Крім того, як і для інших вікон, ширина основного пелюстка або перехідна смуга може регулюватися вибором числа M. Для цього вікна використовується апроксимація за рахунок швидкозбіжного ряду

$$I_0(\beta) = 1 + \sum_{k=1}^{\infty} [1/k! \cdot (\beta/2)^2] \qquad (5.14)$$

Програма на FORTRANі для обчислення $I_0(x)$ по (5.14) наведена в [23].

Квантування коефіцієнтів ІХ. Коефіцієнти ІХ розраховуються, як правило, на ЕОМ з високою розрядністю. Для запису їх у пам'ять мікропроцесора вони округляються (квантуються) до заданого числа розрядів r. Приймається, що максимальне значення модуля коефіцієнта h_0 пропорційне величині $2^{r-1}-1$, тоді

$$h^*_m = [(\hat{h}_m/h_0)\cdot(2^{r-1}-1)], \qquad (5.15)$$

де [·] - ціла частина. При збільшенні r результуюча АЧХ H(f) більш точно відповідає заданій, але зростає використовуваний для коефіцієнтів ІХ обсяг пам'яті і довше виконується програма фільтрації за формулою (5.1).

Перевірка розрахунку ЦФ. Після квантування коефіцієнтів ІХ результуюча АЧХ змінюється. Вона розраховується за допомогою ДПФ квантованої і згладженої послідовності $\{h^*_m\}$ [1,31].

Для непарного і парного M маємо:

$$\hat{H}(f) = \sum_{m=0}^{(M-1)/2} a_m / (2^{r-1} - 1) \cos(2\pi m f \Gamma),$$

$$\hat{H}(f) = \sum_{m=0}^{M/2} b_m / (2^{r-1} - 1)\cos((m - \frac{1}{2})2\pi mfT),$$

де $a_0 = h^*_0$; $b_m = a_m = 2h^*_m$; $m = \overline{1, (M-1)/2-1}$ – для непарного M; $m = \overline{1, M/2-1}$ – для парного M.

Таким чином, задача розрахунку ЦФ зводиться до знаходження коефіцієнтів ІХ із заданої АЧХ, накладання згладжувального вікна, квантування згладжених коефіцієнтів ІХ, знаходження результуючої АЧХ з квантованими коефіцієнтами.

5.3.2. Програмна реалізація нерекурсивних цифрових фільтрів

ЦФ може бути реалізований як апаратно, так і програмно. В першому випадку використовуються апаратні перемножувачі, суматори і реґістри затримки (рис.5.11).

Очевидно, що апаратні затрати такої реалізації зростають із збільшенням розрядності оброблюваних сиґналів $\{x_n\}$ та $\{h_m\}$, а також із збільшенням порядку фільтра M. Тому для зменшення апаратних затрат при заданій і порівняно невисокій швидкодії ЦФ використовується програмна реалізація на мікропроцесорах чи мікро-ЕОМ.

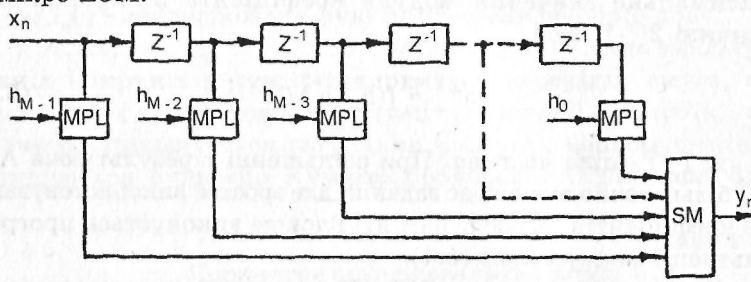


Рис. 5.11. Апаратна реалізація нерекурсивного ЦФ

Основою програмної реалізації функціонування ЦФ є формула (5.1). Вона передбачає запис у постійну пам'ять (ПЗП) послідовності вагових коефіцієнтів ІХ $\{h_m\}$, $m=\overline{0, M-1}$ і запис в оперативну пам'ять (ОЗП) значень відліків вхідного сиґналу $\{x_n\}$, $n>0$. Для обчислень згідно (5.1) необхідно запам'ятати в ОЗП M значень $\{x_n\}$, тому загальна використана пам'ять ОЗП та ПЗП складає 2M комірок пам'яті. Алгоритм роботи фільтра показаний на рис.5.12.

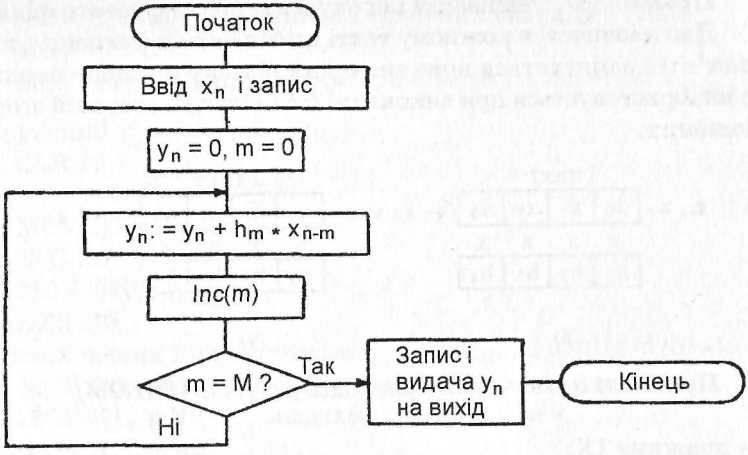


Рис. 5.12. Алгоритм роботи ЦФ

Значення x_n записується в комірку ОЗП з відносним номером q=n mod M. На початку роботи фільтра комірки ОЗП з відносними номерами $q = \overline{0, M-1}$ повинні бути обнулені. Це повинно бути зроблено на етапі ініціалізації системи.

Величина y_n є сумою накопичених добутків, тому її розрядність може вийти за розрядну сітку ОМЕОМ. Для попередження цього значенню y_n відводять кілька послідовних комірок ОЗП, кількість яких визначається із максимально можливої величини сиґналу. Остання знаходиться за допомогою L_p – норми, яка записується у вигляді [23]:

$$\|x\|_p = (1/f \cdot \int_0^{2\pi f} |x(f)|^p df)^{1/p}. \qquad (5.16)$$

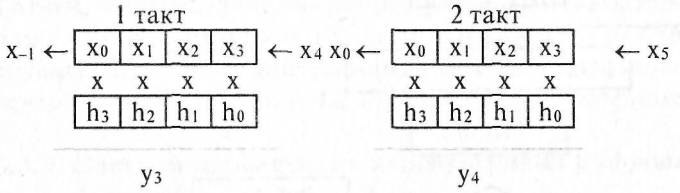
Для $p \to \infty$ $\|x\|\infty = \max|x|$, тому із (5.16) отримуємо:

$$y_{max} = x_{n\,max} \cdot \sum_{m=0}^{M-1} |h_m|. \qquad (5.17)$$

Необхідна кількість комірок ОЗП для зберігання проміжних даних і результатів визначається із (5.17) і розрядності ОМЕОМ (серія КР1816 є восьмирозрядна, тому для одного байта: $y_{max} < 2^8/2 = 128$, для 2-х байтів - $y_{max} < 2^{16}/2 = 32768$ і т.д.).

Приклад 87. Реалізація нерекурсивного цифрового фільтра.

Для наочності в кожному такті здійснюється зміщення даних в пам'яті і записується нове значення відліку на місце першого, що використовується при виконанні (обчисленні) операцій згортки і подібних.



Прийняті позначення і розподіл ресурсів ОМЕОМ:

M - довжина IX;
початкова адреса пам'яті IX в ПЗП - M0;
коефіцієнти $\{h_m\}$ однобайтові;
результат накопичення y_n записується за адресою в R5;
передача y_n на вихід здійснюється за адресою в R4;
зчитування однобайтового відліку x_n проводиться із комірки за адресою в R7;
значення $x_n \ldots x_{n-M+1}$ зберігаються в РПД, починаючи з адреси X0.

```
; підготовка фільтра

; очистка РПД в межах від X0 до X0+M-1
      MOV R1, #X0
      MOV R0, #M
      CLR A
  L0: MOV @R1, A ; скинути один байт
      INC R1 ; змістити вказівник
      DJNZ R0, L0 ; цикл, якщо не всі комірки обнулені

; фільтрація, зчитування і зміщення даних
ST: MOV R6, #M
L1: DEC R1
      MOV A, @R1 ; зчитування xn
      INC R1
      MOV @R1, A ; запис на місце xn+1
      DJNZ R6, L1 ; цикл переміщення
      MOV R0, R7
```

```
    MOV A, @R0 ; зчитування значення сиґналу з АЦП
    MOV R0, R6
    MOV @R0, A ; пересилка x_n в комірку з адресою x_X0+M-1
; обнулення y_n
    CLR A
    MOV R0, R5
    MOV @R0, A
    INC R0
    MOV @R0, A
    DEC R0
; перемноження і накопичення y_n в комірках @R0 і @R0+1
    MOV R6, #M      ; ініціалізація лічильника на M
    MOV R1, #X0     ; задавання адреси для x_n
    MOV R2, #M0
  L2: MOV 22H, @R1
    MOV 23H, @R0
    ACALL MULZ      ; {x_n h_m} - результат перемноження в
    MOV R0, R5                  доповняльному коді
    ADD A, @R0      ; накопичення молодшого байта
    MOV A, @R0      ; запис молодшого байта
    INC R0          ; зміщення вказівника
    XCH A, B        ; в A - старший байт, в B - молодший
    ADDC A, @R0     ; накопичення старшого байта
    MOV A, @R0      ; запис старшого байта
    DEC R0          ; відновлення вказівника
    INC R1          ; зміщення вказівника
    INC R2
    DJNZ R6, L2     ; цикл по M відліках
; передача y_n на вихід
    MOV R1, R4
    MOV @R1, A      ; передача старшого байта результату
    INC R1          ; зміщення вказівника
    MOV A, @R0
    MOV @R1, A      ; передача молодшого байта результату
    JMP ST          ; повторення процедури фільтрації
```

Підпрограма перемноження цілих чисел зі знаком.

```
MULZ : MOV A,22H
    RLC A           ; виділення знака першого числа
    MOV 20.7H,C     ; запам'ятовування знака
    RR A            ; виділення модуля першого числа
    MOV B,A
```

```
     MOV A,23H
     RLC A ; виділення знака другого числа
     JNB 20.7H,C ; сума за модулем два знаків першого і другого
                   чисел
     CPL C
L1 : MOV 20.7H,C ; знак добутку
     MUL AB ; перемноження модулів чисел
     JNB 20.7H,L2 ; перехід, якщо результат перемноження
                    додатний
     CPL A ; інверсія акумулятора
     ADD A,#1 ; додавання одиниці
     XCH A,B ; обмін акумулятора і доповнення
     CPL A ; інверсія доповнення
     ADDC A,#0 ; додавання нуля з врахуванням переносу
     SETB A.7 ; встановлення знака
     XCH A,B ; в акумуляторі і доповненні доповняльний код
               результату множення
L2 : RET ; повернення в основну програму
```

5.4. Реалізація рекурсивних цифрових фільтрів

Приклад 88. Реалізація рекурсивних цифрових фільтрів.

Рекурсивні фільтри в порівнянні з нерекурсивними мають ряд недоліків [23].

1. Мають полюси в функції передачі, тому можуть бути нестійкими.

2. Чутливі до точності подання коефіцієнтів, що вимагає при заданій точності фільтрації великої (значно більшої, ніж у КІХ-фільтрах) розрядності подання даних і коефіцієнтів.

Однак у порівнянні з нерекурсивними фільтрами ці фільтри суттєво простіші в реалізації і реалізують задану передаточну функцію при значно меншій кількості коефіцієнтів. Затримка, яку вносить такий фільтр, значно менша, ніж у КІХ-фільтрів.

Використання ОМЕОМ дозволяє суттєво скоротити затрати апаратури при такій реалізації. Швидкодія фільтра буде визначатись часом виконання відповідної програми, що реалізує алгоритм фільтрації. Такий фільтр також може бути реалізований на основі дискретних елементів, як показано на рис. 5.13. Як правило, фільтри високих порядків складають на основі послідовного з'єднання ланок першого і другого порядків.

Нехай передаточна функція рекурсивного ЦФ 2-го порядку,

який ще також називають біквадратною ланкою, задана таким різницевим рівнянням:

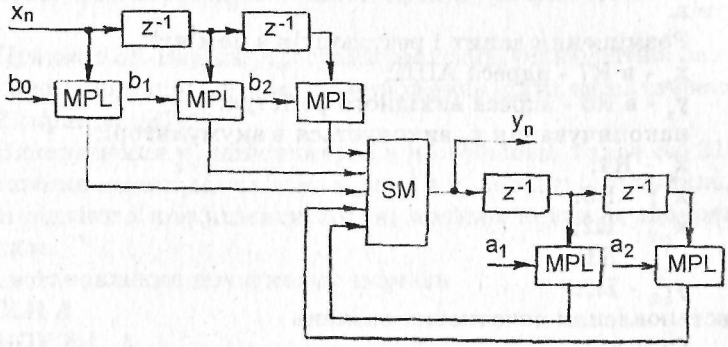


Рис. 5.13. Апаратна реалізація рекурсивного ЦФ 2-го порядку (біквадратної ланки)

$$y_n = b_0x_n + b_1x_{n-1} + b_2x_{n-2} + a_1y_{n-1} + a_2y_{n-2}\ ; \qquad (5.18)$$

Для конкретної програмної реалізації задамо коефіцієнти різницевого рівняння в двійковій формі:

b_0=1;
b_1=-0,111111;
b_2=1;
a_1=1,0011;
a_2=-0,111011.

Подамо коефіцієнти в прямому коді, а відліки вхідного {x} і вихідного {y} сиґналів у форматі з фіксованою комою в доповняльному коді. Розрядність оброблюваних сиґналів вибираємо 8.

Для скорочення загального числа виконуваних операцій необхідно виконати перетворення виразу (5.18) з метою мінімізації числа одиничних значень в поданні коефіцієнтів.

Коефіцієнти з однаковою вагою розміщуємо один під одним.

y_n = +1.000000 x_n
- 1.000000 x_{n-1}+
+0.000001 x_{n-1}+
+1.000000 x_{n-2}+
+1.001100 y_{n-1}-
- 1.000000 y_{n-2}+
+0.000101 y_{n-2}.

Весь алгоритм обробки в цьому випадку виконується без використання операції множення, а лише на основі сумувань і зсувів.

Розміщення даних і результатів в пам'яті:

x_n - в R7 - адреса АЦП;

y_n - в R6 - адреса вихідного регістра;

накопичування y_n виконується в акумуляторі;

x_n - R4;

x_{n-1} - R3;

x_{n-2} - R2;

y_{n-1} - R1;

y_{n-2} - R0.

```
; встановлення початкових значень
      CLR A
      MOV R3, A
      MOV R2, A
      MOV R1, A
      MOV R0, A
; фільтрація
ST: CLR A ; y_n ← 0
      ADD A, R3 ; y_n = y_n + x_n-1
      ADD A, R0 ; y_n = y_n + y_n-2
      RRC A ; зсув A на 2 розряди вправо
      RRC A ;
      ADD A, R0 ; y_n = y_n + y_n-2
      ADD A, R1 ; y_n = y_n + y_n-1
      RRC A ; зсув A на 1 розряд вправо
      ADD A, R1 ; y_n = y_n + y_n-1
      RRC A ; зсув A на 3 розряди вправо
      RRC A ;
      RRC A ;
      ADD A, R2 ; y_n = y_n + x_n-2
      ADD A, R1 ; y_n = y_n + y_n-1
      ADD A, R4 ; y_n = y_n + x_n
      SUBB A, R3 ; y_n = y_n - x_n-1
      SUBB A, R0 ; y_n = y_n - y_n-2
      MOV R0, R6
      MOV @R0, A ; вивід y_n
      MOV R0, R7
      MOV R5, @R0
      MOV R0, R1 ; y_n-2 = y_n-1
      MOV R1, A ; y_n-1 = y_n
      MOV R2, R3 ; x_n-2 = x_n-1
```

```
        MOV R3, R4 ; x_{n-1} = x_n
        MOV R4, R5 ; ввід x_n
        SJMP ST ; перехід на початок процедури фільтрації
```

Приклад 89. Варіант програми реалізації біквадратної ланки з використанням підпрограми перемноження цілих чисел із знаком MULZ (приклад 87).

Накопичення y_n здійснюється в R5 і регістрі з адресою 21H. Розміщення інших даних таке ж, як і в попередньому прикладі. Вхідні відліки і коефіцієнти подані прямим кодом із знаковим розрядом.

```
        ; встановлення початкових значень
        CLR A
        MOV R4, A
        MOV R3, A
        MOV R2, A
        MOV R1, A
        MOV R0, A
; фільтрація
ST: MOV 23H, R3 ; x_{n-1} → 23H
        MOV 22H, #b_1 ; b_1 → 22H
        ACALL MULZ ; x_{n-1}·b_1
        ADD A, R4 ; x_n + x_{n-1}·b_1
        XCH A, B
        ADDC A, #0 ; корекція старшого байта
        XCH A, B
        ADD A, R2 ; x_n + x_{n-1}·b_1 + x_{n-2}
        XCH A, B
        ADDC A, #0 ; корекція старшого байта
        MOV 21H, B
        MOV R5, A ; запис в R5
        MOV 22H, #a_1 ; a_1 → 22H
        MOV 23H, R1 ; y_{n-1} → 23H
        ACALL MULZ ; a_1·y_{n-1}
        ADD A, R5 ; проміжна сума
        MOV R5, A ; запам'ятовування проміжної суми
        XCH A, B
        ADDC A, 21H
        MOV 21H, A
        MOV 23H, R0 ; y_{n-2} → 23H
        MOV 22H, #a_2 ; a_2 → 22H
```

```
ACALL MULZ ; a2·yn-2
ADD A, R5 ; отримання yn
XCH A, B
ADDC A, 21H
MOV R0, R6
MOV @R0, A ; вивід yn ;пересилання проміжних значень
MOV R0, R7
MOV R5, @R0
MOV R0, R1 ; yn-2 ← yn-1
MOV R1, A ; yn-1 ← yn
MOV R2, R3 ; xn-2 ← xn-1
MOV R3, R4 ; xn-1 ← xn
MOV R4, R5 ; ввід xn
SJMP ST ; перехід на початок процедури фільтрації
```

Хоч у другому варіанті програма коротша, але виконується вона довше. Причому, при $b_0 \neq 1$ і $b_2 \neq 1$ необхідно виконати ще дві операції перемноження.

Для мінімізації часу виконання програми в першому варіанті необхідно мінімізувати число одиниць в поданні коефіцієнтів. Відповідна програма на мові TURBO PASCAL 7.0 для виконання такої мінімізації наведена в [29]. Перший варіант програми рекурсивної фільтрації (приклад 88) зручний також при розрядності подання даних і коефіцієнтів, більшій 8.

5.5. Проектування і програмна реалізація кореляторів

Основним призначенням кореляторів є обчислення міри взаємозв'язку між сигналами [15].

Кореляційна функція дозволяє виявити скриту періодичність, часовий зсув та інші взаємозв'язки між вхідними сигналами.

Взаємна кореляційна функція (коваріація) двох центрованих вхідних сигналів $\{x_n\}$, $\{y_n\}$, $n > 0$ без врахування коригуючих множників записується у вигляді:

$$K_{xy}(m) = \sum_{k=1}^{N} x_k y_{k+m} \qquad (5.19)$$

де N - довжина інтервалу реалізації, $m = \overline{0, M}$; M - довжина (кількість зсувів) кореляційної функції.

Кореляційна функція також може обчислюватися для одного з сигналів $\{x_n\}$, $n>0$. В цьому випадку вона носить назву автокореляційної функції (автоковаріації) і записується з наведеними обмеженнями так:

$$K_{xx}(m) = \sum_{k=1}^{N} x_k x_{k+m} \qquad (5.20)$$

Автокореляційна функція дозволяє визначити потужність вхідного сиґналу, його зашумленість, наявність періодичних компонентів, співвідношення їх потужностей і т.п. Для обчислення $K_{xy}(m)$ по (5.19, 5.20) необхідно використати 2M комірок пам'яті в ОЗП з відповідною розрядністю для запам'ятовування сиґналів $\{x_n\}$, $\{y_n\}$ і відповідну кількість комірок ОЗП для зберігання проміжних значень $K_{xy}(m)$. Розрядність останніх визначається за допомогою L_p - норми (5.16) з умови:

$$K_{xy\,max} = N \cdot |x_{max}| \cdot |y_{max}| . \qquad (5.21)$$

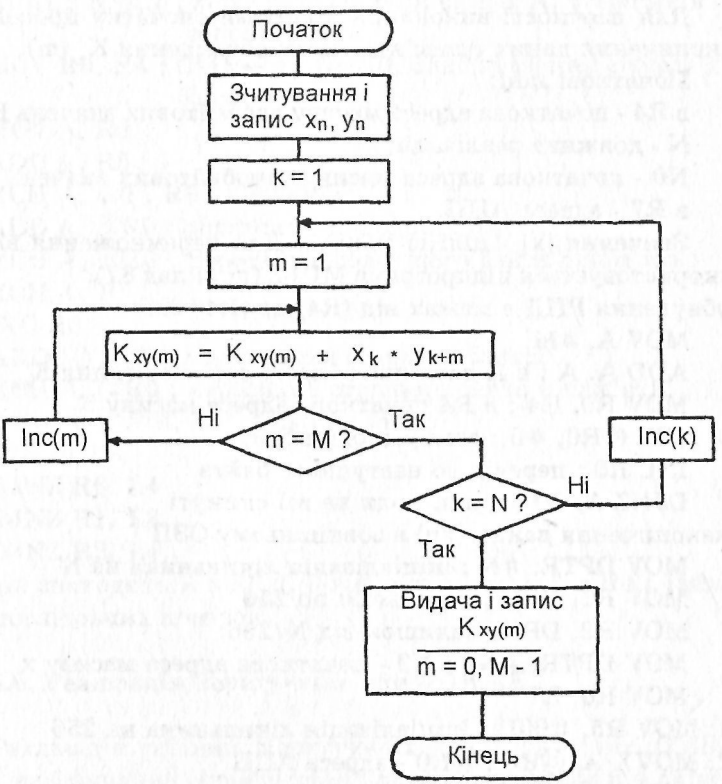


Рис. 5.14. Алгоритм роботи кроскорелятора

і розрядності ОМЕОМ. Наприклад, для N=256, $x_{max}=y_{max}=127$ маємо $K_{xymax}=2^{22}$, тобто необхідно використати 3 байти для зберігання кожного значення $K_{xy}(\cdot)$.

Алгоритм роботи корелятора показаний на рис.5.14. На початку роботи комірки ОЗП, в яких будуть зберігатися значення $\{x_n\}$, $\{y_n\}$, $\{K_{xy}\}$ - необхідно обнулити.

Значення x_n та y_n записуються в комірки ОЗП з відносними номерами $q_x = n \bmod N$; та $p_y = n \bmod N$. Фізичні адреси відповідно дорівнюють $q = q_0 + q_x$ та $p = p_0 + p_y$, де q_0 та p_0 - початкові адреси для зберігання відповідних значень, $|2p_0 - q_0| \geq N$.

У випадку обчислення автокореляційної функції (5.20) необхідний обсяг пам'яті ОЗП зменшується до N комірок, в яких зберігаються лише значення $\{x_n\}$. Алгоритм роботи автокорелятора відповідає рис.5.14 із зміною значень y_n на x_n.

Приклад 90. Програма обчислення автокореляційної функції.

Для наочності виконання програми спочатку проводиться накопичення даних в пам'яті, потім обчислення $K_{xx}(m)$.

Початкові дані:

в R4 - початкова адреса масиву двобайтових значень $K_{xx}(m)$;

N - довжина реалізації;

N0 - початкова адреса масиву однобайтових значень x;

в R7 - адреса АЦП.

Значення {x} - цілі із знаком. Для перемноження відліків використовується підпрограма MULZ (приклад 87).

```
; обнулення РПД в межах від (R4) до (R4+2M)
      MOV A, #M
      ADD A, A ; в А довжина області для зберігання Кxx
      MOV R0, R4 ; в R4 початкова адреса масиву Кxx
L0: MOV @R0, #0 ; скинути один байт
      INC R0 ; перехід до наступного байта
      DJNZ A, L0 ; цикл, коли не всі скинуті
; накопичення даних {xn} в зовнішньому ОЗП
      MOV DPTR, #N ; ініціалізація лічильника на N
      MOV R1, DPH ; лічильник по 256
      MOV R2, DPL ; залишок від N/256
      MOV DPTR, #N0 ; N0 - початкова адреса масиву x
      MOV R0, R7
L1: MOV R5, #00H ; ініціалізація лічильника на 256
L2: MOVX A, @R0 ; @R0 - адреса АЦП
      MOVX @DPTR,A ; запис x
      INC DPTR ; дані x в комірках від N0 до N0+N-1
```

```
      DJNZ R5, L2 ; цикл по 256 відліках
      DJNZ R1, L1 ; цикл по N/256
      DJNZ R2, L2 ; цикл по залишку від N/256
; обчислення K_xx(m)
      MOV R2, DPL ; в DPTR - кінцеве значення адреси
      MOV R1, DPH
      MOV DPTR, #N0 ; в DPTR - початкове значення адреси
L3: MOV R6, #00H ; ініціалізація лічильника на 256
L4: MOV R5, #00H ; цикл по M
L5: INC R5
      MOVX A, @DPTR ; x_n → A
      MOV 22H, A ; x_n → 22H
      MOV A, R5 ; m → A
      MOV A, @R5+DPTR ; x_n+m → A
      MOV 23H, A ; x_n+m → 23H
      ACALL MULZ ; x_n·x_n+m, молодший байт в A, старший в
                   доповненні B
      MOV R0, R4 ; (R4) ← K_xx(m=0), одне значення займає два
                   байти
      MOV A, R0
      ADD A, R5
      XCH A, R0 ; R0 ← R4+R5
      ADD A, @R0 ; додавання молодших байтів
      XCH A, @R0 ; пересилка молодшого байта в пам'ять
      XCH A, B
      INC R0
      ADDC A, @R0 ; додавання старших байтів
      XCH A, @R0 ; пересилка старшого байта в пам'ять
      CJNE R5, #M, L5 ; цикл по m
      INC DPTR
      DJNZ R6, L4
      DJNZ R1, L3
      DJNZ R2, L4
; K_xx(m) знаходиться в РПД, починаючи з адреси в R4 і займає
; 2M послідовних комірок
```

5.6. Реалізація коригуючих ланок ЦСАК

Реалізація типових коригуючих ланок на основі ОМЕОМ МК51 надзвичайно спрощується і може бути виконана лише за допомогою одного банку регістрів для зберігання і накопичення

проміжних значень. В прикладах реалізації прийнято, що коефіцієнти різницевого рівняння і сиґнал розузгодженості подані однобайтовими числами із знаком, а результат - двобайтове число в доповняльному коді. На вихід ланки може бути видано не лише старший байт результату, а і значення з заданою розрядністю подання.

Приклад 91. Реалізація інтеґруючої ланки.

В даному випадку реалізується функція

$$y_n = y_{n-1} + b_1 e_{n-1}, \qquad (5.22)$$

де y_n - вихідний сиґнал ланки; e_n - сиґнал похибки системи; b_1 - коефіцієнт пропорційності.

Функція (5.22) представляє рекурсивну ланку (фільтр) першого порядку.

Розміщення даних і результатів в пам'яті мікро-ЕОМ:

в R1 - адреса АЦП, зчитування значень e_n подано числом зі знаком;

в R0 - y_n - адреса вихідного реґістра;

R5 - y_{n-1} - старший байт, R4 - y_{n-1} - молодший байт;

b_1 - число зі знаком.

```
; підготовка
CLR A          ; скидання акумулятора
MOV R4, A      ; обнулення робочої комірки
MOV R5, A
; робота інтеґруючої ланки
ST_I: MOV B, #b1; значення b1 в реґістрі B
      MOV A, @R1 ;значення e(n-1) в акумуляторі
      ACALL MUL_TZ   ; формування додаткового коду
                     добутку b1 e(n-1) і накопичення значення yn
      MOV @R0, A ; видача результату у вихідний реґістр
      JMP ST_I; перехід на початок програми
```

Підпрограма перемноження двох чисел зі знаком, результат поданий доповняльним кодом

```
MUL_TZ:RLC A ; виділення знака e(n-1)
    JNB B.7, L1 ; сума за модулем два знакових
    CPL C ; розрядів b1 та en
L1: MOV 20.7 H,C ; запам'ятовування знака добутку
    MUL AB ; перемноження модулів
    JNB 20.7 H,L2 ; перехід, якщо результат перемноження
                    додатний
```

```
        CPL A ; інверсія акумулятора
        ADD A, #1 ; додавання одиниці
        XCH A, B ; обмін акумулятора і доповнення
        CPL A ; інверсія доповнення
        ADDC A, #0 ; додавання суми з урахуванням переносу
        SETB A.7 ; встановлення знака в акумуляторі і
                   доповнення
        XCH A, B ; доповняльний код добутку
L2:     ADD A, R4 ; накопичення молодшого байта
        MOV R4 A; запам'ятовування молодшого байта
        XCH A, B ; формування результату
        ADDC A, R5 ; накопичення старшого байта
        MOV R5, A ; запам'ятовування старшого байта
                    результату
        RET; повернення в програму
```

Приклад 92. Реалізація ізодромної коригуючої ланки зводиться до реалізації функції

$$y_n = y_{n-1} + b_0\varepsilon_n + b_1\varepsilon_{n-1},$$

що представляє рекурсію першого порядку.

Розміщення даних і результатів у пам'яті

в R1 - адреса АЦП (значення ε_n),

в R0 - адреса вихідного регістра y_n;

R5, R4 - комірки для зберігання значення y_{n-1} відповідно старшого і молодшого байтів;

R3 – комірка для зберігання значення ε_{n-1}

$\#b_1$, $\#b_0$ - числа зі знаком.

```
        ; підготовка
            CLR A ; скидання акумулятора
            MOV R4, A ; обнулення робочих комірок
            MOV R5, A
            MOV R3, A
        ; робота пропорційно-інтегруючої ланки
ST_PI : MOV B, #b1 ;значення b1 в регістрі B
            MOV A, R3 ; значення ε(n-1) в акумуляторі
            ACALL MUL_TZ ; формування додаткового коду добутку
            ; b1, ε(n-1) і накопичення проміжної суми в R5, R4
            MOV B, #b0 ;значення b0 в регістрі B
            MOV A, @R1; зчитування значення εn
            MOV R3, A ; підготовка ε(n-1) для наступного циклу
                        роботи
```

```
        ACALL MUL_TZ  ; формування доповняльного
                      ; коду добутку b0 εn і
                      ; накопичення значення yn в R5, R4
        MOV @R0, A ; видача yn (старшого байта на вихід)
          JMP ST_PI ; перехід на початок програми
```

Приклад 93. Реалізація пропорційно-інтеґродиференціальної функції зводиться до реалізації функції

$$y_n = y_{n-1} + b_0\varepsilon_n + b_1\varepsilon_{n-1} + b_2\varepsilon_{n-2},$$

і відрізняється від реалізації ізодромної ланки одним додатковим добутком $b_2\varepsilon_{n-2}$.

Значення ε_{n-2} розміщуємо в комірці R2, решта величин, як в попередньому прикладі.

```
        ; підготовка
        CLR A ; скидання акумулятора
        MOV R5, A ; обнулення робочих комірок
        MOV R4, A
        MOV R3, A
        MOV R2, A
        ; робота пропорційно-інтеґродиференціальної ланки
ST_PID : MOV B, #b2 ; значення b2 в реґістрі B
          MOV A, R2 ; значення εn-2 в акумуляторі
        ACALL MUL_TZ ; формування добутку b2. εn-2 і накопичення
                       значень в R4 і R5
          MOV B, #b1 ; значення b1 в реґістрі B
          MOV A, R3; значення εn-1 в акумуляторі
          MOV R2, A ; формування εn-2 для наступного такту
          ACALL MUL_TZ ; формування добутку b1 εn-1 і
                         накопичення в R4 і R5
          MOV B, #b0 ;значення b0 в реґістрі B
          MOV A, @R1; значення εn-1 в акумуляторі
          MOV R3, A ; формування εn-1 для наступного такту
          ACALL  MUL_TZ ; формування добутку b0 εn і
                          накопичення в R4 і R5
           MOV @R0, A ; видача результату у вихідний реґістр
           JMP ST_PID; перехід на початок програми
```

Приклад 94. Реалізація пропорційної ланки є найпростішою і описується функцією

$y_n = b_0\varepsilon_n$.

Розміщення даних таке ж, як і в попередніх прикладах.

```
ST_P : MOV B, #b0
       MOV A, @R1
       ACALL MUL_TZ1; (використовується тіло підпрограми
                      MUL_TZ до мітки L2 - лише операція
                      перемноження чисел зі знаком)
    MOV @R0, A ; видача старшого байта добутку b0εn на вихід
    JMP ST_P ; перехід на початок програми
    Підпрограма перемноження чисел зі знаком :
    MUL_TZ1 : RLC A; виділення знаку εn-1
    JNB B.7, L1        ; сума за модулем два знакових розрядів
L1 : MOV 20.7H,C; запам'ятовування знака добутку
    MUL AB             ; перемноження модулів
    JNB 20.7H, L2      ; відновлення доповняльного коду
    CPL A              ; результату перемноження
    ADD A, #1
    XCH A, B
    CPL A
    ADDC A, #0
    SETB A.7
    XCH A, B
L2 : RET               ; повернення в основну програму.
```

Приклад 95. Реалізація пропорційно-диференціальної ланки описується функцією

$y_n = b_0\varepsilon_n + b_1\varepsilon_{n-1}$.

Програма має наступний вигляд.

```
    ; підготовка
    CLR A ; скидання акумулятора
    MOV R3, A ; скидання робочої комірки
ST_PD : MOV B, #b1
    MOV A, R3
    ACALL MUL_TZ ; формування добутку b1 εn-1 і накопичення
                   в R4 і R5
    MOV R4, A ; запам'ятовування добутку
    MOV R5, B
    MOV B, #b0
    MOV A, @R1
    MOV R3, A
    ACALL MUL_TZ ; формування добутку b0 εn і його
                   накопичення в R4 і R5
    MOV @R0, A ; видача yn на вихід (старшого байта)
    JMP ST_PD ; перехід на початок програми.
```

Приклад 96. Реалізація спеціальних законів керування технологічними об'єктами

При побудові ЦСК технологічними об'єктами поряд з типовими законами реґулювання часто застосовують спеціальні закони керування стосовно до конкретного об'єкту.

При регулюванні кількості рідини (фарби, проявника, травильного розчину, розчинника і т.п.) або газу, широко розповсюджені діафрагмові вимірювачі. Для розширення діапазону вимірювань застосовують вимірювачі з отвором , який змінюється у відповідності до зміни швидкості потоку. В якості такого пристрою служить клапанний пристрій, пропускна здатність якого описується коефіцієнтом $K_в$ (рис.5.15) [8].

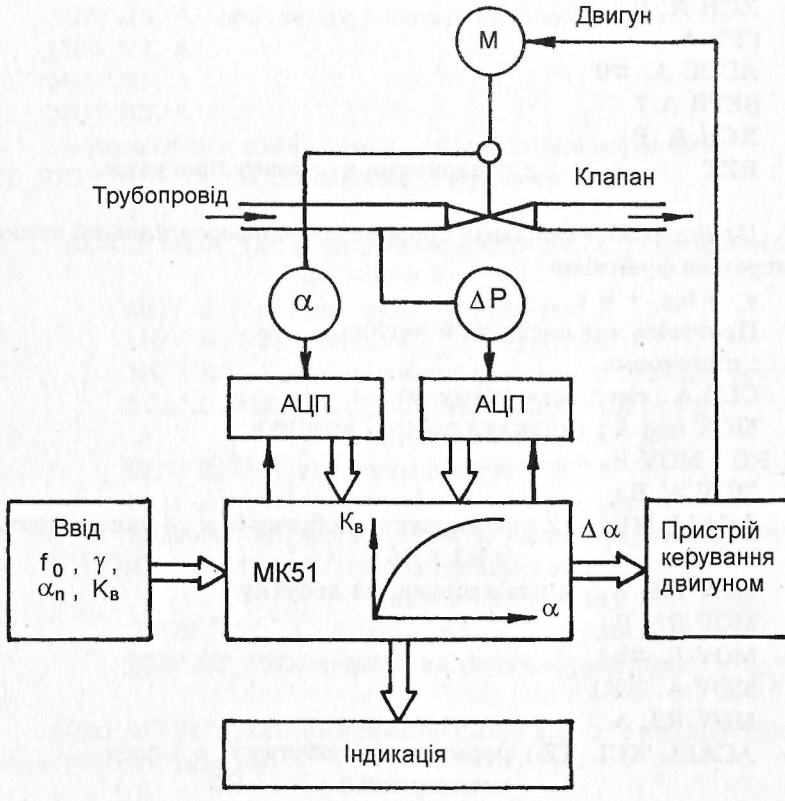


Рис. 5.15. Схема керування потоком рідини (газу)

Цей коефіцієнт залежить від положення заслонки і визначається з рівняння

$$f = K_B\sqrt{\Delta P / \gamma}\ ,$$

де f - витрата рідини через клапан;

γ - питома вага речовини;

ΔP - перепад тиску.

Коефіцієнт K_B і, відповідно, розмір отвору вимірювача відповідає певному перепаду тиску DP на клапані. При зміні перепаду тиску необхідно проводити відповідну корекцію значення K_B за допомогою керуючої дії Ка, яку виконавчий механізм перетворює в положення клапана а. Розхід рідини є функцією перепаду тиску ΔP та положення заслонки а. Оцінку потоку знаходять з рівняння

$$f = K_B(\alpha)\sqrt{\Delta P / \gamma}\ .$$

ЦСК включає в себе вимірювальний комплекс: давачі перепаду тиску і положення клапану; виконавчий пристрій - електродвигун Д; два АЦП, що перетворюють неперервні (аналогові) сиґнали з виходів давачів; ОМЕОМ (МК51), яка формує керуючу послідовність; пульт керування і схему індикації (при необхідності).

Для вимірювання положення заслонки клапана використовується потенціометричний або ємнісний давач, зв'язаний з віссю заслонки клапана. Вимірювання витрати рідини здійснюється за допомогою диференційних манометрів. Положення заслонки клапана керується двигуном за допомогою тиристорного пристрою керування.

В РПП МК51 зберігається залежність $K_B = \Omega(\alpha)$, програми реґулювання, а також усі необхідні константи і дані. В МК проводяться всі обчислення, по зміні яких ґенерується керуюча дія $\Delta\alpha$.

Заміна параметрів або запуск сервісних програм (градуювання, повного відкривання або закривання клапана, перевірки параметрів) забезпечується програмою переривань. Ввід інформації - з пульта керування.

Таким чином, реалізація ЦСАК зводиться до складної системи з можливістю втручання оператора в процес регулювання в залежності від ходу технологічного процесу.

5.7. Локальна керуюча мікромережа на основі МК51

При керуванні складними технологічними об'єктами використовуються системи з розподіленим керуванням, які складаються з групи контролерів, що керують окремими аґреґатами об'єкта (наприклад, багатосекційною друкарською машиною, папероробною установкою, палітурно-обробною лінією, прокатним станом і т.п.). Між окремими підсистемами повинен бути забезпечений інформаційний зв'язок. Тому обов'язковим елементом розподіленої системи керування є локальна мережа, що об'єднує певні контролери в систему.

На таку мережу покладаються звичайно прості функції передачі повідомлень за ґарантований час. Протяжність ліній зв'язку звичайно не перевищує десятків метрів, розмір повідомлення - декількох десятків байтів, а час доставки повідомлень - в межах від 0.01 до 1 с. Типовими є два режими інформаційного обміну в мережі: *широкомовний*, коли повідомлення, яке надсилається, призначається для всієї решти підсистем (мікроконтролерів мережі), та *абонентський*, коли повідомлення призначається лише одному МК. Звичайно першим способом передаються різні інформаційні параметри, які використовуються багатьма підсистемами. Це дозволяє зменшити завантаження мережі за рахунок вилучення численних передач одного і того ж повідомлення різним адресатам. Другим способом звичайно передаються команди керування від центрального пристрою до виконавчих засобів або повідомлення екстренного характеру.

Найбільш широко розповсюджені локальні мережі двох структур: *кільцеві* та *моноканальні* (типу BITBUS). Останні є більш зручними для керуючих мікромереж, бо допускають просте нарощування та модифікацію системи. Крім цього, в моноканальній мережі час доставки повідомлення не залежить від загального числа МК і вони мають високу живучість та надійність.

В мережах з єдиним моноканалом всі МК зв'язані між собою однією спільною (розподільною) лінією зв'язку (рис. 5.16). В лініях передачі даних звичайно використовується коаксіальний кабель або вита пара з резисторами узгодження на їх кінцях.

Відомі різні методи доступу до розподільної лінії (протоколи), які дозволяють здійснювати обмін даними між багатьма МК мережі, тобто забезпечувати розділення каналу зв'язку між багатьма підсистемами [5,21].

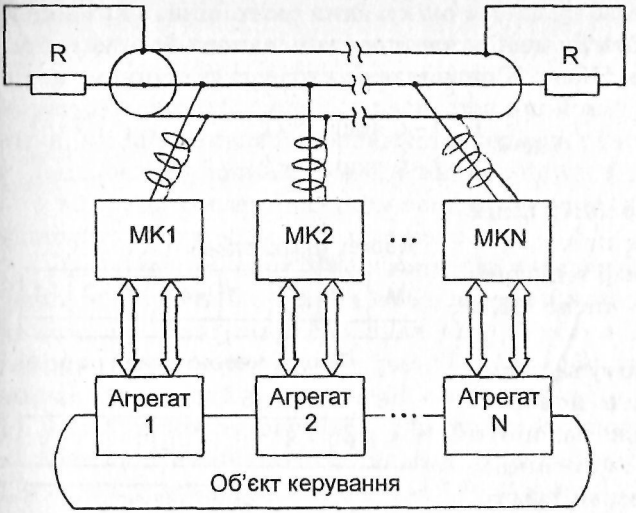


Рис. 5.16. Структура локальної керуючої мікромережі на основі моноканалу

Розглянемо *інтервально-маркерний метод* доступу до розподільного моноканалу, який дозволяє усунути конфлікти в каналі і достатньо повно використати пропускну здатність каналу. Суть методу полягає в наступному:

1. При нульовому завантаженні в каналі періодично з'являється маркер, який ґенерується одним з МК мережі. Маркер містить номер МК, який є ведучим. Головне призначення ведучого - підтримувати синхронізм в мережі за рахунок періодичної видачі маркера в канал.

2. Період ґенерації маркера складається з певного числа «вікон», число яких рівне числу МК в мережі. Кожне вікно має свій номер і належить одній із підсистем.

3. У процесі захоплення каналу МК, який бажає видати свій пакет (повідомлення), повинен діждатися появи маркера і відрахувати від нього своє вікно.

Якщо при цьому його не випередять інші МК, то, діждавшись свого вікна, підсистема може, не побою-ючись конфліктів, почати передачу даних (рис. 5.17,а).

4. Після видачі повідом-лення МК ґенерує свій маркер і стає новим ведучим. Старий ведучий мікромережі, розпізнавши, що моноканал є захоплений, звільняється від цієї ролі.

5. Відрахунок моменту часу від маркера свого вікна проводиться за таким пра-вилом. Тривалість вікна приймається рівною часу передачі одного байта даних. Якщо ведучий мав номер l, то перше вікно буде належати МК з номером $l+1$, потім

МК з номером $l+2$ і т.д. Час очікування свого вікна (Т) можна визначити як $T=t \cdot X$, де t - час передачі одного байта, тобто тривалість вікна. Число X визначається наступним чином:

$$X = \begin{cases} k - l - 1, & \text{якщо } k > l; \\ k - l + 1, & \text{якщо } k < l; \end{cases}$$

де k- номер МК, який пробує захопити канал, $k = 0 \div (n-1)$; l – номер ведучого, $l = 0 \div (n-1)$; n - число МК в мережі.

6. Якщо самому ведучому необхідно видати повідомлення, то він може захопити канал під час свого вікна, тобто замість ґенерації чергового маркера почати передавати своє повідомлення.

7. Видавши маркер, ведучий МК запускає таймер на час $t \cdot (n-1)$, а якщо за цей час жодний МК не захопить канал, то весь цикл повторюється ще раз і т.д.

Рис. 5.17. Інтервально-маркерний доступ до моноканалу: а) діаграма одного періоду мережі; б) маркер; в) формат повідомлення

8. Кожний МК приймає всі байти, які передаються по каналу. Для контролю пропадання маркера кожний МК після прийому кожного байта запускає таймер на затримку $t \cdot (n+1)$. Таким чином, пропадання маркера (а значить, і режиму синхронізму мікромережі) фіксується, якщо за час $t \cdot (n+1)$, не було передано ні одного повідомлення.

9. При виявленні пропадання маркера для відновлення синхронізму в мікромережі кожний МК виконує наступні прості дії: витримує паузу тривалістю $t \cdot (i+1)$, де i - власний номер МК; якщо під час паузи знову не було прийнято жодної інформації, то даний МК стає ведучим і ґенерує новий маркер.

Цією процедурою забезпечується автоматичне відновлення роботи мікромережі при відмові МК, який є в даний момент ведучим.

10. При інтервально-маркерному методі вдається уникнути

будь-яких конфліктів в колі силою таких причин:

контроль пропадання маркера здійснюється постійно усіма МК і воно повністю синхронне, тому що лічильники паузи коригуються приблизно одночасно при прийомі кожного байта і, отже, всі МК виявлять пропадання маркера одночасно;

одночасно починається відрахунок паузи t $(i+1)$ всіма МК;

мікроконтролер з меншим номером першим ґенерує маркер і відновить синхронізм в мікромережі.

Послідовний порт МК51 допускає передачу 9-бітних кодів. Використовуючи це, можна легко ввести ознаку маркера таким чином, що байт маркера буде відрізнятися від будь-якого інформаційного байта. На рис. 5.17,б зображена структура маркера; старший біт визначає ознаку маркера (для маркера - 1). Біт 7 використовується для найпростішого контролю за паритетом. Семибітне поле адреси дозволяє мати в системі до 127 підсистем з номерами від 0 до 126. Адреса 127 зарезервована для широкомовної передачі.

Рекомендований формат повідомлення представлений на рис.5.17,в і передбачає такі поля : A_0 - адреса одержувача; A_B - адреса відправника ; L - довжина поля даних (0-255); CRC - байт контрольної суми.

Можна визначити чотири стани, в яких буде знаходитися кожний МК мікромережі.

Приймач (R). В цьому стані МК прослуховує канал, приймає повідомлення і вибирає з них необхідну йому інформацію.

Передавач (W). В цьому стані МК, захопивши канал, передає своє повідомлення.

Ведучий (H). МК є ведучим і підтримує синхронізм в мережі.

Спеціальний стан (RM). МК реалізує процедуру відновлення синхронізму в мережі.

Граф станів МК мікромережі представлений на рис. 5.18.

Запит на передачу пові-домлення формується в МК прикладною програмою керу-вання об'єктом і позначений RQ. Затримки, що реалізу-ються таймером, мають такий зміст:

ТМ1 - контроль зникнення маркера, затримка рівна τ $(n+1)$;

ТМ2 - очікування свого вікна (τ x);

ТМ3 - очікування закінчення періоду мережі (τ n);

ТМ4 - пауза перед вида-чею маркера при відновленні синхронізму, затримка рівна τ $(i+1)$.

Для реалізації підсистеми необхідні наступні ресурси: УАПП, таймер, два рівні переривань. Цими ресурсами володіє МК51,

який дозволяє вести передачу та прийом даних з швидкістю до 375 кбіт/с. Час передачі одного байта, обрамленого стартовими і стоповими бітами (плюс 9-й розряд), складає 58,7 мкс. Пропускна здатність мікромережі при цьому рівна приблизно 17 кбайт/с.

Мікроконтролер, який працює в складі розподіленої системи керування на основі локальної мікромережі, повинен, крім прикладної програми керування, мати ще програмні засоби доступу до моноканалу. Таким чином, МК повинен працювати в двопрограмному режимі з розподіленням усіх ресурсів між цими двома співпрограмами. Зрозуміло, що при цьому повинен бути реалізований механізм взаємодії між мережною та прикладною програмами. Найчастіше цей механізм реалізується шляхом присвоєння мережній програмі більш високого пріоритету.

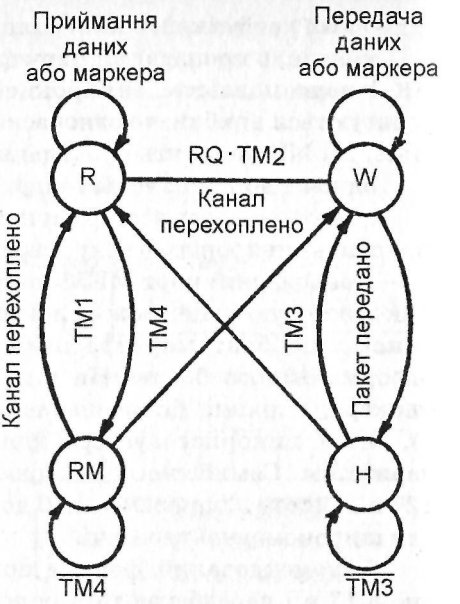


Рис. 5.18. Граф станів МК мікромережі

ПРОГРАМИ ПЕРЕТВОРЕННЯ ДЕСЯТКОВОГО ЧИСЛА В ДВІЙКОВЕ

```
program D_B;
{Програма перетворення десяткового числа в двійкове}
{розрядність двійкового числа 25}
uses Crt;
  var i,r:integer;
      s,s1,s2:string;
      d:real;
procedure Vvid_d(var d0:real);
        {ввід числа для перетворення}
var s:string;
   i:integer;
begin;
   repeat
     write(#10#13'Число десяткове = ');
     readln(s);       val(s,d0,i);
   until i=0;
end;
begin
   textcolor(yellow);   textbackground(blue);  ClrScr;
 repeat
    Vvid_d(d);
     write('Число двійкове = ');
     if d<0 then
       begin    {друк знака}
         d:=-d; write('-');
       end;
   {перетворення модуля числа}
     C_ch(d,r,s1);   {ціла частина}
      for i:=1 to r do write(s1[i]);  write('.');
     Dr_ch(d,25-r,s2);   {дробова частина}
      for i:=1 to 25-r do write(s2[i]);
    s:=s1+s2;
 until readkey=#27;
end.
```

ПРОГРАМА ПЕРЕТВОРЕННЯ ДЕСЯТКОВОГО ЧИСЛА З ЗАДАНИМИ МЕЖАМИ В ДВІЙКОВЕ З МІНІМАЛЬНИМ ЧИСЛОМ ОДИНИЦЬ

```
program D_BM;
{Програма перетворення десяткового числа з заданими межами
в двійкове з мінімальним числом одиниць}
uses Crt;
const rmax=25;
{Розрядність двійкового числа 25}
  var i,j,jmax,k,r,rc:integer;
      s,s1,s2,sopt:string;
      d,d0,dd,dopt:real;
 label L1,L2;
{крок десяткового числа 2^-(rmax - 2)}
procedure Vvid_d(var d0,dd:real);
var s:string;
   i:integer;
begin;
   repeat
     write(#10#13'Число десяткове = ");
     readln(s);       val(s,d0,i);
   until i=0;
   repeat
     write(#10#13'Межі числа□= ");
     readln(s);       val(s,dd,i);
   until i=0;
end;

procedure C_ch(Vx:real;var r:integer;var s:string);
{10-2 перетворення цілої частини числа}
{r – розрядність двійкового числа}
var d,o,j:integer;
   i:byte absolute s;
    ch:char;
begin
  i:=0;  d:=round(int(Vx));
  while d>0 do
    begin
      o:=d mod 2; d:=d div 2; i:=i+1;
      if o=0 then s[i]:='0' else s[i]:='1';
    end;
```

```
r:=i;
  for j:=1 to i div 2 do
    begin
      ch:=s[j]; s[j]:=s[i+1-j]; s[i+1-j]:=ch;
    end;
end;

procedure Dr_ch(Vx:real;r:integer;var s:string);
{визначення дробової частини двійкового числа}
{r – задана розрядність дробової частини двійкового числа}
var d:real;
   i:byte absolute s;
begin
  d:=frac(Vx); s:=''; i:=0;
  while i<r do
    begin
      i:=i+1; d:=d*2;
      if d>1 then begin s[i]:='1'; d:=d-1; end
            else
        if d<1 then s[i]:='0'
              else
          if d=1 then
            begin
              s[i]:='1';
              while i<r do begin s[i+1]:='0'; i:=i+1; end;
              exit;
            end;
    end;
end;

begin
   textcolor(yellow);    textbackground(blue);    Clrscr;
 repeat
     Vvid_d(D0,dd);
     d:=d0-dd;
     jmax:=rmax; sopt:='';
     while d<d0+dd do
       begin      {10-2 перетворення}
           i:=rmax; C_ch(d,rc,s1); Dr_ch(d,rmax-rc,s2); s:=s1+s2;
L1: k:=0;
{Заміна груп двійкових одиниць на знакорозрядний код}
```

```
        if i<=2 then goto L2;
        while s[i]='1' do begin inc(k); dec(i); end;
        if k>=3 then
          begin
{заміна групи одиниць на групу 10...0-1}
            s[i+k]:='m';
            for j:=i+k-1 downto i+1 do s[j]:='0';
            if i=0 then
              begin
                for r:=rmax-1 downto 1 do s[r+1]:=s[r];
                inc(i); inc(rc);
              end;
            s[i]:='1'; j:=0;
            for r:=1 to rmax do if s[r]<>'0' then inc(j);
            if j<jmax then
              begin
{вибір двійкового числа з мінімальним числом одиниць}
                sopt:=s; jmax:=j; dopt:=d;
                Write(#10#13'Локальний мінімум := ',jmax,
                      ' для  d= ',dopt:1:8,' s= ');
                for r:=1 to rc do write(sopt[r]);
                write('.');
                for r:=rc+1 to rmax do write(sopt[r]);
              end;
          end   else
          dec(i);
          goto L1;
   L2:     d:=d+exp((rmax-2)*ln(0.5));
      end;
{друк двійкового числа з мінімальним числом одиниць}
{замість -1 друкується буква m}
  Write(#10#13'Мінімум одиниць = ',jmax,' для  d= ',dopt:1:8,'
s= ');
   for r:=1 to rc do write(sopt[r]);
   write('.');
   for r:=rc+1 to rmax do write(sopt[r]);
 until readkey=#27;
end.
```

ОПИС МАШИННИХ КОМАНД ОМЕОМ МК51

Команда ACALL (addr 11)

Команда *абсолютний виклик підпрограми* викликає безумовно підпрограму, розташовану за вказаною адресою. При цьому лічильник команд збільшується на 2 для одержання адреси наступної команди , після чого отримане 16-бітове значення PC поміщається в стек (спочатку йде молодший байт), і вміст вказівника стека також збільшується на два. Адреса переходу отримується за допомогою конкатенації старших бітів збільшеного вмісту лічильника команд, бітів старшого байта команди і молодшого байта команди.

Асемблер: ACALL (мітка)

Код: | A10 A9 A8 0 0 0 0 1 | A7 A6 A5 A4 A3 A2 A1 A0 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+2
(SP):=(SP)+1
((SP)):=(SP)+1
(SP):=(PC[7-0])
(SP):=(SP)+1
(SP):=(PC[15-8])
(PC[10-0]):=A10A9A8 II A7A6A5A4A3A2A1A0,
де II - знак конкатенації (зчіплення)

Приклад: ; ДО ВИКОНАННЯ КОМАНДИ ACALL
;(SP)=07H
;мітка MT1 відповідає адресі: 0345H, тобто
;(PC)=0345H
ACALL MT1 ;розташована за адресою 028DH, тобто
;(PC)=028DH
;ПІСЛЯ ВИКОНАННЯ КОМАНДИ
;(SP)=09H, (PC)=0345H,
;ОЗП [08]=8FH, ОЗП [09]=02H.

Команда ADD A, (байт-джерело)

Ця команда *додавання* додає вміст акумулятора А з вмістом байта-джерела, залишаючи результат в акумуляторі. При появі переносів із розрядів 7 і 3, встановлюються прапорці переносу «С» і додаткового переносу «АС» відповідно, в іншому випадку ці прапорці скидаються. При додаванні цілих чисел без знака прапорець переносу «С» вказує на появу переповнення. Переповнення «OV» встановлюється, якщо є перенос із біта 7, або є перенос із біта 7 і немає - із біта 6, в іншому випадку прапорець OV скидається. При додаванні цілих чисел із знаком прапорець OV вказує на від'ємну величину, одержану при сумуванні двох додатних операндів або на додатну суму для двох від'ємних операндів.

Для команди додавання дозволені наступні режими адресації байта-джерела:

1) регістровий;
2) непрямо-регістровий;
3) прямий;
4) безпосередній.

1) Асемблер: ADD A,Rn ; де n=0-7

Код: | 0 0 1 0 1 r r r | , де rrr=000-111

Час: 1 цикл

Алгоритм: (A):=(A)+(Rn), де n=0-7
C:=X, OV:=X, AC:=X, де X=(0 або 1)

Приклад: ;(A)=C3H, (R6)=AAH
ADD A,R6 ;(A)=6DH, (R6)=AAH
;(AC)=0, (C)=1, (OV)=1

2) Асемблер: ADD A,@Ri ; де i=0,1

Код: | 0 0 1 0 0 1 1 i | , де i=0,1

Час: 1 цикл

Алгоритм: (A):=(A)+((Ri)), де i=0,1
C:=X, OV:=X, AC:=X, де X=(0 або 1)

Приклад: ;(A)=95H, (R1)=31H, (ОЗП [31])=4CH
ADD A,@R1 ;(A)=E1H, (ОЗП [31])=4CH,
;(C)=0, (AC)=1, (OV)=0

3) Асемблер: ADD A, (direct)

Код: | 0 0 1 0 0 1 0 1 | direct address |

Час: 1 цикл

Алгоритм: (A):=(A)+(direct)
C:=X, OV:=X, AC:=X, де X=(0 або 1)

Приклад: ;(A)=77H, (ОЗП [90])=FFH
ADD A,90H ;(A)=76H, (ОЗП [90])=FFH,
;(C)=1, (OV)=0, (AC)=1

4) Асемблер: ADD A,(#data)

Код: | 0 0 1 0 0 1 0 0 | #data |

Час: 1 цикл

Алгоритм: (A):=(A)+#data
C:=X, OV:=X, AC:=X, де X=(0 або 1)

Приклад: ;(A)=09H
ADD A,#0D3H ;(A)=DCH,
;(C)=0, (OV)=0, (AC)=0

Команда ADDC A, (байт-джерело)

Ця команда *додавання з переносом* одночасно додає вміст байта-джерела, прапорця переносу і вміст акумулятора A, залишаючи результат в акумуляторі. При цьому прапорець переносу і прапорець додаткового переносу встановлюються, якщо є перенос із біта 7 або біта 3, і скидаються в протилежному випадку. При додаванні цілих чисел без знака прапорець переносу

вказує на переповнення. Переповнення (OV) встановлюється, якщо є перенос біта 6 і немає переносу із біта 7 або є перенос із біта 7 і немає переносу із біта 6, в протилежному випадку OV скидається. При додаванні цілих чисел із знаком прапорець OV вказує на від'ємну величину, одержану при сумуванні двох додатних операндів або на додатну суму від двох від'ємних операндів.

Для цієї команди дозволені наступні режими адресації байта-джерела:

1) регістровий;
2) непрямо-регістровий;
3) прямий;
4) безпосередній.

1) Асемблер: ADDC A,Rn ; де n=0-7

Код: 0 0 1 1 1 rrr , де rrr=000-111

Час: 1 цикл

Алгоритм: (A):=(A)+(C)+(Rn)
C:=X, OV:=X, AC:=X, де X=(0 або 1)

Приклад: ;(A)=B2H, (R3)=99H,
;(C)=1
ADDC A,R3 ;(A)=4CH, (R3)=99H
;(AC)=0, (C)=1, (OV)=1

2) Асемблер: ADDC A,@Ri ; де i=0,1

Код: 0 0 1 1 0 1 1 i , де i=0,1

Час: 1 цикл

Алгоритм: (A):=(A)+(C)+((Ri)), де i=0,1
C:=X, OV:=X, AC:=X, де X=(0 або 1)

Приклад: ;(A)=D5H, (R0)=3AH,
;(ОЗП [3A])=1AH, (C)=1
ADDC A,@R0 ;(A)=F0H, (ОЗП [3A])=1AH,
;(C)=0, (AC)=1, (OV)=0

3) Асемблер: ADDC A, (direct)

Код: | 0 0 1 1 0 1 0 1 | direct address |

Час: 1 цикл

Алгоритм: (A):=(A)+(C)+(direct)
C:=X, OV:=X, AC:=X, де X=(0 або 1)

Приклад: ;(A)=11H, (ОЗП [80])=DFH, (C)=1
ADDC A,80H ;(A)=F1H, (C)=0, (OV)=0, (AC)=1

4) Асемблер: ADDC A,(#data)

Код: | 0 0 1 1 0 1 0 0 | #data |

Час: 1 цикл

Алгоритм: (A):=(A)+(C)+#data
C:=X, OV:=X, AC:=X, де X=(0 або 1)

Приклад: ;(A)=55H, (C)=0
ADDC A,#55H ;(A)=55H, (C)=0, (OV)=1, (AC)=0

Команда AJMP (addr11)

Команда *абсолютний перехід* передає керування за вказаною адресою, яка отримується шляхом конкатенції п'яти старших бітів лічильника команд PC (після збільшення на два), 7-5 бітів коду операції і другого байта команди. Адреса переходу повинна знаходитися всередині однієї сторінки ємністю 2 Кбайти пам'яті програми, яка визначається п'ятьма старшими бітами лічильника команд.

Асемблер: AJMP (мітка)

Код: | A10 A9 A8 0 0 0 0 1 | A7 A6 A5 A4 A3 A2 A1 A0 |

Час: 2 цикли

Алгоритм: (PC[15-0]):=(PC[15-0])+2,

(PC[10-0]):=(addr11)

Приклад: ;(PC)=028FH
;мітці MT2 відповідає адреса 034AH

AJMP MT2 ;(PC)=034AH

Команда ANL (байт-джерело)

Команда *логічне «І» для змінних типу байтів* виконує операцію логічного «І» над бітами вказаних змінних і розміщує результат у байт призначення. Ця операція не впливає на стан прапорців.

Два операнди забезпечують наступні комбінації шести режимів адресації:

- байтом призначення є акумулятор (A):

1) регістровий;
2) прямий;
3) непрямо-регістровий;
4) безпосередній;

- байтом призначення є пряма адреса (direct):

5) прямий акумуляторний;
6) безпосередній (байт-джерело дорівнює константі).

1) Асемблер: ANL A,Rn ; де n=0-7

Код: | 0 1 0 1 1 rrr | , де rrr=000-111

Час: 1 цикл

Алгоритм: (A):=(A) AND (Rn)

Приклад: ;(A)=FEH, (R2)=C5H
ANL A,R2 ;(A)=C4H, (R2)=CAH

2) Асемблер: ANL A,(direct)

Код: | 0 1 0 1 0 1 0 1 | | direct address |

Час: 1 цикл

Алгоритм: (A):=(A) AND (direct)

Приклад: ;(A)=A3H, (PSW)=86H
ANL A,PSW ;(A)=82H, (PSW)=86H

3) Асемблер: ANL A,@Ri ; де i=0,1

Код: | 0 1 0 1 0 1 1 i | , де i=0,1

Час: 1 цикл

Алгоритм: (A):=(A) AND ((Ri))

Приклад: ;(A)=BCH, (ОЗП [35])=47H, (R0)=35H
ANL A,@R1 ;(A)=04H, (ОЗП [35])=47H

4) Асемблер: ANL A,(#data)

Код: | 0 1 0 1 0 1 0 0 | #data8 |

Час: 1 цикл

Алгоритм: (A):=(A) AND #data

Приклад: ;(A)=36H
ANL A,#0DDH ;(A)=14H

5) Асемблер: ANL (direct),A

Код: | 0 1 0 1 0 0 1 0 | direct address |

Час: 1 цикл

Алгоритм: (direct):=(direct) AND (A)

Приклад: ;(A)=55H, (P2)=AAH
ANL P2,A ;(P2)=00H, (A)=55H

6) Асемблер: ANL (direct),#data

Код: | 0 1 0 1 0 0 1 1 | direct address | #data8 |

Час: 2 цикли

Алгоритм: (direct):=(direct) AND #data

Приклад: ;(P1)=FFH
ANL P1,#73H ;(P1)=73H

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

Команда ANL C, (біт джерела)

Команда *логічне «І» для змінних бітів*, виконує операцію логічного «І» над вказаними бітами. Якщо біт-джерело рівний «0», то відбувається скидання прапорця переносу, в протилежному випадку прапорець переносу не змінює поточного значення. Косий дріб «/» перед операндом у мові асемблера вказує на те, що за значення використовується логічне заперечення адресованого біта, але сам біт джерела при цьому не змінюється. На інші прапорці ця команда не впливає.

Для операнда-джерела дозволена лише пряма адресація до бітів.

1) Асемблер: ANL C (bit)

Код: | 1 0 0 0 0 0 1 0 | bit address |

Час: 2 цикли

Алгоритм: (C):=(C) ANL (bit)

Приклад: ;(C)=1, P1(0)=0
ANL C,P1.0 ;(C)=0, P1(0)=0

2) Асемблер: ANL C,(/bit)

Код: | 1 0 1 1 0 0 0 0 | bit address |

Час: 2 цикли

Алгоритм: (C):=(C) AND (/bit)

Приклад: ;(C)=1, (AC)=0
ANL C,/AC ;(C)=1, (AC)=0

Команда CJNE (байт-призначення), (байт-джерело), (зміщення)

Команда *порівняння і перехід, якщо не рівне* порівнює значення перших двох операндів і виконує розгалуження, якщо операнди не рівні. Адреса переходу (розгалуження) обчислюється за допомогою додавання значення (зі знаком), вказаного в останньому байті команди, з вмістом лічильника команд після збільшення його на три.

Прапорець переносу «C» встановлюється в «1», якщо значення цілого без знака <байта призначення> менше, ніж значення цілого без знака <байта джерела>, в протилежному випадку перенос скидається (якщо значення операндів рівні, то прапорець переносу скидається). Ця команда не впливає на операнди.

Операнди, які стоять в команді, забезпечують комбінації чотирьох режимів адресації:

- якщо байтом-призначення є акумулятор:

1) прямий

2) безпосередній,

- якщо байтом-призначення є будь-яка комірка ОЗП з непрямо-регістровою або регістровою адресацією:

3) безпосередній до регістрового

4) безпосередній до непрямо-регістрового.

1) Асемблер: CJNE A,(direct),(мітка)

Код: | 1 0 1 1 0 1 0 1 | direct address | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
якщо (direct) < (A) то (PC):=(PC)+(rel8), C:=0
якщо (direct) > (A) то (PC):=(PC)+(rel8), C:=1

Приклад: ;(A)=97H, (P2)=F0H, (C)=0
CJNE A,P2,MT3
...
MT3: CLR A ;(A)=97H, (P2)=F0H, (C)=1
;Адреса відповідна мітці
;MT3 обчислюється, як
;(PC):=(PC)+3+(rel8)

2) Асемблер: CJNE A, #data, (мітка)

Код: | 1 0 1 1 0 1 0 0 | #data8 | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
якщо #data < (A) то (PC):=(PC)+(rel8), C:=0
якщо #data8 > (A) то (PC):=(PC)+(rel8), C:=1

Приклад: ;(A)=FCH, (C)=1
CJNE A,#0BFH,MT4
...
MT4: INC A ;(A)=FDH, C=0,
;(PC):=(PC)+3+(rel8)

3) Асемблер: CJNE Rn,#data,(мітка); де n=0-7

Код: | 1 0 1 1 1 rrr | #data | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
якщо #data < (Rn) то (PC):=(PC)+(rel8), C:=0
якщо #data8 > (Rn) то (PC):=(PC)+(rel8), C:=1

Приклад: ;(R7)=80H, (C)=0
CJNE R7,#81H,MT5
...
MT5: NOP ;(R7)=80H,(C)=1
;(PC):=(PC)+3+(rel8)

4) Асемблер: CJNE @Ri,#data,(мітка); де i=0,1

Код: | 1 0 1 1 1 rrr | | #data | | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
якщо #data < ((Ri)) то (PC):=(PC)+(rel8), C:=0
якщо #data8 > ((Rn)) то (PC):=(PC)+(rel8), C:=1

Приклад: ;(R0)=41H, (C)=1, (ОЗП[41])=57H
CJNE @R0,#29H,MT6
...
MT6: DEC R0
;(ОЗП[41])=57H, (C)=0,
;(PC):=(PC)+3+(rel8)

Команда CLR A

Команда *скидання акумулятора* скидає (обнулює) вміст акумулятора A. На прапорці команда не впливає.

Асемблер: CLR A

Код: | 1 1 1 0 0 1 0 0 |

Час: 1 цикл

Алгоритм: (A):=0

Приклад: ;(A)=6DH, (C)=0, (AC)=1
CLR A ;(A)=00H, (C)=0, (AC)=1

Команда CLR (біт)

Команда *скидання біта* скидає вказаний біт в нуль. Ця команда працює з прапорцем переносу «C» або будь-яким бітом з прямою адресацією.

1) Асемблер: CLR C

Код: | 1 1 0 0 0 0 1 1 |

Час: 1 цикл

Алгоритм: (C):=0

Приклад: ;(C)=1
CLR C ;(C)=0

2) Асемблер: CLR (bit)

Код: | 1 1 0 0 0 0 1 0 | bit address |

Час: 1 цикл

Алгоритм: (bit):=0

Приклад: ;(P1)=5EH, (01011110B)
CLR P1.3 ;(P1)=56H, (01010110B)

Команда CPL A

Команда *інверсія акумулятора* інвертує (змінює на протилежний) кожний біт акумулятора. Біти, які вміщують «одиниці», після цієї команди будуть вміщувати «нулі», і навпаки. На прапорці ця операція не впливає.

Асемблер: CPL A

Код: | 1 1 1 1 0 1 0 0 |

Час: 1 цикл

Алгоритм: (A):=/(A)

Приклад: ;(A)=65H, (01100101B)
CLR A ;(A)=9AH, (10011010B)

Команда CPL (біт)

Команда *інверсія біта* інвертує (змінює на протилежне значення) вказаний біт. Біт, який був «одиницею», змінюється в «нуль», і навпаки. Команда CPL може працювати з прапорцем

переносу чи з будь-яким прямо адресованим бітом. На інші прапорці команда не впливає.

1) Асемблер: CPL (bit)

Код: 1 0 1 1 0 0 1 0 1

Час: 1 цикл

Алгоритм: (bit):=/(bit)

Приклад: ;(P1)=39H (00111001B)
CPL P1.1
CPL P1.3 ;(P1)=33H (00110011B)

2) Асемблер: CPL C

Код: 1 0 1 1 0 0 1 1

Час: 1 цикл

Алгоритм: (C):=/(C)

Приклад: ;(C)=0, (AC)=1, (OV)=0
CPL C ;(C)=1, (AC)=1, (OV)=0

Примітка: Якщо ця команда використовується для зміни інформації на виході порту, значення, що використовуються як вхідні дані, зчитуються з «фіксатора» порту, а не з виводів BIC.

Команда DA A

Команда *десяткова корекція акумулятора для додавання* упорядковує 8-бітову величину в акумуляторі після виконання попередньої команди додавання двох змінних (кожна в запакованому двійково-десятковому форматі). Для виконання додавання може бути використана будь-яка із типів команд ADD або ADDC. Якщо значення бітів 3-0 акумулятора (A) перевищує 9 (XXXX1010 - XXXX1111) або, якщо прапорець AC дорівнює «1», то до вмісту (A) додається 06, отримуючи відповідну двійково-десяткову цифру в молодшому півбайті. Це внутрішнє

побітове додавання встановлює прапорець переносу, якщо перенос із поля молодших чотирьох бітів розповсюджується через всі старші біти, а в протилежному випадку - не змінює прапорець переносу. Якщо після цього прапорець переносу дорівнює «1», або якщо значення чотирьох старших бітів (7-4) перевищує 9 (1010XXXX - 1111XXXX), значення цих бітів збільшується на 6, створюючи відповідну двійково-десяткову цифру в старшому півбайті. При цьому прапорець переносу встановлюється, якщо перенос одержується із старших бітів, і не змінюється в протилежному випадку. Таким чином, прапорець переносу вказує на те, що отримана сума двох двійково-десяткових змінних більше ніж 100. Ця команда виконує десяткове перетворення за допомогою додавання 06, 60, 66 з вмістом акумулятора в залежності від початкового стану акумулятора і слова стану програми (PSW).

Асемблер: DA A

Код: 1 1 0 1 0 1 0 0

Час: 1 цикл

Алгоритм: якщо ((A[3-0]) > 9 або (AC)=1), то A[3-0]:=
=A[3-0]+6
якщо ((A[7-4]) > 9 або (C)=1), то A[7-4]:=A[7-4]+6

Приклад: а) ;(A)=56H, (R3)=67H, (C)=1
ADDC A, R3 ;(A)=30H, (C)=0
б) ;(A)=30H, (C)=0
ADD A, #99H
DA A ;(A)=29, (C)=1

Примітка: Команда DA A не може просто перетворювати шістнадцятирозрядне значення в акумуляторі в двійково-десяткове представлення і не застосовується, наприклад, для десяткового віднімання.

Команда DEC (байт)

Команда *декремент* виконує віднімання «1» із вказаного операнда. Початкове значення 00H перейде в 0FFH. Команда

DEC не впливає на прапорці. Цією командою допускається чотири режими адресації операнда:

1) до акумулятора
2) регістровий
3) прямий
4) непрямо-регістровий.

1) Асемблер: DEC A

Код: | 0 0 0 1 0 1 0 0 |

Час: 1 цикл

Алгоритм: (A):=(A)-1

Приклад: ;(A)=11H, (C)=1, (AC)=1
DEC A ;(A)=10H, (C)=1, (AC)=1

2) Асемблер: DEC Rn ; де n=0-7

Код: | 0 0 0 1 1 rrr |, де rrr=000-111

Час: 1 цикл

Алгоритм: (Rn):=(Rn)-1

Приклад: ;(R1)=7FH,
;(ОЗП[7F])=40H, (ОЗП[7F])=00H
DEC @R1
DEC R1
DEC @R1 ;(R1)=7EH,
;(ОЗП[7F])=3FH, (ОЗП[7F])=FFH

3) Асемблер: DEC (direct)

Код: | 0 0 0 1 0 1 0 1 | | direct address |

Час: 1 цикл

Алгоритм: (direct):=(direct)-1

Приклад: ;(SCON)=A0H, (C)=1, (AC)=1
DEC SCON ;(SCON)=9FH, (C)=1, (AC)=1

4) Асемблер: DEC @Ri ; де i=0,1

Код: | 0 0 0 1 0 1 1 i | ; де i = 0,1

Час: 1 цикл

Алгоритм: ((Ri)):=((Ri)-1)

Приклад: ;(R1)=7FH,
;(ОЗП[7F])=40H, (ОЗП[7F])=00H
DEC @R1
DEC R1
DEC @R1 ;(R1)=7EH,
;(ОЗП[7F])=3FH, (ОЗП[7F])=FFH

Примітка: Якщо ця команда використовується для зміни інформації на виході порту, значення, яке використовується як вихідні дані, зчитується із «фіксатора» порту, а не з виводів ВІС.

Команда DIV AB

Команда *ділення* ділить 8-бітове ціле число без знака із акумулятора А на 8-бітове ціле число без знака в регістрі А. Акумулятору присвоюється ціла частина частки (старші розряди), а регістрові В - залишок. Прапорці переносу (С) і переповнення (OV) скидаються. Якщо (А) < (В), то прапорець додаткового переносу (АС) не скидається. Прапорець переносу скидається в будь-якому випадку.

Асемблер: DIV AB

Код: | 1 0 0 0 0 1 0 0 |

Час: 4 цикли

Алгоритм: (А):=((А)/(В)) [15-8],
(В):=((А)/(В)) [7-0]

Приклад: Нехай акумулятор вміщує число 251 (0FBH або 11111011B), а регістр B - число 18 (12H або 00010010B). Після виконання команди

DIV AB

в акумуляторі буде число 13 (0DH або 000011011B) а в регістрі B - число 17 (11H або 00010001B), так як 251=(13*18)+17. Прапорці C і OV будуть скинуті.

Примітка: Якщо B вміщує 00, то після команди DIV вміст акумулятора A і регістра B не будуть визначені. Прапорець переносу скидається, а прапорець переповнення встановлюється в «1».

Команда DJNZ (байт), (зміщення)

Команда *декремент і перехід, якщо не дорівнює нулю* виконує віднімання «1» із вказаної комірки і здійснює розгалуження за вирахуваною адресою, якщо результат не рівний нулеві. Початкове значення 00H перейде в 0FFH. Адреса переходу (розгалуження) вираховується додаванням значення зміщення (із знаком), вказаного в останньому байті команди, з вмістом лічильника команд, збільшеним на довжину команди DJNZ. На прапорці ця команда не впливає і допускає наступні режими адресації:

1) регістровий
2) прямий.

1) Асемблер: DJNZ Rn,(мітка); де n=0-7

Код: | 1 1 0 1 1 rrr | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+2,
(Rn):=(Rn)-1,
якщо ((Rn)>0 або (Rn)<0), то (PC):=(PC)+(rel8)

Приклад: ;(R2)=08H, (P1)=FFFH (11111111B)
LAB4: CPL P1.7
DJNZ R2, LAB4 ;(R2)={07-00}
;Ця послідовність команд переключає P1.7
;вісім разів і приводить до появи чотирьох імпульсів

;на виводі BIC, відповідному біту P1.7.

2) Асемблер: DJNZ (direct), (мітка)

Код: | 1 1 0 1 0 1 0 1 | direct address | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
(direct):=(direct)-1,
якщо ((direct)>0 або (direct)<0), то
(PC):=(PC)+(rel8)

Приклад: ;(ОЗП[40])=01H, (ОЗП[50])=80H,
;(ОЗП[60])=25H
DJNZ 40H, LAB1 ;(ОЗП[40]):=00H
DJNZ 50H, LAB2 ;(ОЗП[50]):=7FH
DJNZ 40H, LAB3 ;(ОЗП[40]):=25H
...
LAB1: CLR A
...
LAB2: DEC R1 ;виконався перехід на мітку LAB2

Примітка: Якщо команда DJNZ використовується для зміни виходу порту, значення, використане як операнд, зчитується із «фіксатора» порту, а не з виводів BIC.

Команда INC (байт)

Команда *інкремент* виконує додавання «1» до вказаної змінної і не впливає на прапорці. Початкове значення 0FFH перейде в 00H. Ця команда допускає чотири режими адресації:

1) до акумулятора
2) регістровий
3) прямий
4) непрямо-регістровий.

1) Асемблер: INC A

Код: | 0 0 0 0 0 1 0 0 |

Час: 1 цикл

Алгоритм: (A):=(A)+1

Приклад: ;(A)=1FH, (AC)=1
INC A ;(A)=20H, (AC)=0

2) Асемблер: INC Rn ; де n=0-7

Код: | 0 0 0 1 1 rrr | , де rrr=000-111

Час: 1 цикл

Алгоритм: (Rn):=(Rn)+1

Приклад: ;(R4)=FFH, (C)=0, (AC)=0
INC R4 ;(R4)=00H, (C)=0, (AC)=0

3) Асемблер: INC (direct)

Код: | 0 0 0 0 0 1 0 1 | | direct address |

Час: 1 цикл

Алгоритм: (direct):=(direct)+1

Приклад: ;(ОЗП[43])=22H
INC 43H ;(ОЗП[43])=23H

4) Асемблер: INC @Ri ; де i=0,1

Код: | 0 0 0 0 0 1 1 i | , де i=0,1

Час: 1 цикл

Алгоритм: ((Ri)):=((Ri)+1)

Приклад: ;(R1)=41H, ;(ОЗП [41])=4FH, (AC)=0
INC @R1 ;(R1)=41H, ;(ОЗП [41])=50H, (AC)=0

Примітка: При використанні команди INC для зміни вмісту

порту, величина, використана як операнд, зчитується із «фіксатора» порту, а не з виводів ВІС.

Команда INC DPTR

Команда *інкремент вказівника даних* виконує інкремент (додавання «1») вмісту 16-бітового вказівника даних (DPTR). Додавання «1» здійснюється до 16 бітів, причому переповнення молодшого байта вказівника даних (DPL) із FFH в 00H призводить до інкременту старшого байта вказівника даних (DPH). На прапорці ця команда не впливає.

Асемблер: INC DPTR

Код: | 1 0 1 0 0 0 1 1 |

Час: 2 цикли

Алгоритм: (DPTR):=(DPTR)+1

Приклад: ;(DPH)=12H, (DPL)=FEH
INC DPTR
INC DPTR
INC DPTR ;(DPH)=13H, (DPL)=01H

Команда JB (bit), (rel8)

Команда *перехід, якщо біт встановлений* виконує перехід за адресою розгалуження, якщо вказаний біт дорівнює «1», в протилежному випадку виконується наступна команда. Адреса розгалуження розраховується за допомогою додавання відносного зміщення із знаком у третьому байті команди (rel8) до вмісту лічильника команд після додавання до нього 3. Перевірений біт не змінюється. Ця команда на прапорці не впливає.

Асемблер: JB (bit),(мітка)

Код: | 0 0 1 0 0 0 0 0 | bit address | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
якщо (bit)=1, то (PC):=(PC)+(rel8)

Приклад: ;(A)=96H (10010110B)
JB ACC.2,LAB8 ;ця команда забезпечує перехід
;на мітку LAB8
...
LAB8: INC DPTR

Команда JBC (bit), (rel8)

Команда *перехід, якщо біт встановлений, і скидання цього біта*, виконує розгалуження за розрахованою адресою, якщо біт дорівнює «1». В протилежному випадку виконується наступна за JBC команда. В будь-якому випадку вказаний біт скидається. Адреса переходу вираховується додаванням відносного зміщення із знаком у третьому байті команди (rel8) і вмісту лічильника команд після додавання до нього 3. Ця команда не впливає на прапорці.

Асемблер: JBC (bit),(мітка)

Код: | 0 0 1 0 0 0 0 0 | bit address | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
якщо (bit)=1, то (bit):=0, (PC):=(PC)+(rel8)

Приклад: ;(A)=76H (01110110B)
JBC ACC.3,LAB6 ;Переходу на LAB6 немає, бо
;(A[3])=0
JBC ACC.2,LAB7 ;(A)=72H (01110010B) і перехід за
;адресою, відповідною мітці LAB7.

Примітка: Якщо ця команда використовується для перевірки бітів порту, то значення, використане як операнд, зчитується із «фіксатора» порту, а не з виводів BIC.

Команда JC (rel8)

Команда *перехід, якщо перенос встановлений,* виконує розгалуження за адресою, якщо прапорець переносу дорівнює «1», в протилежному випадку виконується наступна команда. Адреса розгалуження вираховується за допомогою додавання відносного зміщення із знаком у другому байті команди (rel8) і вмісту лічильника команд, після додавання до нього 2. Ця команда на прапорці не впливає.

Асемблер: JC (мітка)

Код: | 0 0 0 1 0 0 0 0 | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+2
якщо (C)=1, то (PC):=(PC)+(rel8)

Приклад: ;(C)=0
JC LAB8 ;немає переходу не мітку LAB8
CPL C ;(C):=1
LAB8: JC LAB9 ;перехід на мітку LAB9, тому що (C)=1
...
LAB9: NOP

Команда JMP @A+DPTR

Команда *непрямий перехід* додає 8-бітовий вміст акумулятора без знака з 16-бітовим вказівником даних (DPTR) і завантажує отриманий результат в лічильник команд, вміст якого є адресою для виборки наступної команди. 16-бітове додавання виконується за модулем 2^{16}, перенос із молодших восьми бітів розповсюджується на старші біти програмного лічильника. Вміст акумулятора і вказівника даних не змінюється. Ця команда на прапорці не впливає.

Асемблер: JMP @A+DPTR

Код: | 0 1 1 1 0 0 1 1 |

Час: 2 цикли

Алгоритм: (PC):=(A)[7-0]+(DPTR)[15-0]

Приклад: ;(PC)=034EH, (A)=86H, (DPTR)=0329H
JMP @A+DPTR ;(PC)=03AFH, (A)=86H, (DPTR)=0329H

Команда JNB (bit),(rel8)

Команда *перехід, якщо біт не встановлений* виконує розгалуження за адресою, якщо вказаний біт дорівнює нулеві, в протилежному випадку виконується наступна команда. Адреса розгалуження розраховується за допомогою додавання відносного зміщення із знаком у третьому байті команди (rel8) і вмісту лічильника команд після додавання до нього 3. Перевірений біт не змінюється. Ця команада на прапорці не впливає.

Асемблер: JNB (bit),(мітка)

Код: | 0 0 1 1 0 0 0 0 | bit address | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
якщо (bit)=0, то (PC):=(PC)+(rel8)

Приклад: ;(P2)=CAH (11001010B)
;(A)=56H (01010110B)
JNB P1.3,LAB10 ;немає переходу LAB10
JNB ACC.3,LAB11 ;перехід на мітку LAB11
...
LAB11: INC A

Команда JNC (rel8)

Команада *перехід, якщо перенос не встановлений* виконує розгалуження за адресою, якщо прапорець переносу дорівнює нулеві, в протилежному випадку виконується наступна команда. Адреса розгалуження розраховується за допомогою додавання відносного зміщення зі знаком у другому байті команди (rel8) і вмісту лічильника команд після додавання до нього 2. Прапорець переносу не змінюється. Ця команда на інші прапорці не впливає.

Асемблер: JNC (мітка)

Код: | 0 1 0 1 0 0 0 0 | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+2
якщо (C)=0, то (PC):=(PC)+(rel8)

Приклад: ;(C)=1
JNC LAB12 ;немає переходу LAB12
CPL C
LAB12: JNC LAB13 ;перехід на мітку LAB13

Команда JNZ (rel8)

Команда *перехід, якщо вміст акумулятора не дорівнює нулеві* виконує розгалуження за адресою, якщо хоча б один біт акумулятора дорівнює «1», в протилежному випадку виконується наступна команда. Адреса розгалуження розраховується додаванням відносного зміщення зі знаком у другому байті команди (rel8) і вмісту лічильника команд (PC) після додавання до нього 2. Вміст акумулятора не змінюється. Ця команда на прапорці не впливає.

Асемблер: JNZ (мітка)

Код: | 0 1 1 1 0 0 0 0 | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+2
якщо (A)=0, то (PC):=(PC)+(rel8)

Приклад: ;(A)=00H
JNC LAB14 ;немає переходу LAB14
INC A
LAB14: JNZ LAB15 ; перехід на мітку LAB15
...
LAB15: NOP

Команда JZ (rel8)

Команда *перехід, якщо вміст акумулятора дорівнює «0»* виконує розгалуження за адресою, якщо всі біти акумулятора дорівнюють «0», в протилежному випадку виконується наступна команда. Адреса розгалуження розраховується додаванням відносного зміщення зі знаком у другому байті команди (rel8) і вміст лічильника команд після додавання до нього 2. Вміст акумулятора не змінюється. Ця команда на прапорці не впливає.

Асемблер: JZ (мітка)

Код: | 0 1 1 0 0 0 0 0 | rel8 |

Час: 2 цикли

Алгоритм: (PC):=(PC)+2
якщо (A)=0, то (PC):=(PC)+(rel8)

Приклад: ;(A)=01H
JZ LAB16 ;немає переходу на LAB16
DEC A
LAB16: JZ LAB17 ; перехід на мітку LAB17
...
LAB17: CLR A

Команда LCALL (addr16)

Команда *довгий виклик* викликає підпрограму, яка знаходиться за вказаною адресою. По команді LCALL до лічильника команд (PC) додається 3 для отримання адреси наступної команди і після цього отриманий 16-бітовий результат заноситься в стек (спочатку йде молодший байт, за ним - старший), а вміст вказівника стека (SP) збільшується на 2. Потім старший і молодший байти лічильника команд завантажуються відповідно другим і третім байтами команди LCALL. Виконання програми продовжується командою, яка знаходиться за отриманою адресою. Підпрограма, відповідно, може починатися в будь-якому місці адресного простору пам'яті програм ємністю до 64 Кбайтів. Ця команда на прапорці не впливає.

Асемблер: LCALL (мітка)

Код: | 0 0 0 1 0 0 1 0 | addr[15-8] | addr[7-0] |

Час: 2 цикли

Алгоритм: (PC):=(PC)+3
(SP):=(SP)+1
((SP)):=(PC[7-0])
(SP):=(SP)+1
((SP)):=(PC[15-8])
(PC):=(addr[15-0])

Приклад: ;(SP)=07H,
;мітці PRN відповідає адреса 1234H,
;за адресою 0126H знаходиться команда
;LCALL
LCALL PRN ;(SP)=09H, (PC)=1234H,
;(ОЗП[08])=26H, (ОЗП[09])=01H

Команда LJMP (add16)

Команда *довгий перехід* виконує безумовний перехід за вказаною адресою, завантажуючи старший і молодший байти лічильника команд (PC) відповідно другим і третім байтами, які знаходяться у коді команди. Адреса переходу, таким чином, може знаходитися за будь-якою адресою простору пам'яті програм в 64 Кбайти. Ця команда на прапорці не впливає.

Асемблер: LJMP (мітка)

Код: | 0 0 0 0 0 0 1 0 | addr[15-8] | addr[7-0] |

Час: 2 цикли

Алгоритм: (PC):=(addr[15-0])
Приклад: ; мітці M1 відповідає адреса 1234H,
; за адресою 0126H знаходиться команда LJMP
LJMP M1; (PC)=1234H

Команда MOV (байт-призначення),(байт-джерело)

Команда *переслати змінну-байтів* пересилає змінну-байтів, вказану у другому операнді, в комірку, вказану в першому операнді. Вміст байта джерела не змінюється. Ця команда на прапорці переносу і інші регістри не впливає. Команда «MOV» допускає 15 комбінацій адресації байта-джерела і байта-призначення.

1) Асемблер: MOV A,Rn ; де n=0-7

Код: | 1 1 1 0 1 rrr |, де rrr=000-111

Час: 1 цикл

Алгоритм: (A):=(Rn)

Приклад: ;(A)=FAH, (R4)=93H
MOV A,R4 ;(A)=93H, (R4)=93H

2) Асемблер: MOV A,(direct)

Код: | 1 1 1 0 0 1 0 1 | | direct adderss |

Час: 1 цикл

Алгоритм: (A):=(direct)

Приклад: ;(A)=93H, (ОЗП[40])=10H, (R0)=40H
MOV A,40H ;(A)=10H, (ОЗП[40])=10H, (R0)=40H

3) Асемблер: MOV A,@Ri ; де i=0,1

Код: | 1 1 1 0 0 1 1 i | , де i = 0,1

Час: 1 цикл

Алгоритм: (A):=((Ri))

Приклад: ;(A)=10H, (R0)=41H, (ОЗП[41])=0CAH
MOV A,@R0 ;(A)=CAH, (R0)=41H, (ОЗП[41])=0CAH

4) Асемблер: MOV A,#data

Код: | 0 1 1 1 0 1 0 0 | | #data8 |

Час: 1 цикл

Алгоритм: (A):=(#data8)

Приклад: ;(A)=C9H (11001001B)
MOV A,#37H ;(A)=37H (00110111B)

5) Асемблер: MOV Rn,A ; де n=0-7

Код: | 1 1 1 1 1 rrr | , де rrr=000-111

Час: 1 цикл

Алгоритм: (Rn):=(A)

Приклад: ;(A)=38H, (R0)=42H
MOV R0,A ;(A)=38H, (R0)=38H

6) Асемблер: MOV Rn,(direct) ; де n=0-7

Код: | 1 0 1 0 1 rrr | | direct address |, де rrr = 000-111

Час: 2 цикли

Алгоритм: (Rn):=(direct)

Приклад: ;(R0)=39H, (P2)=0F2H
MOV R0,P2 ;(R0)=F2H

7) Асемблер: MOV Rn,#data ; де n=0-7

Код: | 0 1 1 1 1 rrr | | #data8 | , де rrr = 000-111

Час: 1 цикл

Алгоритм: (Rn):=(#data8)

Приклад: ;(R0)=0F5H
MOV R0,#49H ;(R0)=49H

8) Асемблер: MOV (direct),A

Код: | 1 1 1 1 0 1 0 1 | direct address |

Час: 1 цикл

Алгоритм: (direct):=(A)

Приклад: ;(P0)=FFH, (A)=4BH
MOV P0,A ;(P0)=4BH, (A)=4BH

9) Асемблер: MOV (direct),Rn ; де n=0-7

Код: | 1 0 0 0 1 rrr | direct address |, де rrr = 000-111

Час: 2 цикли

Алгоритм: (direct):=(Rn)

Приклад: ;(PSW)=C2H, (R7)=57H
MOV PSW,R7 ;(PSW)=57H, (R7)=57H

10) Асемблер: MOV (direct),(direct)

Код: | 1 0 0 0 0 1 0 1 | direct address | direct address |

Час: 2 цикли

Алгоритм: (direct):=(direct)

Приклад: ;(ОЗП[45])=33H, (ОЗП[48])=0DEH
MOV 48H,45H ;(ОЗП[45])=33H, (ОЗП[48])=33H

11) Асемблер: MOV (direct),@Ri ; де i=0,1

Код: | 1 0 0 0 0 1 1 i | direct address |, де i = 0,1

Час: 2 цикли

Алгоритм: (direct):=((Ri))

Приклад: ;(R1)+49H, (ОЗП[49])=0E3H
MOV 51H,@R1 ;(ОЗП[51])=0E3H, (ОЗП[49])=0E3H

12) Асемблер: MOV (direct),#data

Код: | 0 1 1 1 0 1 0 1 | direct address | #data8 |

Час: 2 цикли

Алгоритм: (direct):=(#data8)

Приклад: ;(ОЗП[5F])=9BH
MOV 5FH,#07H ;(ОЗП[5F])=07H

13) Асемблер: MOV @Ri,A ; де i=0,1

Код: | 1 1 1 1 0 1 1 i | , де i=0,1

Час: 1 цикл

Алгоритм: ((Ri)):=(A)

Приклад: ;(R1)=48H, (ОЗП[48])=75H, (A)-0BDH
MOV @R1,A ;(ОЗП[48])=0BDH

14) Асемблер: MOV @Ri,(direct) ; де i=0,1

Код: | 1 0 1 0 0 1 1 i | direct address | , де i = 0,1

Час: 2 цикли

Алгоритм: ((Ri)):=(direct)

Приклад: ;(R0)=51H, (ОЗП[51])=0E3H, (P0)=0ACH
MOV @R0,P0 ;(ОЗП[51])=0ACH

15) Асемблер: MOV@Ri,#data ; де i=0,1

Код: | 0 1 1 1 0 1 1 i | , де i=0,1

Час: 1 цикл

Алгоритм: ((Ri)):=(#data8)

Приклад: ;(ОЗП[7E])=67H, (R1)7EH
MOV @R1,#0A9H ;(ОЗП[7E])=0A9H, (R1)=7EH

Команда MOV (біт призначення),(біт джерела)

Команда *переслати біт даних* бітову змінну, вказану у другому байті, копіює в розряд, який вказаний у першому операнді. Одним із операндів повинен бути прапорець переносу C, а другим може бути будь-який із бітів, до якого можлива адресація.

1) Асемблер: MOV C,(bit)

Код: | 1 0 1 0 0 0 1 0 | bit address |

Час: 1 цикл

Алгоритм: (C):=(bit)

Приклад: ;(C)=0, (P3)=D5H (11010101B)
MOV C,P3.0 ;C:=1
MOV C,P3.3 ;C:=0
MOV C,P3.7 ;C:=1

2) Асемблер: MOV (bit),C

Код: | 1 0 0 1 0 0 1 0 | bit address |

Час: 2 цикли

Алгоритм: (bit):=(C)

Приклад: ;(C)=1, (P0)=20H (00100000B)
MOV P0.1,C
MOV P0.2,C
MOV P0.3,C ;(C)=1, (P0)=2EH (00101110B)

Команда MOV DPTR,#data16

Команда *завантажити вказівник даних 16-бітовою константою* завантажує вказівник даних DPTR 16-бітовою константою, вказаною у другому і третьому байтах команди. Другий байт команди завантажується в старший байт вказівника даних (DPH), а третій байт - в молодший байт вказівника даних (DPL). Ця команда на прапорці не впливає і є єдиною командою, яка одночасно завантажує 16 бітів даних.

Асемблер: MOV DPTR,#(data16)

Код: | 1 0 0 1 0 0 0 0 | #data[15-8] | #data[7-0] |

Час: 2 цикли

Алгоритм: (DPTR):=#data[15-0], DPL:=#data[7-0]

Приклад: ;(DPTR)=01FDH
MOV DPTR,#1234H ;(DPTR)=1234H,
;(DPH)=12H, (DPL)=34H

Команда MOVC A,@A+((R16))

(R16) - 16-розрядний регістр.

Команда *переслати байт із пам'яті програм* завантажує акумулятор байтом коду або константою із пам'яті програми. Адреса зчитаного байта розраховується як сума 8-бітового вмісту акумулятора без знака і вмісту 16-бітового регістра. В ролі 16-бітового регістра може бути:

1) вказівник даних DPTR
2) лічильник команд PC.

У випадку, коли використовується PC, його вміст збільшується до значення адреси наступної команди перед тим, як його вміст додається з вмістом акумулятора. 16-бітове додавання виконується так, що перенос із молодших восьми бітів може розповсюджуватися через старші біти. Ця команда на прапорці не впливає.

1) Асемблер: MOVC A,@A+DPTR

Код: 1 0 0 1 0 0 1 1

Час: 2 цикли

Алгоритм: (A):=((A)+(DPTR))

Приклад: ;(A)=1BH, (DPTR)=1020H,
;(ОЗП[103B])+48H,
MOVC A,@A+DPTR ;(A)=48H, (DPTR)+1020H

2) Асемблер: MOVC A,@A+PC

Код: 1 0 0 0 0 0 1 1

Час: 2 цикли

Алгоритм: (A):=((A)+(PC))

Приклад: ;(A)=FAH, (PC)=0289,
;(ОЗП[0384])=9BH MOVC A,@A+PC
;(A)=9BH, (PC)=028AH

Команда MOVX (байт приймача),(байт джерела)

Команда *переслати у внутрішню пам'ять (із зовнішньої пам'яті) даних* пересилає дані між акумулятором і байтом зовнішньої пам'яті даних. Є в наявності два типи команд, які відрізняються тим, що забезпечують 8-бітову або 16-бітову непряму адресу до зовнішнього ОЗП даних.

У першому випадку вміст R0 або R1 в біжучому банку регістрів забезпечує 8-бітову адресу, яка мультиплексується з порту P0. Для розширення дешифрації вводу/виводу або адресації невеликого масиву ОЗП достатньо восьми бітів адресації. Якщо застосовуються ОЗП з ємністю, не набагато більшою ніж 256 байтів, то для фіксації старших бітів адреси можна використовувати будь-які інші виходи портів, які переключаються командою, що передує команді MOVX.

У другому випадку при виконанні команди MOVX вказівник даних DPTR ґенерує 16-бітову адресу. Порт P2 виводить старші вісім бітів адреси (DPH), а порт P0 мультиплексує молодші 8 бітів адреси (DPL) з даними. Ця форма є ефективною при доступі

до великих масивів даних (до 64 Кбайтів), тому що для встановлення портів виводу не вимагається використання додаткових команд.

1) Асемблер: MOVX A,@Ri ; де i=0,1

Код: | 1 1 1 0 0 0 1 i | ; де i = 0,1

Час: 2 цикли

Алгоритм: (A):=((Ri))

Приклад: ;(A)=32H, (R0)=83H, комірка зовнішнього
;ОЗП за адресою 83H містить B6H
MOVX A,@R0 :(A)=B6H, (R0)=83H

2) Асемблер: MOVX A,@DPTR

Код: | 1 1 1 0 0 0 0 0 |

Час: 2 цикли

Алгоритм: (A):=((DPTR))

Приклад: ;(A)=5CH, (DPTR)=1ABEH,
;комірка зовнішнього ОЗП за адресою
;1ABEH містить 72H
MOVX A,@DPTR ;(A)=72H, (DPTR)=2ABEH

3) Асемблер: MOVX @Ri,A ; де i=0,1

Код: | 1 1 1 1 0 0 1 i | ; де i = 0,1

Час: 2 цикли

Алгоритм: ((Ri)):=(A)

Приклад: ;(A)=95H, (R1)=FDH,
;комірка зовнішнього ОЗП за адресою
;FDH містить 00
MOVX @R1,A ;(A)=95H, (R1)=FDH,

;комірка зовнішнього ОЗП за адресою
;FDH вміщає 95H

4) Асемблер: MOVX @DPTR,A

Код: 1 1 1 1 0 0 0 0

Час: 2 цикли

Алгоритм: ((DPTR)):=(A)

Приклад: ;(A)=97H, (DPTR)=1FFFH,
;комірка зовнішнього ОЗП за адресою
;1FFFH містить 00
MOVX @DPTR,A ;(A)=97H, (DPTR)=1FFFH
;комірка зовнішнього ОЗП за адресою
;1FFFH містить 97H

Команда MUL AB

Команда *множення* перемножує 8-бітові цілі числа без знака із акумулятора і регістра B. Старший байт 16-бітового добутку розміщується в регістр B, а молодший в акумулятор A. Якщо результат добутку більший, ніж 0FFH(255), то встановлюється прапорець переповнення (OV), в протилежному випадку він скидається. Прапорець переносу завжди скидається.

Асемблер: MUL AB

Код: 1 0 1 0 0 1 0 0

Час: 4 цикли

Алгоритм: (A)[7-0]=(A)*(B),
(B)[15-8]=(A)*(B)

Приклад: а) ;(A)=50H (50H=80 DEC), (C)=1,
;(B)=0A0H (A0H=160 DEC), (OV)=0
MUL AB ;(A)=00H, (B)=32H, (C)=0, (OV)=1
б) ;(A)=2HH, (OV)=1, (B)=06H, (C)=1
MUL AB ;(A)=0D8H, (B)=00H, (OV)=0, (C)=0

Команда NOP

Команда *немає операції* виконує холостий хід і не впливає на регістри і прапорці, крім лічильника команд (PC).

Асемблер: NOP

Код: 0 0 0 0 0 0 0 0

Час: 1 цикл

Алгоритм: (PC):= (PC)+1

Приклад: Нехай необхідно створити від'ємний вихідний імпульс на порті P1[6] тривалістю 3 цикли. Це виконує наступна послідовність команд:

```
CLR P1.6 ;P1[6]:=0
NOP
NOP
NOP
SETB P1.6 ;P1[6]:=1
```

Команда ORL (байт призначення),(байт джерела)

Команда *логічне «АБО» для змінних-байтів* виконує операцію логічного «АБО» над бітами вказаних змінних, записуючи результат в байт призначення. Ця команда на прапорці не впливає. Допускається шість комбінацій режимів адресації:

- якщо байтом призначення є акумулятор:

1) регістровий
2) прямий
3) непрямо-регістровий
4) безпосередній

- якщо байтом призначення є пряма адреса:

5) до акумулятора
6) до константи

1) Асемблер: ORL A,Rn ; де n=0-7

Код: 0 1 0 0 1 rrr , де rrr=000-111

Час: 1 цикл

Алгоритм: (A):=(A) OR (Rn), де OR - операція логічного «АБО»

Приклад: ;(A)=15H, (R5)=6CH
ORL A,R5 ;(A)=7DH, (R5)=6CH

2) Асемблер: ORL A,(direct)

Код: | 0 1 0 0 0 1 0 1 | direct adderss |

Час: 1 цикл

Алгоритм: (A):=(A) OR (direct)

Приклад: ;(A)=84H, (PSW)=C2H
ORL A,PSW ;(A)=C6H, (PSW)=C2H

3) Асемблер: ORL A,@Ri ; де i=0,1

Код: | 0 1 0 0 0 1 1 i | , де i = 0,1

Час: 1 цикл

Алгоритм: (A):=(A) OR ((Ri))

Приклад: ;(A)=52H, (R0)=6DH, (ОЗП[6D])=49H
ORL A,@R0 ;(A)=5BH, (ОЗП[6D])=49H

4) Асемблер: ORL A,#(data)

Код: | 0 1 0 0 0 1 0 0 | #data8 |

Час: 1 цикл

Алгоритм: (A):=(A) OR #(data)

Приклад: ;(A)=F0H
ORL A,#0AH ;(A)=FAH

5) Асемблер: ORL (direct),A

Код: | 0 1 0 0 0 0 1 0 | direct address |

Час: 1 цикл

Алгоритм: (direct):=(direct) OR (A)

Приклад: ;(A)=34H, (IP)=23H
ORL IP,A ;(IP)=37H, (A)=34H

6) Асемблер: ORL (direct),#(data)

Код: | 0 1 0 0 0 0 1 1 | direct address | #data8 |

Час: 2 цикли

Алгоритм: (direct):=(direct) OR #(data)

Приклад: ;(P1)=00H
ORL P1,#0C4H ;(P1)=11000100B (C4H)

Примітка: Якщо команда використовується для роботи з портом, величина, яка використовується в ролі вихідних даних порту, зчитується із «фіксатора» порту, а не з виводів ВІС.

Команда ORL C,(біт джерела)

Команда *логічне «АБО» для змінних-бітів* встановлює прапорець переносу С, якщо буліва величина дорівнює логічній «1», в протилежному випадку встановлює прапорець С в «0». Косий дріб («/») перед операндом на мові асемблера вказує на те, що в ролі операнда використовується логічне заперечення значення адресованого біта, але сам біт джерела не змінюється. Ця команда на інші прапорці не впливає.

1) Асемблер: ORL C,(bit)

Код: | 0 1 1 1 0 0 1 0 | bit address |

Час: 2 цикли

Алгоритм: (C):=(C) OR (bit)

Приклад: ;(C)=0, (P1)=53H (01010011B)
ORL C,P1.4 ;(C)=1, (P1)=53H (01010011B)

2) Асемблер: ORL C,/(bit)

Код: | 1 0 1 0 0 0 0 0 | bit address |

Час: 2 цикли

Алгоритм: (C):=(C) OR /(bit)

Приклад: ;(C)=0, (ОЗП[25])=39H (00111001B)
ORL C,/2A ;(C)=1, (ОЗП[25])=39H (00111001B)

Команда POP (direct)

Команда *зчитування з стека* зчитує вміст комірки, яка адресується за допомогою вказівника стека в прямоадресовану комірку ОЗП, при цьому вказівник стека зменшується на одиницю.

Ця команда не діє на прапорці і часто використовується для зчитування із стека проміжних даних.

Асемблер: POP (direct)

Код: | 1 1 0 1 0 0 0 0 | bit address |

Час: 2 цикли

Алгоритм: (direct):=((SP)), (SP):=(SP)-1

Приклад: ;(SP)=32H, (DPH)=01, (DPL)=ABH,
;(ОЗП[32])=12H, (ОЗП[31])=56H,
;(ОЗП[30])=20H
POP DPH
POP DPL ;(SP)=30H, (DPH)=12H, (DPL)=56H,
;(ОЗП[32])=12H, (ОЗП[31])=56H,
;(SP)=20H, (ОЗП[30])=20H

Команда PUSH (direct)

Команда *запис в стек* збільшує вказівник стека на одиницю і після цього вміст вказаної прямоадресованої змінної копіюється в комірку внутрішнього ОЗП, який адресується за допомогою вказівника стека. На прапорці ця команда не впливає і використовується для запису проміжних даних в стек.

Асемблер: PUSH (direct)

Код: | 1 1 0 0 0 0 0 0 | direct address |

Час: 2 цикли

Алгоритм: (SP):=(SP)+1 ((SP)):=(direct)

Приклад: ;(SP)=09H, (DPTR)=1279H
PUSH DPL
PUSH DPH ;(SP)=0BH, (DPTR)=1279H,
;(ОЗП[0A])=79H, (ОЗП[0B])=12H

Команда RET

Команда *повернення із підпрограми* послідовно вивантажує старший і молодший байти лічильника команд із стека, зменшуючи вказівник стека на 2. Виконання основної програми звичайно продовжується за адресою команди, яка йде за ACALL або LCALL. На прапорці ця команда не впливає.

Асемблер: RET

Код: | 0 0 1 0 0 0 1 0 |

Час: 2 цикли

Алгоритм: (PC)[15-8]:=((SP)),
(SP):=(SP)-1,
(PC)[7-0]:=((SP)),
(SP):=(SP)-1

Приклад: ;(SP)=0DH (ОЗП[0C])=93H, (ОЗП[0D])=02H

RET ;(SP)=0BH, (PC)=0293H

Команда RETI

Команда *повернення із переривання* вивантажує старший і молодший байти лічильника команд із стека і встановлює «логіку переривань», дозволяючи прийом інших переривань з рівнем пріоритету, який дорівнює рівню пріоритету щойно обробленого переривання. Вказівник стека зменшується на 2. Слово стану програми (PSW) не відновлюється автоматично. Виконання основної програми продовжується з команди, що йде за командою, на якій виник перехід до виявлення запиту на переривання. Якщо при виконанні команди RETI виявлено переривання з таким же або меншим рівнем пріоритету, то така одна команда основної підпрограми встигає виконатися до обробки такого переривання.

Асемблер: RETI

Код: 0 0 1 1 0 0 1 0

Час: 2 цикли

Алгоритм: (PC)[15-8]:=((SP)),
(SP):=(SP)-1,
(PC)[7-0]:=((SP)),
(SP):=(SP)-1

Приклад: ;(SP)=0BH (ОЗП[0A])=2AH, (ОЗП[0B])=03H
;(PC)=YYYYH, де Y=0-FH
RETI ;(SP)=09H, (PC)=032AH

Команда RL A

Команда *зсув вмісту акумулятора вліво* зсуває вісім бітів акумулятора на один біт вліво, біт 7 засилається на місце біта 0. На прапорці команда не впливає.

Асемблер: RL A

Код: 0 0 1 0 0 0 1 1

Час: 1 цикл

Алгоритм: (A[N+1]):=(A[N]), де N=0-6
(A[0]):=(A[7])

Приклад: ;(A)=0D5H (11010101B), (C)=0
RL A ;(A)=0ABH (10101011B), (C)=0

Команда RLC A

Команда *зсув вмісту акумулятора вліво через прапорець переносу* зсуває вісім біт акумулятора і прапорець переносу вліво на один біт. Вміст прапорця переносу переміщується на місце біта 0 акумулятора, а вміст біта 7 акумулятора переписується в прапорець переносу. На прапорці ця команда на впливає.

Асемблер: RLC A

Код: | 0 0 1 1 0 0 1 1 |

Час: 1 цикл

Алгоритм: (A[N+1]):=(A[N]), де N=0-6
(A[0]):=(C)
(C):=(A[7])

Приклад: ;(A)=56H (01010110B), (C)=1
RLC A ;(A)=0ADH (10101101B), (C)=0

Команда RR A

Команда *зсув вмісту акумулятора вправо* зсуває вправо на один біт всі вісім бітів акумулятора. Вміст біта 0 переміщується на місце біта 7. На прапорці ця команда не впливає.

Асемблер: RR A

Код: | 0 0 0 0 0 0 1 1 |

Час: 1 цикл

Алгоритм: (A[N]):=(A[N+1]), де N=0-6
(A[7]):=(A[0])

Приклад: ;(A)=0D6H (11010110B), (C)=1
RR A ;(A)=6BH (01101011B), (C)=1

Команда RRC A

Команда *зсув вмісту акумулятора вправо через прапорець переносу* зсуває вісім бітів акумулятора і прапорець переносу на один біт вправо. Біт 0 переміщується в прапорець переносу, а вихідний стан прапорця переносу переміщується в біт 7. На інші прапорці ця команда не впливає.

Асемблер: RRC A

Код: | 0 0 0 1 0 0 1 1 |

Час: 1 цикл

Алгоритм: (A[N]):=(A[N+1]), де N=0-6
(A[7]):=(C)
(C):=(A[0])

Приклад: ;(A)=95H (10010101B), (C)=0
RRC A ;(A)=4AH (01001010B), (C)=1

Команда SETB (біт)

Команда *встановити біт* встановлює вказаний біт в «1». Адресується:

1) до прапорця переносу (C);
2) до біта з прямою адресацією.

1) Асемблер: SETB C

Код: | 1 1 0 1 0 0 1 1 | bit address |

Час: 1 цикл

Алгоритм: (C):=1

Приклад: ;(C)=0
SETB C ;(C)=1

2) Асемблер: SETB (bit)

Код: | 1 1 0 1 0 0 1 0 | bit address |

Час: 2 цикли

Алгоритм: (bit):=1

Приклад: ;(P2)=38H (00111000B)
SETB P2.0 SETB P2.7
;(P2)=B9H (10111001B)

Команда SJMP (мітка)

Команда *короткий перехід* виконує безумовне розгалуження в програмі за вказаною адресою. Адреса розгалуження вираховується додаванням зміщення зі знаком в другому байті команди з вмістом лічильника команд після додавання до нього 2.

Таким чином, адреса приходу повинна знаходитися в діапазоні від 128 байтів, які передують команді, до 127 байтів, які слідують за нею.

Асемблер: SJMP (мітка)

Код: | 10000000 | rel8 |

Час: 2 цикли

Алгоритм: (PC)=(PC)+2
(PC)=(PC)+(rel8)

Приклад: ;(PC)=0418H
; мітка HIT1 відповідає адресі
; 039AH
SJMP HIT1; (PC)=039AH, де
; (rel8)=80H= −128D
SJMP HIT2; (PC)=041AH, де мітка

; HIT2 відповідає адресі 041AH,
; (rel8)=7D= +125D

Команда SUBB A,(байт джерела)

Команда *віднімання із позикою* віднімає вказану змінну разом з прапорцем переносу від вмісту акумулятора, пересилаючи результат в акумулятор. Ця команда встановлює прапорець переносу (позики), якщо при відніманні для біта 7 необхідна позика, в протилежному випадку прапорець переносу скидається. Якщо прапорець переносу встановлений перед виконанням цієї команди, то це вказує на те, що позика необхідна при відніманні з підвищеною точністю на попередньому кроці, тому прапорець переносу віднімається від вмісту акумулятора разом з операндом джерела. (AC) встановлюється, якщо позика необхідна для біта 3 і скидається в протилежному випадку. Прапорець переповнення (OV) встановлюється, якщо позика необхідна для біта 6, але його немає для біта 7, або якщо є для біта 7, але немає для біта 6.

При відніманні цілих чисел зі знаком (OV) вказує на від'ємне число, яке отримується при відніманні від'ємної величини від додатної, або додатне число, яке отримується при відніманні додатного числа від від'ємного.

Операнд джерела допускає чотири режими адресації:
1) регістровий
2) прямий
3) непрямо-регістровий
4) безпосередній (до констант)

1) Асемблер: SUBB A,Rn ;де n=0-7

Код: 1 0 0 1 1 rrr , де rrr=000-111

Час: 1 цикл

Алгоритм: (A):=(A)-(C)-(Rn);
(C):=X, (AC):=X, (OV):=X, де X=(0 або 1)

Приклад: ;(A)=C9H, (R2)=54H, (C)=1
SUBB A,R2 ;(A)=74H, (R2)=54H, (C)=0,

;(AC)=0, (OV)=1

2) Асемблер: SUBB A,(direct)

Код: | 1 0 0 1 0 1 0 1 | direct address |

Час: 1 цикл

Алгоритм: (A):=(A)-(C)-(direct);
(C):=X, (AC):=X, (OV):=X, де X=(0 або 1)

Приклад: ;(A)=97H, (B)=25H, (C)=0
SUBB A,B ;(A)=72H, (B)=25H, (C)=0,
;(AC)=0, (OV)=1

3) Асемблер: SUBB A,@Ri ; де i=0,1

Код: | 1 0 0 1 0 1 1 i | ; де i = 0,1

Час: 1 цикл

Алгоритм: (A):=(A)-(C)-((Ri));
(C):=X, (AC):=X, (OV):=X, де X=(0 або 1)

Приклад: ;(A)=49H, (C)=1, (R0)=33H,
;(ОЗП[33])=68H SUBB A,@R0
;(A)=E0H, (C)=1, (AC)=0, (OV)=0

4) Асемблер: SUBB A,#data

Код: | 1 0 0 1 0 1 0 0 | #data8 |

Час: 1 цикл

Алгоритм: (A):=(A)-(C)-(#data8);
(C):=X, (AC):=X, (OV):=X, де X=(0 або 1)

Приклад: ;(A)=0BEH, (C)=0
SUBB A,#3FH ;(A)=7FH, (C)=0, (AC)=1, (OV)=1

Команда SWAP

Команда *обмін тетрадами всередині акумулятора* здійснює обмін між молодшими чотирма і старшима чотирма бітами акумулятора (між старшою і молодшою тетрадами).

Ця команда може розглядатися так, як команда чотирибітового циклічного зсуву. На прапорці ця команда не впливає.

Асемблер: SWAP A

Код: 1 1 0 0 0 1 0 0

Час: 1 цикл

Алгоритм: (A[3-0]):=(A[7-4]), (A[7-4]):=(A[3-0])

Приклад: ;(A)=0D7H (111010111B)
SWAP A ;(A)=7DH (01111101B)

Команда XCH A,(байт)

Команда *обмін вмісту акумулятора зі змінною-байтом* здійснює обмін вмісту акумулятора з вмістом джерела, вказаного в команді. Операнд джерела може використовувати наступні режими адресації:

1) регістровий
2) прямий
3) непрямо-регістровий

1) Асемблер: XCH A,Rn ;де n=0-7

Код: 1 1 0 1 1 rrr , де rrr=000-111

Час: 1 цикл

Алгоритм: (A):=(Rn), (Rn):=(A)

Приклад: ;(A)=3CH, (R4)=15H
XCH A,R2 ;(A)=15H, (R6)=3CH

2) Асемблер: XCH A,(direct)

Код: | 1 1 0 0 0 1 0 1 | direct address |

Час: 1 цикл

Алгоритм: (A):=(direct), (direct):=(A)

Приклад: ;(A)=0FEH, (P3)=0DAH
XCH A,P3 ;(A)=0DAH, (B)=0FEH

3) Асемблер: XCH A,@Ri , де i=0,1

Код: | 1 1 0 0 0 1 1 i | ; де i = 0,1

Час: 1 цикл

Алгоритм: (A):=((Ri)), ((Ri)):=(A)

Приклад: ;(R1)=39H, (ОЗП[39])=44H, (A)=0BCH
XCH A,@R1 ;(ОЗП[39])=0BCH, (A)=44H

Команда XCHD A,@Ri

Команда *обмін тетрадою* здійснює обмін молодшої тетради (біти 3-0) акумулятора з вмістом молодшої тетради (біти 3-0) комірки внутрішнього ОЗП, непряма адресація до якої здійснюється за допомогою вказаного регістра. На старші біти (біти 7-4) ця команда не впливає (також, як і на прапорці).

Асемблер: XCHD A,@Ri , де i=0,1

Код: | 1 1 0 1 0 1 1 i | ; де i = 0,1

Час: 1 цикл

Алгоритм: (A[3-0]):=((Ri[3-0])),
((Ri[3-0])):=(A[3-0])

Приклад: ;(R0)=55H, (A)=89H, (ОЗП[55])=0A2H
XCHD A,@R0 ;(A)=82H, (ОЗП[55])=0A9H

Команда XRL (байт призначення),(байт джерела)

Команда *логічне «ВИКЛЮЧНЕ АБО» для змінних-байтів* виконує операцію «ВИКЛЮЧНЕ АБО» над бітами вказаних змінних, записуючи результат в байт призначення. На прапорці ця команда не впливає.

Допускається шість режимів адресації:
- байтом призначення є акумулятор:
1) регістровий
2) прямий
3) непрямо-регістровий
4) безпосередній
- байтом призначення є пряма адреса:
5) до акумулятора
6) до константи.

1) Асемблер: XRL A,Rn ;де n=0-7

Код: | 0 1 1 0 1 rrr | , де rrr=000-111

Час: 1 цикл

Алгоритм: (A):=(A) XOR (Rn)

Приклад: ;(A)=C3H, (R6)=0AAH
XRL A,R6 ;(A)=69H, (R6)=0AAH

2) Асемблер: XRL A,(direct)

Код: | 0 1 1 0 0 1 0 1 | | direct address |

Час: 1 цикл

Алгоритм: (A):=(A) XOR (direct)

Приклад: ;(A)=0FH, (P1)=0A6H
XRL A,P1 ;(A)=A9H, (P1)=0A6H

3) Асемблер: XRL A,@Ri ; де i=0,1

Код: | 0 1 1 0 0 1 1 i | ; де i = 0,1

Час: 1 цикл

Алгоритм: (A):=(A) XOR ((Ri))

Приклад: ;(A)=55H, (R1)=33H, (ОЗП[77])=5AH
XRL A,@R1 ;(A)=0FH, (ОЗП[77])=5AH

4) Асемблер: XRL A,#data

Код: | 0 1 1 0 0 1 0 0 | #data8 |

Час: 1 цикл

Алгоритм: (A):=(A) XOR (data)

Приклад: ;(A)=0C3H
XRL A,#0F5H ;(A)=36H

5) Асемблер: XRL (direct),(A)

Код: | 0 1 1 0 0 0 0 1 | direct address |

Час: 1 цикл

Алгоритм: (direct):=(direct) XOR (A)

Приклад: ;(A)=31H, (P1)=82H
XRL P1,A ;(A)=31H, (P1)=B3H

6) Асемблер: XRL (direct),#data

Код: | 0 1 1 0 0 1 0 0 | direct address | #data8 |

Час: 2 цикли

Алгоритм: (direct):=(direct) XOR #data

Приклад: ;(IP)=65H
XRL IP,#65H ;(IP)=00H

Примітка: Якщо ця команда використовується для роботи з портами, то значення, яке використовується в якості операнда, зчитується із «фіксатора» порту, а не з виводів ВІС.

ПЕРЕЛІК ІНТЕРФЕЙСНИХ МІКРОСХЕМ ДЛЯ ОРГАНІЗАЦІЇ МК СИСТЕМ

Характеристики мікросхем АЦП

N п/п	Тип АЦП	Розрядність	Час перетворення, мкс	Опорна напруга, В	Функція і тип логіки на виході
1	К572 ПВ1	12	60	-15...+15 (зовн.)	АЦП послідовного наближення (КМОН, ТТЛ)
2	К572 ПВ2	3,5 дес.	---	---	Інтеґруючий з виходом на 7-сегм. індикатор
3	К572 ПВ3	8	15	-10	З виходом на МП (КМОН, ТТЛ)
4	К572 ПВ4	8	32	-2,5...2,5	З виходом на МП (КМОН, ТТЛ) 8-кан.
5	К572 ПВ5	3,5 дес.	---	---	Інтеґруючий з виходом на 7-сегм. індикатор
6	К1107 ПВ1	6	0,1	-1,9...-2,1 (зовн.)	Одночасного перетворювання (ТТЛ)
7	К1107 ПВ2	8	0,1	-1,9...-2,1 (зовн.)	Одночасного перетворювання (ТТЛ)
8	К1107 ПВ3	6	0,02	-2,5...2,5	Одночасного перетворювання (ТТЛ) (ЕЗЛ)
9	К1107 ПВ4	8	0,03	-2,5...2,5	Одночасного перетворювання (ТТЛ) (ЕЗЛ)
10	К1107 ПВ6	10	0,06	-3,0...0	Одночасного перетворювання (ТТЛ) (ЕЗЛ)
11	К1108 ПВ1	8-10	1	внутр.	Програмованої розрядності, що спрягаються з МП (ТТЛ)
12	К1108 ПВ2	12	2	2,5 або зовн.	Програмованої розрядності, що спрягаються з МП (ТТЛ)
13	К1113 ПВ1	10	25	-10...+10	Програмованої розрядності, що спрягаються з МП (ТТЛ)
14	К512 ПВ1	12	170	-15...+15	Універсальний (АЦП і ЦАП)

Характеристики мікросхем ЦАП

п/п	Тип ЦАП	Розрядність	Час перетворення, мкс	Опорна напруга, В	Функція і тип логіки на виході
1	К572ПА1	10	5	-15...+15	Перемножуючий (КМОН)
2	К572ПА2	12	15	-15...+15	З записом і зберіганням вхідного сигналу (КМОН/ТТЛ)
3	К594ПА1	12	3,5	-15...+15	З зап. і збер.вхід.сиг. (КМОН/ТТЛ)
4	К1108ПА1	12	0,4	10	Швидкодіючий (ТТЛ)
5	К1108ПА2	8	1,5	10	Швидкодіюч. (ТТЛ)
6	К1118ПА1	8	0,02	10	Швидкодіюч. (ТТЛ)
7	К1118ПА2	10	0,03	-1	Швидкодіюч. (ТТЛ)
8	К1118ПА3	8	0,01	1,2...1,3	Швидкодіюч. (ТТЛ)
9	К1118ПА4	10	0,03	1,2...1,3	Швидкодіюч. (ТТЛ)
10	К417ПА1	12	15	-10...+10	Перемножуючий (ТТЛ)
11	К417ПА2	12	15	-10...+10	Перемнож. (ТТЛ)
12	К427ПА1	16	30	-10...+10	Прецизійний (ТТЛ)

Характеристики аналогових комутаторів

N п/п	Тип мікросхеми	Організація
1	К561КП1	2(4 → 1) комутатор напруги із схемою керування
2	К561КП2	8 → 1 комутатор напруги із схемою керування
3	К561КТ3	4 одноканальні ключі
4	К176КТ1	4 одноканальні ключі
5	КР590КН1	8-канальний комутатор з дешифратором (8 → 1)
6	КР590КН2	4-канальний ключ із схемою керування
7	КР590КН3	2(4 → 1)-канальний комутатор з дешифратором
8	КР590КН4	4-канальний аналоговий ключ із схемою керув.
9	КР590КН6	8-канальний аналоговий комутатор із схемою керування
10	КР590КН8	4-канальний аналоговий ключ із схемою керув.
11	КР590КН9	2-канальний аналоговий ключ із схемою керув.
12	КР590КТ1	2(2 → 1)аналоговий комутатор
13	К591КН2	16-канальний 2(8 → 1) аналоговий комутатор
14	К591КН3	16-канальний (16 → 1) аналоговий комутатор

Характеристики мікросхем ОЗП

N п/п	Тип мікросхеми	Ємність	Організація	Примітка
1	К155РУ5	256	256 х 1	статичне
2	КР185РУ9	512	64 х 9	статичне
3	К155РУ7	1к	1к х 1	статичне
4	КМ537РУ1А	1к	1к х 1	статичне
5	К565РУ2	1к	1к х 1	динамічне
6	КР132РУ2А	1к	1к х 1	статичне
7	КР132РУ3	1к	1к х 1	статичне
8	КР132РУ4	1к	1к х 1	статичне
9	КМ185РУ7	1к	256 х 4	статичне
10	КМ1603РУ1	1к	256 х 4	статичне
11	КМ132РУ5А	4к	4к х 1	статичне
12	КМ537РУ14А	4к	4к х 1	статичне
13	КР537РУ2А	4к	4к х 1	статичне
14	КР537РУ4	4к	4к х 1	статичне
15	КР541РУ1	4к	4к х 1	статичне
16	КМ132РУ8	4к	1к х 4	статичне
17	КМ132РУ9	4к	1к х 4	статичне
18	КР537РУ13	4к	1к х 4	статичне
19	КР541РУ2	4к	1к х 4	статичне
20	КР565РУ1А	4к	4к х 1	динамічне
21	К541РУ31-34	8к	8к х 1	статичне
22	КР537РУ11	4к	256 х 16	статичне
23	КР132РУ6	16к	16к х 1	статичне
24	КР565РУ6	16к	16к х 1	динамічне
25	КС581РУ4	16к	16к х 1	динамічне
26	КМ581РУ5Б	16к	2к х 8	динамічне
27	КР537РУ10	16к	2к х 8	статичне
28	КР537РУ8А	16к	2к х 8	статичне
29	К1809РУ1	16к	1к х 16	статичне
30	КР565РУ5Д1,Д2	32к	32к х 1	динамічне
31	КР565РУ5	64к	64к х 1	динамічне
32	КР565РУ7	256к	256к х 1	динамічне

Характеристики мікросхем ПЗП

N п/п	Тип мікросхеми	Ємність	Організація	Примітка
1	К155РЕ3	256	32 х 8	ВК
2	К556РТ4	1к	256 х 4	ВК
3	К556Т11	1к	256 х 4	ТС
4	К1601РР2	1к	512 х 2	ТС
5	К1601РР11	1к	512 х 2	ТС
6	К1601РР12	1к	512 х 2	ТС
7	К556РТ5	2к	512 х 8	ВК
8	К556Т17	2к	512 х 8	ТС
9	К1608РТ2	2к	512 х 8	ТС
10	К573РФ11	2к	512 х 8	ТС
11	К573РФ12	2к	512 х 8	ТС
12	К556РТ12	4к	1к х 4	ВК
13	К556РТ13	4к	1к х 4	ТС
14	К556РТ1 (ПЛМ)	4к	1к х 4	ВК
15	К556РТ14	8к	2к х 4	ВК
16	К556РТ15	8к	2к х 4	ТС
17	К1601РР33	8к	2к х 4	ТС
18	К573РФ23	8к	2к х 4	ТС
19	К573РФ24	8к	2к х 4	ТС
20	К558РР21	8к	1к х 8	ТС
21	К1601РР31	8к	1к х 8	ТС
22	К1609РР11	8к	1к х 8	ТС
23	К573РФ1	8к	1к х 8	ТС
24	К556РТ6	16к	2к х 8	ВК
25	К556РТ7	16к	2к х 8	ТС
26	К556РТ18	16к	2к х 8	ТС
27	К558РР2	16к	2к х 8	ТС
28	К573РФ2	16к	2к х 8	ТС
29	К573РФ5	16к	2к х 8	ТС
30	К1601РР1	16к	2к х 8	ТС
31	К1601РР3	16к	2к х 8	ТС
32	К573РР21	8к	1к х 8	ТС
33	К573РФ21	8к	1к х 8	ТС
34	К573РФ22	8к	1к х 8	ТС
35	К573РФ33	16к	1к х 16	ТС
36	К573РФ34	16к	1к х 16	ТС
37	К573РФ31	32к	1к х 16	ТС
38	К573РФ32	32к	1к х 16	ТС
39	К573РФ41	32к	4к х 8	ТС

Продовження характеристики мікросхем ПЗП

40	К573РФ42	32к	4к х 8	ТС
41	К573РФ61	32к	4к х 8	ТС
42	К573РФ62	32к	4к х 8	ТС
43	К573РФ43	32к	8к х 4	ТС
44	К573РФ44	32к	8к х 4	ТС
45	К573РФ63	32к	8к х 4	ТС
46	К573РФ64	32к	8к х 4	ТС
47	К573РФ3	64к	4к х 16	ТС
48	К1607РФ1	64к	4к х 16	ТС
49	К1610РЕ1	16к	2к х 8	ТС
50	К1611РЕ1	64к	8к х 8	ТС
51	К558РР3	64к	8к х 8	ТС
52	КР1801РЕ21	64к	4к х 16	ТС
53	КР1801РЕ26	64к	4к х 16	ТС
54	К1809РЕ1	64к	4к х 16	ТС

Розшифровка позначень ПЗП:

РЕ - масочні;

РТ - одноразово програмовані;

РР - багаторазово програмовані;

РФ - багаторазово програмовані з ультрафіолетовим стиранням інформації.

Виходи:

ВК - відкритий колекторний вивід;

ТС - з трьома станами.

Характеристики цифрових дешифраторів для побудови ОЗП і ПЗП

N п/п	Тип мікросхеми	Організація і вихід
1	К555ИД4	2(2– 4), ТС
2	К555ИД18	двійковий в семисегментний
3	К555ИД5	2(2 – 4), ВК
4	К555ИД6	4 – 10, ТС
5	К555ИД7	3 – 8, ТС
6	К555ИД10	4 – 10, ВК
7	КМ533ИД3	4 – 16
8	К155ИД1	4 – 16 - високовольтний ВК

Список літератури

1. *Антонью А.* Цифровые фильтры: анализ и проектирование.- М.: Радио и связь, 1983. –320 с.

2. *Бесекерский В.А., Изранцев В.В.* Системы автоматического управления с микро-ЭВМ. -М.: Наука, 1987. –320 с.

3. *Бойко Н.П., Стеклов В.К.* Системы автоматического управления на базе микро-ЭВМ. -К.: Техника, 1989. –182 с.

4. *Вершинин О.Е.* Применение микропроцессоров для автоматизации технологических процессов. -М.: Энергоатомиздат. Лен. отд-ние, 1986. –208 с.

5. *Ги К.* Введение в локальные вычислительные сети. -М.: Радио и связь, 1986. -176 с.

6. *Каган Б.М., Сташин В.В.* Основы проектирования микропроцессорных устройств автоматики. -М.: Мир, 1987. -304 с.

7. *Корнейчук В.И., Тарасенко В.П.* Вычислительные устройства на микросхемах: Справочник. -К.: Техника, 1988. – 351 с.

8. *Корячко В.П.* Микропроцессоры и микро-ЭВМ в радиоэлектронных средствах. –М.: Высш. шк., 1990.- 407 с.

9. *Куо Б.* Теория и проектирование цифровых систем управления. -М.: Машиностроение, 1986. – 448 с.

10. *Макаревич О.Б., Спиридонов Б.Г.* Цифровые процессоры обработки сигналов на основе БИС//Зарубежная электронная техника.- 1983.- N1.- С.58-92.

11. *Макклеллан Дж., Рейдер Ч.* Применение теории чисел в цифровой обработке сигналов. -М.: Радио и связь, 1983. –264 с.

12. Микропроцессоры/Под ред. Преснухина Л.Н., т. 1, 2, 3. -М.: Высшая школа, 1986.

13. Микропроцессорное управление технологическим оборудованием микроэлектроники: Учеб. пособие /А.А.Сазонов, Р.В.Корнилов, Н.П.Кохан и др.; Под ред. А.А.Сазонова. -М.: Радио и связь, 1988. –264 с.

14. Микропроцессорные системы автоматического управления/Под ред. Бесекерского В.А. -Л.: Машиностроение, 1988. –365 с.

15. *Мирский Г.Я.* Характеристики стохастической взаимосвязи и их измерения. -М.: Энергоиздат, 1982. –320 с.

16. Недорогие однокристальные процессоры сиґналов. – Электроника.- 1983.- N7.- С.76-79.

17. Однокристальные микрокомпьютеры в системах управления. /В.П.Захаров, Ю.М.Польский, Л.М.Солдатенко и др. -К.: Техніка, 1984. –95 с.

18. Однокристальные микро-ЭВМ: Справочник. -М.: МИКАП, 1994. –400 с.

19. *Погрибной В.А.* Дельта-модуляция в цифровой обработке сиґналов. -М.: Радио и связь, 1990. –216 с.

20. *Погрібний В.О., Рожанківський І.В., Юрченко Ю.П.* Основи інформаційних принципів в роботизованому виробництві /За ред. В.О.Погрібного. -Львів: Світ, 1995. –304 с.

21. *Прангишвили И.В.* Микропроцессоры и локальные сети микро-ЭВМ в распределённых системах управления. -М.: Энергоатомиздат, 1985. –272 с.

22. Проектирование цифровых устройств на однокристальных микроконтроллерах/В.В.Сташин и др. -М.: Энергоатомиздат, 1990. –224 с.

23. *Рабинер Л., Гоулд Б.* Теория и применение цифровой обработки сигналов. -М.: Мир, 1978. –848 с.

24. СверхБИС универсальных однокристальных микро-ЭВМ /А.В.Кобылинский, Г.П.Липовецкий, Н.Г.Сабадаш и др. -К.: Техніка, 1987. –166 с.

25. *Солодовников В.В.* и др. Микропроцессорные автоматические системы регулирования. -М.: Высшая школа, 1991. –255 с.

26. Справочник по устройствам цифровой обработки информации/Н.А.Виноградов и др. -К.: Техніка, 1988. –415 с.

27. Справочник по цифровой схемотехнике/В.И.Зубчук, В.П.Сигорский, А.Н.Шкуро. -К.: Техніка, 1990. – 448 с.

28. *Стеклов В.К.* Проектування систем автоматичного керування. -К.: Вища школа, 1995. –231 с.

29. *Стрепко І.Т., Тимченко О.В.* Проектування систем керування на основі однокристальних мікро-ЕОМ з аналоговими каналами вводу і виводу сиґналів. -Львів: УАД, 1995. –54 с.

30. *Стрепко І.Т., Тимченко О.В.* Обробка сиґналів в системах автоматичного керування поліграфічним обладнанням на однокристальних мікро-ЕОМ серії КР1816. -Львів: УАД, 1996.– 56 с.

31. *Стрепко І.Т., Тимченко О.В.* Проектування систем керування поліграфічним обладнанням на однокристальних мікро-ЕОМ. -Львів: УАД, 1996. –34 с.

32. *Стрепко І.Т., Тимченко О.В.* Використання повноекранного відлагоджувача-симулятора однокристальної

мікро-ЕОМ типу КР1816ВЕ51. -Львів: УАД, 1996. –40 с.

33. *Стрепко І.Т., Тимченко О.В.* Двовідковий давач коду кута // Поліграфія і видавнича справа. -Львів: УАД, 1997. –№33.- С.36-39.

34. *Хвощ С.Т., Верлинский Н.Н., Попов Е.А.* Микропроцессоры и микро-ЭВМ в системах автоматического управления: Справочник /Под ред. С.Т.Хвоща. -Л.: Машиностроение, 1987. –640 с.

35. Цифровые и аналоговые интегральные микросхемы: Справочник /Под ред. С.В.Якубовского. -М.: Радио и связь, 1990.–496 с.

36. Цифровой процессор обработки сигналов с аналоговыми устройствами ввода-вывода/Под ред. А.А.Ланнэ. -Л.: ВАС, 1985.– 88 с.

37. Intel 8051. User's manual. Intel Corp., 1980. –110 p.

38. ISIS-II MCS51. Macro-assembler. Intel Corp., 1980. –80p.

39. R.E.Holm, I.S.Rittenhouse. Implementation of a Scanning Spectrum Analyzer Using the 2920 Signal Processor. Intel, 1980.– 24 p.

40. *So J.* TMS-320 - a step forward in digital signal processing.- Microprocessors and microsystems. V.7.- N10.- 1983. -pp.451-460.

СТРЕПКО Ігор Теодорович
ТИМЧЕНКО Олександр Володимирович
ДУРНЯК Богдан Васильович

ПРОЕКТУВАННЯ СИСТЕМ КЕРУВАННЯ НА ОДНОКРИСТАЛЬНИХ МІКРО-ЕОМ

Літературний редактор *М.В.Старовойт*
Художній редактор *Р.Р.Мервінський*
Макетування та верстка *О.І.Стрепко*
Коректор *О.М.Михайлович*

Підписано до друку 14.04.98. Формат 60x84/16.
18 ум.друк.арк., 19,2 обл.-вид.арк. 19,76
Папір офсетний №1. Друк офсетний. Гарнітура School Book.
Зам. № 1691. Наклад 700 прим.
Видавництво "Фенікс"
290020, Львів, вул. Підголоско, 19

Віддруковано з готових діапозитивів
на обладнанні видавництва "Київська правда"
254136, Київ, вул. Маршала Гречка, 13