**Міністерство транспорту та зв'язку України**
**Державна адміністрація зв'язку України**

ОДЕСЬКА НАЦІОНАЛЬНА АКАДЕМІЯ ЗВ'ЯЗКУ ім. О.С. ПОПОВА

**Кафедра обчислювальної техніки та мікропроцесорів**

**Хіхловська І.В., Антонов О.С.**

# ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА МІКРОПРОЦЕСОРИ

***Підручник***
**з дисципліни**
**"Обчислювальна техніка та мікропроцесори"**
**напряму Телекомунікації**

ЗАТВЕРДЖЕНО
Методичною радою
Одеської національної академії
зв'язку ім. О.С. Попова
Протокол № 6
від 10 лютого 2008 р.

**Одеса 2009**

УДК 004.31 (075)
ББК 32.973.2
Х 52

Гриф «Затверджено Міністерством освіти і науки України як підручник для студентів вищих навчальних закладів, які навчаються за напрямом підготовки «Телекомунікації»», надано Міністерством освіти і науки України, лист №1/11-26 від 05.01.2011 року.

Рецензенти: *В.В. Поповський,* д-р техн. наук, професор
*І.П. Лісовий,* д-р техн. наук, професор
*М.М. Климаш,* д-р техн. наук, професор

**Хіхловська І.В.** Обчислювальна техніка та
**Х 52** мікропроцесори : підручник / **Хіхловська І.В., Антонов О.С.** –
Одеса: ОНАЗ ім. О.С. Попова, 2011. – 440 с.: іл.

**ISBN 978-966-7598-58-7**

Підручник призначено для самостійної роботи студентів з дисципліни “Обчислювальна техніка та мікропроцесори”, яка викладається за модульним принципом та має два модулі:

Перший модуль складається з двох частин: 1. Вузли обчислювальної техніки та мікропроцесорних систем; 2. Програмування мікропроцесорів фірми *Intel*.

Другий модуль також складається з двох частин: 1. Мікропроцесорні системи на мікропроцесорах фірми *Motorola* та їхнє програмування; 2. Мікропроцесорні системи на мікроконтролерах, *DSP* фірми *Motorola* та їхнє програмування.

Розглянуто основні принципи побудови й функціонування обчислювальних та мікропроцесорних систем, розглянуто їхні основні вузли, показані принципи проектування мікропроцесорних систем. Наведено приклади застосування мікропроцесорів та мікроконтролерів різних моделей у пристроях телекомунікацій. Кожен розділ супроводжується питаннями вхідного та вихідного контролю.

УДК 004.31 (075)
ББК 32.973.2
Х 52

**ISBN 978-966-7598-58-7**

© Хіхловська І.В., Антонов О.С.,
© ОНАЗ ім. О.С. Попова, 2011.

# ЗМІСТ

ВСТУП ..... 7
МОДУЛЬ 1.
Частина І ВУЗЛИ ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА МІКРОПРОЦЕСОРНИХ СИСТЕМ ..... 9
1 ОБЧИСЛЮВАЛЬНІ ТА МІКРОПРОЦЕСОРНІ СИСТЕМИ ..... 9
1.1 Основні визначення ..... 9
1.2 Принципи побудови та функціонування обчислювальних систем ..... 12
1.2.1 Архітектура обчислювальних систем ..... 12
1.2.2 Класифікація комп'ютерів (Для поглибленого вивчення) ..... 15
1.3 Принципи побудови та функціонування МПС ..... 22
1.4 Функціонування обчислювального пристрою ..... 25
2 ОПЕРАЦІЇ НАД ДАНИМИ В ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ ..... 29
2.1 Подання даних в обчислювальних системах ..... 29
2.2 Подання даних у кодах ..... 36
2.3 Порозрядні операції над даними ..... 39
3 ЦИФРОВІ АВТОМАТИ ..... 43
4 ТИПОВІ ПРИСТРОЇ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ (Для самостійного вивчення) ..... 52
4.1 Суматори ..... 52
4.2 Цифрові компаратори ..... 55
4.3 Арифметично-логічний пристрій ..... 57
4.4 Програмовані логічні інтегральні схеми (ПЛІС) ..... 62
5 ПРИНЦИПИ ПОБУДОВИ ЗАПАМ'ЯТОВУВАЛЬНИХ ПРИСТРОЇВ МПС З ЗАДАНОЮ ОРГАНІЗАЦІЄЮ ..... 64
5.1 Запам'ятовувальні пристрої МПС та їх класифікація ..... 64
5.2 Постійні запам'ятовувальні пристрої–флеш-пам'ять ..... 68
5.3 Оперативні запам'ятовувальні пристрої ..... 72
5.4 Побудова блока запам'ятовувального пристрою МПС з заданою організацією ..... 77
6 ІНТЕРФЕЙС ..... 83
6.1 Організація інтерфейсів ..... 83
6.2 Асинхронний послідовний адаптер *RS*-232-*C* ..... 86
7 МІКРОПРОЦЕСОРИ ..... 93
7.1 Архітектура мікропроцесорів ..... 93
7.2 МП фірми *Intel* ..... 95
7.2.1 Історична довідка про розвиток мікропроцесорів фірми *Intel* (Для самостійного вивчення) ..... 95
7.2.2 Організація 16-розрядних мікропроцесорів ..... 109
7.2.3 Програмна модель МП *I*8086 ..... 113
7.2.4 Режим переривань МП *I*8086 ..... 116

7.2.5 Організація 32-розрядних мікропроцесорів (Для самостійного вивчення) ... 120
7.3 Архітектура сучасних мікропроцесорів ... 130
7.3.1 Тенденції розвитку архітектури сучасних мікропроцесорів ... 130
7.3.2 Мікропроцесори *Pentium* ... 133
7.3.3 Процесори фірми *AMD* ... 139
7.3.4 Продуктивність мікропроцесорів та її оцінювання ... 141
8 ВИКОРИСТАННЯ СУЧАСНИХ МІКРОПРОЦЕСОРІВ У ТЕЛЕКОМУНІКАЦІЙНОМУ ОБЛАДНАННІ (Для поглибленого вивчення) ... 144
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ДО Частини І 1-го МОДУЛЯ ... 154
Частина ІІ ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРІВ ФІРМИ *INTEL* ... 156
9 ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРІВ ФІРМИ *INTEL* ... 156
9.1 Сегментування пам'яті мікропроцесорами ... 156
9.2 Способи адресування операндів МП фірми *Intel* ... 161
9.3 Мова програмування Асемблер-86 ... 169
9.3.1 Формат команди ... 175
9.3.2 Команди пересилань ... 179
9.3.3 Команди перетворення даних мови Асемблер-86 ... 189
9.3.4 Команди умовних та безумовних переходів ... 203
9.3.5 Команди організації циклів ... 207
9.4 Створення програм на мові Асемблер-86 ... 209
9.4.1 Лінійні програми ... 209
9.4.2 Розгалужені програми ... 215
9.4.3 Циклічні програми ... 221
10 ПРОГРАМНА РЕАЛІЗАЦІЯ ВУЗЛІВ ТЕЛЕКОМУНІКАЦІЙНОГО ОБЛАДНАННЯ МОВОЮ АСЕМБЛЕР-86 ... 229
10.1 Способи реалізації алгоритмів ... 229
10.2 Розробка апаратно-програмних комплексів ... 230
10.3 Приклади реалізації простих вузлів телекомунікацій ... 233
10.3.1 Ініціалізація послідовного асинхронного адаптера *RS*-232-*C* ... 233
10.3.2 Фрагмент програми передавання даних через асинхронний адаптер *RS*-232-*C* ... 235
10.3.3 Фрагмент програми приймання даних через асинхронний адаптер *RS*-232-*C* ... 235
10.3.4 Приклад програми ініціалізації *RS*-232-*C* та введення-виведення даних, написаної у програмному середовищі *TURBO ASSEMBLER (TASM*) ... 235
10.3.5 Програмна реалізація генератора імпульсних послідовностей ... 237

10.3.6 Програмне вимірювання періоду імпульсної послідовності *DET* ..... 238
10.3.7 Програмна реалізація мультиплексора ..... 240
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ДО Частини II 1-го МОДУЛЯ ..... 242
МОДУЛЬ 2.
Частина I МІКРОПРОЦЕСОРНІ СИСТЕМИ НА МІКРОПРОЦЕСОРАХ ФІРМИ *MOTOROLA* ТА ЇХНЄ ПРОГРАМУВАННЯ ..... 243
11 МІКРОПРОЦЕСОРНІ СИСТЕМИ НА УНІВЕРСАЛЬНИХ МП ФІРМИ *MOTOROLA* ..... 243
11.1 16-розрядні мікропроцесори фірми *Motorola* ..... 243
11.2 Побудова МПС на 16-розрядних мікропроцесорах фірми *Motorola* ..... 256
11.2.1 Підсистема центрального процесорного елемента *MC*68000 ..... 256
11.2.2 Розподіл адресного простору МПС ..... 260
11.2.3 Організація підсистеми пам’яті ..... 262
11.2.4 Організація підсистеми введення-виведення ..... 266
11.3 32-розрядні мікропроцесори сімейства *M*680*XX* фірми *Motorola* ..... 284
11.4 Побудова МПС на 32-розрядних мікропроцесорах фірми *Motorola* ..... 288
11.4.1 Підсистеми центрального процесорного елемента ..... 288
11.4.2 Розподіл адресного простору МПС ..... 292
11.4.3 Організація підсистеми пам’яті МПС ..... 295
11.4.4 Організація підсистеми введення-виведення ..... 298
11.4.5 Підключення співпроцесора ..... 304
12 ПРОГРАМУВАННЯ УНІВЕРСАЛЬНИХ МП ФІРМИ *MOTOROLA* ..... 307
12.1 Мова програмування Асемблер МП фірми *Motorola* ..... 307
12.2 Система команд МП *MC*680*X*0 (Для самостійного вивчення) ..... 312
12.2.1 Команди пересилань ..... 312
12.2.2 Команди арифметичних операцій ..... 313
12.2.3 Команди логічних операцій ..... 318
12.2.4 Команди зсувів ..... 319
12.2.5 Команди безумовних переходів ..... 321
12.2.6 Команди умовних переходів ..... 322
12.2.7 Команди організації програмних циклів ..... 325
12.2.8 Команди звернення до підпрограм ..... 325
12.3 Побудова програм з різною структурою мовою Асемблер МП фірми *Motorola* ..... 328
12.3.1 Лінійні програми ..... 328
12.3.2 Розгалужені та циклічні програми. Підпрограми ..... 329

12.4 Створення програмного забезпечення МПС на МП фірми *Motorola* ..... 332
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ДО Частини I 2-го МОДУЛЯ..... 342
Частина II МІКРОПРОЦЕСОРНІ СИСТЕМИ НА МІКРОКОНТРОЛЕРАХ, *DSP* ФІРМИ *MOTOROLA* ТА ЇХНЄ ПРОГРАМУВАННЯ ..... 343
13 МІКРОПРОЦЕСОРНІ СИСТЕМИ НА МІКРОКОНТРОЛЕРАХ ФІРМИ *MOTOROLA* ТА ЇХНЄ ПРОГРАМУВАННЯ ..... 343
13.1 Типові мікроконтролери фірми *Motorola* ..... 344
13.1.1 8-розрядні мікроконтролери ..... 344
13.1.2 16-розрядні мікроконтролери ..... 375
13.1.3 32-розрядні мікроконтролери ..... 380
13.2 Система команд мікроконтролерів фірми *Motorola* ..... 384
13.3 Налаштовування вбудованих засобів мікроконтролерів ..... 394
14 *RISC*-ПРОЦЕСОРИ ФІРМИ *MOTOROLA* ..... 401
14.1 *RISC*-процесори *PowerPC* ..... 401
14.2 *RISC*-процесори *ColdFire* ..... 407
14.3 Система команд *RISC*-мікропроцесорів сімейства *PowerPC* ..... 411
15 АРХІТЕКТУРА ТА ПРИНЦИПИ ПОБУДОВИ ПРОЦЕСОРІВ ЦИФРОВОГО ОБРОБЛЕННЯ СИГНАЛІВ ..... 420
15.1 Основні напрямки цифрового оброблення сигналів (ЦОС) . 420
15.2 Узагальнена архітектура процесорів сімейства *DSP*563*XX*.. 424
15.3 Організація циклічного буфера в *DSP* ..... 426
15.4 Програмна реалізація цифрового фільтра СІХ ..... 429
16 МПС НА МІКРОКОНТРОЛЕРАХ, МІКРОПРОЦЕСОРАХ ТА *DSP* ..... 432
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ДО Частини II 2-го МОДУЛЯ..... 437
ПРЕДМЕТНИЙ ПОКАЖЧИК ..... 438

# ВСТУП

На етапі розвитку сучасних інформаційних мереж нового покоління уже неможливо собі уявити телекомунікаційне обладнання без сучасних мікропроцесорів та мікроконтролерів. Широкий спектр функцій, які реалізують системи комутації, шлюзи, маршрутизатори, інтегровані платформи, сервери, робочі станції, вимагає від процесорів та мікроконтролерів високої продуктивності та багатофункційності. Десятки років можна було спостерігати процес взаємного стимулювання розвитку процесорів, з одного боку, та побудованого на їх основі телекомунікаційного обладнання, з іншого.

Підручник призначено для самостійної роботи студентів напряму Телекомунікації з дисципліни “Обчислювальна техніка та мікропроцесори”. Дисципліна має 216 годин і складається з двох модулів:

1 Вузли обчислювальної техніки та мікропроцесорних систем. Програмування мікропроцесорів фірми *Intel*.

2 Мікропроцесорні системи на універсальних МП та мікроконтролерах. Програмування МПС.

За вивчення першого модуля студенти отримують такі знання та уміння: подавати та трактувати вхідні та вихідні чисельні дані для подальшого цифрового оброблення. Співвідносити логічні змінні та функції з цифровими сигналами, що їх реалізують. Синтезувати цифрові пристрої, використовуючи типові цифрові блоки, вузли та елементи. Ставити та розв’язувати задачі, пов’язані з вибором засобів обчислювальної техніки, мікропроцесорів та мікроконтролерів за їх технічними, експлуатаційними та економічними характеристиками для систем телекомунікацій. Створювати та налагоджувати програмне забезпечення для мікропроцесорів фірми *Intel*. Створювати та налагоджувати програмне забезпечення для програмної реалізації вузлів телекомунікаційного обладнання.

За вивчення другого модуля студенти отримують такі знання та уміння: ставити та розв’язувати задачі, пов’язані з аналізом, розробленням та експлуатацією мікропроцесорних систем у складі інформаційних та телекомунікаційних систем і мереж, створенням та налагодженням програмного забезпечення до них. Аналізувати та розробляти окремі вузли систем телекомунікацій, які використовують засоби обчислювальної техніки, мікропроцесори та мікроконтролери. Створювати та налагоджувати програмне забезпечення для пристроїв управління, комутації, оброблення цифрових сигналів у системах телекомунікацій мовами конкретних мікропроцесорів та мікроконтролерів.

Підручник відрізняється від уже раніше виданих тим, що має розділи, присвячені застосуванню мікропроцесорів в обладнанні телекомунікацій, створенню програмного забезпечення для пристроїв та систем телекомунікацій.

Знання архітектури сучасних мікропроцесорів та їх основних характеристик дасть можливість майбутнім фахівцям вибирати апаратуру інформаційних мереж та систем з урахуванням можливостей застосовуваних у

ній засобів обчислювальної техніки та мікропроцесорів, а також проектувати цю апаратуру на сучасному рівні.

Для розуміння викладаного у підручнику матеріалу студенти повинні знати такі теми з попередньо вивчених дисциплін:

І З дисципліни **“Основи схемотехніки”**:

1 Усі розділи цифрової техніки: логічні елементи, таблиці істинності, що описують їхню роботу, шифратори, дешифратори, мультиплексори, демультиплексори, перемикальні функції, що описують їх роботу, регістри, лічильники імпульсів та таблиці переходів, що їх описують.

2 Інтегральна схемотехніка, технологія МОП та КМОП, великі (ВІС) та надвеликі (НВІС) інтегральні схеми, конструктивна реалізація.

3 Класифікація систем пам’яті: постійні запам’ятовувальні пристрої, ОЗП (статичний та динамічний), принципи зберігання інформації, ПЛІС, ВІС пам’яті, адресні, інформаційні та керувальні сигнали, що подаються на ВІС пам’яті, доступ до пам’яті, ємність ВІС пам’яті.

ІІ З дисципліни **“Інформатика”**:

1 Призначення обчислювальних систем, задачі, які можуть розв’язувати обчислювальні системи.

2 Подання даних, які обробляються в обчислювальних системах: подання даних у різних системах числення, подання даних з плаваючою та фіксованою точками, подання даних зі знаком.

3 Алгоритмізація задач, які розв’язуються обчислювальними системами, складання структурних схем алгоритмів розв’язуваних задач.

4 Мови високого рівня *Delphi*, $C^{++}$.

5 Мати навички складання та налагодження програм, які мають розгалуження та цикли, написаних мовами високого рівня.

ІІІ З дисципліни **“Дискретна математика”**:

1 Позиційні системи числення – двійкова, десяткова, шістнадцяткова, двійкова-десяткова, – перехід від одної системи числення до іншої.

2 Алгебра логіки – поняття логічної змінної та логічної функції, закони алгебри логіки, мінімізація логічних функцій методом Квайна та методами координатних діаграм.

# МОДУЛЬ 1.
# Частина І ВУЗЛИ ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА МІКРОПРОЦЕСОРНИХ СИСТЕМ

## 1 ОБЧИСЛЮВАЛЬНІ ТА МІКРОПРОЦЕСОРНІ СИСТЕМИ

### 1.1 Основні визначення

**Вхідний контроль:**

1 Які засоби обчислювальної техніки Ви знаєте?
2 Які задачі Ви розв'язували за допомогою цих засобів?
3 Якими мовами програмування Ви пишете програми?
4 Опишіть відомі Вам великі інтегральні схеми (ВІС).
5 За якими технологіями виготовлені відомі Вам ВІС?

**Обчислювальна техніка (ОТ)** (*computer science, computing machinery*) – це систематизована сукупність наукових дисциплін і галузей техніки, що досліджує обчислювальні машини, принципи їхньої побудови і використання; займається розробленням і тестуванням апаратних засобів для оброблення і зберігання інформації; архітектури обчислювальних систем; різні аспекти програмування, в тому числі питання розроблення і створення будь-якого програмного забезпечення; інформаційні структури; мови програмування тощо.

При цьому обчислювальна техніка розв'язує задачі оптимізації апаратних засобів з точки зору їхньої побудови, енергоспоживання, надійності, вартості тощо.

Крім того, термін "**обчислювальна техніка"** застосовується для описування сукупності обчислювальних засобів, призначених для автоматизації розв'язання будь-яких задач керування процесами і збирання даних, таких як мікропроцесор, абонентські термінали, пристрої обміну даними тощо, а також окремі функціональні вузли цих засобів.

**Електронна обчислювальна машина (ЕОМ), комп'ютер** – комплекс технічних засобів, призначених для автоматизованого оброблення інформації в процесі розв'язання обчислювальних та інформаційних задач.

Під **обчислювальним пристроєм** зазвичай розуміють будь-який пристрій оброблення цифрової інформації: електронна обчислювальна машина (ЕОМ), мікропроцесор, персональний комп'ютер (ПК), мікропроцесорна система тощо.

**Обчислювальна система (ОС)** – це сукупність програм та технічних засобів, призначених для оброблення інформації.

**Архітектура обчислювальної системи** – це загальна логічна організація обчислювальної системи, яка визначає процес оброблення даних у ній та поєднує методи кодування даних, склад, призначення, принципи взаємодії технічних засобів і програмного забезпечення.

**Процесор** – це функціональний пристрій, що забезпечує конкретне застосування сукупності команд.

**Мікропроцесор (МП)** – це оброблювальний та керувальний цифровий пристрій, виконаний за технологією великих інтегральних схем (ВІС), який під програмним керуванням здатний виконувати оброблення інформації, а саме арифметичні та логічні операції, введення-виведення та зберігання інформації, а також приймати рішення.

За типом архітектури розрізняють МП з фоннеймановською архітектурою і МП з гарвардською архітектурою.

За типом побудови мови програмування розрізняють *CISC*-процесори (*Complete Instruction Set Computing*) з повним набором команд та *RISC*-процесори (*Reduced Instruction Set Computing*) – зі зменшеним набором команд.

Сучасні мікропроцесори мають ознаки, як *CISC*, так і *RISC*-архітектури.

**Мікропроцесорна система (МПС)** – це багатофункційна програмно-керована система обробки інформації, яка складається з підсистеми центрального процесора, підсистеми пам'яті та підсистеми введення-виведення, об'єднаних інформаційними каналами. Мікропроцесорні системи будують на мікропроцесорних комплектах і поділяють на МПС керувальні, обчислювальні, контрольно-вимірювальні, збирання даних.

Комп'ютер сам по собі є також мікропроцесорною системою.

Різниця між обчислювальною системою та мікропроцесорною системою є тільки у масштабах розв'язуваних задач, кількості та складності обладнання.

Класичним варіантом ОС є багатокомп'ютерний або багатопроцесорний комплекс.

Вимоги до сучасних ОС – це, перш за все, здатність до інформаційного обслуговування користувачів, сервіс та якість цього обслуговування. Для суперкомп'ютерів, які будуються на основі багатопроцесорних систем, найбільш важливими вимогами є продуктивність та надійність.

Обчислювальні системи можуть будуватись на базі кількох комп'ютерів або на базі кількох процесорів.

У багатокомп'ютерних системах інформаційно комп'ютери взаємодіють один з одним по мережі, і кожен з них може працювати під керуванням своєї операційної системи, що знижує динамічні характеристики та надійність ОС.

Багатопроцесорні ОС працюють під керуванням однієї операційної системи, мають більш високу швидкодію та надійність.

**Універсальні мікропроцесори** призначені для застосування в обчислювальних системах та керувальних персональних комп'ютерах, робочих станціях, серверах, у суперкомп'ютерах. Основною характеристикою універсальних мікропроцесорів є наявність розвинених пристроїв для ефективної реалізації операцій з плаваючою точкою над 64-розрядними і більш довгими операндами та можливість підімкнення розвиненої системи введення-виведення інформації, яка забезпечує різноманітні види зовнішніх пристроїв. Вони призначені, в основному, для розв'язання науково-технічних, економічних, математичних, інформаційних та інших задач, які характеризуються складністю алгоритмів та великим обсягом даних, що обробляються.

**Процесори цифрового оброблення сигналів** (сигнальні процесори) – розраховані на оброблення у реальному часі цифрових потоків. Сучасні сигнальні процесори здатні виконувати цілочислові операції та операції з плаваючою точкою над 32-40-розрядними операндами.

Сигнальні процесори апаратно підтримують фільтрацію та згортання сигналів, обчислення кореляційної функції двох сигналів, пряме та обернене Фур'є-перетворення сигналу тощо.

**Медійні та мультимедійні мікропроцесори** забезпечують апаратну підтримку оброблення аудіосигналів, графічної інформації, відеозображень тощо. Вони застосовуються у мультимедіакомп'ютерах, приставках для ігор, побутовій техніці тощо.

**Мікроконтролер** є інтегрована на одному кришталі ВІС мікропроцесорна система, яка складається з одного або кількох мікропроцесорів, іноді з різною архітектурою та призначенням, підсистеми пам'яті та набору периферійних пристроїв. Мікроконтролери знаходять широке застосування у телекомунікаціях (комунікаційні мікроконтролери), засобах автоматизації, апаратурі зв'язку, контрольно-вимірювальній техніці тощо.

**Мікропроцесорний комплект** – це сукупність інтегральних мікросхем різного ступеня інтеграції: ВІС мікропроцесорів, ВІС пристроїв пам'яті, ВІС пристроїв введення-виведення, ВІС контролерів зовнішніх пристроїв, службових інтегральних схем – тактовий генератор, модулі, з яких складається інтерфейс з пам'яттю, контролери шин, арбітри шин тощо, які є сумісні за своїми електричними, інформаційними та конструктивними параметрами. Мікропроцесорні комплекти призначені, в основному, для побудови мікропроцесорних систем.

**Контролером** називається пристрій, часто вбудований, який призначений для керування технологічним пристроєм або процесом. Контролер будується на основі одного або кількох процесорів або мікроконтролера.

**Трансп'ютери** – це мікрокомп'ютери з власною внутрішньою пам'яттю та каналами (лінками) для підключення до інших трансп'ютерів з метою створення багатопроцесорних систем та паралельних обчислювальних систем.

**Контрольні питання:**

1 З яких пристроїв можуть складатись ОС?

2 Які вимоги пред'являються до сучасних ОС?

3 Які задачі можуть вирішуватись за допомогою обчислювальної техніки?

4 Як можна класифікувати МПС за їх функціональним призначенням?

5 Чим різняться між собою мікропроцесори, мікроконтролери та мікропроцесорні системи?

6 Що таке мікропроцесорний комплект?

**Контрольні питання підвищеної складності:**

1 Назвіть відомі Вам моделі ЕОМ; в якому році вони були виготовлені?

2 Назвіть відомі Вам моделі ПК; в якому році вони були виготовлені?

## 1.2 Принципи побудови та функціонування обчислювальних систем

**Вхідний контроль:**

1 Які задачі розв'язують обчислювальні системи?
2 У чому різниця між поняттями "кількість інформації" та "обсяг даних"?
3 Які задачі розв'язують інформаційні системи?

### *1.2.1 Архітектура обчислювальних систем*

За архітектурою обчислювальні системи поділяються на **однорідні** та **неоднорідні**.

**Однорідні** обчислювальні системи побудовані на базі однотипних комп'ютерів або процесорів. Вони використовують стандартні набори технічних, програмних засобів, стандартні протоколи сполучення пристроїв.

**Неоднорідні** обчислювальні системи мають у своєму складі різні типи комп'ютерів або процесорів, і при побудові такої системи треба враховувати їхні різні технічні та функціональні характеристики, що ускладнює їх створення та обслуговування. Прикладом неоднорідної інформаційно-обчислювальної системи є мережа Інтернет.

Обчислювальні системи працюють у двох режимах – **оперативному** та **неоперативному**.

**Оперативні** системи працюють у реальному масштабі часу, в них реалізується оперативний режим обміну інформацією – відповіді на запитання надходять негайно.

У **неоперативних** системах можливий режим затриманої відповіді, коли результати запиту можна отримати з затримкою.

Розрізняють обчислювальні системи з **централізованим** та **децентралізованим керуванням**. У першому випадку керування виконує окремий комп'ютер або процесор, у другому – кожний компонент може брати керування на себе і вони є рівноправні.

Обчислювальні системи можуть бути розподіленими територіально, зосередженими, структурно однорівневими, тобто мати один загальний рівень обробки даних, та багаторівневими, ієрархічними структурами; у ієрархічних структурах комп'ютери або процесори розподілені за різними рівнями оброблення інформації і кожний з них може ініціалізуватись для виконання певних функцій.

За способом організації оброблення даних багатопроцесорні системи можуть бути **конвеєрними** (магістральними), **векторними** або **матричними**.

У **конвеєрних** багатопроцесорних системах кожен процесор одночасно виконує різні операції над послідовним потоком оброблюваних даних. За прийнятою класифікацією такі системи є системами з множинним потоком команд та поодиноким потоком даних (МКПД) – *Multiple Instruction Single Data, MISD*. Структура *MISD* показана на рис. 1.1.

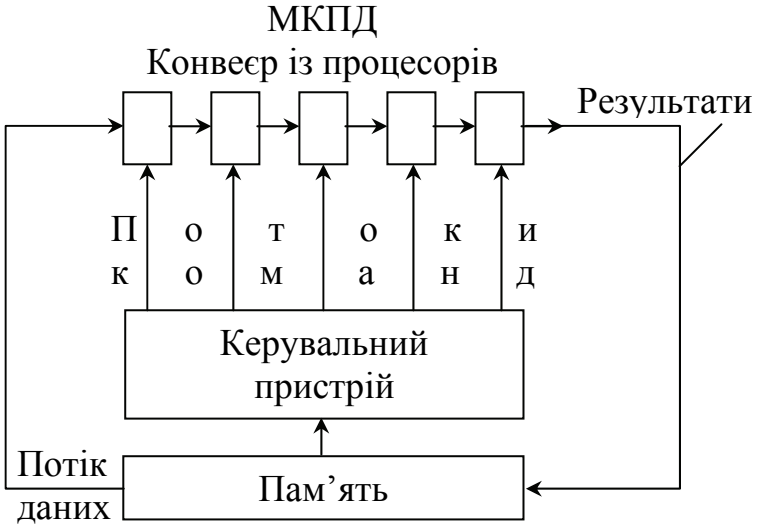



Рисунок 1.1 – Конвеєрна багатопроцесорна система
(множинний потік команд та поодинокий потік даних)

У **векторних** багатопроцесорних системах усі процесори одночасно виконують одну команду над різними даними – поодинокий потік команд з множинним потоком даних (ПКМД) – *Single Instruction Multiple Data, SIMD*. Структура *SIMD* показана на рис. 1.2. Принцип *SIMD* використовується також для підвищення продуктивності мікропроцесорів – **суперскалярні** (векторні) МП *Pentium* III, *Pentium* 4, *Power PC* тощо.

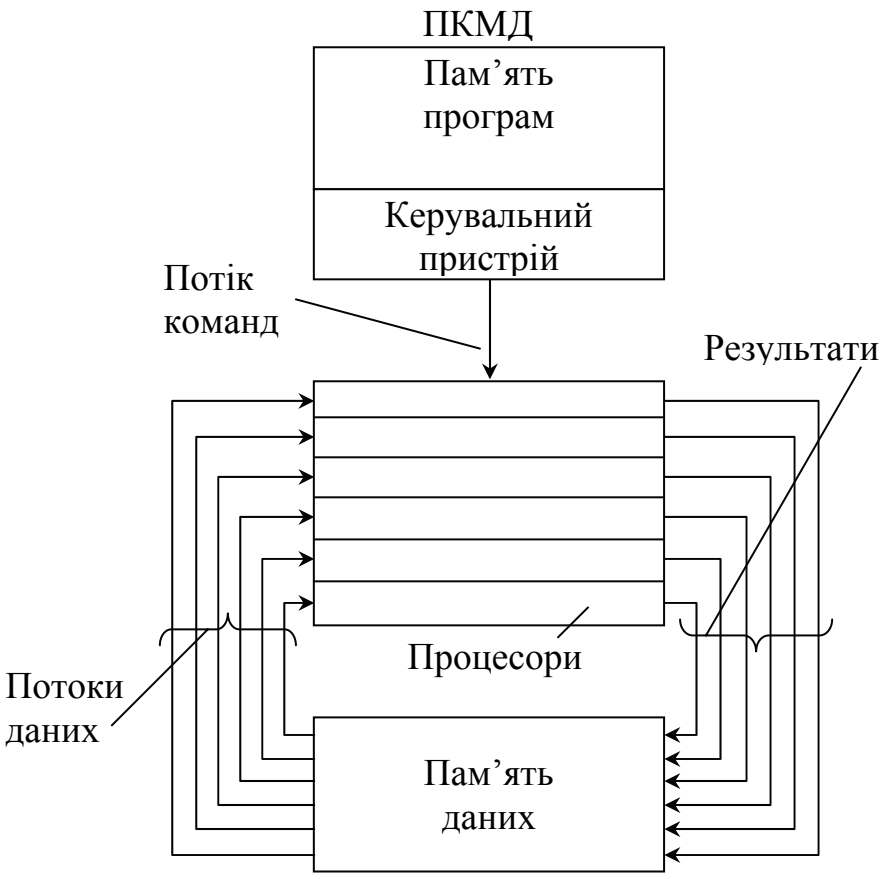



Рисунок 1.2 – Векторна багатопроцесорна система
(поодинокий потік команд та множинний потік даних)

У **матричних** багатопроцесорних системах кожний мікропроцесор одночасно виконує різні операції над послідовними потоками оброблюваних даних – множинний потік команд з множинним потоком даних (МКМД) – *Multiple Instruction Multiple Data, MIMD*. Структура *MIMD* показана на рис. 1.3.

Структура однопроцесорної системи (ПКПД) – *Single Instruction Single Data, SISD* показана на рис. 1.4.

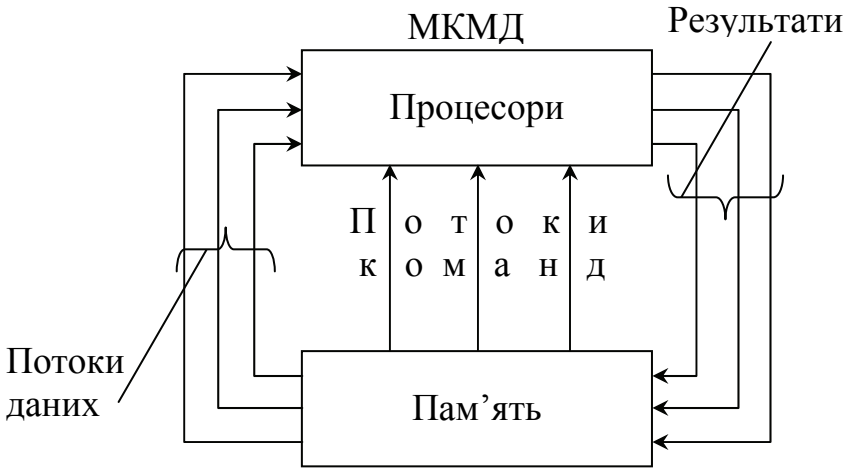



Рисунок 1.3 – Матрична багатопроцесорна система
(множинні потоки команд та даних)

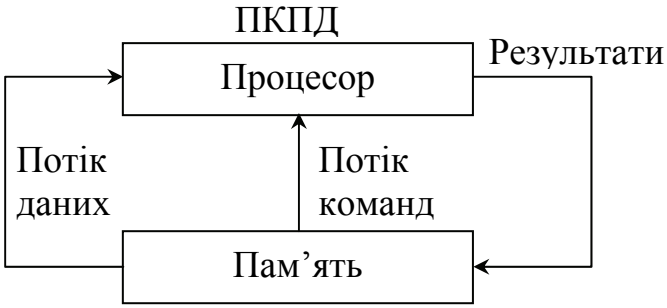



Рисунок 1.4 – Однопроцесорна система
(поодинокі потоки команд та даних)

**Контрольні питання:**

1 Що є архітектурою обчислювальних систем?
2 Які режими роботи обчислювальних систем Ви знаєте?
3 Які різновиди багатопроцесорних систем Ви знаєте?

**Контрольні питання підвищеної складності:**

1 За яким принципом працює конвеєрна багатопроцесорна система?
2 За яким принципом працює векторна багатопроцесорна система?
3 За яким принципом працює матрична багатопроцесорна система?

### *1.2.2 Класифікація комп'ютерів (Для поглибленого вивчення)*

**Комп'ютери** можна класифікувати за багатьма ознаками, що призводить іноді до різного тлумачення термінів.

За призначенням комп'ютери можна поділити на універсальні, проблемно-орієнтовані та спеціалізовані.

**Універсальні комп'ютери** використовують для розв'язання задач, які характеризуються складністю алгоритмів, великим обсягом оброблюваної інформації та вимагають високої продуктивності обчислювальних засобів. Це задачі математичні, інженерно-технічні, інформаційні, космічної галузі тощо.

**Проблемно-орієнтовані комп`ютери** призначені для керування телекомунікаційними пристроями, роботами, різними технологічними процесами і потребують меншої продуктивності та обчислювальних ресурсів. До них можна віднести робочі станції, які призначені для виконання графічних, інженерних, проектувальних, видавницьких робіт тощо.

**Спеціалізовані комп'ютери** призначені для керування простими технічними пристроями, процесами, використовуються для спрощення та узгодження роботи вузлів обчислювальних систем.

За розміром і обчислювальною потужністю комп'ютери можна поділити на суперкомп'ютери, великі, малі, мікрокомп'ютери.

**Суперкомп'ютери** призначені для розв'язання задач керування складними оборонними комплексами, моделювання екологічних систем, прогнозування погоди тощо.

**Великі комп'ютери** часто називають також майнфреймами. Вони призначені для розв'язання науково-технічних задач, роботи з великими базами даних, керування мережами та їхніми ресурсами.

**Малі комп'ютери** орієнтовані на використання у керувальних обчислювальних комплексах, системах автоматизованого проектування, моделювання та штучного інтелекту.

**Мікрокомп'ютери** можна поділити на багатокористувацькі, які оздоблені кількома відеотерміналами, та персональні комп'ютери, однокористувацькі, які задовольняють вимоги універсальності використання.

Структурна схема персонального комп'ютера показана на рис. 1.5.

Мікропроцесор є центральним блоком персонального комп'ютера. Він призначений для керування роботою усіх блоків ПК та виконання арифметичних і логічних операцій над даними.

**До складу мікропроцесора входять:**

1 Арифметично-логічний пристрій (АЛП), який призначений для виконання усіх арифметичних та логічних операцій над числовими та символьними даними. У деяких моделях ПК для прискорення виконання операцій до АЛП підключається додатковий математичний співпроцесор.

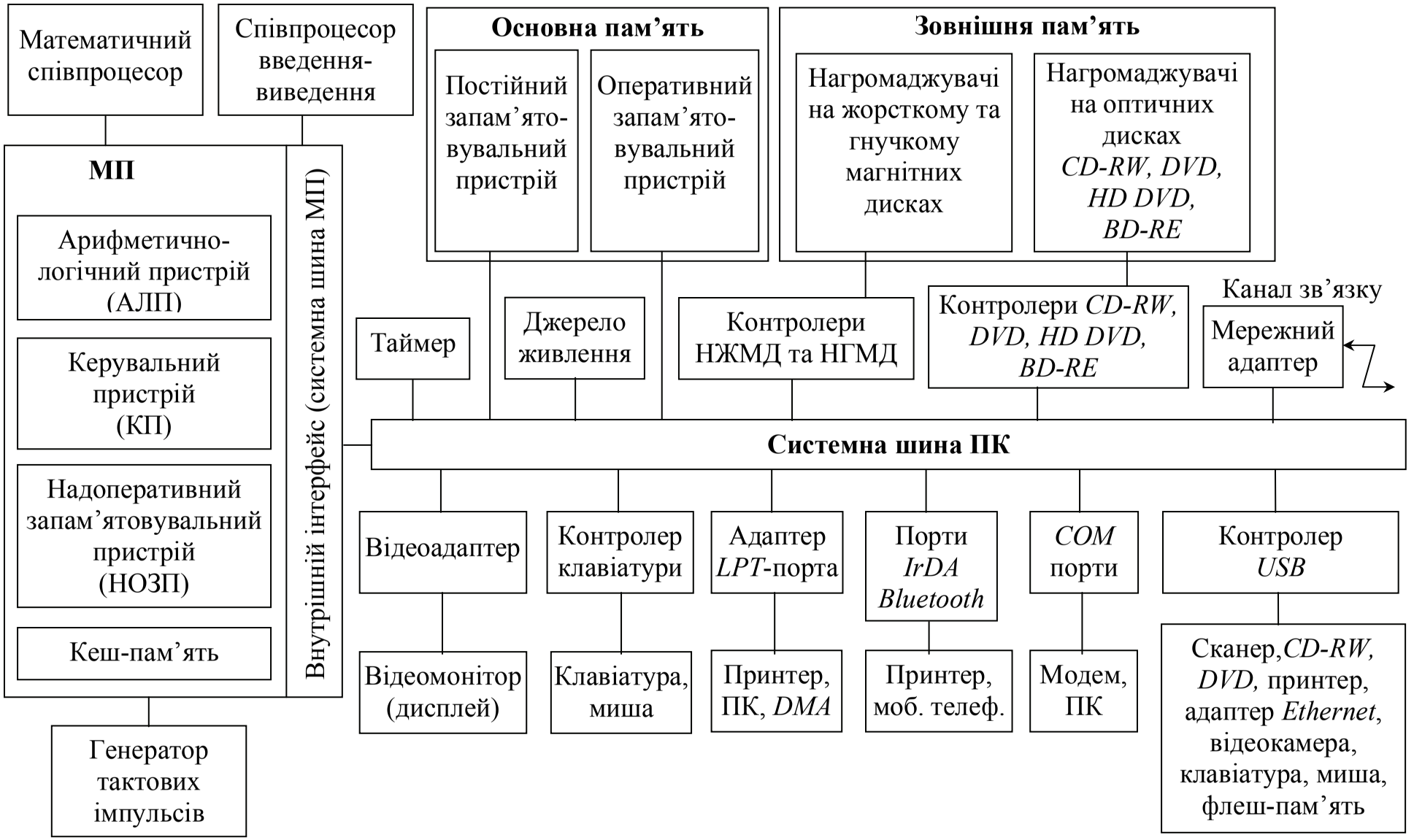



Рисунок 1.5 – Структурна схема ПК

2 Керувальний пристрій (КП) формує та подає на всі блоки ПК у визначені моменти часу певні сигнали керування, які обумовлені специфікою виконуваної операції і результатами попередніх операцій; формує адреси пам’яті і передає їх до відповідних пристроїв та вузлів комп’ютера; опорну послідовність імпульсів керувальний пристрій отримує від генератора тактових імпульсів.

3 Надоперативний запам’ятовувальний пристрій (НОЗП) призначений для короткочасного зберігання, записування та видавання інформації безпосередньо у найближчі такти роботи мікропроцесора. НОЗП побудований на регістрах.

4 Кеш-пам’ять команд та даних – це буферна пам’ять між мікропроцесором та оперативним запам’ятовувальним пристроєм або мікропроцесором та зовнішніми нагромаджувачами.

5 Інтерфейсна система мікропроцесора призначена для сполучення та зв’язку з іншими пристроями ПК; вона вміщує внутрішній інтерфейс мікропроцесора, буферні запам’ятовувальні регістри і схеми керування портами введення-виведення та системною шиною ПК.

6 Порт введення-виведення (*I/O port*) – це апаратура сполучення, яка дозволяє підключати до мікропроцесора інший пристрій ПК або периферійні пристрої.

7 Генератор тактових імпульсів генерує послідовність електричних імпульсів, частота яких визначає тактову частоту ПК. Частота генератора тактових імпульсів є однією з основних характеристик ПК і значною мірою визначає його продуктивність; кожна операція у ПК виконується за фіксовану кількість тактів.

Системна шина – це основна інтерфейсна система комп’ютера, яка забезпечує сполучення та зв’язок усіх його вузлів між собою.

Системна шина має у своєму складі:

− шину даних (ШД), яка створюється зі з’єднувальних ліній та схем сполучення для паралельного передавання усіх розрядів числового коду операнда;

− шину адреси (ША), яка вміщує з’єднувальні лінії та схеми сполучення для паралельного передавання коду адреси комірок основної пам’яті та порту введення-виведення зовнішнього пристрою;

− шину керування (ШК), яка складається зі з’єднувальних ліній та схем сполучення для передавання керувальних сигналів в усі блоки ПК;

− шину живлення, яка вміщує також з’єднувальні лінії та схеми сполучення для підключення блоків ПК до системи енергоживлення.

Системна шина забезпечує обмін даними між мікропроцесором та основною пам’яттю, між мікропроцесором та портами введення-виведення зовнішніх пристроїв, між основною пам’яттю та портами введення-виведення зовнішніх пристроїв у режимі прямого доступу до пам’яті.

Усі зовнішні пристрої ПК через їх порти введення-виведення підключаються до системної шини або безпосередньо, або через контролери (адаптери). Керування системною шиною ПК здійснюється частіш за все через

додаткову мікросхему – контролер шини, який формує основні сигнали керування.

Основна пам’ять призначена для керування та оперативного обміну інформацією з іншими блоками ПК. Основна пам’ять складається з постійного (ПЗП) та оперативного (ОЗП) запам’ятовувальних пристроїв:

− ПЗП (*ROM – Read Only Memory*) призначений для зберігання програмної та довідкової інформації, яка не змінюється; дозволяє тільки зчитувати інформацію, яка в ньому зберігається;

− ОЗП (*RAM – Random Access Memory*) динамічного та статичного типу призначений для оперативного записування, зберігання та зчитування інформації (програм та даних), які безпосередньо беруть участь в інформаційно-обчислювальному процесі.

Вимогами до оперативної пам’яті є її висока швидкодія та можливість доступу до будь-якої комірки пам’яті окремо. ОЗП є енергозалежною.

Крім основної пам’яті у ПК є енергонезалежна пам’ять *CMOS RAM (Complementary Metall-Oxide Semiconductor RAM*), яка живиться від свого акумулятора; у ній зберігається інформація про апаратну конфігурацію ПК, яка перевіряється при кожному включенні ПК.

Зовнішня пам’ять є зовнішнім пристроєм ПК і використовується для зберігання усього програмного забезпечення ПК. Найбільш розповсюдженими видами зовнішньої пам’яті є нагромаджувачі на жорстких (НЖМД) та гнучких (НГМД) магнітних дисках, нагромаджувачі на оптичних дисках *CD-ROM (Compact Disk Read Only Memory*) та *CD-RW (Re Writable CD*), які дозволяють багаторазовий запис інформації.

У сучасних ПК для установлення програмного забезпечення, виконання певних додатків, наприклад, ігор, прослуховування музики, перегляду фільмів використовується *DVD (Digital Versatile Disk*) – універсальні цифрові диски, *HD (High Density DVD*) – диски *DVD* з високою щільністю запису на *BD (Blu-ray Disk*), які отримали таку назву від кольору променя лазера, який зчитує інформацію з диску.

Флеш (*Flash*) – пам’ять з багаторазовим електричним стиранням та записуванням інформації.

Призначення зовнішніх нагромаджувачів – зберігання великих обсягів інформації, записування та видавання її по запиту в ОЗП. Зовнішня пам’ять є енергонезалежною.

Джерело живлення – це блок автономного та мережного енергоживлення.

Таймер – вбудований у ПК електронний годинник реального часу, який дозволяє автоматично видавати поточний момент часу (рік, місяць, години, хвилини, секунди та частки секунд). Таймер підключається до автономного джерела живлення – акумулятора, і при відімкнені ПК від мережі продовжує працювати.

Зовнішні пристрої (ЗП) ПК забезпечують його взаємодію з зовнішнім інформаційним середовищем: користувачами, об’єктами керування, іншими комп’ютерами тощо.

До зовнішніх пристроїв відносяться:
- зовнішня пам’ять ПК;
- пристрої введення інформації;
- пристрої виведення інформації;
- діалогові засоби користувача;
- засоби зв’язку та телекомунікацій.

До пристроїв введення інформації відносяться:
- клавіатура;
- сканери;
- миша тощо.

До пристроїв виведення інформації відносяться:
- принтери;
- графобудувачі тощо.

*LPT*-порт (*Line Printer Terminal*) було введено у ПК для підключення принтера, але у розширеному режимі він може використовуватись як двоспрямований порт обміну даними і, як порт прямого доступу до пам’яті ПДП.

Контролер *USB (Universal Serial Bus*) дозволяє підключення до універсальної послідовної шини *Flach*-пам’яті, цифрових телевізійних камер, 10-Мбіт адаптерів *Ethernet*, багатопортових конверторів (*USB to COM/USB-to-LBT*), ноутбуків, клавіатур, цифрових фотоапаратів, *CD-ROM, CD-RW*, цифрових відеокамер тощо.

Деякі периферійні пристрої, найчастіше принтери, можуть мати безпроводове з’єднання з ПК через інфрачервоний послідовний порт *IrDA (Infrared Data Assotiation*).

Радіоінтерфейс *Bluetooth* використовується для передавання інформації на невеликі відстані між ПК, їх периферійними пристроями, мобільними телефонами тощо.

Пристрої зв’язку та телекомунікацій використовуються для зв’язку з приладами та іншими засобами автоматизації (цифро-аналогові та аналого-цифрові перетворювачі, адаптери тощо), а також для підключення ПК до каналів зв’язку з метою обміну інформацією по мережі, у складі обчислювальних систем (мережні інтерфейсні плати та карти, модеми, мультиплексори передавання даних).

До діалогових засобів відносяться відеотермінал, керований відеокартою, та пристрої мовного введення-виведення інформації (мікрофони та акустичні системи), керовані звуковою картою. До засобів мультимедіа відносяться мікрофони, відеокамери, акустичні та відеосистеми тощо.

Математичний співпроцесор використовується для виконання операцій над двійковими числами з фіксованою та плаваючою точками, для обчислення трансцендентних, у тому числі тригонометричних функцій. Співпроцесор працює паралельно з основним мікропроцесором, що значно прискорює виконання операцій. Сучасні моделі мікропроцесорів часто інтегрують співпроцесор до своєї структури.

Співпроцесор введення-виведення працює паралельно з мікропроцесором і обслуговує кілька зовнішніх пристроїв (відеотермінал, принтер, НЖМД, НГМД тощо) та звільняє процесор від оброблення процедур введення-виведення, у тому числі реалізує режим прямого доступу до пам'яті.

Контролер прямого доступу до пам'яті (*DMA – Direct Memory Access*) забезпечує обмін між зовнішніми пристроями та оперативною пам'яттю без участі мікропроцесора, що підвищує ефективність роботи ПК у цілому. Процесор під час обміну даними між зовнішніми пристроями та оперативною пам'яттю може обробляти інші дані або навіть вирішувати іншу задачу.

Контролер переривань обслуговує запити переривань від зовнішніх пристроїв за допомогою процедур переривань. Контролер приймає запит, визначає пріоритет цього запиту та визначає права конкретного зовнішнього пристрою на його обслуговування. Він посилає у мікропроцесор сигнал переривання та повідомляє його про номер або адресу обслуговуваного пристрою або про адресу першої команди підпрограми оброблення цього запиту. Мікропроцесор призупиняє виконання поточної програми та виконує підпрограму обслуговування переривання. Після завершення підпрограми мікропроцесор повертається до виконання перерваної програми.

Контролер переривань є програмований. Режим переривань використовується у ПК постійно, усі процедури введення-виведення виконуються за запитами зовнішніх пристроїв на переривання. Системний таймер здійснює переключення задач, вирішуваних на ПК у багатозадачному режимі, кожні 2 мс.

Слід зазначити особливості термінології щодо комп'ютерів, об'єднаних в обчислювальні та інформаційні мережі.

Використання комп'ютерів у мережах визначається їх програмним забезпеченням та установленим додатковим устаткуванням.

**Мережні комп'ютери** – це спрощені мікрокомп'ютери, які забезпечують роботу у мережі та доступ до мережних ресурсів. Вони часто оснащені відеотерміналом, клавіатурою, а іноді не мають навіть жорсткого диску. Такі комп'ютери можуть спеціалізуватись на виконанні певних робіт: захисту мережі від несанкціонованого доступу, перегляду мережних ресурсів, організації електронної пошти тощо.

**Робочі станції** у мережах – це персональні комп'ютери, які об'єднані у мережу і є вузлами цієї мережі.

**Сервер** – це багатокористувацький потужний мікрокомп'ютер, призначений для оброблення запитів від усіх робочих станцій мережі.

**Проксі-сервер** – це робоча станція, на якій установлено спеціалізоване програмне забезпечення для безпосереднього зв'язку локальних мереж з Інтернет.

Наведена термінологія складалась протягом кількох десятків років і відображає стани розвитку обчислювальних систем та мереж на базі електронних обчислювальних машин (ЕОМ). Розвиток потужних мікропроцесорів призвів до того, що в інформаційних та телекомунікаційних мережах як кінцеві вузли найчастіше використовуються персональні

комп'ютери. Для розв'язування складних задач керування та моделювання актуальним є застосування великих та суперкомп'ютерів.

Найбільш перспективною технологією побудови великих та суперкомп'ютерів є кластерні обчислювальні системи – групи високоефективних процесорів, об'єднаних у кластери. Їх перевагою є можливість гнучкого регулювання необхідної потужності системи шляхом підключення до кластера звичайних серійних серверів за допомогою спеціальних апаратних та програмних інтерфейсів. Кластерізація дозволяє маніпулювати групою серверів як одною системою, забезпечувати доступ будь-якого сервера до будь-якого блока оперативної та дискової пам'яті. Кластерні системи характеризуються спрощеним керуванням під операційними системами, наприклад, *Windows* 2000 *Enterprise* фірми *Microsoft*; її компонент *Wolfpack* забезпечує також функції діагностики збоїв та відновлення системи.

Створити високопродуктивні комп'ютери на одному мікропроцесорі неможливо через обмеження, зумовлені кінцевою швидкістю поширення електромагнітних хвиль (300 000 км/с), тому що час розповсюдження сигналу на відстань кілька міліметрів, яка складає розмір сторони МП при швидкодії 100 млрд. операцій за секунду, стає вже відповідним часу виконання однієї операції. Тому суперкомп'ютери створюються як високопаралельні системи.

У 2005 році Національна академія наук України створила в Інституті кібернетики ім. В.М. Глушкова (Київ) суперкомп'ютерний обчислювальний центр (СОЦ) на базі двох високопродуктивних кластерних систем СКІТ-1 та СКІТ-2 – 32-процесорного кластера на процесорах *Intel Xeon* та 64-процесорного – на процесорах *Intel Itanium2*.

**Кластерна система** СКІТ-1 (суперкомп'ютер для інформаційних технологій) – це 32-процесорний 16-вузловий кластер на основі 32-розрядних мікропроцесорів *Intel Xeon*, з піковою потужністю не менше 170 Гігафлопс (мільярдів операцій з плаваючою точкою за секунду) та можливістю підвищення продуктивності до 0,5-1 Терафлопс (трильйони операцій з плаваючою точкою за секунду). Кластер працює під керуванням головної операційної системи *ALT Linux*.

Система СКІТ-2 – це 64-процесорний 32-вузловий кластер на основі мікропроцесорів *Intel Itanium2* з частотою 1,4 ГГц з розрядністю 64 біти і можливістю виконувати обчислення із 128 та 256-бітовою інформацією.

Пікова потужність кластера до 300 Гігафлопс з можливістю її підвищення до 2-2,5 Гігафлопс, підсистемою пам'яті 1 Гбайт і можливістю нарощування потужності до 10-15 Гбайт. Кластер працює під керуванням головної операційної системи *Red Hat Enterprise*.

Основними перевагами кластерних суперкомп'ютерних систем є:

- висока сумарна потужність;
- висока надійність системи;
- найкраще відношення потужність/вартість;
- можливість динамічного перерозподілу навантажень між серверами;
- легка масштабованість за рахунок підключення додаткових серверів;
- зручність керування та контролю роботи системи.

Суперкомп'ютери можуть мати також модифіковані структури – *MMISD*, паралельно-конвеєрна *MISD*-структура у суперкомп'ютері Ельбрус; *MSIMD*, паралельно-векторна модифікація, яка застосована у суперкомп'ютері *Cray*2.

Слід зазначити, що багато ідей, втілених протягом десятків років у розроблених у різних країнах ЕОМ та комп'ютерах, знайшли застосування у сучасних мікропроцесорах.

**Контрольні питання:**

1 З яких вузлів складається мікропроцесор?

2 З яких вузлів складається ПК?

3 З якою метою будуються кластерні суперкомп'ютери?

4 Які сучасні операційні системи керують кластерними системами?

5 Які новітні розробки українських вчених в області суперкомп'ютерів Ви знаєте?

6 Яку потужність має сучасний суперкомп'ютер, розроблений українськими вченими?

7 Які переваги кластерних суперкомп'ютерних систем Ви знаєте?

8 Які параметри комп'ютера слід враховувати при його виборі?

**Контрольні питання підвищеної складності:**

1 На якій підставі може взаємодіяти персональний комп'ютер з периферійними пристроями через інфрачервоний послідовний порт?

2 На якій саме відстані може взаємодіяти мобільний телефон з ПК через радіоінтерфейс *Bluetooth*?

## 1.3 Принципи побудови та функціонування МПС

**Вхідний контроль:**

1 Перерахуйте підсистеми МПС.

2 Наведіть приклади об'єктів автоматичного контролю й управління.

3 Які пристрої введення та відображення інформації можуть використовуватись у керувальній МПС?

МПС будується за принципами "трьох М" – модульності, магістральності та мікропрограмованості. Модулем називається функціонально, електрично та конструктивно завершений цифровий пристрій, який призначено для виконання задач певного типу: процесорний модуль, модуль пам'яті тощо. Модульний підхід спрощує процес проектування МПС, орієнтованих на конкретні області використання, тобто найбільш ефективні, надійні, економічні.

Магістральний спосіб обміну інформацією в МПС реалізується у вигляді шинної організації, яка здійснює зв'язки між підсистемами МПС шинами (електричними лініями). Магістральність забезпечує регулярність структури МПС, можливість масштабування, змінення конфігурації, мінімізує кількість зв'язків між окремими пристроями. Зазвичай більшість універсальних мікропроцесорів забезпечують при побудові МПС тришинну організацію за

допомогою шин адреси (ША), шини даних (ШД) та шини керування (ШК), які утворюють системну шину.

Мікропрограмне керування може забезпечити найбільшу гнучкість у застосуванні МПС, але частіше використовують командний рівень керування через складність мікропрограмування.

Підсистеми МПС можуть складатися з кількох модулів мікропроцесорів, пам'яті, пристроїв введення-виведення.

Незважаючи на різноманіття МПС різного призначення, усі вони мають подібну структуру й однотипний склад устаткування. Їх характерною рисою є наявність розвиненої периферії або зовнішніх пристроїв: блоків датчиків, блоків керування, комутаторів, пристроїв введення та відображення інформації, наприклад, клавіатура, монітор. Зовнішні пристрої підключаються до системної шини МПС за допомогою її підсистеми введення-виведення за допомогою інтерфейса. На рис. 1.6 показана структурна схема МПС автоматичного контролю і керування технологічним процесом. До її складу входять блок датчиків $Д_1 \ldots Д_m$, власне МПС та блок керування $К_1 ... К_p$. Датчики слугують для вимірювання параметрів стану об'єкта автоматичного контроля та управління і можуть бути, як аналоговими ($Д_1 \ldots Д_n$), так і цифровими ($Д_{n+1} \ldots Д_m$). Аналогові та цифрові сигнали з датчиків підключаються до входів МПС за допомогою комутатора 1, після якого, у разі необхідності, ставиться АЦП.

МПС за заданими алгоритмами обробляє дані про стан процесу і видає цифрові керувальні сигнали через АЦП, якщо це потрібно, на блок керування $К_1 ... К_p$. Підключення керувальних сигналів до блока керування здійснюється за допомогою комутатора 2. МПС може бути побудована на комп'ютерах та мікроконтролерах.

Блок керування може складатись з формувачів аналогових сигналів $К_1 ... К_n$ або цифрових сигналів $К_{n+1} ... К_p$ керування.

Керувальна МПС функціонує таким чином. Комутатор, керований МПС, опитує датчики за адресами, які задаються програмою, й інформація у цифровій формі надходить до МПС. На основі цієї інформації відповідно до програми роботи МПС формується модель стану об'єкта і видається інформація на пристрій відображення для оператора або на виконавчий механізм, наприклад, переключення ліній зв'язку тощо. Таймер формує відліки часу, мітки, які прив'язують процес до внутрішнього часу МПС. Датчик переривань забезпечує переривання поточної програми, наприклад, при виникненні аварійних ситуацій і перехід до програм їхнього оброблення.

Обмін даними між МПС та зовнішніми пристроями може реалізовуватись трьома способами: програмно керованим, за перериваннями та прямим доступом до пам'яті.

Програмно керований спосіб обміну даними ініціюється будь-яким процесором МПС за основною програмою, яка вміщує команди введення-виведення. Перед обміном процесор перевіряє готовність до роботи зовнішніх пристроїв. Цей спосіб є простий, але не забезпечує термінову реакцію МП на готовність зовнішніх пристроїв до обміну. Відповідно, такий програмно

керований спосіб використовується при роботі з повільно діючими різнорідними пристроями.

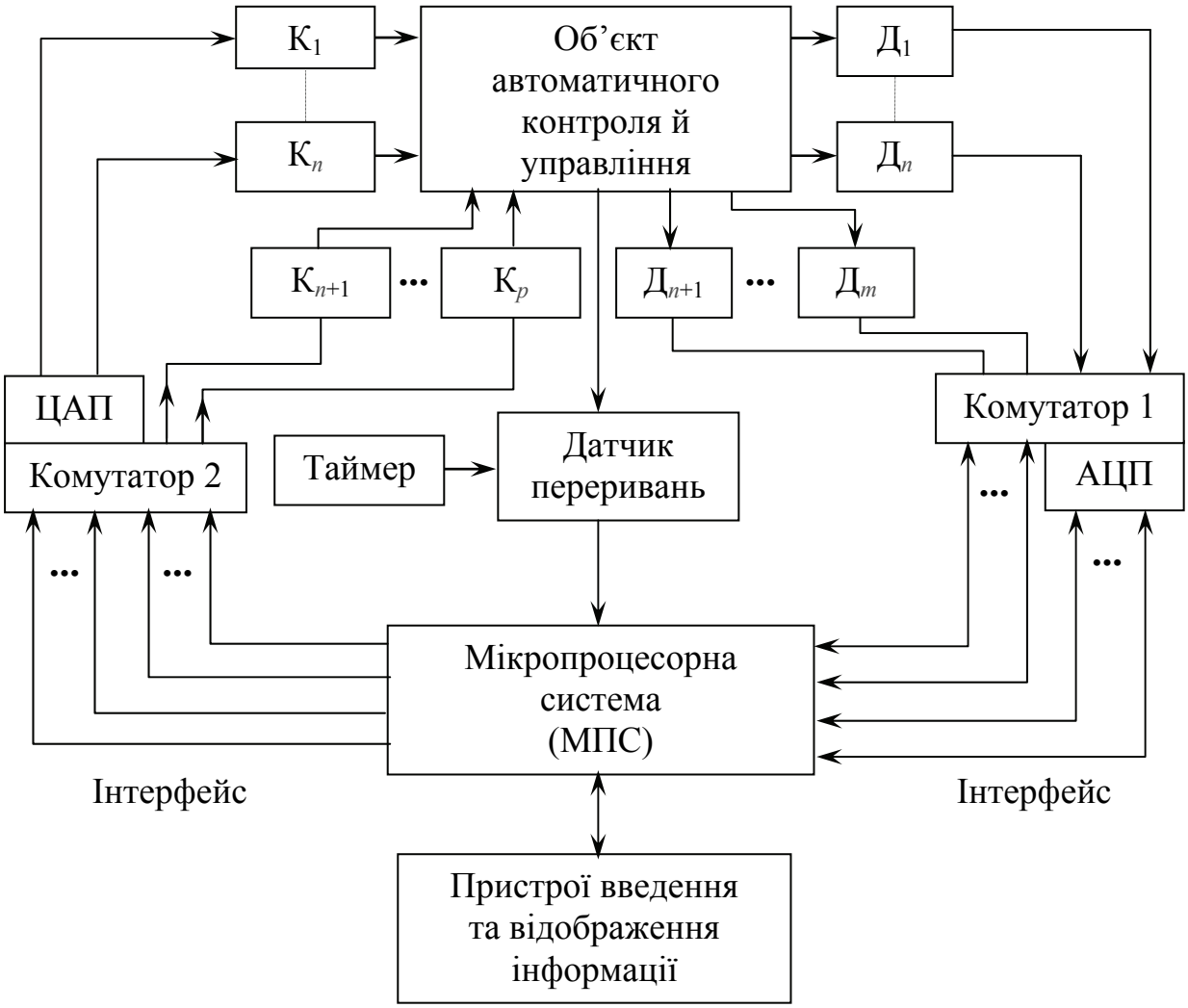



Рисунок 1.6 – Структурна схема МПС автоматичного контролю та управління технологічним процесом

При роботі з асинхронно, по відношенню до процесора, діючими пристроями доцільним є спосіб обміну за перериваннями. Зовнішній пристрій, коли він готовий до роботи, посилає сигнал *INT* – запит на переривання (від *Interrupt* – переривання) на відповідний вхід процесора. За цим сигналом, якщо переривання дозволені, процесор припиняє виконання поточної програми, посилає пристрою сигнал підтвердження *INTA* (від *Interrupt Acknowledge* – підтвердження переривання) і переходить до виконання підпрограми обслуговування переривання від даного пристрою. Метод переривань забезпечує швидку реакцію МПС на запити зовнішніх пристроїв, але також здійснює програмно керований обмін і потребує значної кількості команд на пересилання одного байта даних, оскільки обмін відбувається через процесор.

Найбільш високу швидкодію обміну забезпечує режим прямого доступу до пам'яті ПДП (*DMA – Direct Memory Access*) по каналу зовнішній пристрій-

пам’ять. По запиту захоплення шин *HOLD* (від *Hold* – захоплення) від зовнішнього пристрою процесор завершує поточний машинний цикл виконуваної команди і переводить системні шини адреси, даних і керування у стан високого опору і видає сигнал підтвердження захоплення *HLDA*. Процесор відключається від процесу обміну даними і обмін відбувається з максимальною для всіх учасників обміну швидкістю. Режим ПДП забезпечується за допомогою програмованих контролерів ПДП , які адресують комірки пам’яті, рахують біти та врегульовують конфліктні ситуації, які можуть виникнути при одночасній роботі кількох пристроїв у режимі ПДП.

Як правило, МПС є вузькоспеціалізованою після її розробки та впровадження. Розробка МПС здійснюється стосовно конкретної задачі (алгоритму), як в апаратній частині, так і у програмному забезпеченні. Робоча програма після налаштування МПС завантажується у ПЗП одноразово. Змінення реалізуючого алгоритму потребує зміни апаратної та програмної частини МПС, що не є ефективно.

Слід зазначити, що вбудовані у пристрої керування МПС, які розробляються на мікроконтролерах, допускають багаторазове програмування ПЗП з занесенням різних робочих програм, тобто перепрофілювання.

**Контрольні питання:**

1 Яку роль відіграють АЦП та ЦАП у керувальній МПС?

2 З якою метою у керувальній МПС використовуються комутатори, датчики, керувальні блоки та таймер?

3 Назвіть основні етапи функціонування керувальної МПС.

4 Які вузли входять до складу підсистеми центрального процесорного елементу?

**Контрольні питання підвищеної складності:**

1 Яку роль відіграє датчик переривань?

2 Наведіть приклад режиму переривань у ПК.

3 Чим на Ваш погляд відрізняється МПС збирання даних від керувальної МПС?

### 1.4 Функціонування обчислювального пристрою

**Вхідний контроль:**

1 Що таке обчислювальний пристрій?

2 Які обчислювальні пристрої Ви знаєте?

3 Які периферійні пристрої обчислювального пристрою Ви знаєте?

У 60-х роках минулого століття академік В. М. Глушков довів, що у будь-якому пристрої обробки цифрової інформації можна виділити операційний та керувальний блоки, що це є принцип декомпозиції обчислювального пристрою. Операційний блок складається з регістрів, суматорів та інших пристроїв, які приймають з запам’ятовувального пристрою, зберігають операнди, виконують

над ними операції та видають результати операції у запам'ятовувальний пристрій. У керувальний блок з операційного блоку надходять відомості про знак та інші особливості результату, наприклад, чи дорівнює він нулю тощо. Такі відомості називаються ознаки або прапорці *F* (від *Flags*) результату. На рис. 1.7 показано декомпозицію обчислювального пристрою.

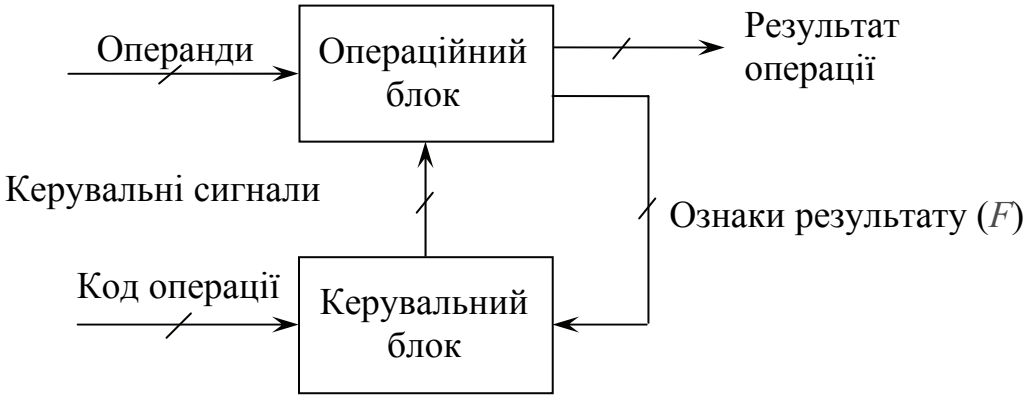



Рисунок 1.7 – Декомпозиція обчислювального пристрою

Процес функціонування пристрою оброблення цифрової інформації, зокрема обчислювального пристрою, складається із послідовності елементарних перетворень, які виконуються за інтервали часу, визначені частотою тактового генератора, такти. До елементарних операцій можуть відноситись зсув даних у регістрі, знаходження оберненого коду, пересилання операнда з одного регістра в інший тощо. Виконання елементарних операцій ініціюється поданням в операційний блок відповідних керувальних сигналів з керувального блока.

Елементарна функціональна операція, виконувана за один такт, називається **мікрооперацією**.

У деякі такти з керувального блоку можуть надходити кілька керувальних сигналів, які ініціюють виконання мікрокоманд у різних вузлах обчислювального пристрою. Сукупність одночасно виконуваних мікроопераций називається **мікрокомандою**.

Послідовність керувальних сигналів визначається сигналами коду операції, які надходять після декодування у керувальний блок з пам'яті, і сигналами, які залежать від операндів та проміжних результатів обчислень.

Операційний блок задається його структурою, тобто складом вузлів та зв'язками між ними і виконуваним операційним блоком набором мікроопераций.

Послідовність мікрокоманд, які забезпечують виконання даної операції, називається **мікропрограмою** даної операції.

Функціонування обчислювального пристрою може описуватись сукупністю мікропрограм, які в ньому реалізуються.

При створенні ВІС МП використовується ідея функціонування обчислювального пристрою. Спрощена схема обчислювального пристрою, або ЕОМ, ПК, МПС, наведена на рис. 1.8. Вона складається з п'яти блоків:

арифметично-логічного пристрою (АЛП) з надоперативним запам’ятовувальним пристроєм (НОЗП), який складається з регістрів, керувального пристрою (КП), підсистеми пам’яті (ЗП) та підсистеми введення-виведення (ПВВ-ПВИВ) і побудована за апаратно-програмним принципом. Апаратна частина виконує обмежений набір простих операцій під керуванням програмного забезпечення. Обмін інформацією між підсистемами здійснюється за допомогою шини даних (ШД), шини адреси (ША) та шини керування (ШК).

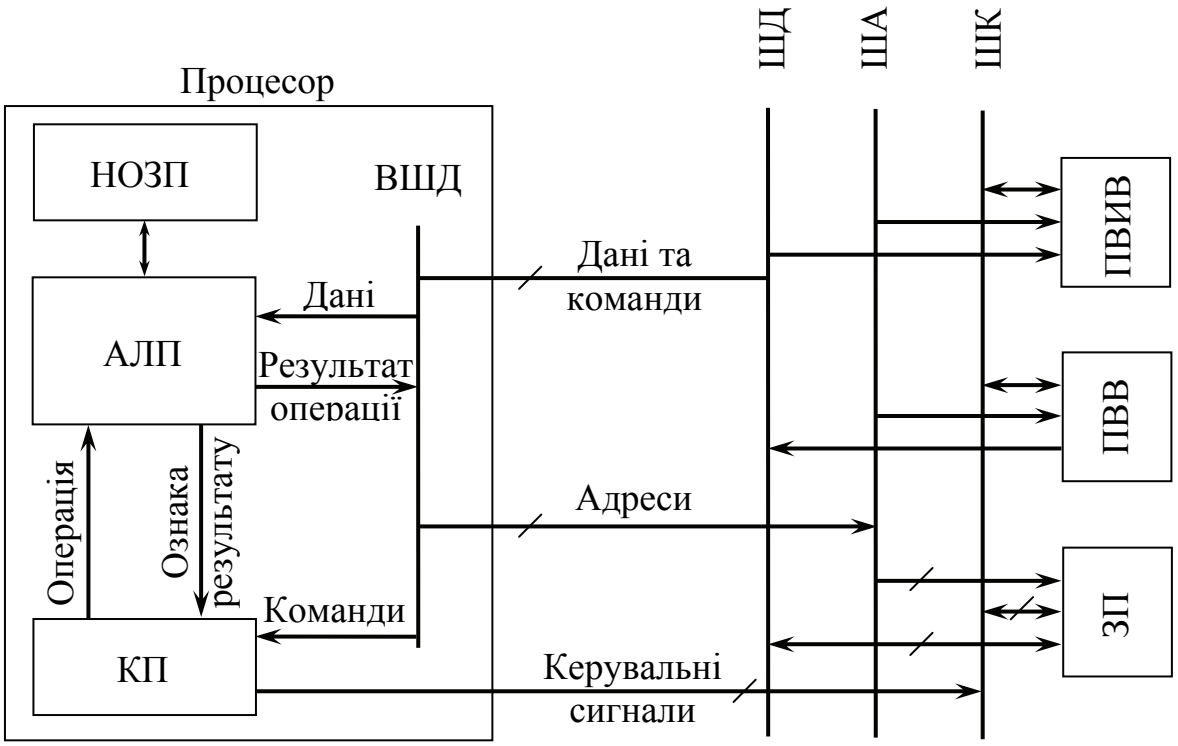



Рисунок 1.8 – Функціональна схема обчислювального пристрою

АЛП, НОЗП та КП входять до складу процесора і взаємодіють за допомогою внутрішньої шини даних (ВШД). Вхідні дані та програма задачі, яка розв’язується, надходять до запам’ятовувального пристрою через пристрій введення.

Виконувана програма складається зі списку команд (інструкцій), спрямованих на виконання серії послідовних дій (операцій). Кожна команда у свою чергу складається з операційної (код операції – КОП) та адресної частини, в якій вказуються самі операнди, або адреси операндів, над якими виконується задана операція. ЗП побудований за адресним принципом – кожна комірка пам’яті ЗП, в якій зберігаються дані або команди, окрім кеш-пам’яті, має постійний номер, адресу або ім’я, за якими можна однозначно до неї звертатись. Функціонування МПС починається з того, що КП центрального процесора видає адресу першої команди програми у ЗП, команда зчитується і направляється у КП. Тут вона декодується і на основі КОП команди виробляються сигнали “Операція”, які налаштовують АЛП на виконання заданої операції. За інформацією адресної частини команди з ЗП до АЛП

надходять операнди. Після завершення операції її результат надходить для зберігання у ЗП (“Результат операції”), а установлювані АЛП ознаки результату надходять до КП, що дає можливість вказувати розгалуження у програмі (приймати рішення). На лінійних ділянках програми далі виставляється адреса наступної команди, при розгалуженнях і циклах – залежно від реалізації заданих умов переходу.

Керувальний пристрій за результатами розшифровування команди виробляє паралельно-послідовну серію керувальних сигналів (КС) для керування, як блоками самого процесора, так і підсистемами МПС.

**Контрольні питання:**

1 Яка роль призначена апаратній частині і яка – програмній при розв’язанні задачі на ЕОМ?

2 Як можна трактувати твердження, що процесор “Приймає рішення”?

3 Який пристрій у складі процесора називається надоперативним і чому?

4 Як використовуються у виконуваній програмі ознаки результату виконаної команди?

**Контрольні питання підвищеної складності:**

1 МПС призначена для оброблення сигналів, які надходять з цифрової ТВ камери. Який спосіб обміну даними між МП та камерою Ви б обрали?

2 Керувальна МПС призначена для керування системою комутації. Який спосіб обміну даними між МП та зовнішніми пристроями Ви б обрали?

# 2 ОПЕРАЦІЇ НАД ДАНИМИ В ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

**Вхідний контроль:**

1 Які позиційні системи числення і їх особливості Ви знаєте?

2 Як перевести число з однієї позиційної системи числення до іншої позиційної системи?

## 2.1 Подання даних в обчислювальних системах

**Вхідний контроль:**

1 Яке мінімальне та максимальне число без знака можна зберігати у 8-розрядному регістрі?

2 Яке мінімальне та максимальне число зі знаком можна зберігати у 8-розрядному регістрі?

3 Запишіть основу двійкової, десяткової та шістнадцяткової систем числення символами своєї ж системи?

4 У мережі Інтернет використовується єдиний для всіх країн світу універсальний 16-розрядний код (*Unicod*). Скільки символів можна відобразити за допомогою цього коду?

В обчислювальних системах виконуються необхідні дії по збиранню, зберіганню, обробленню і передаванню дискретної інформації, яка подана у вигляді потоку цифрових даних. Дані цифрового потоку можуть бути числами зі знаком або без нього, логічними змінними (масивами), адресами (номерами) будь-яких об’єктів і символами алфавітів, що використовуються. Усі ці види даних безпосередньо в обчислювальних системах представлені однаково – у вигляді двійкового коду, однак виконання арифметичних і логічних операцій виконується різними вузлами арифметико-логічного пристрою з використанням різних принципів. Так виконання арифметичних операцій виконується у позиційній двійковій системі числення, а виконання логічних операцій – за законами булевої алгебри.

Крім двійкової системи числення в обчислювальних системах також використовуються вісімкова і шістнадцяткова позиційні системи числення, двійково-десяткова система з кодованим поданням чисел.

**Двійково-десяткова система числення з кодованим поданням чисел** (***BCD*-**система) – це система числення, в якій цифри десяткової системи числення кодуються за допомогою символів двійкової. Це система числення зі звичайною вагою двійкових розрядів – (8–4–2–1), в якій кожна десяткова цифра кодується двійковою тетрадою за допомогою двох символів двійкової системи. Слід зауважити, що двійково-десяткова система має певну надлишковість, тому що для кодування десяткових цифр використовується лише 10 комбінацій із 16 можливих у кожній тетраді. Це відрізняє її від інших позиційних систем числення, в яких надлишковості немає.

Використання цієї системи дозволяє зменшити програмні та апаратні витрати при перетворенні двійкових чисел на десяткові для виведення результатів на пристрої відображення і при введені інформації.

У комп'ютерах використовують упакований та розпакований формати подання двійково-десяткових даних.

Упакований формат передбачає подання двох цифр десяткової системи числення у двох тетрадах одного байта. Наприклад,

$$\overbrace{29_D \leftrightarrow \underbrace{0010}_{\text{Старша тетрада}}\ \underbrace{1001}_{\text{Молодша тетрада}}{}_{BCD}}^{\text{Байт}}$$

Таким чином, діапазон подання десяткових чисел у такій системі сягає від 00 до 99 і такі числа можливо використовувати при виконанні операцій додавання і віднімання з корекцією.

Розпакований формат передбачає подання десяткових чисел у вигляді двохрозрядних десяткових цифр від 00 до 09 і використовується при виконанні операцій додавання, віднімання, множення і ділення з корекцією. У такому форматі при кодуванні вхідних даних у старшій тетраді двійково-десяткового числа повинно бути 0000.

Крім двійково-десяткової системи числення зі звичайною вагою двійкових розрядів в обчислювальній техніці використовується двійково-десяткова система числення з надмірністю 3 (8–4–2–1 + 3)

**Шістнадцяткова системи числення** (*H*-система) – система числення, яка дозволяє легко переходити від неї до двійкової і навпаки. Використовується для подання адреси пристроїв при написанні програм і для виводу результатів на екран. Так лістинг програми виводиться на екран у шістнадцятковій системі числення. Крім того, використання шістнадцяткової системи числення дозволяє зменшити обсяг тексту програми при її написанні мовою Асемблера.

Для подання алфавітно-цифрової інформації, що включає цифри, літери різних абеток, розділові знаки, математичні та інші символи, використовуються різні коди для оброблення такої інформації. Так у персональних комп'ютерах використовується код системи *ASCII*.

**Система *ASCII*** (*American Standard Code for Information Interchange* – Американській „стандартний код для обміну інформацією") є угодою за стандартом кодування символів, що схвалена Асоціацією стандартів США. Наявність такого коду спрощує обмін інформацією між різними пристроями комп'ютера. Фірма *IBM* запропонувала для використання у персональних комп'ютерах розширену версію системи *ASCII*, яка відрізняється від стандартної тим, що всі вісім біт символу використовуються для кодування інформації (у стандартній системі один біт призначався для перевірки правильності передавання інших семи бітів – перевірка на парність). Це дало змогу збільшити кількість символів на 128 і включити до їх складу символи національних алфавітів.

Стандартні коди *ASCII* можна розподілити на три групи:

– коди керування передаванням даних;
– коди керування форматом;
– коди друкованих символів.

Коди керування передаванням даних подають інформацію, що призначена для управління процесами обміну даними. Наприклад, код *SOH* (*start of header* – початок заголовка), код *STX* (*start of text* – початок тексту) і *ETX* (*end of text* – кінець тексту). Ці символи використовуються в комунікаційних протоколах для розмежування різних блоків даних, що передаються. Фірма *IBM* також модифікувала стандартне використання цих кодів, надавши їх друкованим графічним символам.

Коди керування форматом зображення використовуються для керування принтером або екраном монітору. Прикладами служать коди звукового сигналу, переходу на новий рядок, повернення каретки і прогону на початок сторінки. Коди керування передаванням даних і керування форматом зображення наведено у табл. 2.1. Друкарські символи стандартного набору наведено у табл. 2.2, а друкарські символи розширеного набору – у табл. 2.3.

Таблиця 2.1 – Коди керування передаванням даних і керування форматом зображення

| Мнемо-ніка | Зображення при друкуванні | Призначення у системі *ASCII* | Код у системі числення | |
|---|---|---|---|---|
| | | | десятко-ва | шістнад-цяткова |
| *NUL* | | Нульовий байт | 0 | 0 |
| *SOH* | | Початок заголовку | 1 | 1 |
| *STX* | | Початок тексту | 2 | 2 |
| *ETX* | | Кінець тексту | 3 | 3 |
| *EOT* | | Кінець передачі | 4 | 4 |
| *ENQ* | | Запит | 5 | 5 |
| *ACK* | | Підтвердження | 6 | 6 |
| *BEL* | Сигнал | | 7 | 7 |
| *BS* | Повернення на один символ | | 8 | 8 |
| *HT* | Табуляція | | 9 | 9 |
| *LF* | Перехід на новий рядок | | 10 | *A* |
| *VT* | Переміщення курсора в лівий верхній кут | | 11 | *B* |
| *FF* | Прогін сторінки | | 12 | *C* |
| *CR* | Повернення каретки | | 13 | *D* |

Закінчення табл. 2.1

| Мнемо-ніка | Зображення при друкуванні | Призначення у системі *ASCII* | Код у системі числення | |
|---|---|---|---|---|
| | | | десяткова | шістнадцяткова |
| *SO* | | Нижній регістр | 14 | *E* |
| *SI* | | Верхній регістр | 15 | *F* |
| *DLE* | | Кінець сеансу зв’язку | 16 | 10 |
| *DC1* | | Керування пристроєм 1 | 17 | 11 |
| *DC2* | | Керування пристроєм 2 | 18 | 12 |
| *DC3* | | Керування пристроєм 3 | 19 | 13 |
| *DC4* | | Керування пристроєм 4 | 20 | 14 |
| *NAK* | | Збій передавання даних | 21 | 15 |
| *SYN* | | Синхронізація | 22 | 16 |
| *ETB* | | Кінець блока передавання | 23 | 17 |
| *CAN* | | Відміна | 24 | 18 |
| | | | | |
| | | | | |
| *EM* | | Кінець носія даних | 25 | 19 |
| *SUB* | | Підстановка | 26 | 1*A* |
| *ESC* | | Початок керування | 27 | 1*B* |
| *FS* | Курсор праворуч | Розподілювач файлів | 28 | 1*C* |
| *GS* | Курсор ліворуч | Розподілювач груп | 29 | 1*D* |
| *RS* | Курсор вгору | Розподілювач записів | 30 | 1*E* |
| *US* | Курсор вниз | Розподілювач полів | 31 | 1*F* |

Таблиця 2.2 – Друкарські символи стандартного набору

| Символ | Код у системі числення | | Символ | Код у системі числення | | Символ | Код у системі числення | |
|---|---|---|---|---|---|---|---|---|
| | *D* | *H* | | *D* | *H* | | *D* | *H* |
| Пропуск | 32 | 20 | @ | 64 | 40 | ` | 96 | 60 |
| ! | 33 | 21 | *A* | 65 | 41 | *a* | 97 | 61 |
| “ | 34 | 22 | *B* | 66 | 42 | *b* | 98 | 62 |
| # | 35 | 23 | *C* | 67 | 43 | *c* | 99 | 63 |
| $ | 36 | 24 | *D* | 68 | 44 | *d* | 100 | 64 |
| % | 37 | 25 | *E* | 69 | 45 | *e* | 101 | 65 |
| & | 38 | 26 | *F* | 70 | 46 | *f* | 102 | 66 |
| ‘ | 39 | 27 | *G* | 71 | 47 | *i* | 103 | 67 |
| ( | 40 | 28 | *H* | 72 | 48 | *h* | 104 | 68 |
| ) | 41 | 29 | *I* | 73 | 49 | *i* | 105 | 69 |
| * | 42 | 2*A* | *J* | 74 | 4*A* | *j* | 106 | 6*A* |
| + | 43 | 2*B* | *K* | 75 | 4*B* | *k* | 107 | 6*B* |
| , | 44 | 2*C* | *L* | 76 | 4*C* | *l* | 108 | 6*C* |
| – | 45 | 2*D* | *M* | 77 | 4*D* | *m* | 109 | 6*D* |
| | 46 | 2*E* | *N* | 78 | 4*E* | *n* | 110 | 6*E* |
| / | 47 | 2*F* | *0* | 79 | 4*F* | *o* | 111 | 6*F* |
| 0 | 48 | 30 | *P* | 80 | 50 | *p* | 112 | 70 |
| 1 | 49 | 31 | *Q* | 81 | 51 | *q* | 113 | 71 |
| 2 | 50 | 32 | *R* | 82 | 52 | *r* | 114 | 72 |
| 3 | 51 | 33 | *S* | 83 | 53 | *s* | 115 | 73 |
| 4 | 52 | 34 | *T* | 84 | 54 | *t* | 116 | 74 |
| 5 | 53 | 35 | *U* | 85 | 55 | *u* | 117 | 75 |
| 6 | 54 | 36 | *V* | 86 | 56 | *v* | 118 | 76 |
| 7 | 55 | 37 | *W* | 87 | 57 | *w* | 119 | 77 |
| 8 | 56 | 38 | *X* | 88 | 58 | *x* | 120 | 78 |
| 9 | 57 | 39 | *Y* | 89 | 59 | *y* | 121 | 79 |
| : | 58 | 3*A* | *Z* | 90 | 5*A* | *z* | 122 | 7*A* |
| ; | 59 | 3*B* | [ | 91 | 5*B* | { | 123 | 7*B* |
| < | 60 | 3*C* | \ | 92 | 5*C* | \| | 124 | 7*C* |
| = | 61 | 3*D* | ] | 93 | 5*D* | } | 125 | 7*D* |
| > | 62 | 3*E* | ^ | 94 | 5*E* | ~ | 126 | 7*E* |
| ? | 63 | 3*F* | _ | 95 | 5*F* | | 127 | 7*F* |

Таблиця 2.3 – Друкарські символи розширеного набору

| Символ | Код у системі числення | | Символ | Код у системі числення | | Символ | Код у системі числення | |
|---|---|---|---|---|---|---|---|---|
| | *D* | *H* | | *D* | *H* | | *D* | *H* |
| А | 128 | 80 | а | 160 | *A*0 | └ | 192 | *C*0 |
| Б | 129 | 81 | б | 161 | *A*1 | ┴ | 193 | *C*1 |
| В | 130 | 82 | в | 162 | *A*2 | ┬ | 194 | *C*2 |
| Г | 131 | 83 | г | 163 | *A*3 | ├ | 195 | *C*3 |
| Д | 132 | 84 | д | 164 | *A*4 | ─ | 196 | *C*4 |
| Е | 133 | 85 | е | 165 | *A*5 | ┼ | 197 | *C*5 |
| Ж | 134 | 86 | ж | 166 | *A*6 | ╞ | 198 | *C*6 |
| З | 135 | 87 | з | 167 | *A*7 | ╟ | 199 | *C*7 |
| И | 136 | 88 | и | 168 | *A*8 | ╚ | 200 | *C*8 |
| Й | 137 | 89 | й | 169 | *A*9 | ╔ | 201 | *C*9 |
| К | 138 | 8*A* | к | 170 | *AA* | ╩ | 202 | *CA* |
| Л | 139 | 8*B* | л | 171 | *AB* | ╦ | 203 | *CB* |
| М | 140 | 8*C* | м | 172 | *AC* | ╠ | 204 | *CC* |
| Н | 141 | 8*D* | н | 173 | *AD* | ═ | 205 | *CD* |
| О | 142 | 8*E* | о | 174 | *AE* | ╬ | 206 | *CT* |
| П | 143 | 8*F* | п | 175 | *AF* | ╧ | 207 | *CF* |
| Р | 144 | 90 | ▒ | 176 | *B0* | ╨ | 208 | *D*0 |
| С | 145 | 91 | ▒ | 177 | *B*1 | ╤ | 209 | *D*1 |
| Т | 146 | 92 | ▓ | 178 | *B*2 | ╥ | 210 | *D*2 |
| У | 147 | 93 | │ | 179 | *B*3 | ╙ | 211 | *D*3 |
| Ф | 148 | 94 | ⌡ | 180 | *B*4 | ╘ | 212 | *D*4 |
| Х | 149 | 95 | ╕ | 181 | *B*5 | ╒ | 213 | *D*5 |
| Ц | 150 | 96 | ╣ | 182 | *B*6 | ╓ | 214 | *D*6 |
| Ч | 151 | 97 | ╗ | 183 | *B*7 | ╫ | 215 | *D*7 |
| Ш | 152 | 98 | ╝ | 184 | *B*8 | ╪ | 216 | *D*8 |
| Щ | 153 | 99 | ╜ | 185 | *B*9 | ┘ | 217 | *D*9 |
| Ъ | 154 | 9*A* | ║ | 186 | *BA* | ┌ | 218 | *DA* |
| Ы | 155 | 9*B* | ╖ | 187 | *BB* | █ | 219 | *DB* |
| Ь | 156 | 9*C* | ╛ | 188 | *BC* | ▄ | 220 | *DC* |
| Э | 157 | 9*D* | ╙ | 189 | *BD* | ▌ | 221 | *DD* |
| Ю | 158 | 9*E* | ╛ | 190 | *BE* | ▐ | 222 | *DE* |
| Я | 159 | 9*F* | ┐ | 191 | *BF* | ▀ | 223 | *DF* |

Закінчення табл. 2.3

| Символ | Код у системі числення | | Символ | Код у системі числення | | Символ | Код у системі числення | |
|---|---|---|---|---|---|---|---|---|
| | *D* | *H* | | *D* | *H* | | *D* | *H* |
| р | 224 | *E*0 | ы | 235 | *EB* | → | 246 | *F*6 |
| с | 225 | *E*1 | ь | 236 | *EC* | ← | 247 | *F*7 |
| т | 226 | *E*2 | э | 237 | *ED* | ↑ | 248 | *F*8 |
| у | 227 | *E*3 | ю | 238 | *EE* | ↓ | 249 | *F*9 |
| ф | 228 | *E*4 | я | 239 | *EF* | ÷ | 250 | *FA* |
| х | 229 | *E*5 | Ё | 240 | *F*0 | ± | 251 | *FB* |
| ц | 230 | *E*6 | ё | 241 | *F*1 | № | 252 | *FC* |
| ч | 231 | *E*7 | ′ | 242 | *F*2 | ¤ | 253 | *FD* |
| ш | 232 | *E*8 | ` | 243 | *F*3 | * | 254 | *FE* |
| щ | 233 | *E*9 | ´ | 244 | *F*4 | пропуск | 255 | *FF* |
| ъ | 234 | *EA* | ` | 245 | *F*5 | | | |

Двійкові числа у комірках пам’яті та регістрах мікропроцесора розміщуються таким чином, що для кожного біта (двійкового розряду) призначено окремий елемент пам’яті. Сукупність елементів, які утворюють комірку пам’яті, визначають довжину двійкового числа, називаються **розрядною сіткою**. Довжина розрядної сітки завжди обмежена і визначається конструктивними особливостями МПС. Тому значення чисел, що оброблюються, завжди обмежені. Ще більше обмежень стає при поданні дробових чисел.

Для подання дробових чисел в обчислювальній техніці використовуються дві форми:

– форма з фіксованою точкою;
– форма з плаваючою точкою.

При використанні форми з фіксованою точкою розрядна сітка конструктивно розподіляється на три частини: для подання знака, цілої і дробової частини числа. Використання такої форми пов’язано з труднощами при підготовці даних для оброблення, для забезпечення необхідного діапазону поданих даних. Також при обробленні даних трудно забезпечити необхідну точність оброблення (розрахунків).

Значно ефективніше використання форми з плаваючою точкою. Таке подання даних дозволяє розширити діапазон чисел і забезпечити необхідну точність подання результатів.

У цій формі дробове число подається у вигляді мантиси *M* і порядку *E (Exponent)*

$$D = \pm M \cdot B^{\pm E},$$

де $B$ – основа системи числення, у нашому випадку $B = 2$ – двійкова система числення.

Для однакового подання різних чисел мантису завжди нормалізують у межах

$$0{,}1 \le M < 1,$$

що виключає наявність нульового розряду системи числення після коми. Якщо у результаті обчислення результат перестає бути нормалізованим, то перед збереженням у пам'яті його необхідно нормалізувати, виконавши необхідний зсув мантиси і змінити значення порядку.

У формі з плаваючою точкою кількість розрядів порядку визначає діапазон оброблюваних чисел, а кількість розрядів мантиси – точність їх подання.

**Контрольні питання:**

1 Для чого використовуються коди *ASCII*?

2 Які форми використовуються для подання дробових чисел?

3 Чим визначається точність подання чисел при використанні форми з плаваючою точкою?

**Контрольні питання підвищеної складності:**

1 Подати двійкове число $111010101_B$ у шістнадцятковій системі числення.

2 Подати десяткові числа $06_D$, $45_D$, $60_D$ у двійково-десятковій системі числення в упакованому та розпакованому форматах.

3 Подайте число $12{,}25_D$ у вигляді з плаваючою точкою.

## 2.2 Подання даних у кодах

**Вхідний контроль:**

1 В чому полягає різниця між послідовними і паралельними кодами подання інформації?

2 Як записується дробове число у формі з фіксованою точкою?

3 Як записується дробове число у формі з плаваючою точкою?

З одного боку, в обчислювальній техніці існують два види кодів подання інформації – послідовний і паралельний, які обумовлюють порядок надходження даних для оброблення. Це пов'язано з конструктивними особливостями ліній передавання інформації, які бувають **двопроводовими** (телефонна лінія, радіорелейна лінія тощо) і **багатопроводовими** шинами. У двопроводовій лінії інформація передається побітно (у кожний момент часу на лінії є лише один біт) і код, що використовується, називається **послідовним**. Оброблення послідовного коду відбувається за декілька тактів (визначається розрядною сіткою), в міру надходження окремих бітів. Навпаки, у багатопроводовій шині у кожний момент

часу інформація подана кількома розрядами числа, які можуть оброблятися одночасно, за один такт. Такий код називається **паралельним**.

З іншого боку, числа в обчислювальній техніці можуть подаватися у вигляді різних кодів у залежності від вимог до подання числа і необхідного його оброблення.

Отже, цілі беззнакові двійкові числа можуть бути подані натуральним кодом числа.

**Натуральний код числа** передбачає подання даних як цілих беззнакових двійкових чисел. Діапазон подання чисел у натуральному коді визначається довжиною розрядної сітки. При цьому максимальне число, що може бути подане у натуральному коді розрядної сітки, становить $2^n$–1, де $n$ – кількість розрядів.

Необхідність виконання алгебраїчного додавання передбачає зображення числа сумісно зі своїм знаком, тому для подання таких чисел використовуються прямий, обернений і додатковий коди. Спосіб побудування цих кодів передбачає забезпечення таких вимог:

– запис алгебраїчного знака числа;

– подання від'ємних чисел за допомогою допоміжних додатних чисел, які відрізняються від відображення вихідних додатних чисел таким чином, щоб їх області зображення не збігалися;

– повна ідентичність алгоритмів виконання операцій над числами з однаковими і різними знаками, для забезпечення однотипності апаратного забезпечення для виконання цих операцій.

**Прямий код (ПК)** передбачає подання від'ємного числа таким чином, щоб у старшому розряді числа було показано його знак у вигляді 1, а в інших розрядах – його модуль. Старший розряд при цьому називають **знаковим**. Зображення додатного числа співпадає з його зображенням у натуральному коді, тому що знак + кодується у вигляді 0. При запису зручно знаковий розряд відокремлювати точкою після нього. Наприклад, $-98_D = 1.1100010_{ПК}$.

Діапазон представлення від'ємних чисел у прямому коді сягає від 0 до $2^{n-1} - 1$, а для додатних чисел – $2^{n-1}$.

Операція додавання для чисел, поданих у прямому коді виконується по-різному, в залежності від їх знаків. Так, якщо числа мають однакові знаки, то вони додаються і сумі надається знак, який мають обидва числа. Якщо числа мають різні знаки, то спочатку знаходиться більше за модулем число, потім виконується віднімання від нього меншого і результату привласнюється знак більшого з них.

До недоліків використання прямого коду можна віднести:

– не забезпечується ідентичність алгоритмів виконання операцій над числами з однаковими і різними знаками, що приводить до збільшення кількості операцій для отримання результату;

– нуль може мати два значення: додатне число (у вигляді байта) – 0.0000000 і від'ємне число -1.000000, що також потребує виконання додаткових операцій при обчисленнях.

Для усунення цих недоліків і виконання операції віднімання (алгебраїчного додавання) в обчислювальній техніці використовуються **обернений** і

**доповнювальний** коди. Додатні числа у цих кодах подаються аналогічно прямому, а від'ємні обчислюються за певними алгоритмами.

**Обернений код (ОК)** від'ємного двійкового числа *A*, для певної розрядної сітки, обчислюється за виразом

$$A_{об} = N - |A|,$$

де *N* – значення найбільшого беззнакового числа, що можливо розташувати у певній розрядній сітці – значення *N* для десяткових дробів становить

$$N = 2 - 2^{-(n-1)},$$

а для цілих чисел

$$N = 2^n - 1.$$

У цих виразах *n* – кількість бітів розрядної сітки для подання числа. Діапазон подання чисел в оберненому коді, що можливо подати у розрядній сітці, такий самий, як і у прямому коді.

За визначенням обернений код від'ємного числа є доповненням його модуля до найбільшого беззнакового числа, яке можливо розмістити у розрядній сітці. Таким чином, взаємне перетворення прямого й оберненого кодів від'ємного числа виконується як операція порозрядної інверсії всіх розрядів числа крім знакового, в якому необхідно записати 1.

Наприклад,

$$-98_D = 1.1100010_{ПК} \Rightarrow 1.0011101_{ОК}.$$

До переваг оберненого коду можливо віднести простий взаємозв'язок прямого й оберненого кодів, у результаті чого взаємне перетворення цих кодів є порозрядною операцією, що спрощує і прискорює її виконання.

Недоліками цього коду є необхідність урахування перенесення зі старшого розряду, яке виникає при виконанні додавання, і наявність двох значень для нуля: додатне – 0.0000000 та від'ємне – 1.1111111.

**Доповнювальний код (ДК)** від'ємного двійкового числа *A* для певної розрядної сітки обчислюється за виразом

$$A_{об} = K - |A|,$$

де *K* – значення ваги розряду, який знаходиться за старшим розрядом використовуваної розрядної сітки – значення *K* для десяткових дробів становить

$$K = 2,$$

а для цілих чисел

$$K = 2^n.$$

Діапазон подання чисел у доповнювальному коді для від'ємних чисел становить $0...2^{n-1}$, де *n* – кількість розрядів подання числа.

Доповнювальний код від'ємного числа легко отримати, додаючи одиницю молодшого розряду до оберненного коду цього числа.

Для попереднього прикладу

$$-98_D = 1.1100010_{\text{ПК}} \Rightarrow 1.0011101_{\text{ОК}}$$
$$+ \begin{array}{r} 1.0011101 \\ \underline{\qquad\qquad 1} \\ 1.0011110 \end{array}$$
$$1.0011101_{\text{ОК}} \Rightarrow 1.0011110_{\text{ДК}}$$

Для визначення прямого коду числа, яке подано у доповнювальному коді, віднімається одиниця із молодшого розряду, в результаті чого отримаємо оберненний код, а далі виконуємо інверсію всіх розрядів числа.

Використання доповнювального коду ліквідує недоліки оберненного коду: перенесення із старшого розряду втрачається і не враховується у подальших обчисленнях; нуль у доповнювальному коді – тільки додатне число.

**Контрольні питання:**

1 Якими причинами обумовлене використання прямого оберненого і довнювального кодів?

2 Сформулюйте правило формування доповнювального кода.

3 Які недоліки існують при формуванні числа 0 в оберненому коді?

**Контрольні питання підвищеної складності:**

1 Подати число +92 у вигляді восьмирозрядного (з урахуванням знака) прямого, зворотного та доповнювального кодів.

2 Подати число -121 у вигляді восьмирозрядного (з урахуванням знака) прямого, зворотного та доповнювального кодів.

## 2.3 Порозрядні операції над даними

**Вхідний контроль:**

1 Які правила виконання арифметичних операцій у позиційних системах числення Ви знаєте?

Виконання операції додавання у кодах дещо відрізняється від звичайного:

– розряди знаковий і розряди числа є рівноправними і перенесення у знаковий розряд необхідно враховувати і додавати до знакових розрядів;

– при виконанні операції додавання в ОК перенесення із знакового розряду необхідно додавати до молодшого розряду числа (операція циклічного перенесення);

– при виконанні операції додавання в ДК перенесення із знакового розряду ігнорується і відкидається.

Розглянемо операцію додавання цілих двійкових чисел зі знаком у кодах. Для подання у кодах необхідно користуватися восьмирозрядною сіткою (з урахуванням знака) у вигляді байта.

1 Для чисел з різними знаками

$$27_D - 30_D = 27_D + (-30_D) = 11011_B - 11110_B$$

представимо двійкові числа у кодах

| ОК | ДК |
|---|---|
| $+11011_B \Rightarrow 0.0011011$ | $+11011_B \Rightarrow 0.0011011$ |
| $-11110_B \Rightarrow 1.1100001$ | $-11110_B \Rightarrow 1.1100010$ |

Виконаємо операцію додавання у кодах

| ОК | ДК |
|---|---|
| 0.0011011 | 0.0011011 |
| + 1.1100001 | + 1.1100010 |
| 1.1111100 | 1.1111101 |

Результати отримано у відповідних кодах. Перетворимо їх на ПК і подамо у десятковій системі числення

$$1.1111100_{\text{ОК}} \Rightarrow 1.0000011_{\text{ПК}} \Rightarrow -3_D.$$

Результат при роботі з оберненним кодом правильний.

Для перевірки результату, поданого у ДК, його треба спочатку перевести в ОК, для чого від молодшого розряду необхідно відняти одиницю.

$$1.1111101_{\text{ДК}} - 1 = 1.1111100_{\text{ОК}} \Rightarrow 1.0000011_{\text{ПК}} \Rightarrow -3_D.$$

Результат також правильний.

Слід зазначити, що мікропроцесори виконують операцію додавання у доповнювальному коді, тому на етапі підготовки даних їх слід подати у вигляді ДК. Результат операції також подається в ДК. Для переведення його в ПК слід використовувати спеціальні програми, які виконують це перетворення.

2 Для двох інших чисел з різними знаками

$$-27_D + 30_D = -1011_B + 11110_B$$

представимо двійкові числа у кодах

| ОК | ДК |
|---|---|
| $-11011_B \Rightarrow 1.1100100$ | $-11011_B \Rightarrow 1.1100101$ |
| $+11110_B \Rightarrow 0.0011110$ | $+11110_B \Rightarrow 0.0011110$ |

Виконаємо операцію додавання у кодах

| ОК | ДК |
|---|---|
| 1.1100100 | 1.1100101 |
| + 0.0011110 | + 0.0011110 |
| 1 0.0000010 | 1 0.0000011 |
| + 1 | |
| 0.0000011 | |

При виконанні операції в ОК перенесення із знакового розряду було враховано в результаті, а при виконанні операції в ДК – відкинуто. Результат в обох випадках отримано у вигляді додатного числа $11_B = 3_D$, що є правильним.

При виконанні додавання даних, поданих у двійково-десятковій системі числення, виникає необхідність коригування результату в таких випадках:

– при формуванні суми у тетраді отримано число, яке не є цифрою десяткової системи числення, наприклад,

$$\begin{array}{r} 0000\ 1001_{BCD} \\ +\ \underline{0000\ 0110_{BCD}} \\ 0000\ 1111_{BCD} \end{array},$$

у молодшій тетраді отримано двійкове число $1111_B$, яке неможливо подати у десятковій системі числення одним розрядом;

– при формуванні суми виникає перенесення до старшого розряду (старшої тетради), а значення результату в тетраді є невірне, наприклад,

$$\begin{array}{r} 0000\ 1001_{BCD} \\ +\ \underline{0000\ 1000_{BCD}} \\ 0001\ 0001_{BCD} \\ \circlearrowleft \end{array},$$

у молодшій тетраді отримано двійкове число 0001*B*, замість 0111*B*, яке є правильним результатом.

В обох випадках корекція результату в тетраді буде виконуватися при використанні коду з надлишком 3 для обох складових, або при додаванні $6_D$ ($0110_B$) до результату. У першому випадку:

$$\begin{array}{r} 0000\ 1111_{BCD} \\ +\ \underline{0000\ 0110_{BCD}} \\ 0001\ 0101_{BCD} \\ \circlearrowleft \end{array}.$$

**Контрольні питання:**

1 Чому може з’явитися необхідність корекції результату при виконанні арифметичних операцій над числами, які подані у двійково-десятковій системі числення?

2 Який код використовується для корекції результату арифметичної операції над числами, які подані у двійково-десятковій системі числення?

3 Чим відрізняється виконання арифметичних операції над числами зі знаком над числами, які подані в оберненому і доповнювальному кодах?

**Контрольні питання підвищеної складності:**

1 Виконати операцію (+92 – 121) у доповнювальному коді. Отримати результат і перевести його у десяткову систему числення.

2 Виконати операцію (+92 – 121) в оберненному коді. Отримати результат і перевести його у десяткову систему числення.

3 Виконати операцію (+ 121 – 92) у доповнювальному коді. Отримати результат і перевести його у десяткову систему числення.

4 Виконати операцію (+ 121 – 92) в оберненному коді. Отримати результат і перевести його у десяткову систему числення.

5 Перевести число +8,73 у двійкову систему числення і подати у вигляді з плаваючою точкою (у вигляді *мантиси і порядку*). Мантису подати у вигляді округленого восьмирозрядного (з урахуванням знака) двійкового числа у прямому, зворотному та доповнювальному кодах.

6 Перевести число –5,63 у двійкову систему числення і подати у вигляді з плаваючою точкою (у вигляді *мантиси і порядку*). Мантису подати у вигляді округленого восьмирозрядного (з урахуванням знака) двійкового числа у прямому, зворотному та доповнювальному кодах.

7 Виконати операцію додавання чисел, отриманих у попередніх пунктах в оберненному коді.

8 Виконати операцію додавання (+ 8.73 –5,63) у доповнювальному коді.

9 Представити число 95 у двійково-десятковій системі числення.

10 Пояснити, чому при виконанні арифметичних операцій над числами, поданими у двійково-десятковій системі числення, необхідно використовувати код з залишком 3?

11 Як перевести десяткові числа 95 і 55 у двійково-десяткову систему числення? Виконати над отриманими числами операцію додавання. Скоригувати результат.

# 3 ЦИФРОВІ АВТОМАТИ

**Вхідний контроль:**

1 Які логічні операції виконуються в обчислювальній техніці?
2 Які властивості логічних елементів Ви знаєте?
3 Яка різниця між комбінаційними і послідовнісними схемами?
4 Які логічні функції Ви знаєте?
5 Які способи мінімізації логічних функцій Ви знаєте?

Перетворення інформації в обчислювальній техніці відбувається за допомогою електронних цифрових пристроїв (ЦП) – логічних схем, які будуються з логічних елементів і забезпечують виконання арифметичних і логічних операцій над сукупністю цифрових вхідних сигналів $X_i$, перетворюючи їх у сукупність вихідних сигналів $Y_j$. Розподіляють два класи таких пристроїв: **комбінаційні схеми** (КС) і **послідовнісні схеми** – цифрові автомати (ЦА).

**В комбінаційних схемах** результат перетворення – сукупність цифрових вихідних сигналів у будь-який момент часу $t_n$ залежить лише від сукупності (комбінації) цифрових сигналів $X_i$, які надходять на її входи у даний момент часу і не залежать від стану схеми, який передував надходженню сигналів $X_i$. В таких схемах відсутні елементи пам'яті, тому сигнали, які діють у такий схемі, не зберігаються. Такі ЦП ще називають цифровими автоматами без пам'яті (примітивними автоматами). До КС відносяться досить прості елементи, вузли та блоки мікропроцесорних систем: шифратори, дешифратори, мультиплексори, демультиплексори, комбінаційні суматори, перетворювачі кодів, схеми контролю тощо.

До **послідовністних схем** відносяться ЦП, в яких результат перетворення $Y_j$ залежить не лише від комбінації цифрових сигналів $X_i$, які надходять на її входи у даний момент часу $t_n$, а й від послідовності попередніх цифрових станів входів і виходів схеми – внутрішніх станів $Z_f$. Для запам'ятовування попередніх станів така схема повинна мати елементи пам'яті. Подібні схеми називають **цифровими автоматами.** Можна стверджувати, що кількість внутрішніх станів і кількість елементарних запам'ятовувальних пристроїв в цифрових автоматах зв'язані між собою такою залежністю $Z = 2^k$, де $k$ – кількість запам'ятовувальних елементів.

Загальна теорія ЦА поділяється на **абстрактну** і **структурну**. Абстрактна теорія досліджує поведінку автомата відносно зовнішнього середовища, на рівні алгоритмів його роботи, не вирішуючи задач його побудови. Абстрактна теорія ЦА показує принципові відміни КС від ЦА для можливості подання дискретних процесів і обмеження, які є при цьому. Важливий висновок, який отримано у цих дослідженнях, полягає в тому, що в ЦА можливо реалізувати нескінченні регулярні події, на відміну від КС, де є можливість реалізувати лише скінченні події. Регулярними подіями вважаються такі події, які у скінченному алфавіті $X = \{x_1, x_2, \ldots, x_n\}$ можливо отримати з елементарних слів алфавіту $X$ при здійсненні над ними операцій диз'юнкції, перемноження та

ітерації. Нерегулярні події реалізувати у ЦА неможливо, тому що це вимагає нескінченного обсягу пам'яті. На практиці звичайно розглядаються ЦА, які мають обмежений обсяг пам'яті і називаються **скінченними ЦА.**

Структурна теорія досліджує проблеми побудування ЦА, кодування вхідних сигналів і реакції схеми на них. Використовуючи апарат булевої алгебри, структурна теорія надає важливі рекомендації щодо розробки схем пристроїв обчислювальної техніки.

У процесі проектування будь-якого ЦП доводиться вирішувати задачі аналізу або синтезу. Розв'язання задачі аналізу передбачає описання роботи принципової схеми ЦП в аналітичному вигляді, можливості його мінімізації й оцінку деяких параметрів готової структури ЦП.

Розв'язання задачі синтезу передбачає побудування структурної і електронної схеми пристрою відповідно до опису алгоритму його роботи, для чого необхідно зробити вибір певних логічних елементів і передбачити їхнє з'єднання таким чином, щоб забезпечити виконання правил його функціонування, які подано в аналітичній формі. При розв'язанні цієї задачі необхідно зробити мінімізацію синтезованої схеми і виконати оцінку параметрів принципової схеми ЦП.

**Синтез логічних схем**

Правила функціонування логічних схем (ЛС) можуть задаватися у різний спосіб:

- словесне описування;
- за допомогою таблиць істинності;
- за допомогою часових діаграм;
- аналітичний – за допомогою логічних (булевих) виразів;
- у вигляді координатних діаграм.

Усі ці способи описують один і той самий процес, тому по одному з них легко уявити функціонування ЛС і перейти від одного виду описування до іншого.

Слід зазначити, що для спрощення розв'язання задачі синтезу загальна ЛС поділяється на декілька схем, кожна з яких має один вихід (із сукупності вихідних сигналів $Y_j$) і стільки входів, скільки входить у сукупність вхідних сигналів $X_i$, необхідних для формування $Y_j$. Для більшого спрощення дозволяється подальше роздрібнення схеми таким чином, щоб зменшити кількість вхідних сигналів $X$. Після закінчення синтезу схем усі ці дрібні схеми об'єднують в одну загальну схему ЦП. Розв'язання задачі синтезу виконується за декілька етапів.

На першому етапі відбувається описування функціонування ЛС, як правило, у вигляді таблиці істинності, яка описує кожен зі стійких станів ЦП, як співвідношення вхідних і вихідного сигналів і кожен рядок таблиці відповідає певному часовому інтервалу роботи. Описування виконується переважно для схем, які мають один вихід $Y$, а кількість входів рекомендується вибирати не більше ніж п'ять. Таблиця істинності повинна описувати всі

можливі стани роботи ЛС, тому кількість рядків такої таблиці дорівнює кількості можливих станів. Якщо ЛС є частково визначеною, то значення вихідної функції позначається як ~.

На другому етапі відбувається подання описування роботи ЛС в аналітичному виді – у досконалій диз'юнктивній нормальній формі (ДДНФ), або у вигляді досконалої кон'юнктивної нормальної форми (ДКНФ). Вибір форми подання можливо обумовити співвідношенням кількості нулів та одиниць у вихідному сигналі (логічній функції) та їх розташуванням за номерами рядків таблиці істинності або технічним завданням на проведення синтезу схеми.

На третьому етапі відбувається мінімізація схеми ЦП. Найбільш просто і наочно рекомендується проводити мінімізацію за допомогою координатних діаграм (карти Карно або діаграми Вейча). Різниця між цими способами полягає лише у поданні системи координат на діаграмі. На рис. 3.1, а показано вид карти Карно, а на рис. 3.1, б – вид діаграми Вейча.

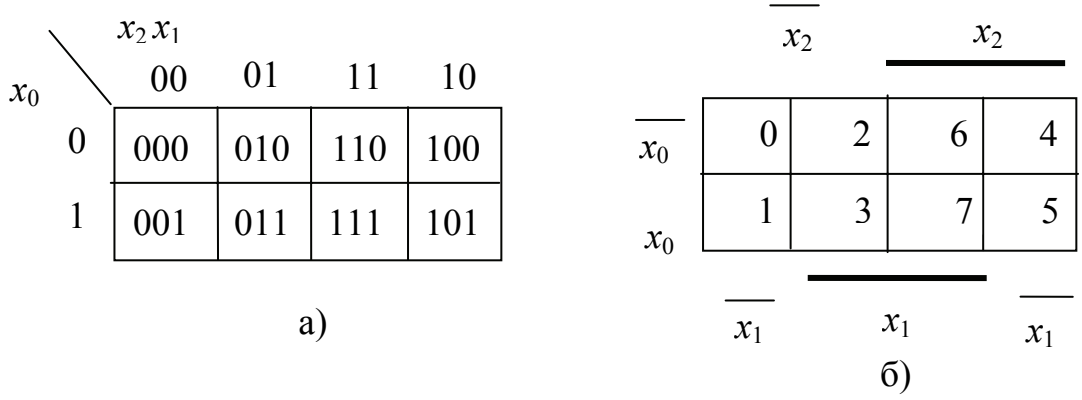



Рисунок 3.1 – Координатні діаграми

Кількість можливих комбінацій значень вхідних сигналів (логічних змінних) $X$, а також кількість можливих стійких станів (логічних функцій) дорівнює $K = 2^N$, де $N$ – кількість вхідних сигналів. Слід зазначити, що деякі стійкі стани можуть бути відсутніми, тобто вихідний сигнал $Y$ (логічна функція) індиферентний до комбінації вхідних сигналів $X_i$ у цей час. Це можливо трактувати, як відсутність деяких комбінацій вхідних сигналів $X_i$ у повному наборі можливих. Так, наприклад, дешифратор для керування цифровим індикатором, який відображує десять десяткових цифр 0...9, має чотири входи для подання кодів двійкових чисел від 0000 до 1001, а кількість стійких станів для такої схеми дорівнює $K = 2^4 = 16$. Таким чином, шість комбінацій сигналів ($16_D - 10_D = 6_D$) не можуть бути сформовані на виходах такого пристрою. Значення сигналу карти Карно (рис. 3.1, а) і діаграми Вейча (рис. 3.1, б) для трьох вхідних змінних $x_0$ , $x_1$ , $x_2$, при однаковому розміщенні вхідних змінних на сторонах прямокутників, які є осями координат (загалом, вибір системи координат може бути довільним, обов'язковим є тільки зсув координат на одну клітинку відповідно одна до одної на тих напрямках, на яких необхідно розміщувати дві змінні). У клітинках на рис. 3.1, а показано набори вхідних

сигналів, а на рис. 3.1, б номери рядків таблиці істинності, які відповідають один одному.

У результаті мінімізації отримуємо аналітичну функцію, яка складається з мінімальної кількості логічних функцій, за допомогою яких вона реалізується у вигляді мінімальної диз'юнктивної нормальної форми (МДНФ) або мінімальної кон'юнктивної нормальної форми (МКНФ).

Критеріями оптимальності для цих способів є розробка схеми ЛС з мінімальною кількістю логічних елементів і мінімальною кількістю з'єднань між ними, внаслідок чого схема буде споживати мінімум енергії і вартість її буде менше, ніж у інших можливих.

На четвертому етапі за отриманими МДНФ або МКНФ можливо побудувати принципову схему ЦП у базисі ТА-АБО-НІ. Якщо принципову схему необхідно мати в іншому базисі, то виконуємо необхідне перетворення і будуємо схему у відповідному базисі ТА-НІ (АБО-НІ). За необхідності на цьому етапі можливо зробити розрахунок часу затримки сигналу й інших параметрів схеми.

Як приклад зробимо синтез схеми дешифратора для керування одним із елементів семисегментного індикатора, який відображує десяткові цифри 0...9.

1 Таблиця істинності, що завдає алгоритм роботи цього дешифратора, – табл. 3.1.

Таблиця 3.1 – Таблиця істинності дешифратора

| № набору вхідних змінних | $x_3$ | $x_2$ | $x_1$ | $x_0$ | $Y$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | ~ |
| 11 | 1 | 0 | 1 | 1 | ~ |
| 12 | 1 | 1 | 0 | 0 | ~ |
| 13 | 1 | 1 | 0 | 1 | ~ |
| 14 | 1 | 1 | 1 | 0 | ~ |
| 15 | 1 | 1 | 1 | 1 | ~ |

2 За даними цієї таблиці запишемо вираз у вигляді досконалої диз'юнктивної нормальної форми (ДДНФ) або у вигляді досконалої кон'юнктивної нормальної форми (ДКНФ). Досконаліть логічних функцій передбачає включення у вираз, який отримується, всіх логічних функцій, які використовуються для формування логічної функції. Будь-який з цих двох способів однозначно описує алгоритм роботи ЛС.

Для подання у вигляді ДДНФ вибираємо рядки таблиці, на яких значення вихідної функції $Y$ дорівнює $L1$ (логічній одиниці) і записуємо у вигляді кон'юнкції (логічного добутку) набори вхідних змінних кожного рядка, які об'єднуємо у вигляді диз'юнкції (логічної суми). Математичний вираз, що описує функцію на одному рядку, називається **термом**. Вхідні змінні повинні входити у кон'юнкцію таким чином, щоб значення логічного добутку дорівнювало $L1$, для чого вхідні змінні, які мають значення $L0$ (логічного нуля) необхідно інвертувати. Так, для нульового набору (рядок таблиці з номером 0) кон'юнкція вхідних змінних матиме вигляд:

$$Y_0 = \bar{x}_3 \wedge \bar{x}_2 \wedge \bar{x}_1 \wedge \bar{x}_0$$

Рядки таблиці, на яких логічна функція не визначена, не беруться до уваги. Тоді логічна функція у вигляді ДДНФ буде записана як

$$Y_{\text{ДДНФ}} = \left(\overline{x_3} \wedge \overline{x_2} \wedge \overline{x_1} \wedge \overline{x_0}\right) \vee \left(\overline{x_3} \wedge \overline{x_2} \wedge x_1 \wedge \overline{x_0}\right) \vee \left(\overline{x_3} \wedge x_2 \wedge x_1 \wedge \overline{x_0}\right) \vee \left(x_3 \wedge \overline{x_2} \wedge \overline{x_1} \wedge x_0\right).$$

Для подавання логічної функції у вигляді ДКНФ вибираємо рядки, на яких значення вихідної функції $Y$ дорівнює логічному нулю $L0$, і записуємо логічну функцію у вигляді кон'юнкції (логічного добутку) диз'юнкцій вхідних логічних змінних кожного рядка. Для опису вихідної логічної функції $Y$ у вигляді $L1$ логічні зміні кожного рядка, які входять у вираз ДКНФ, необхідно проінвертувати. ДКНФ логічної функції має вигляд:

$$Y_{\text{ДКНФ}} = \left(x_3 \vee x_2 \vee x_1 \vee \overline{x_0}\right) \wedge \left(x_3 \vee x_2 \vee \overline{x_1} \vee \overline{x_0}\right) \wedge \left(x_3 \vee \overline{x_2} \vee x_1 \vee x_0\right) \wedge \left(x_3 \vee \overline{x_2} \vee \overline{x_1} \vee \overline{x_0}\right) \wedge \left(\overline{x_3} \vee x_2 \vee x_1 \vee \overline{x_0}\right).$$

За виразами ДДНФ і ДКНФ можливо оцінити їхню **складність** у вигляді коефіцієнта зв'язку $K_{\text{зв}}$, який визначає кількість елементарних логічних функцій, які необхідно виконати для отримання результатів:

$$K_{\text{зв ДДНФ}} = 4_{\text{НІ}} + 4_{\text{ТА}} + 1_{\text{АБО}} = 9.$$

$$K_{\text{зв ДКНФ}} = 4_{\text{НІ}} + 6_{\text{АБО}} + 1_{\text{ТА}} = 11.$$

Таким чином видно, що для реалізації функції у ДДНФ необхідно 9, а для реалізації у ДКНФ – 11 логічних елементів.

3 Мінімізацію досконалих логічних функцій виконуємо за допомогою діаграми Вейча, яка для логічної функції з чотирма вхідними змінними матиме вигляд квадрата $4 \times 4$ клітинки. Діаграма Вейча для логічної функції, яка описана табл. 3.1, наведена на рис. 3.2.

Для мінімізації логічної функції за одиницями або за нулями на діаграмі Вейча необхідно визначити мінімальну кількість найбільших груп (контурів), що складаються з одиниць або нулів, у вигляді квадратів або прямокутників (об'єднувати можливо лише сусідні клітинки). Сусідніми вважаємо клітинки, що мають спільні сторони, при цьому клітинки, розташовані по горизонтальних та вертикальних сторонах квадрата, є сусідніми. Взагалі, ознакою сусідства можливо вважати відміну двійкового коду номера клітинки лише в одному розряді. Таким чином, аналізуючи номери клітинок у верхньому і нижньому рядках, а також у правому і лівому, можливо зробити висновок, що відповідні клітинки у цих рядках також є сусідніми і діаграму Вейча можна згорнути у циліндр по горизонталі і вертикалі. Кількість клітинок у групі повинна дорівнювати цілому ступеню двійки, тобто 1, 2, 4, 8, 16. На рис. 3.2 у вигляді кіл і овалів показано контури, які об'єднують одиниці і нулі поданої логічної функції. Контури об'єднань можуть будь-яку кількість разів перетинатися, але не можуть повністю суміщатися. Якщо функція має невизначені стани (символ – ~), то функції цієї клітинки можливо надати значення 0 або 1 таким чином, щоб отримати контур, до якого буде входити більша кількість клітинок з однаковими значеннями 0 або 1.

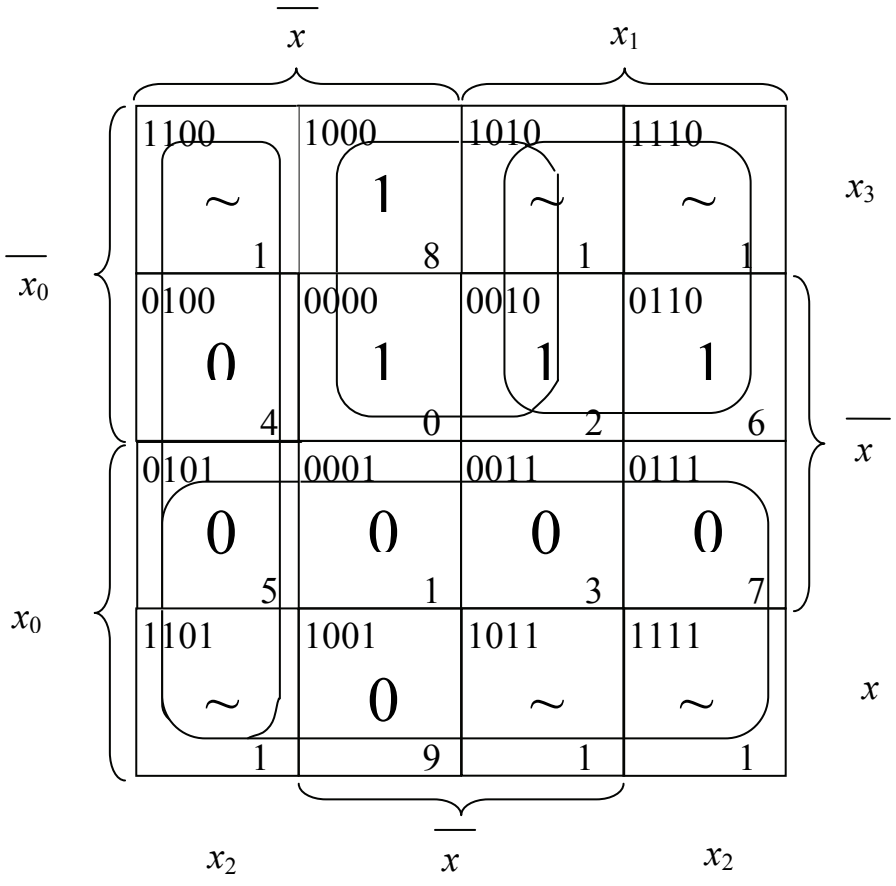



Рисунок 3.2 – Діаграма Вейча з контурами для одиниць і нулів

Таким чином, об'єднавши 1 і невизначені стани, отримали два контури, до яких входять клітинки з номерами 0, 2, 8, 10 і 2, 6, 10, 14. Мінімальною формою логічної функції для кожного з контурів будуть координати контурів на діаграмі з відсутніми координатами, на яких функція $Y$ змінює своє значення. Так, координата $x_1$ у контурі з номерами клітинок 0, 2, 8, 10 змінює своє значення; у стовпчику з клітинками 0, 8 вона має значення 0, а у стовпчику з номерами клітинок 2, 10 – 1. Таким чином, координата $x_1$ буде відсутня при записуванні мінімальної ДНФ (МДНФ).

Запишемо мінімальні логічні функції, які отримуємо після мінімізації

$$Y_{\text{МДНФ}} = \left(\overline{x_2} \wedge \overline{x_0}\right) \vee \left(x_1 \wedge \overline{x_0}\right),$$

$$Y_{\text{МКНФ}} = \left(x_2 \vee \overline{x_1}\right) \wedge x_0 .$$

Оцінимо $K_{\text{зв}}$ для мінімізованих функцій:

для МДНФ $K_{\text{зв}}$ = 2НІ + 2ТА + 1АБО = 5,

для МКНФ $K_{\text{зв}}$ = 1НІ + 1АБО + 1ТА = 3.

4 За одним із отриманих у попередньому пункті виразах МДНФ або МКНФ можемо побудувати схему логічного пристрою у базисі ТА-АБО-НІ. Критерієм вибору є менше значення $K_{\text{зв}}$ або завдання на розробку пристрою.

Логічні схеми будують, як правило, на елементах одного типу в базисах ТА-НІ, АБО-НІ. Це робиться для того, щоб запобігти розкиду параметрів часових затримок у схемі.

Для перетворення виразу до необхідного базису використовується правило подвійної інверсії $\overline{\overline{x}} = x$ та закон де Моргана у такому виді:

$$\overline{x_1 \vee x_0} = \overline{x_1} \wedge \overline{x_0}$$
$$\overline{x_1 \wedge x_0} = \overline{x_1} \vee \overline{x_0}$$

Простіше перетворювати вираз МДНФ до базису ТА-НІ, а вираз МКНФ до базису АБО-НІ, взагалі можливі будь-які перетворення, але при перетворенні МДНФ до базису АБО-НІ і навпаки МКНФ до базису ТА-НІ, схеми будуть більші на один логічний елемент.

Виконаємо перетворення МДНФ у базис ТА-НІ

$$Y_{\text{МДНФ}} = \left(\overline{x_2} \wedge \overline{x_0}\right) \vee \left(x_1 \wedge \overline{x_0}\right) = \overline{\overline{\left(\overline{x_2} \wedge \overline{x_1}\right) \vee \left(x_1 \wedge \overline{x_0}\right)}} = \overline{\overline{\left(\overline{x_2} \vee \overline{x_0}\right) \vee \left(x_1 \vee \overline{x_0}\right)}},$$

і перетворення МКНФ у базис АБО-НІ

$$Y_{\text{МКНФ}} = \left(x_2 \vee \overline{x_1}\right) \wedge x_0 = \overline{\overline{\left(x_2 \vee \overline{x_1}\right) \wedge x_0}} = \overline{\overline{\left(x_2 \vee \overline{x_1}\right)} \vee \overline{x_0}}\,.$$

За цими виразами будується схема дешифратора. На рис. 3.3 схема подана у базисі ТА-НІ, а на рис. 3.4 – у базисі АБО-НІ.

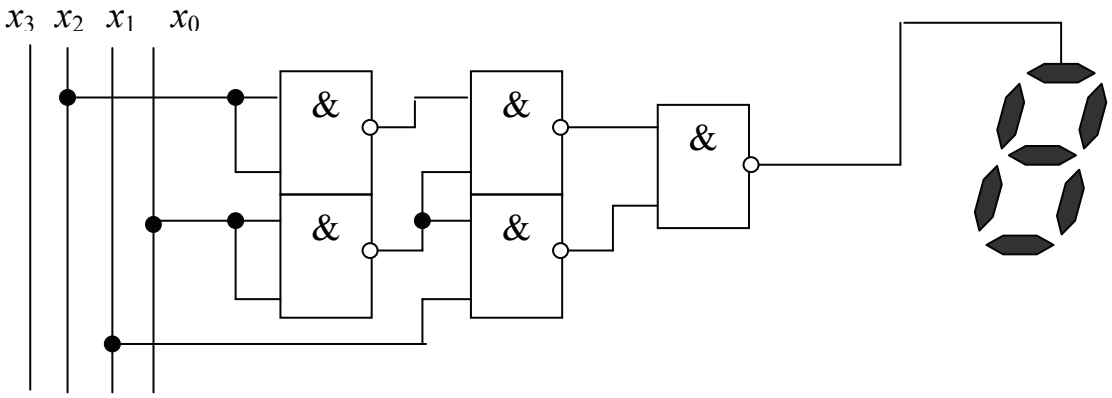



Рисунок 3.3 – Схема дешифратора у базисі ТА-НІ

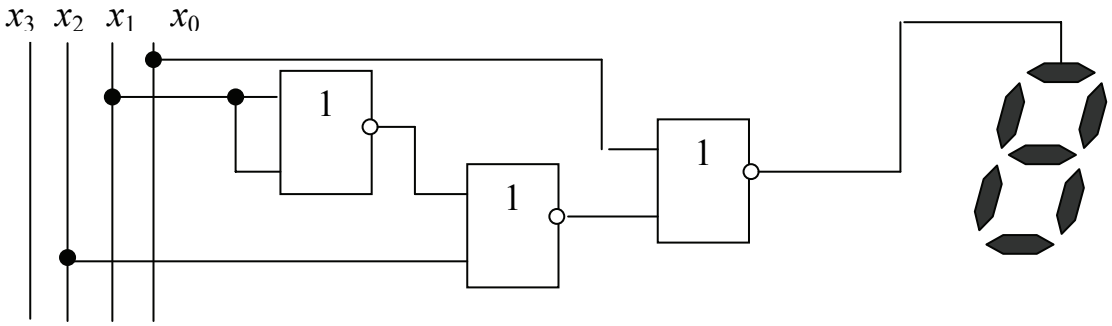



Рисунок 3.4 – Схема дешифратора у базисі АБО-НІ

Час затримки сигналів розраховується за найдовшим шляхом (за максимальної кількості мікросхем, через які проходить хоча б один із сигналів) і становить

$$T_{\text{затр}} = t_{\text{зм}} \times N_{\text{мс}},$$

де $T_{\text{затр}}$ – час затримки сигналів схемою;

$t_{\text{зм}}$ – час затримки сигналів однією мікросхемою (визначається технологією виготовлення мікросхеми);

$N_{\text{мс}}$ – максимальна кількість мікросхем від входу схеми до її виходу.

Для розроблених схем

$$T_{\text{затр}} = t_{\text{зм}} \times N_{\text{мс}} = 20 \cdot 10^{-6} \times 3 = 60 \cdot 10^{-6} \text{ с},$$

де $t_{\text{зм}} = 20 \cdot 10^{-6}$ секунди – часова затримка однієї мікросхеми, вважаючи, що мікросхема виготовлена за технологією ТТЛ;

$N_{мс} = 3$ – кількість мікросхем, через які проходять сигнали $x_2$ і $x_0$ на рис. 3.3, а також сигнал $x_1$ на рис. 3.4.

**Контрольні питання:**

1 Синтезувати схему цифрового пристрою, алгоритм роботи якого задано не повністю визначеною логічною функцією, що подана у вигляді таблиці істинності:

| № набору | $X_3$ | $X_2$ | $X_1$ | $X_0$ | $Y$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 | 0 |
| 14 | 1 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 |

Визначити складність логічної функції.

**Контрольні питання підвищеної складності:**

1 Виконати мінімізацію логічної функції, що подана у вигляді:

$$Y = (x_3 \wedge x_2 \wedge x_1) \vee (x_3 \wedge \overline{x}_2 \wedge x_1) \vee (x_3 \wedge x_2 \wedge \overline{x_1}) \vee (\overline{x_3} \wedge \overline{x_2} \wedge x_1)$$

за допомогою координатної діаграми і результат подати у вигляді схеми, що реалізована:

– у базисі ТА–НІ;

– у базисі АБО–НІ.

2 Визначити складність логічної функції до і після мінімізації для кожного з базисів.

# 4 ТИПОВІ ПРИСТРОЇ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ
## (Для самостійного вивчення)

**Вхідний контроль:**

1 Які типові цифрові пристрої Ви знаєте?
2 Чим відрізняються комбінаційні і послідовнісні пристрої?
3 Які елементи є основою для побудови цифрових пристроїв?

До типових пристроїв мікропроцесорних систем можна віднести такі пристрої: двійкові суматори, цифрові компаратори, арифметично-логічний пристрій, запам'ятовувальні пристрої, пристрої введення-виведення інформації, таймери, програмовані логічні інтегральні схеми (ПЛІС) тощо.

### 4.1 Суматори

**Суматор** – це вузол обчислювальної системи, що виконує арифметичне додавання кодів двійкових чисел. Суматор будується з комбінаційних схем, які виконують додавання двох однорозрядних двійкових чисел *a* і *b*, формують однорозрядний сигнал їхньої суми – *S* і сигнал перенесення в наступний старший розряд – *C*. Такі пристрої називають напівсуматорами і вони є елементною базою для побудови повних суматорів. Умовне графічне позначення такої схеми показано на рис. 4.1.

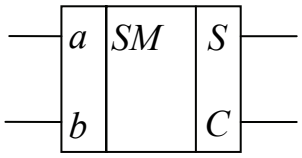



Рисунок 4.1 – Умовне графічне позначення комбінаційного напівсуматора

Схема повного однорозрядного суматора буде різнитися тим, що одним із вхідних сигналів, додатково до операндів *a* і *b*, буде значення вхідного перенесення, яке буде додаватися до результату. Алгоритм роботи повного однорозрядного суматора можливо описати за допомогою таблиці істиності – табл. 4.1.

Таблиця 4.1 – Таблиця істиності однорозрядного суматора

| Вхідне перенесення *c* | Додаток *a* | Додаток *b* | Вихідне перенесення *C* | Сума *S* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |

Закінчення табл. 4.1

| Вхідне перенесення $c$ | Додаток $a$ | Додаток $b$ | Вихідне перенесення $C$ | Сума $S$ |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

За даними цієї таблиці можливо виконати синтез схеми повного суматора. Після чого буде видно, що ця схема є поєднанням схем двох напівсуматорів. Схему повного суматора показано на рис. 4.2.

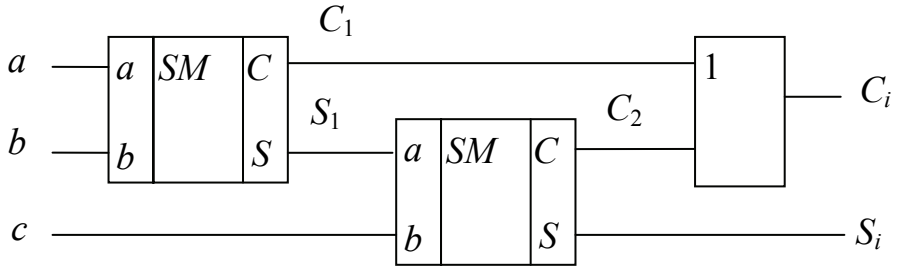



Рисунок 4.2 – Схема повного однорозрядного суматора

У цій схемі формуються два проміжні перенесення і одна проміжна сума, які далі беруть участь у формуванні результату. Вихідне перенесення формується як результат виконання логічної функції АБО сигналів двох проміжних перенесень.

Багаторозрядні суматори будуються як схеми паралельного з'єднання повних однорозрядних суматорів з послідовними лініями міжрозрядних перенесень. Приклад схеми такого суматора показано на рис. 4.3.

Комбінаційний суматор можливо також використовувати для виконання операції віднімання чисел зі знаками, для чого операнди, що надходять на схему, повинні подаватися у доповнювальному коді.

Важливою рисою комбінаційного паралельного суматора є затримка у колах формування перенесень. За характером розповсюдження перенесення суматори поділяють на три класи: суматори з порозрядним паралельним перенесенням, з паралельним перенесенням і з груповим перенесенням.

Паралельні суматори будуються за схемою, що показана на рис. 4.3, і характеризуються послідовним розповсюдженням сигналу перенесення від розряду до розряду, за ступенем формування результату. Затримка формування суми, за наявності перенесень буде дорівнювати

$$T_{\text{пер}} = \tau \cdot n,$$

де $\tau$ – затримка сигналу в одному розряді суматора; $n$ – кількість розрядів. Видно, що ця величина швидко зростатиме зі збільшенням кількості розрядів.

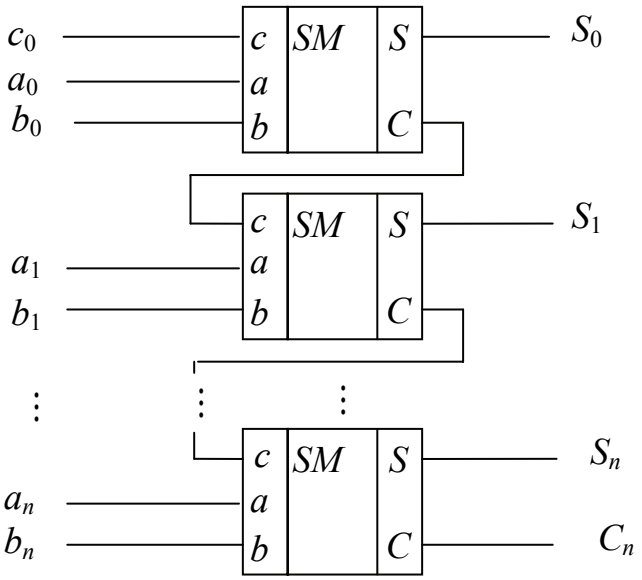



Рисунок 4.3 – Схема *n*-розрядного комбінаційного паралельного суматора

Таким чином, такий тип суматора характеризуються простотою схеми формування перенесення, але має низьку швидкодію.

Для усунення цього недоліку, можливо використовувати схему суматора з паралельним перенесенням. В такому суматорі додавання виконується як порозрядна операція і вхідне перенесення для кожного старшого розряду формується незалежно від формування перенесення у попередньому молодшому розряді. Для всіх розрядів сигнали перенесення також формуються паралельно. Формування сигналів перенесення відбувається в результаті формування й оброблення додаткових двох сигналів: розповсюдження перенесення (*CRP – Carry Propagation*) і генерування перенесення (*Carry Generation*). Затримка отримання суми в такому суматорі складається із однакових значень затримки перенесення для всіх розрядів, які не залежать від кількості розрядів. Слід зазначити, що апаратні затримки в такій схемі швидко зростають відповідно до збільшення кількості розрядів, тому в чистому вигляді такий тип суматора майже не використовується. Для незначної кількості розрядів (до 8) промисловість випускає спеціальні мікросхеми – схеми прискореного перенесення.

Для прискорення формування сигналів перенесення у суматорах зі значною кількістю розрядів використовується принцип групового перенесення, відповідно до якого розрядна сітка розбивається на декілька груп з однаковою кількістю розрядів у кожній. Кожна група являє собою суматор, усередині якого перенесення може формуватися, як послідовно, так і паралельно, а перенесення із групи в групу відбувається за трактом міжгрупового перенесення, який може буде побудований, як паралельний, так і послідовний. У паралельному тракті перенесення між групами формуються як функції лише доданків. У послідовному тракті сигнал перенесення формується в кожній групі і з виходу молодшої групи надходить на вхід перенесення наступної старшої групи.

Використання паралельного перенесення всередині групи з паралельним перенесенням між групами дозволяє будувати найбільш швидкодіючі суматори з розрядністю 24...64 біти. Взагалі, вибір схеми суматора і формування перенесення в кожному випадку виконується за результатами аналізу апаратних витрат і забезпеченням необхідної швидкодії.

Суматори є елементною базою для побудови цифрових компараторів і арифметично-логічних пристроїв.

**Контрольні питання:**

1 Яке призначення схеми суматора?
2 Які умови формування сигналів на виходах однорозрядного суматора?
3 В чому полягає причина низької швидкодії паралельного суматора?
4 Якими способами можливо підвищити швидкодію суматора?

**Контрольні питання підвищеної складності:**

1 В чому полягає принцип групового перенесення?
2 Поясніть причину формування затримки сигналу у схемі паралельного суматора?

## 4.2 Цифрові компаратори

**Вхідний контроль:**

1 Який пристрій називається компаратором?
2 Для чого використовується компаратор?

Цифровим компаратором називають функціональний вузол обчислювальної техніки, який порівнює два багаторозрядних числа $A$ і $B$ між собою і результат порівняння подає у вигляді сигналів співвідношення між ними $A = B$, $A < B$, $A > B$.

Для порівняння сигналів використовується суматор, який виконує операцію віднімання (алгебраїчного додавання) вхідних чисел і за цими результатами формуються сигнали, що показують співвідношення між ними. Для виконання операції віднімання одне із вхідних чисел подається у доповнювальному коді на відповідні входи суматора, тому до складу цифрового компаратора входить схема, яка формує доповнювальний код одного з операндів. За результатами виконання операції $A - B$ можна зробити такі висновки:

– якщо результат дорівнює 0, то це є ознакою того, що значення $A$ і $B$ співпадають;

– якщо результат не дорівнює 0, а вихідне переповнення дорівнює 1, то $A < B$;

– якщо вихідне переповнення дорівнює 0, то це є ознакою, що $A > B$.

Сигнали, що показують співвідношення між вхідними числами формуються за допомогою логічних схем. Функціональна схема цифрового компаратора для чотирирозрядних чисел наведена на рис. 4.4.

Умовне графічне позначення компаратора, який показано на рис. 4.4, наведено на рис. 4.5.

Для нарощування розрядності вхідних даних цифрові компаратори дозволяється каскадувати, для чого набір вхідних сигналів компаратора доповнюють вихідними сигналами попереднього каскаду $A = B$, $A < B$, $A > B$ (рис. 4.5). Формування кінцевого результату порівняння відбувається з урахуванням цих сигналів.

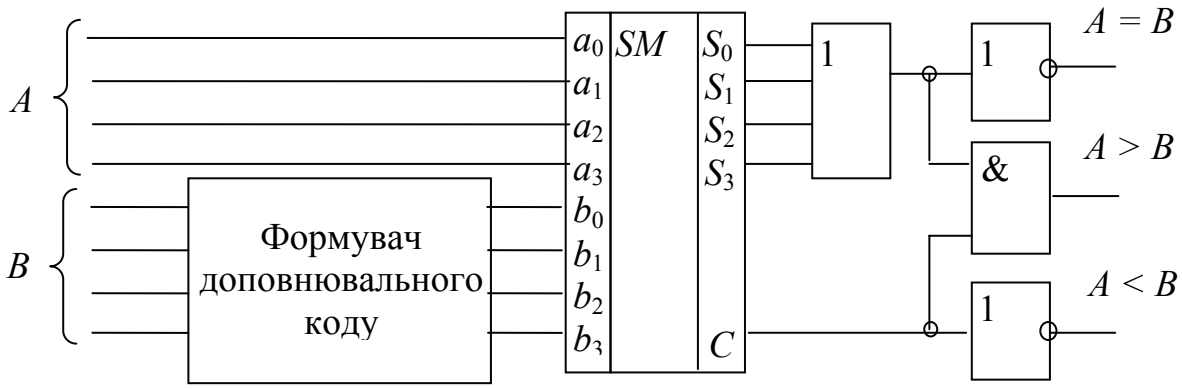



Рисунок 4.4 – Функціональна схема чотирирозрядного цифрового компаратора

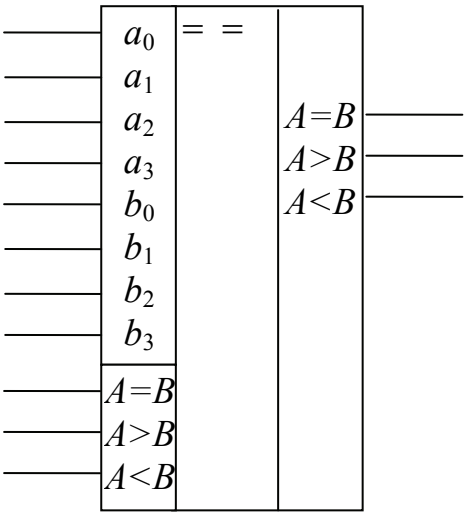



Рисунок 4.5 – Умовне графічне позначення чотирирозрядного цифрового компаратора

**Контрольні питання:**

1 Як виконується порівняння двох двійкових чисел у цифровому компараторі?

2 Поясніть призначення формувача доповнювального коду у схемі, що наведена на рис. 4.4.

**Контрольні питання підвищеної складності:**

1 На підставі якої ознаки приймається рішення, що $A = B$?

2 На підставі яких ознак приймається рішення, що $A > B$?

## 4.3 Арифметично-логічний пристрій

**Вхідний контроль:**

1 Які арифметичні операції використовуються в обчислювальній техніці?

2 Які логічні операції використовуються в обчислювальній техніці?

3 Які правила використовуються при виконанні арифметичних операцій в обчислювальній техніці?

Арифметично-логічні пристрої (АЛП) широко використовуються для побудування арифметичних вузлів, зокрема, АЛП є обов'язковою складовою частиною будь-якого процесора. АЛП використовується для виконання арифметичних і логічних операцій з даними, які до нього надходять і являють собою числа або будь-який інший вид інформації. Кількість розрядів у даних, над якими виконується операція, звичайно співпадає з кількістю розрядів в основних регістрах МПС. АЛП випускають як окремі мікросхеми або вони є невід'ємною складовою мікросхеми центрального процесора.

Операції, які виконуються в АЛП, можна поділити на такі групи:

- арифметичні операції над двійковими числами з фіксованою точкою;
- арифметичні операції над двійковим числами з плаваючою точкою;
- операції десяткової арифметики;
- операції індексної арифметики (для модифікації адрес команд);
- логічні операції з операндами, які є логічними змінними;
- спеціальні арифметичні операції;
- операції над алфавітно-цифровими полями.

До арифметичних операцій відносяться додавання, віднімання, множення і ділення двійкових і двійково-десяткових чисел, операції з рядками даних. До групи логічних операцій входять: диз'юнкція (логічне АБО), кон'юнкція (логічне ТА), виключне АБО, інверсія, логічні зсуви, порівняння кодів між собою тощо. До групи спеціальних арифметичних операцій входять операції нормалізації даних, арифметичні зсуви тощо.

Арифметичні операції в АЛП виконуються за допомогою багаторозрядного суматора, а логічні – відповідними логічними схемами. В залежності від характеру використання цих пристроїв АЛП поділяються на **блочні** і **багатофункційні**. В блочних АЛП для виконання різних типів операцій використовуються окремі блоки. Це дає змогу підвищити швидкодію, але збільшує апаратні витрати. В багатофункційних АЛП для виконання операцій різних типів використовуються одні й ті ж пристрої, які комутуються по-різному, в залежності від виконуваної операції. Вибір операції і необхідних блоків здійснюється за допомогою сигналів керування, якими кодується операція, яку необхідно виконати. Крім результату АЛП формує набір спеціальних сигналів – ознаки результату (прапорці). Умовне позначення АЛП на схемах показано на рис. 4.6.

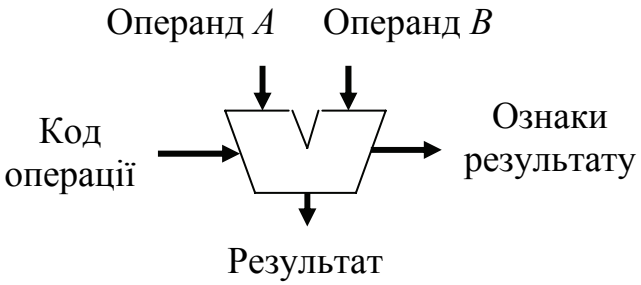



Рисунок 4.6 – Умовне графічне позначення АЛП

Тому що двійковий суматор, який використовується в АЛП для виконання арифметичних операції, є комбінаційним пристроєм, то для одночасного подання двох операндів на його входи необхідно мати запам'ятовувальні пристрої для тимчасового зберігання операндів і результату.

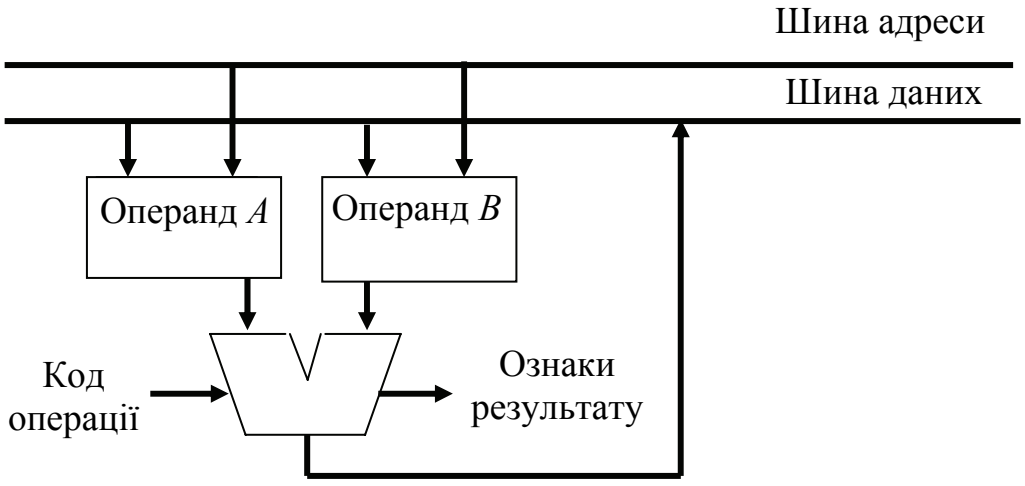
До АЛП операнди можуть надходити з двох регістрів загального призначення (РЗП) або із регістра і комірки пам'яті, а результат надходить до регістра або до комірки пам'яті, які визначені як приймач результату. Така система має назву **двоадресної**. Тому МПС повинна подати адреси обох операндів. Схему організації двоадресної системи показано на рис. 4.7, а.

В інших МПС один з операндів до початку виконання команди обов'язково зберігається в акумуляторі, а після закінчення результат записується на місці операнда, який безповоротно втрачається. Така система має назву **одноадресної**, тому що необхідно адресувати лише один операнд. Схему організації одноадресної системи показано на рис. 4.7, б.

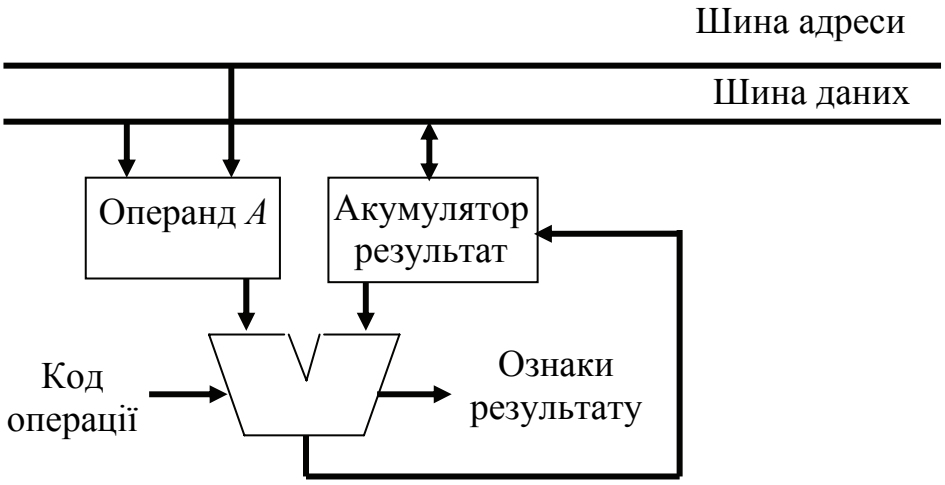
До важливої характерної ознаки АЛП відноситься формування ним ознак результату (прапорців) – тобто деяких властивостей результату, які можуть вплинути на подальше виконання програми. Ознаки результату формуються після отримання результату і записуються у спеціальний регістр – регістр ознак результату (або інші назви цього регістра – регістр прапорців, регістр стану), де вони зберігаються до формування результату наступної операції. Слід зазначити, що ознаки результату формують лише команди арифметичних і логічних операцій. Регістр ознак, у залежності від типу мікропроцесора, може мати різну кількість розрядів, які відповідають властивостям результату і зберігають інформацію про стан деяких апаратних або програмних компонентів процесора. Формат регістра ознак для відповідної мікросхеми може бути таким, як показано на рис. 4.8.

Кожна з ознак результату (прапорець) зберігається у одному розряді регістра і кодує наявність відповідної ознаки значенням 0 або 1. Розглянемо кожну з ознак результату з короткою характеристикою:

– *C* (*carry*) – ознака зберігає значення перенесення до наступного старшого розряду за межами розрядної сітки або значення позики з цього розряду;




а)




б)

Рисунок 4.7 – Схема організації двоадресного (а) і одноадресного (б) АЛП

| *C* | *S* | *Z* | *AC* | *OVR* | *P* |
|---|---|---|---|---|---|
| Ознака перенесення (позики) | Ознака знаку | Ознака нуля | Ознака допоміжного перенесення | Ознака переповнення | Ознака парітету |

Рисунок 4.8 – Формат регістра і назви ознак результату

– *S* (*sign*) – розряд регістра зберігає копію знакового розряду результату операції. Значення цієї ознаки, що дорівнює 1 відповідає від'ємному результату;

– *Z* (*zero*) – ознака нульового результату. Для результату, що дорівнює 0 – ознака набуває значення 1;

– *AC* (*auxiliary carry*) – ознака зберігає значення перенесення зі старшого розряду молодшої тетради у молодший розряд старшої тетради байта

при додаванні двійково-десяткових чисел або – значення позики зі старшої тетради до молодшої при відніманні;

– *OVR* (*overflow*) – ознака, яка сигналізує про втрату старшого біта результату додавання або віднімання чисел зі знаком, що обумовлено наявністю перенесення у знаковий розряд. При додаванні ознака набуває значення 1, якщо є перенесення у старший (знаковий) біт результату, але немає перенесення з нього і навпаки якщо є перенесення у біт *C*, але немає перенесення у старший (знаковий) біт. При відніманні ознака дорівнює 1, якщо є позика зі старшого (знакового) біта результату, але немає позики з нього і навпаки, якщо є позики з біта *C*, але немає позики зі старшого (знакового) біта. Значення ознаки дорівнює результату виконання логічної операції ВИКЛЮЧНЕ АБО над значеннями бітів *C* і перенесенням до старшого (знакового) біта результату

$$C \oplus C_{D7},$$

де $C_{D7}$ – перенесення до старшого (знакового) біта результату;

– *P* (*parity*) – значення цієї ознаки дорівнює 0, якщо результат операції складається з непарної кількості одиниць.

Формування ознак покажемо на прикладі виконання арифметичної і логічної операцій для восьмирозрядних чисел:

– арифметична операція над двійковими числами

$$\begin{array}{r} 11101111 \\ +\ 01111000 \\ \hline 101100111 \end{array} .$$

$C \quad AC$

Якщо вважати, що операція виконується над беззнаковими числами, то це $239_D + 120_D = 359_D$ і результат з урахуванням перенесення правильний, якщо вважати що числа зі знаками, то це $(-17_D) + 120_D = +103_D$ . Враховуючи, що числа зі знаком подані в доповнювальному коді, то результат також правильний. Ознаки результату не залежать від того, над якими числами виконувалась операція. Вони набувають таких значень:

$C = 1$ – перенесення у 9 розряд має місце;

$S = 0$ – восьмий розряд результату, що кодує знак, дорівнює 0, результат додатний;

$Z = 0$ – результат не дорівнює нулю;

$AC = 1$ – відбулося перенесення з молодшої тетради до старшої. Слід сказати, що АЛП виставляє ознаки, формально, за їх наявності, не враховуючи конкретні обставини виконання операції;

$OVR = 1$ – відбулося переповнення розрядної сітки (біт *C*) і не відбулося перенесення до старшого (знакового) розряду.

$$C \oplus D7 = 1 \oplus 0 = 1.$$

Якщо це числа зі знаком і передбачається їх подальша обробка, то необхідно збільшити довжину розрядної сітки;

$P$ = 0 – результат (без урахування перенесення) складається з непарної кількості (5) одиниць;

– логічна операція кон'юнкція над восьмирозрядними логічними змінними

$$\begin{array}{r} 11101111 \\ \wedge\ 01111000 \\ \hline 01101000 \end{array} .$$

При виконанні логічних операцій перенесення не відбувається, тому ознаки *C, AC* і *OVR* не формуються, а зберігають свої значення, яких набули при попередній операції. Ознака *S* формується формально копіюванням старшого розряду результату. Після виконання логічної операції ознаки набувають таких значень:

$C = X$ – зберігає попереднє значення;

$S$ = 0 – восьмий розряд результату, що кодує знак, дорівнює 0;

$Z$ = 0 – результат не дорівнює нулю;

$AC = X$ – зберігає попереднє значення;

$OVR = X$ – зберігає попереднє значення;

$P$ = 0 – результат складається з непарної кількості (3) одиниць.

Умовне графічне позначення мікросхеми чотирирозрядного АЛП наведено на рис. 4.9.

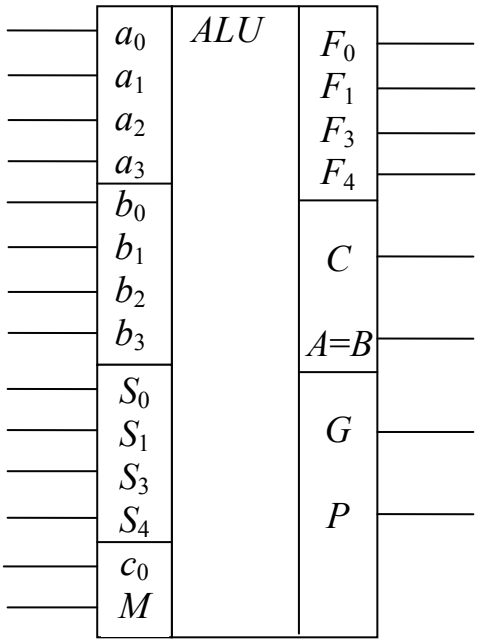



Рисунок 4.9 – Умовне графічне позначення мікросхеми чотирирозрядного АЛП

Виводи цієї мікросхеми мають таке функціональне призначення:

- $a_0 – a_3$ – входи операнда $A$;
- $b_0 – b_3$ – входи операнда $B$;
- $S_0 – S_3$ – входи для подання коду операції, що буде виконуватись;

– $c$ – вхідне перенесення;

– $M$ – код режиму роботи $M = 1$ – АЛП може виконувати 16 логічних операцій, $M = 0$ – 16 арифметичних операцій;

– $F_0 – F_3$ – результат виконання операції;

– $C$ – вихідне перенесення;

– $A = B$ – ознака рівності вхідних операндів;

– $G$ – вихід сигналу генерування перенесення;

– $P$ – вихід сигналу розповсюдження перенесення.

Останні два сигнали використовуються для нарощування розрядності операндів і збільшення швидкодії схеми, яка буде отримана.

**Контрольні питання:**

1 Яке місце займає АЛП у складі мікропроцесора?

2 Чому операнди, над якими необхідно виконати операцію в АЛП, необхідно спочатку записати в два запам’ятовувальні пристрої – регістри?

3 Які ознаки результату формуються після виконання операції в АЛП і яке їх призначення?

4 Які ознаки результату не змінюються при виконанні логічних операцій?

**Контрольні питання підвищеної складності:**

1 Які ознаки результату будуть сформовані при виконанні арифметичної операції

$$+89_D - 120_D.$$

2 Які ознаки результату будуть сформовані при виконанні логічної операції

$$11001110 \vee 01100011.$$

3 Яке значення буде мати ознака $P$, якщо результат виконання операції дорівнює 11100001?

## 4.4 Програмовані логічні інтегральні схеми (ПЛІС)

У мікропроцесорних системах у багатьох випадках, наприклад, у телекомунікаціях, при виконанні цифрового оброблення сигналів можна доповнювати, а іноді навіть заміняти мікропроцесорні засоби на програмовані логічні інтегральні схеми (ПЛІС).

На них можна будувати спеціалізовані цифрові пристрої, керувальні засоби – контролери, і застосовувати у тих областях, де апаратним рішенням задач можна віддати перевагу порівняно з програмними рішеннями, які завжди є послідовними. Застосування ПЛІС забезпечує паралельне оброблення різних гілок одного обчислювального процесу і, таким чином, збільшує продуктивність системи іноді у десятки разів. У той самий час зберігається така ж сама гнучкість реалізації алгоритмів, як і при програмному способі. Задавати алгоритм дії проектованого цифрового пристрою для реалізації на ПЛІС можна

у вигляді часових діаграм, текстового опису, схем на логічних елементах, у вигляді логічних функцій. Для отримання оптимального алгоритму використовується процедура мінімізації, як було показано вище. Використовуючи засоби САПР, розробники отримують файл, який використовується потім при програмуванні ПЛІС на програматорі. ПЛІС – це надвелика інтегральна схема, яка вміщує на кришталі універсальні налаштовувані користувачем функціональні перетворювачі та програмовані зв'язки між ними. ПЛІС можуть реалізовувати блоки пам'яті, блоки цифрової обробки сигналів, вбудовані процесорні ядра з периферією, швидкісні канали введення-виведення.

Окремі цифрові пристрої – шифратори, перетворювачі кодів, спеціальні функціональні пристрої, декодери адрес тощо, як правило, входять до складу мікропроцесорних систем. Вони не є складними й описати алгоритм їхньої роботи можна у вигляді логічної функції не більше як від трьох до п'яти логічних змінних. Синтез таких логічних схем можна здійснити після мінімізації логічних функцій, які їх описують. Їх можливо реалізовувати на трьох типах програмованих логічних пристроїв:

- програмовані логічні матриці ПЛМ (*PLA – programmable logic array*);
- програмована матрична логіка ПЛМ (*PAL – programmable array logic*);
- вентильна матрична логіка ВМЛ (*GAL – gated array logic*).

Останнім часом елементи *PAL* знайшли широке застосування у МПС для побудування адресних дешифраторів. Схема такого пристрою складається з двох матриць логічних пристроїв – матриці логічних елементів АБО та матриці логічних елементів ТА. Програмування такої структури полягає у перепалюванні перемичок між матрицями, в результаті чого формується схема, що відповідає заданому алгоритму роботи.

**Контрольні питання:**

1 Чому використання ПЛІС дозволяє збільшити продуктивність МПС?
2 Які типи мікросхем ПЛІС Ви знаєте?
3 Для чого використовуються мікросхеми *PAL* у МПС*?*

**Контрольні питання підвищеної складності:**

1 Поясніть, для чого необхідно використовувати мінімізацію при розробці алгоритму робити ПЛІС.

# 5 ПРИНЦИПИ ПОБУДОВИ ЗАПАМ'ЯТОВУВАЛЬНИХ ПРИСТРОЇВ МПС З ЗАДАНОЮ ОРГАНІЗАЦІЄЮ

**Вхідний контроль:**

1 Які підсистеми входять до складу МПС?

2 Яке призначення має підсистема пам'яті у складі МПС?

## 5.1 Запам'ятовувальні пристрої МПС та їх класифікація

Підсистема пам'яті є однією зі складових частин МПС, яка значною мірою визначає її продуктивність і обчислювальні можливості. До підсистеми пам'яті входять технічні пристрої, які називаються **запам'ятовувальними пристроями** (ЗП) і призначені для зберігання двійкової інформації. Основними операціями у пам'яті є запис, зберігання і вибірка (читання) інформації. Сукупність операції запису і вибірки називається **зверненням до пам'яті**.

ЗП будуються з двопозиційних **елементів пам'яті** (ЕП)**,** кожен з яких зберігає один біт інформації. Сукупність декількох елементів пам'яті створюють **комірку пам'яті,** яка призначена для зберігання багаторозрядної двійкової інформації і звернення до елементів якої відбувається одночасно. Звернення до ЗП відбувається за адресним принципом, який передбачає наявність у кожній комірці пам'яті відповідного номера, що називається **адресою** і яку необхідно явно чи неявно вказувати при зверненні. Крім адресних, використовують асоціативні ЗП, звернення до комірок яких відбувається за результатами аналізу деяких розрядів інформації, яка зберігається.

Класифікувати і порівнювати ЗП можливо за багатьма різними критеріями в залежності від потреби користувача і технічних вимог до побудови пам'яті. Основними критеріями, які визначають побудови і функціонування ЗП, є: фізичний принцип роботи запам'ятовувальних елементів і технологія їхнього виготовлення, доступ до комірок пам'яті, швидкість обміну інформацією, спосіб зберігання інформації тощо.

Так, в залежності від природи фізичного середовища, в якому зберігається інформація, ЗП поділяються на напівпровідникові, пристрої із зарядним зв'язком (ПЗЗ), магнітні, оптичні тощо. Надалі будемо розглядати напівпровідникові ЗП.

Для порівняння запам'ятовувальних пристроїв між собою розроблена система параметрів, що характеризує їхні властивості. До таких параметрів відносяться:

– **інформаційна ємність** – це кількість одиниць інформації, що одночасно може зберігатися у ЗП і визначається у двійкових одиницях бітах або байтах. У разі використання одиниці вимірювання – байт, ємність пам'яті подають через число $K = 1024$ байт = 1 кбайт;

– **розрядність даних** – визначається вмістом (кількістю розрядів) комірки пам'яті;

– **організація пам’яті** – це параметр, який поєднує два попередніх у вигляді виразу

$$N \times n,$$

де $N$ – кількість комірок пам’яті, які входять у ЗП; $n$ – розрядність даних.

Добуток, який буде отримано при виконанні множення, буде дорівнювати інформаційній ємності в тій одиниці вимірювання, яка визначена розрядністю даних;

– **час вибірки** – інтервал часу з моменту надходження запиту на передачу даних до моменту їхньої появи на виході ЗП;

– **тривалість циклу звернення (цикл пам’яті)** – мінімальний інтервал часу між двома сусідніми зверненнями до ЗП, кожен з яких буде нормально виконуватись;

– **напруга живлення ЗП** – визначає вибір необхідного пристрою живлення;

– **потужність енергоспоживання** – електрична потужність, яку споживає ЗП при роботі. Для деяких типів ЗП розрізняється потужністю енергоспоживання у режимі звернення (запис, читання) й у режимі зберігання. Потужність енергоспоживання у режимі звернення набагато більша за потужність у режимі зберігання;

– **питома вартість –** визначає співвідношення вартості та інформаційної ємності ЗП.

Вимоги до обсягу і швидкодії пам’яті є суперечними. Досягнення високої швидкодії є досить важкою технічною задачею, вирішення якої потребує додаткових витрат для досягнення необхідного обсягу пам’яті. Взагалі вартість ЗП складає значну частину витрат на апаратну частину МПС. Для раціональної організації пам’яті, відповідно до принципів фон Неймана, передбачено класифікацію ЗП зі швидкодії, яка окреслює певну ієрархію пристроїв пам’яті. Це необхідно для забезпечення високої швидкості обміну інформацією між пам’яттю й арифметично-логічним пристроєм (АЛП) мікропроцесора за великої ємності ЗП МПС. Ієрархія ЗП МПС показана на рис. 5.1. Вершину цієї ієрархії складають найбільш швидкодіючі ЗП певної МПС, які входять до складу центрального процесора – надоперативний ЗП (НОЗП) і внутрішня кеш-пам’ять.

**Надоперативний ЗП** (НОЗП) складається з регістрів загального призначення центрального процесору (ЦП), кількість яких залежить від архітектури процесора і складає від 16 до 64 регістрів, кількість розрядів у кожному з котрих визначається регістровою моделлю процесора. Ці регістри не потребують виставлення адреси та її оброблення при зверненні до них, тому швидкість роботи з ними максимальна.

**Внутрішня кеш-пам’ять** – це різновид програмно недоступної оперативної пам’яті ємністю 1 – 16 кбайт, яка вбудована у ЦП. В залежності від типу процесора може бути одна кеш-пам’ять, спільна для даних і команд, або дві окремих.

**Зовнішня кеш-пам’ять** – це також оперативна пам’ять, яка установлюється на системній платі і призначена для прискорення процедури звернення до інших типів пам’яті, що входять до складу МПС. Ємність цієї пам’яті становить 64 кбайт – 1 Мбайт і постійно зростає у кожній наступній моделі комп’ютера. Швидкість роботи цієї пам’яті визначається швидкодією системної шини.

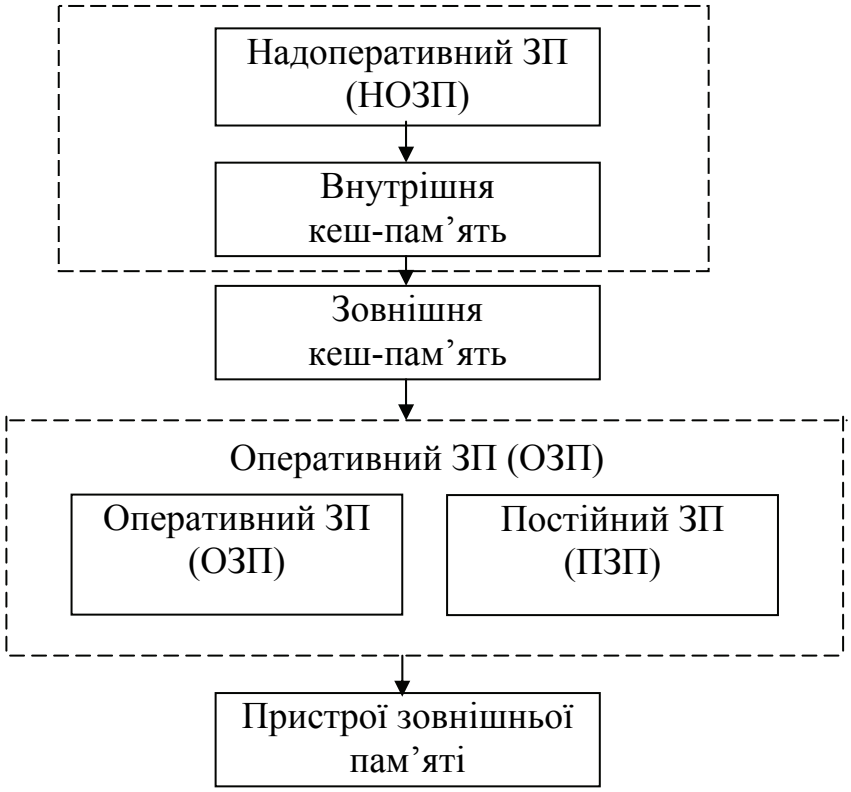



Рисунок 5.1 – Ієрархія пам’яті МПС

**Оперативний запам’ятовувальний пристрій** (ОЗП) – основний тип пам’яті МПС, який значною мірою визначає властивості МПС. Складається з двох видів пам’яті – **постійного запам’ятовувального пристрою** і, власне, **оперативного запам’ятовувального пристрою**. ОЗП будується за адресним принципом і має довільний доступ до кожної з комірок пам’яті.

**Постійний запам’ятовувальний пристрій** (ПЗП) – будується з мікросхем постійних (ПЗП) або перепрограмованих запам’ятовувальних пристроїв (ППЗП). Ці мікросхеми не допускають оперативної зміни інформації, що зберігається в них під час виконання програми, і не втрачають її при відключенні живлення. Інформація до них записується під час виготовлення або при першому програмуванні мікросхеми і надалі змінитися не може. Тому, під час роботи МПС інформація з них лише зчитується. У ПЗП зберігаються таблиці кодів команд, константи, стандартні підпрограми, наприклад, *BIOS* тощо. Ємність ПЗП складає 64...128 кбайт. Вартість біта інформації, що зберігається у ПЗП, може бути майже на порядок нижче ніж у ОЗП.

**Оперативний запам’ятовувальний пристрій** (ОЗП) – пристрій пам’яті призначений для записування, зберігання і зчитування будь-якої інформації під

час виконання програми. В залежності від типу використовуваних мікросхем, ОЗП буває двох видів: **статичний** і **динамічний**. Статичний ОЗП будується з мікросхем, елементом пам'яті яких є транзисторний тригер. Такий елемент пам'яті може зберігати інформацію дуже довго, доки є живлення, незалежно від кількості звернень для читання цієї інформації. ОЗП динамічного типу як елемент пам'яті використовує конденсатор, який є паразитною ємністю деяких схем, що побудовані з транзисторів зі структурою метал-діелектрик-напівпровідник (МДН). У результаті саморозряду такої ємності, інформація, що записана, може пропадати, тому вони потребують періодичного її поновлення (регенерування). При регенеруванні у кожний запам'ятовувальний елемент записується інверсія значення, що зберігалася до читання. В динамічному ПЗП є схема, що вказує, яке значення знаходиться у комірці пам'яті – пряме або інверсне.

До **пристроїв зовнішньої пам'яті** відносяться всі нагромаджувачі зі змінними і незмінними носіями: на твердих і гнучких магнітних дисках, лазерних компакт-дисках (*CD-ROM*) тощо. Обмін інформацією з такими пристроями відбувається з малою швидкістю, що пояснюється наявністю в їхньому складі електромеханічних вузлів. У цілому, ємність пристроїв зовнішньої пам'яті не обмежена. Можливо говорити лише про окремі пристрої, наприклад, ємність нагромаджувачів на твердих дисках становить 1...100 Гбайт.

Наявність того чи іншого типу пам'яті визначається функціональним призначенням МПС і умовами її роботи.

Крім адресної пам'яті до складу МПС можуть входити пристрої з іншим механізмом звернення до окремих комірок. До них відносяться асоціативна та стекова пам'ять.

**Асоціативна пам'ять** відрізняється тим, що звернення до комірок відбувається не за адресою, а за асоціативними ознаками самої інформації, що визначаються у результаті порівняння її з необхідними. До критеріїв визначення необхідної комірки можна віднести: рівність вмісту комірки наперед визначеному числу або будь-які інші. При цьому пошук за асоціативною ознакою (або послідовно за окремими розрядами) виконується паралельно у часі для всіх комірок масиву інформації, що зберігається. Це дозволяє підвищити швидкість оброблення інформації.

**Стекова пам'ять (стек)** – це пам'ять з послідовним доступом, звернення до комірок якої відбувається за безадресним принципом, за алгоритмом *LIFO (Last Input First Output)* – останній увійшов – перший вийшов. У МПС стекова пам'ять широко використовується під час виклику підпрограм, обробленні переривань та обробленні вкладених структур даних.

Стекову пам'ять реалізують за апаратним або апаратно-програмним способами.

При апаратній реалізації стекова пам'ять організована як сукупність регістрів, які зв'язані між собою так, що при зверненні до стека вміст його даних автоматично зсувається у той чи інший бік (залежно від операції, яка відбувається), чим забезпечується виконання алгоритму *LIFO*.

При апаратно-програмній реалізації стек організують в ОЗП статичного типу, а для визначення адреси існує спеціальний регістр *SP* (*Stack Pointer*) – **вказівник стека**, який вміщує адресу комірки пам'яті, з якої починається робота за алгоритмом *LIFO*. Ця комірка пам'яті називається **вершиною стека** і її вміст змінюється в залежності від виконання алгоритму *LIFO* для запису або читання даних у стеку.

**Контрольні питання:**

1 Які параметри ЗП Ви знаєте?

2 Які параметри ЗП характеризують їх швидкодію?

3 Які принципи покладено в основу ієрархії ЗП МПС?

4 За якими принципами можливо здійснити звернення до певної комірки пам'яті у МПС?

**Контрольні питання підвищеної складності:**

1 Чим можливо пояснити меншу швидкодію зовнішніх пристроїв пам'яті?

2 За яким принципом відбувається звернення до комірок пам'яті у стеку?

3 Поясніть призначення вказівника стека.

## 5.2 Постійні запам'ятовувальні пристрої – флеш-пам'ять

**Вхідний контроль:**

1 Які технології виробництва мікросхем Ви знаєте?

2 Чим відрізняються одна від одної мікросхеми з різними технологіями?

**Флеш-пам'ять** (*Flash*-пам'ять). Мікросхеми пам'яті такого типу були розроблені фірмою Intel у 1988 році.

Як ЕП флеш-пам'яті використовується МОН-транзистор з ПЗ, який виготовлено за спеціальною технологією, яка називається *ETOX* (*EPROM Thin Oxide*) і запатентована фірмою *Intel*. У цілому, структура МОН-транзистора з ПЗ подібна описаним вище. Відмінністю, яка забезпечуються технологією *ETOX,* є зменшення товщини прошарку оксиду кремнію більш ніж втричі, що дозволило зменшити напругу програмування до 12 В і зменшити напругу стирання за рахунок тунельного ефекту, також до 12 В. Ці заходи дозволяють виконувати перепрограмування флеш-пам'яті безпосередньо у складі МПС і забезпечують можливість збільшення кількості циклів запису інформації.

Для забезпечення правильної організації роботи флеш-пам'яті фірмою *Intel* розроблена низка заходів, що дозволяють уникнути виходу її з ладу під час програмування. До них можна віднести:

– застосування спеціальних алгоритмів запису і стирання з контролем стану і завершенням процесу за результатами контролю;

– попереднє програмування в режимі стирання, коли перед стиранням усі ЕП матриці установлюються в стан 0;

– включення до складу мікросхеми регістра, який зберігає ідентифікатори фірми-виробника й типу мікросхеми, що дозволяє захистити елемент від помилок вибору алгоритму;

– вбудування в мікросхему кіл, що реалізують алгоритм стирання і запису. Це спрощує зовнішнє керування і захищає від помилок під час перезапису.

Існує три групи мікросхем флеш-пам'яті:

– мікросхеми першого покоління, які виготовлені у вигляді єдиного масиву (блока), інформація в якому стирається цілком (*BULK-ERASE*);

– мікросхеми, масив пам'яті яких поділено на блоки різного розміру, що мають різні рівні захисту від випадкового звернення до них (*BOOT-BLOCK*);

– мікросхеми третього покоління, які мають найбільший розмір масиву, що поділено на блоки однакового розміру з незалежним стиранням (*FLASH-FILE*).

Мікросхеми різних груп мають відмінності в їх використанні. Так, мікросхеми *BULK-ERASE* можуть використовуватись замість традиційних мікросхем *EPROM*, з можливістю перепрограмування безпосередньо у складі обладнання під керівництвом процесора самої системи. Мікросхеми *BOOT-BLOCK* застосовуються для зберігання *BIOS* у персональних комп'ютерах, що дає можливість оновлення системи безпосередньо з зовнішніх носіїв інформації. Мікросхеми *FLASH-FILE* використовуються для зберігання даних великого обсягу в *Flash*-картах, які є альтернативою жорстким магнітним дискам. Очікується, що *Flash*-картки зможуть замінити жорсткі магнітні диски, особливо у системах, що працюють в умовах сильних механічних впливів.

Швидкодія флеш-пам'яті в 125…250 разів перевищує цей параметр для жорсткого диска, але поступається йому щодо інформаційної ємності.

Напруга живлення мікросхем флеш-пам'яті становить – 5 В, а стирання і програмування –12 В. Споживаний струм суттєво залежить від режиму роботи мікросхеми. Так, в режимі очікування (*Standby*) споживаний струм значно менший за струм, який споживається у режимі стирання і запису, переважно у колі джерела 12 В.

Більшість мікросхем флеш-пам'яті працюють з даними у вигляді послідовного коду з використанням шини $I^2C$ (*Inter Integrated Circuit Bus*). Ця шина складається з двох двонаправлених ліній: *SCL* (*Serial Clock*) і *SDA* (*Serial Date*), до яких можна підключати до 128 пристроїв. Один з пристроїв є ведучим (*master*), інші – веденими (*slave*). Ведучий пристрій генерує імпульси синхронізації *SCL* і керує всією роботою шини. Ведені працюють під керуванням ведучого, обслуговуючи його запити.

Типова структурна схема мікросхеми флеш-пам'яті з послідовним введенням-виведенням інформації з використанням шини $I^2C$ показана на рис. 5.1.

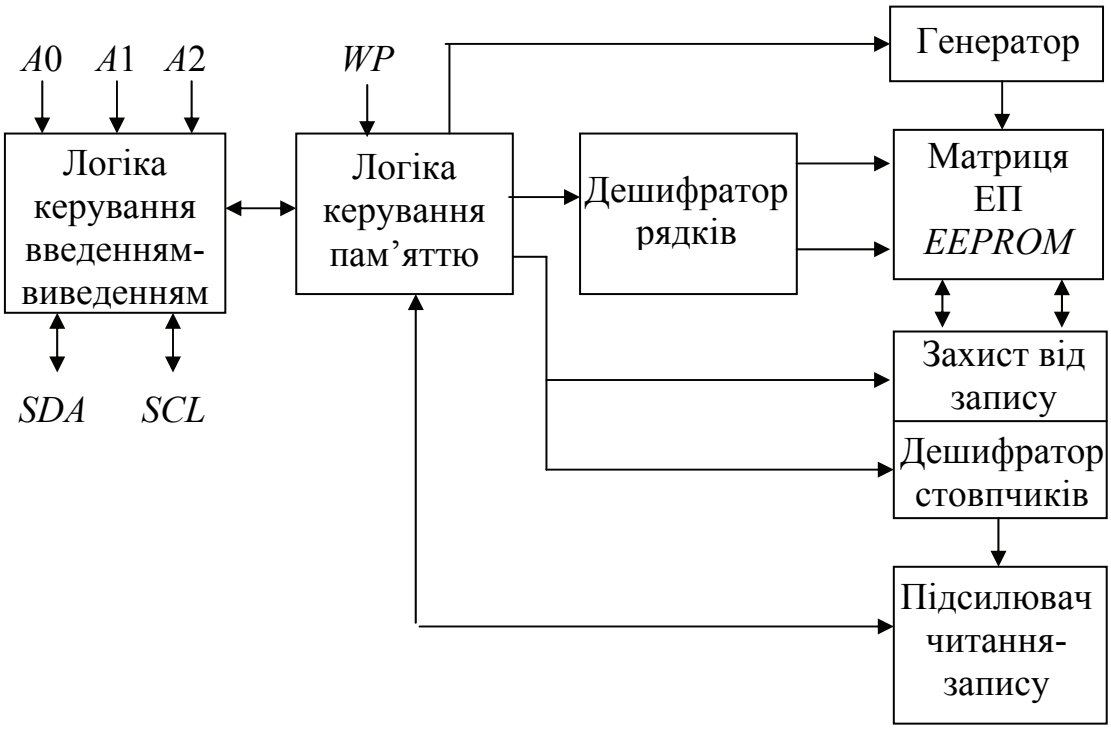



Рисунок 5.1 – Структурна схема ВІС флеш-пам'яті, що працює з шиною $I^2C$

На рис. 5.1 на блок логіки керування введенням-виведенням інформації надходять з шини $I^2C$ сигнали $A0$, $A1$, $A2$, які визначають три молодші розряди адреси веденої ВІС, а старші чотири розряди адреси пам'яті не змінюються і мають значення 1010. По лініях *SCL* і *SDA* проводиться обмін відповідними сигналами. Вхід *WP* (*Write Protect*) блока логіки керування пам'яттю призначений для керування захистом даних, що записані у ВІС. Якщо цей вивід з'єднаний з загальним проводом, то можливо змінювати вміст будь-яких комірок пам'яті. Сигнал високого рівня, що подано на цей вивід захищає увесь масив даних або його частину від змін. Нині випускаються мікросхеми, в яких блокування стирання і запису можливо виконувати в інший спосіб, наприклад, за командами ведучого.

У неактивному стані шини $I^2C$ на лініях *SCL* і *SDA* присутні високі рівні сигналу. Для початку сеансу роботи ведучий змінює стан лінії *SDA* на низький, не змінюючи стану лінії *SCL*. Після установлення низького рівня на лінії *SDA* змінюється стан лінії *SCL,* що відповідає команді СТАРТ. Передавання інформації відбувається побітно. Сеанс передавання закінчується командою СТОП, під час якої на лінії *SCL* установлюється високий рівень сигналу і за його наявності відбувається зміна стану лінії *SDA*. Часові діаграми процесу передавання інформації показано на рис. 5.2.

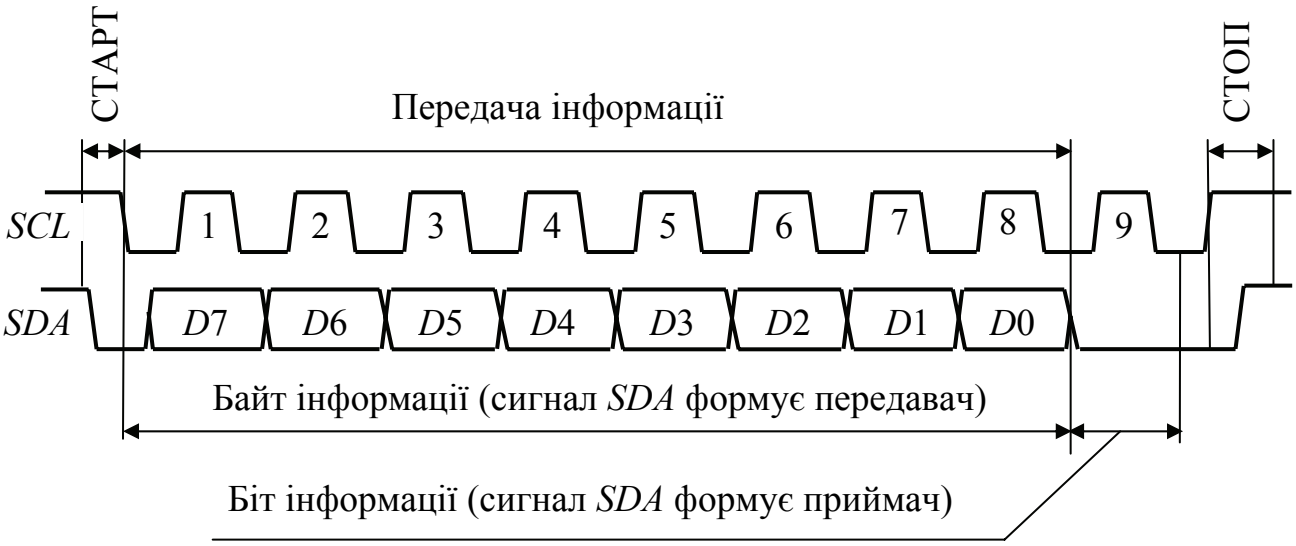



Рисунок 5.2 – Часові діаграми процесу передавання інформації

Частота проходження імпульсів синхронізації *SCL* становить 100 (400) кГц. При передаванні першим передається старший байт, а останнім – молодший. Пристрій, який прийняв байт, підтверджує його прийом, установлюючи на лінії *SDA* сигнал низького рівня, після чого формується сигнал СТОП.

Сім бітів, що передаються безпосередньо за командою СТАРТ, є адресою веденого пристрою, з яким необхідно установити зв’язок. Пристрій з такою адресою підтверджує приймання і готується до наступної роботи. Якщо у МПС немає пристрою з адресою, що передана по шині, то всі пристрої відключаються.

Молодший біт першого байта – це ознака напряму передавання. Значення 0 цього біта відповідає напряму від ведучого до веденого і не може змінитися у поточному сеансі роботи.

**Контрольні питання:**

1 Як проводиться програмування ПЗП, програмованих маскою?

2 Який пристрій використовується як елемент пам’яті у ПЗП з ультрафіолетовим стиранням?

3 Як відбувається робота флеш-пам’яті з шиною $I^2C$?

**Контрольні питання підвищеної складності:**

1 Які заходи застосовуються для уникнення виходу з ладу ВІС флеш-пам’яті?

2 Які типи ВІС флеш-пам’яті Ви знаєте?

3 Чим відрізняються ВІС РПЗП-УФ від ВІС РПЗП-ЕС?

## 5.3 Оперативні запам'ятовувальні пристрої

**Вхідний контроль:**

1 Які типи тригерів Ви знаєте?

2 Які характерні особливості мають тригери різних типів?

3 Сигнали, які необхідно використовувати для керування тригерами різних типів?

**ОЗП статичного типу (*Static Random Access Memory – SRAM*)** призначений для оперативного запису, зберігання і зчитування інформації під час виконання МПС будь-яких програм. У ОЗП статичного типу інформація зберігається у тому місці (комірці пам'яті або запам'ятовувальному елементі), де вона була записана і не руйнується під час її зчитування.

Структура ВІС ОЗП схожа на структуру мікросхем ПЗП з тією різницею, що як ЕП використовується транзисторний статичний тригер. Елементною базою для побудування тригерів можуть бути як біполярні, так і МОН-транзистори. Через те що, для функціонування тригера потрібне живлення, пам'ять такого типу є енергетично залежною (*volatile memory*). При відключенні живлення інформація, що зберігалася, втрачається. Запис інформації у тригер відбувається шляхом установлення в один з двох його можливих станів. Для зміни стану необхідно подати на входи тригера необхідні сигнали запису.

Типова структура ОЗП статичного типу включає: матрицю нагромаджувача і схеми запису-зчитування інформації, схеми дешифрування адреси ЕП або комірки пам'яті, схеми управління режимом тощо, які інтегровані на одному кристалі. У залежності від побудови нагромаджувача розрізняють ОЗП з однорозрядною та багаторозрядною організацією пам'яті. Основна відміна у структурних схемах полягає у тому, що нагромаджувач ОЗП з багаторозрядною організацією пам'яті складається з кількох прошарків однакових матриць і одну комірку пам'яті складають елементи з однаковими адресами в усіх матрицях.

Організація пам'яті однорозрядного ОЗП становить $2^m \times 1$ біт, де $m$ – кількість розрядів шини адреси, що можуть бути підключені до цієї ВІС, а для багаторозрядного ОЗП становить $2^m \times n$ біт, де $n$ – кількість розрядів шини даних. Багаторозрядні *SRAM*, переважно, мають байтову організацію ($2^m \times 8$).

Спрощену структурну схему багаторозрядного ОЗП з організацією $2^m \times 4$ показано на рис. 5.3.

У схемі, яка зображена на цьому рисунку, показано, що звернення проводиться одночасно до чотирьох матриць нагромаджувача по одному ЕП в кожній. Розряди шини адреси розподіляються на дешифратор рядків і стовпчиків для вибору відповідних рядків і стовпчиків одночасно в чотирьох матрицях.

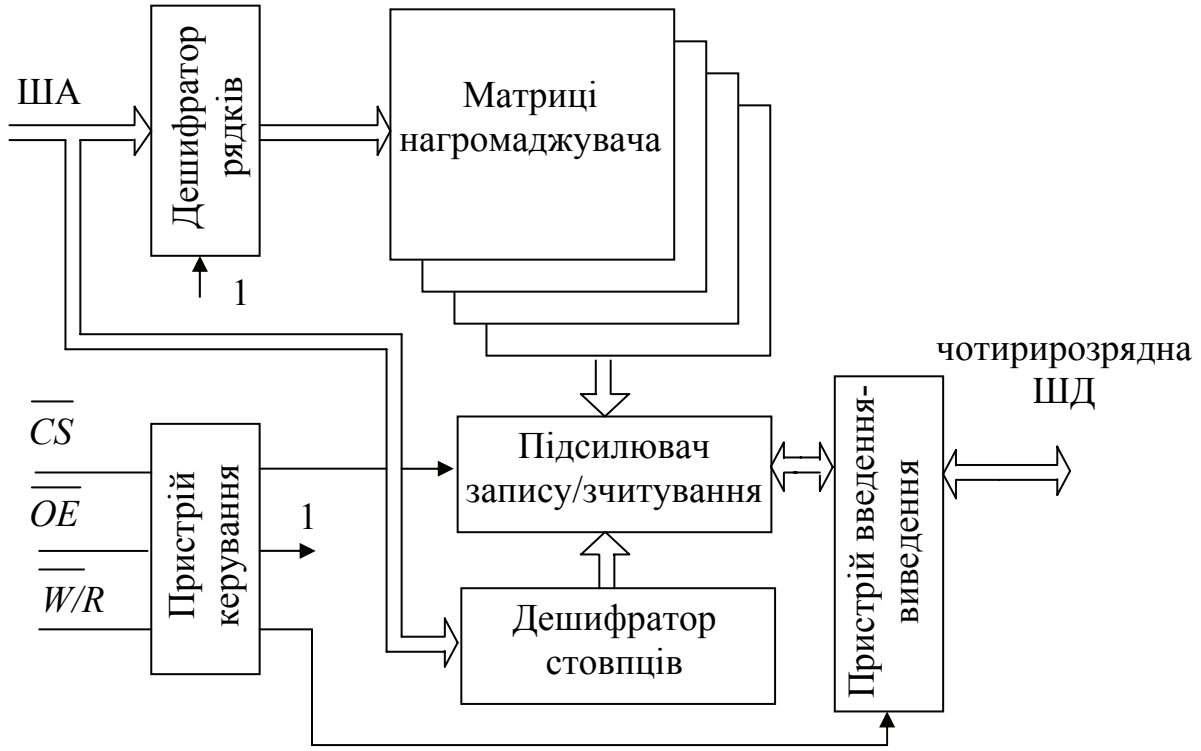



Рисунок 5.3 – Спрощена структурна схема статичного багаторозрядного ОЗП

На пристрій керування надходять такі сигнали:

– $\overline{CS}$ (*Chip Select*) – сигнал вибору мікросхеми. Сигнал логічного 0 на цьому виводі дозволяє роботу вибраної мікросхеми. Відсутність цього сигналу переводить мікросхему в неактивний стан;

– $\overline{OE}$ (*Output Enable*) – сигнал дозволу на вихід. Сигнал логічного 0 (активний рівень для цього входу) дозволяє роботу виходу. Сигнал логічної 1 визначає перехід мікросхеми у *z*-стан;

– $\overline{\text{W/R}}$ ($\overline{Write\ /\ Read}$) – запис / зчитування. Цей сигнал керує режимом роботи мікросхеми, забезпечуючи виконання необхідних функцій мікросхеми.

Для виконання операцій запису/зчитування необхідна одночасна наявність рівнів логічного 0 на виводах $\overline{CS}$ і $\overline{OE}$. Відсутність будь-якого з них переводить мікросхему у режим зберігання інформації.

Запис інформації, яка надходить з чотирирозрядної шини даних, виконується сигналом логічного 0 на вході $\overline{W}/R$ при активних рівнях сигналів $\overline{CS}$ і $\overline{OE}$. Запис проводиться у комірку пам’яті, адреса якої установлена на шині адреси.

Для зчитування вмісту комірки пам’яті необхідно подати активні рівні сигналів $\overline{CS}$ і $\overline{OE}$, на шині адреси установити адресу необхідної комірки, на вхід $\overline{W}/R$ подати сигнал з рівнем логічної 1. Зчитування відбувається на чотирирозрядну шину даних.

Будь-які інші комбінації сигналів на входах керування переводять ВІС у режим зберігання інформації.

Сучасні мікросхеми *SRAM* мають інформаційну ємність до 36 Мбіт, а час вибірки менш ніж 5 нс.

**ОЗП динамічного типу (*Dynamic Random Access Memory – DRAM*)** також призначений для оперативного запису, зберігання і зчитування інформації під час виконання МПС будь-яких програм. Модулі ОЗП сучасних МПС, як правило, будуються на базі мікросхем такого типу.

В якості ЕП ВІС *DRAM* використовується ємність *p-n*-переходу МДН-транзистора, стан заряду якої відповідає інформації, що зберігається у цій комірці. Вважають, що заряджений конденсатор зберігає інформацію логічної 1, а розряджений – логічного 0. Для тривалого зберігання інформації виконується порядкова регенерація (*refresh*) всього вмісту *DRAM* з інтервалом 2 або 4 мс. Поновлення інформації відбувається також під час запису і зчитування інформації, а також під час спеціального циклу регенерації. Порівняно з ВІС *SRAM* мікросхеми ОЗП динамічного типу мають більшу інформаційну ємність. Останнім часом випускаються ВІС з організацією пам'яті 1М × 1, 4М × 1, 16М × 1, 64М × 1. До недоліків мікросхем *DRAM* можливо віднести лише меншу швидкодію.

Для забезпечення збільшення інформаційної ємності, мікросхеми *DRAM* повинні мати адресну шину з більшою кількістю розрядів, що викликає певні труднощі, тому всі ВІС цього типу мають мультиплексовану адресну шину. Звернення до ЕП відбувається за два етапи формування її адреси – окремо для рядка і окремо для стовпчика, що забезпечується наявністю двох спеціальних входів: $\overline{CAS}$ (*Column Address Strobe*) – строб адреси стовпця і $\overline{RAS}$ (*Row Address Strobe*) – строб адреси рядка.

Для запису або зчитування інформації з такої ВІС на адресну шину установлюють код адреси рядків (молодшу частину адреси) і на вхід *RAS* подають активний рівень сигналу (звичайно – логічний 0), який фіксує цю адресу у внутрішній регістр адреси рядків. Після чого, формується потрібний сигнал запису або зчитування. На адресній шині установлюється код адреси стовпчика (старша частина адреси) і на вхід $\overline{CAS}$ подається активний рівень сигналу. Негативний перепад проводить запис інформації у певну комірку. Зчитування інформації здійснюється негативним рівнем сигналу $\overline{CAS}$ після формування адреси стовпчика. Запис інформації в ЕП проводиться з вхідної лінії *DI* (*Date Input*). Часові діаграми процесу запису інформації показано на рис. 5.3.

Після закінчення процесу запису стан внутрішніх кіл ВІС необхідно відновити, подавши на вхід $\overline{RAS}$ високий рівень сигналу. Тривалість дії цього сигналу дорівнює інтервалу між сусідніми сигналами $\overline{RAS}$.

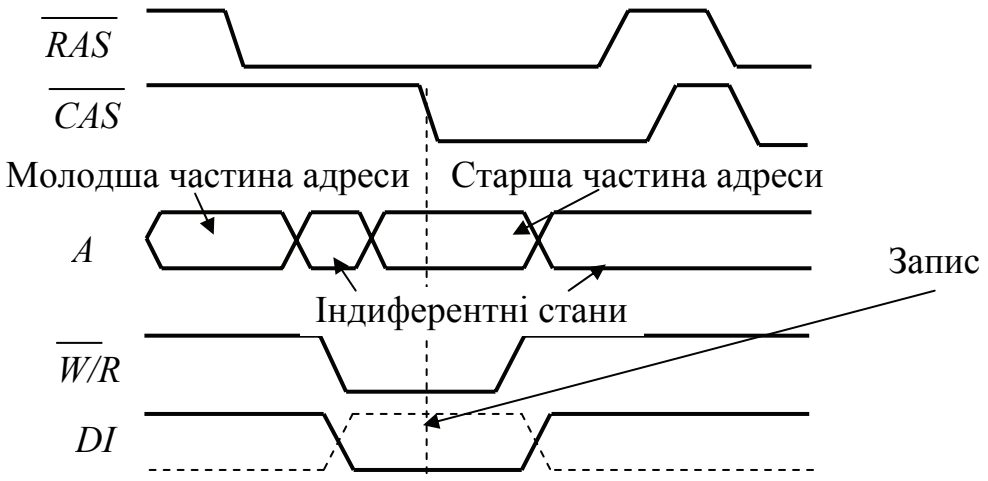



Рисунок 5.3 – Часові діаграми процесу запису інформації

Зчитування інформації проводиться на вихідну лінію *DO* (*Date Output*). Затримку зчитування вихідного сигналу можна відраховувати від негативного перепаду сигналу $\overline{CAS}$. Процес зчитування показано на рис. 5.4.

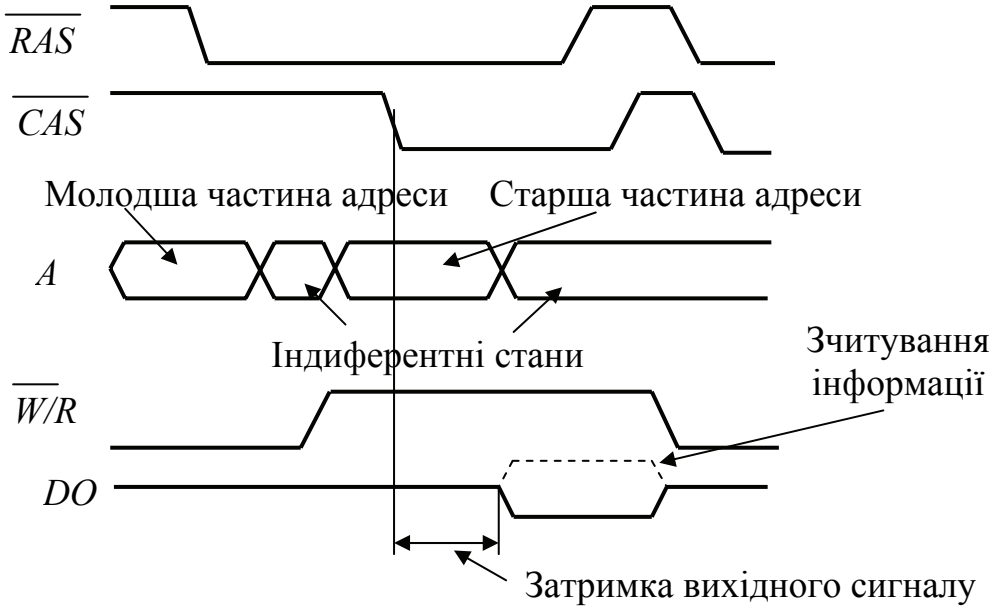



Рисунок 5.4 – Часові діаграми процесу зчитування інформації

Процес регенерації автоматично виконується для всіх ЕП рядка до якого відбувається звернення для запису або зчитування.

Цикл регенерації складається з послідовного перебору адрес усіх рядків і звернення до них. Формування адрес відбувається за допомогою зовнішнього лічильника циклів звернень. Звернення до матриці можливо організувати у кожному з можливих режимів функціонування: запису, зчитування, зчитування/модифікації/запису, а також у спеціальному режимі регенерації – сигналом $\overline{RAS}$ (за наявності сигналу $\overline{CAS}$, що має неактивний рівень). Такий

вид регенерації називається прихованою регенерацією (*hidden refresh*), «прозорою» (*transparent refresh*) або захопленням циклу (*cycle stealing*).

ВІС типу *DRAM* можуть працювати з пам'яттю, що має сторінкову організацію. Сторінкові режими звернення до ВІС *DRAM* зреалізовано з вибіркою вмісту ЕП усього рядка при зміні адреси стовпчиків. У такому режимі зменшується час циклу звернення, тому що звернення до наступного байта відбувається без станів очікування і зміни лише частини адреси.

Для використання в складі комп'ютерів на базі процесорів 80386, *i*486, а також перших моделей *Pentium* використовувалися модулі пам'яті *SIMM* (*Single Inline Memory Modules* – модулі пам'яті з однорядковим розташуванням виводів). Оперативна пам'ять стандарту *SIMM* випускалася у двох модифікаціях: *FPM* (*Fast Page Mode*) з напругою живлення 5 В для комп'ютерів стандарту *IBM PC* 486 і більш сучасний варіант *EDO* (*Extended Data Output* – розширений вивід даних) з живленням 3,3 В. Перші модулі мали 30 виводів і організацію 1М × 8, 1М × 9, 4М × 8 і 4М × 9. Дев'ятий біт – біт контролю парності. Більш сучасні модулі *SIMM* мають 72 виводи й організацію – 1М × 32, 1М × 36 (з контролем на парність). Також є модулі з організацією 2М × 32, 4М × 32, 8М × 32, 16М × 32 тощо.

Зараз стандартними для більшості систем є модулі модифікації *DIMM DDR* (*Dual Inline Memory Modules Double Data Rate* – модулі пам'яті з дворядним розташуванням виводів і подвійним стандартом даних), які мають ємність від 64 М до 1 Гбайта. Існує декілька варіантів реалізації оперативної пам'яті стандарту *DIMM DDR*, що відрізняються пропускною здатністю, яка визначається кількістю біт за секунду, що приймаються і передаються оперативною пам'яттю в процесі її функціонування. Сьогодні випускаються модулі пам'яті *DIMM DDR* стандартів *PC*1600, *PC*2100, *PC*2700 і *PC*3200 (пропускна здатність 1600, 2100, 2700 і 3200 Мбайт/с відповідно).

Ще більш сучасними є модулі пам'яті *RIMM* корпорації *RAMBUS*, які мають більшу пропускну здатність ніж модулі *DIMM*. Пропускна здатність модуля *RIMM* на частоті 400/800 МГц становить 1,6/3,2 Гбайта/с.

**Контрольні питання:**

1 В чому полягає різниця між статичними і динамічними ОЗП?

2 В чому полягає різниця між однорозрядними і багаторозрядними ОЗП?

3 Які сигнали керування забезпечують запис інформації в статичний ОЗП?

4 Яким чином досягнуто збільшення інформаційної ємності динамічних ОЗП?

5 Яке призначення сигналів $\overline{CAS}$ і $\overline{RAS}$ динамічного ОЗП?

**Контрольні питання підвищеної складності:**

1 Які модифікації модулів *SIMM* Ви знаєте?

2 В чому полягає сутність процесу регенерування динамічної пам'яті?

## 5.4 Побудова блока запам'ятовувального пристрою МПС з заданою організацією

Задача побудови блока ЗП у залежності від завдання і початкових умов може вирішуватися у різний спосіб. Умови задачі також можуть мати свої відмінності в залежності від конкретних умов функціонування цього блока.

Технічне завдання на розробку модуля ЗП повинно містити:

– технічні характеристики МПС, для якої буде розроблятися блок: потрібна інформаційна ємність, розрядність та типи сигналів шини адреси та шини даних, розподіл адресного простору, наявність сигналів керування та їхні рівні, довжина ліній проходження сигналів, інформаційна організація блока пам'яті, наявність блоків живлення – рівні напруг та величини електричних струмів, які вони забезпечують, тощо;

– часові характеристики сигналів керування: тривалість імпульсів керування, часові затримки між ними, співвідношення між перепадами цих сигналів;

– часові характеристики сигналів, які формуються блоком, що розробляється, їх співвідношення з внутрішніми і зовнішніми сигналами в МПС ;

– електричні характеристики сигналів на виході блоку ЗП: рівні напруги, навантажувальна здатність виходів блока.

Головна задача, що при цьому вирішується – забезпечення необхідної інформаційної ємності і забезпечення розрядності сигналів даних. Розв'язання цієї задачі, в залежності від наявності мікросхем пам'яті може бути двох видів:

– у номенклатурі мікросхем пам'яті існує ВІС, яка відповідає завданню на інформаційну організацію блока і забезпечує відповідні часові характеристики. В цьому випадку мікросхема установлюється у МПС і виконується узгодження рівней сигналів керування або їх формування в разі необхідності. На цьому побудова блока пам'яті вважається закінченою;

– у номенклатурі мікросхем пам'яті не існує ВІС, яка відповідає завданню на організацію блока. В цьому випадку необхідно з наявних типів ВІС побудувати схему, що буде відповідати завданню. Ця задача має два варіанти: по-перше – у наявності є мікросхеми, що мають необхідну інформаційну ємність, але мають меншу розрядність даних; по-друге – у наявності можуть бути ВІС, які мають меншу інформаційну ємність, ніж задано, але забезпечують розрядність шини даних.

Побудова блоків пам'яті ПЗП і ОЗП відбувається аналогічно. Різниця між ними полягає лише у необхідності забезпечення і формування специфічних для кожного з них сигналів керування.

Розглянемо приклад побудови ПЗП для використання у МПС, яка має 24-розрядну шину адреси, 8-розрядну шину даних і на шині керування якої формуються сигнали:

– $\overline{OE}$ – з активним рівнем логічного 0, окремо для блока ПЗП;

– $\overline{W}/R$ – сигнал керування процесом запису-зчитування, активний рівень логічного 0 має сигнал *W*.

Інформаційна організація блока, що розробляється, становить 115*K* × 8, у блоці пам'яті необхідно використовувати мікросхему РПЗП-УФ типу *AM*27*C*512, що має організацію 64*K* × 8. Початкова адреса комірки пам'яті для блока – $000000_B$.

По-перше, визначимо кількість мікросхем для забезпечення необхідної організації

$$N = \frac{115K \times 8}{64K \times 8} = 1{,}7968 \approx 2 .$$

Якщо в результаті отримано дробове число, то його необхідно **обов'язково** заокруглити до більшого цілого числа. По-друге, визначимо останню адресу комірки пам'яті блока. Дві ВІС блока повинні працювати по-черзі, обробляючи кожен по 64*K* інформації. У сумі обидві ВІС можуть обробити 128*K* інформації, що буде відповідати $2^{17} = 131072_D$ адресам. Якщо подати це число у шістнадцятьковій системі числення, то отримаємо число $1FFFF_H$, яке й буде останньою адресою блока. Це число більше ніж задане – 115*K*, це означає, що останні комірки пам'яті блока ніколи використовуватись не будуть.

ВІС, яку необхідно використовувати, має такі виводи: шістнадцять адресних входів – $A_0...A_{15}$; вісім виходів даних – $DO_0...DO_7$; входи керування – $\overline{OE}$, $\overline{CS}$ і $\overline{CE}$. Таким чином, необхідно з'єднати 2 ВІС з шинами МПС і між собою так, щоб забезпечити чергування їхньої роботи в залежності від установлення адреси.

Схема блока наведена на рис. 5.5.

Таблиця істинності мікросхеми *AM*27*C*512 наведена у табл. 5.1.

Таблиця 5.1 – Таблиця істинності мікросхеми *AM*27*C*512

| Назва сигналу, значення сигналу | | | | | Режим роботи |
|---|---|---|---|---|---|
| $\overline{CE}$ | $\overline{OE}$ | $\overline{CS}$ | $A_0...A_{15}$ | *DO* | |
| 1 | *X* | *X* | *X* | *z* | Неактивна |
| 0 | 1 | *X* | *X* | *z* | Неактивна |
| 0 | 0 | 1 | *X* | *z* | Неактивна |
| 0 | 1 | 0 | *X* | *z* | Зберігання |
| 0 | 0 | 0 | *A* | *D* | Читання |

Робота кожної з мікросхем відбувається відповідно до цієї таблиці за сигналами керування, що надходять від МПС. Робота блока відбувається в діапазоні адрес, який був визначений раніше. Якщо МПС формує адресу більшу ніж $01FFFF_H$, то на виході 7-вхідного елемента АБО (елемент *DD*1) формується сигнал логічної 1, який з'явиться на виході 2-вхідного елемента АБО (елемент *DD*2) і переведе обидві мікросхеми ВІС до неактивного стану.

Вибір однієї з двох мікросхем ПЗП відбувається сигналом $A_{16}$ (лінія 17 адресної шини). Якщо на цій лінії є сигнал логічного 0 адреси в діапазоні

$000000_H$ – $00FFFF_H$), то дозволяється робота мікросхеми *DD*4 на вхід $\overline{CE}$ якої надходить сигнал логічного 0. Якщо діапазон адрес, який виставлено на шину адреси становить $010000_H$ – $01FFFF_H$, то дозволяється зчитування з мікросхеми *DD*5 на вхід $\overline{CE}$ якої надходить сигнал логічного 0 з виходу інвертора (елемент *DD*3). Зчитування інформації відбувається сигналом $\overline{W}/R$, що надходить на входи $\overline{OE}$ обох мікросхем.

Розглянемо приклад побудування ПЗП для використання в МПС, яка має 24-розрядну шину адреси, 8-розрядну шину даних і на шині керування якої формуються сигнали:

– $\overline{OE}$ – з активним рівнем логічного 0, окремо для блока ПЗП;

– $\overline{W}/R$ – сигнал керування процесом запису-зчитування, активний рівень логічного 0 має сигнал *W*.

Інформаційна організація блока, що розроблюється становить 64*К* × 16, але розрядність шини даних становить 16 розрядів (слово). У блоці пам'яті необхідно використовувати мікросхему ОПЗП статичного типу *AM*21*C*512, що має організацію 64*К* × 8. Початкова адреса комірки пам'яті для блока – $000000_B$.

Визначимо кількість мікросхем для забезпечення необхідної організації

$$N = \frac{115K \times 16}{64K \times 8} = 1{,}7968 \approx 2\,.$$

Якщо у результаті отримано дробове число, то його необхідно заокруглювати **обов'язково** до більшого цілого числа.

Визначимо останню адресу комірки пам'яті блока. Тому що інформаційна ємність блока складає 64*К*, то остання адреса буде $00FFFF_{H.}$

ВІС, яку необхідно використовувати має такі виводи: шістнадцять адресних входів – $A_0...A_{15}$; вісім входів/виходів даних – $DIO_0...DIO_7$; входи керування – $\overline{OE}$, $\overline{CS}$, $\overline{CE}$ і $\overline{W}/R$.

Для збільшення кількості розрядів шини даних слід з'єднати мікросхеми таким чином, щоб забезпечити їх одночасну роботі з однаковими адресами.

Схема блока наведена на рис. 5.6.

Робота мікросхем ОЗП відбувається згідно з таблицею істинності (табл. 5.2) відповідно до сигналів керування, що надходять від МПС. Обидві мікросхеми працюють одночасно (паралельно з сигналами адрес і сигналами керування).

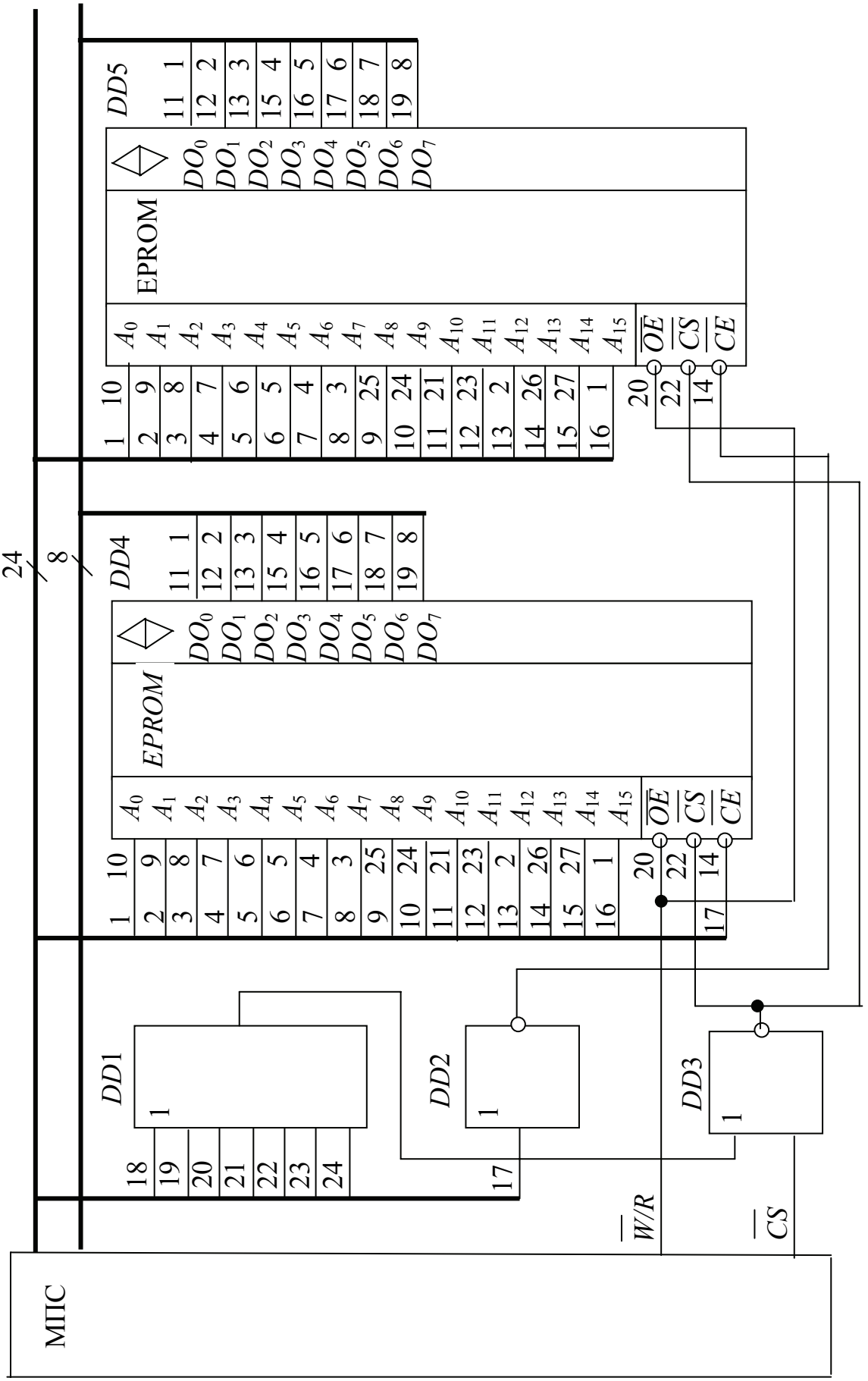



Рисунок 5.5 – Блок ПЗП з організацією $128K \times 8$

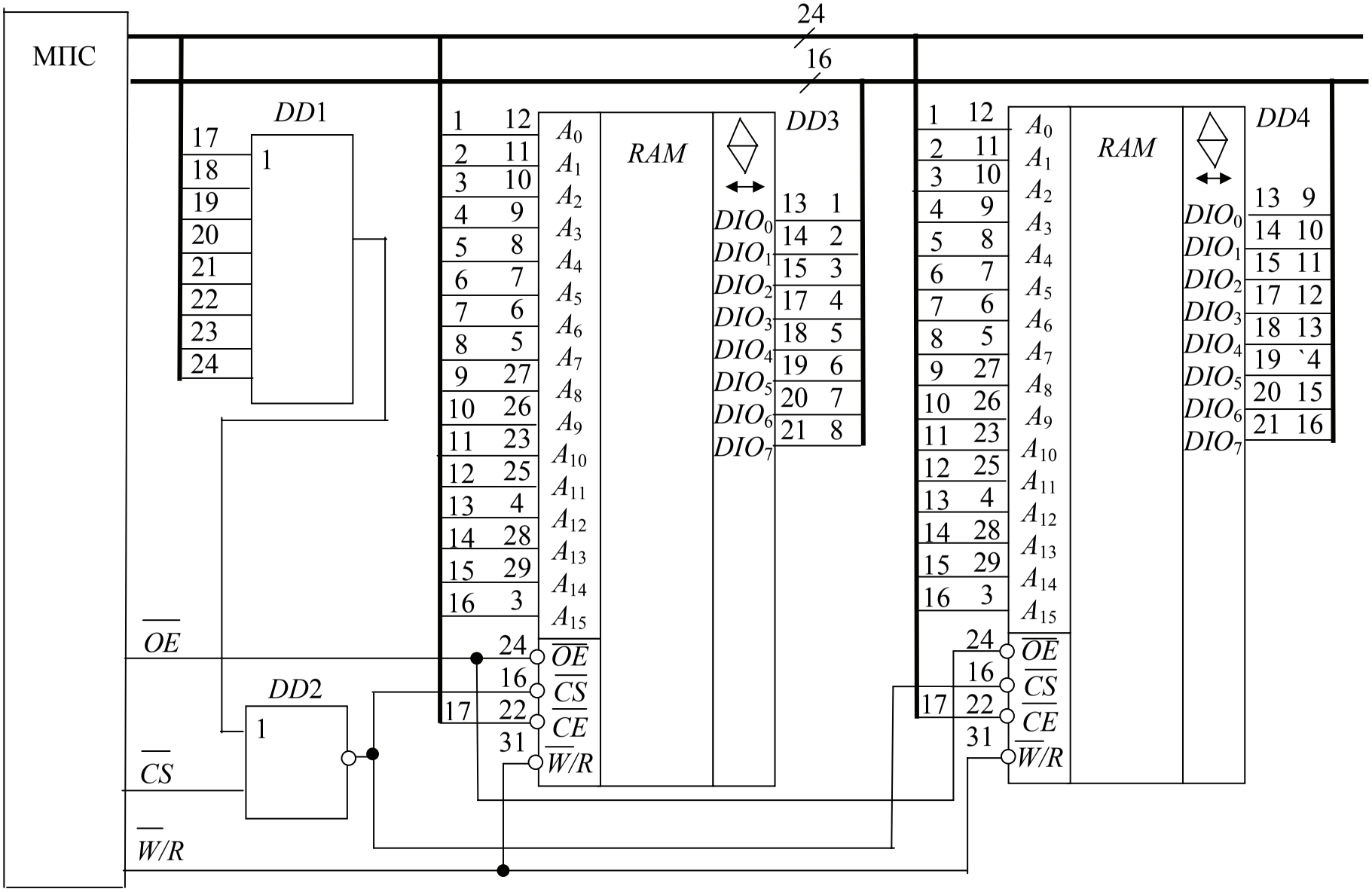



Рисунок 5.6 – Блок ОЗП з організацією 64*K* × 16

Таблиця 5.2 – Таблиця істинності мікросхеми ОЗП типу *AM*21*C*512

| Назва сигналу, значення сигналу | | | | | | Режим роботи |
|---|---|---|---|---|---|---|
| $\overline{CE}$ | $\overline{OE}$ | $\overline{CS}$ | $\overline{W}/R$ | $A_0...A_{15}$ | $DIO_0...DIO_8$ | |
| 1 | 1 | 1 | *X* | *X* | *z* | Неактивний стан |
| 1 | 1 | 0 | *X* | *X* | *z* | Неактивний стан |
| 0 | 1 | 0 | *X* | *X* | *z* | Зберігання |
| 0 | 0 | 0 | 1 | *A* | *DI* | Запис вхідних даних |
| 0 | 0 | 0 | 0 | *A* | *DO* | Зчитування даних |

**Контрольні питання:**

1 Які головні задачі вирішуються при побудуванні пристроїв ЗП?

2 Як визначається кількість мікросхем пам’яті, які необхідно використовувати?

3 Як відбувається нарощування інформаційної ємності ЗП?

4 Яким чином проводиться збільшення кількості розрядів виводів введення-виведення даних?

**Контрольні питання підвищеної складності:**

1 Які типи ЗП входять до складу підсистеми пам’яті МПС?

2 В чому полягає адресний принцип звернення до підсистеми пам’яті МПС?

3 Скільки комірок пам’яті і скільки бітів інформації зберігається в цій комірці, якщо інформаційна організація блока пам’яті становить $16K \times 8$?

4 Чим відрізняється стекова пам’ять, що організована програмно, від ОЗП МПС?

# 6 ІНТЕРФЕЙС

## 6.1 Організація інтерфейсів

**Вхідний контроль:**

1 Які вузли ПК слугують забезпеченню взаємодії між мікропроцесором, пам’яттю та периферійними пристроями?

2 Що таке АЦП?

3 Що таке ЦАП?

**Інтерфейсом** називається система шин, допоміжної апаратури і алгоритмів, реалізованих апаратно, призначених для організації обміну між мікропроцесором, пам’яттю та пристроями введення-виведення. До функцій інтерфейсу входить дешифрування адреси пристрою пам’яті або введення-виведення, а також зовнішнього пристрою, синхронізації обміну інформацією, узгодження форматів слів, електричне узгодження сигналів різних підсистем та вузлів МПС.

Складність задач, вирішуваних інтерфейсом, часто недостатня потужність буферних схем, які входять до складу самої ВІС МП, призводять до розподілу засобів інтерфейсу між різними пристроями:

1 Пристроями керування пам’яттю та введення-виведення.

2 Безпосередньо інтерфейсним пристроєм, який є проміжним ланцюгом між МП і пам’яттю та пристроями введення-виведення (системний інтерфейс).

3 Спеціалізованими пристроями керування (контролерами) введенням-виведенням, призначеними для реалізації алгоритмів керування, специфічних для різних пристроїв (клавіатури, магнітних дисків, моніторів тощо).

4 Контролерами для спряження за допомогою шин з периферійними пристроями (датчиками, виконавчими механізмами, комутаторами, АЦП, ЦАП).

Організація обміну між МП та пам’яттю або введенням-виведенням у простих випадках можлива на основі засобів, які є у самому МП. Ті функції, які не підтримуються апаратно мікропроцесором, реалізуються програмно.

Більш складні запам’ятовувальні пристрої та пристрої введення-виведення підключаються до МП через додаткові інтерфейсні пристрої, контролери, які виконуються у вигляді спеціальних ВІС, що входять до мікропроцесорного комплекту.

Керувальні сигнали МП повинні забезпечувати функціональну та часову взаємодію між МП, пам’яттю та введенням-виведенням у процесі обміну інформацією в основних режимах:

– для організації обміну з пам’яттю, крім адресних сигналів та сигналів даних повинні видаватись сигнали запису, читання, іноді сприйматися сигнал готовності ЗП;

– для програмного обміну даними, крім адресної інформації та самих даних, необхідні код вибирання пристрою введення-виведення, сигнали

керування записом та читанням, повинні сприйматися сигнали готовності введення-виведення тощо;

– для організації прямого доступу до пам’яті (ПДП) повинні бути підключені шина адреси для передавання адреси пам’яті у контролер ПДП та керувальні сигнали: код вибирання пристрою введення-виведення, запити від цього пристрою, дозвіл на обмін даними;

– у процесі роботи МП виробляє сигнали стану (зупину, чекання роботи у ПДП тощо) та синхронізуючі сигнали для інших пристроїв;

– для обміну даними у режимі переривань необхідно видавати з мікропроцесора керувальні сигнали дозволу переривань, підтвердження переривань та отримувати при готовності пристрою введення-виведення сигнали запиту на переривання.

Область використання зумовлює кількість та склад потрібних пристроїв введення-виведення і каналів зв’язку в МПС. Відповідно до цього у підсистемі введення-виведення можна виділити два рівні спряження підсистеми з процесором та пам’яттю. На першому рівні пристрої введення-виведення самі або через контролери поєднуються з процесором і пам’яттю за допомогою системного інтерфейсу МПС, який об’єднує окремі підсистеми в єдину систему. На другому рівні спряження контролери через канали зв’язку поєднуються з відповідними зовнішніми, або периферійними, пристроями (ПП) мікропроцесорної системи.

Незважаючи на широке застосування системних інтерфейсів, можна виділити два способи звернення до пристроїв введення-виведення:

– з застосуванням спеціальних команд введення-виведення;

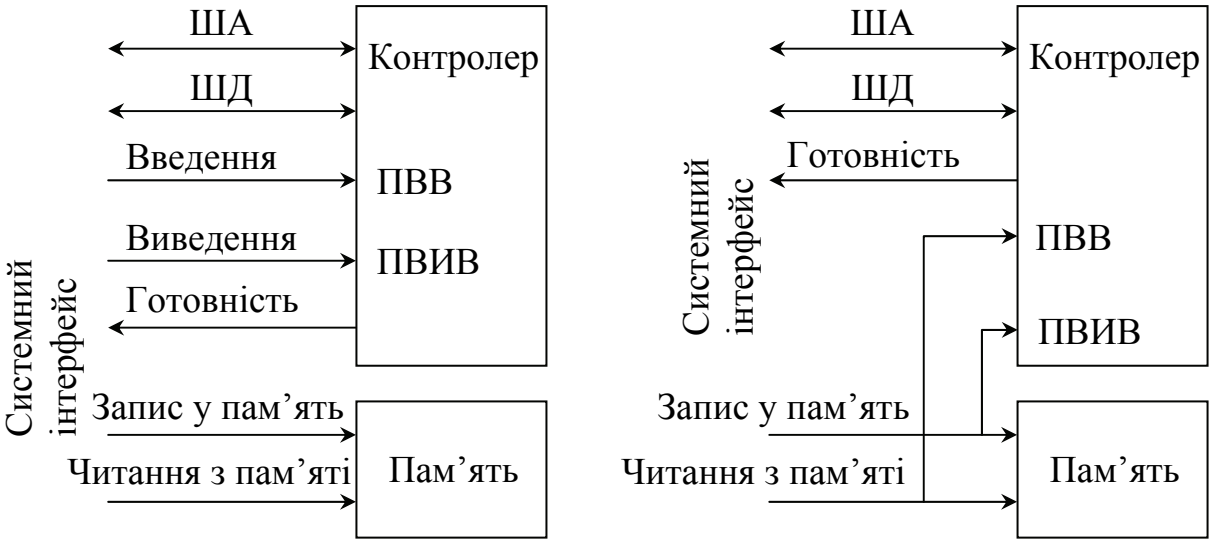
– за аналогією зі зверненням до комірок пам’яті за адресою.

У першому випадку адреса пристрою передається по тій самій шині адреси, що й адреси комірок пам’яті, і трактується як його номер тільки при формуванні спеціальних керувальних сигналів Введення або Виведення на пристрої введення-виведення, який ініціюється відповідними командами МП. Для синхронізації роботи процесора та контролерів, тобто з зазначенням моментів часу, які визначають готовність даних у пристрої введення для передачі їх у МП або підтверджуючих їх приймання при передаванні з МП у пристрій виведення, слугує сигнал Готовність, який надходить з пристроїв введення-виведення.

Для організації програмно-керованого обміну даними з пристроями введення-виведення (ПВВ-ПВИВ) на першому рівні (МП – контролер ПВВ-ПВИВ) достатньо простого набору сигналів (рис. 6.1).

На рис. 6.1, а показано використання керувальних сигналів при виконанні команд введення та виведення. Виконання операції введення починається з того, що МП виставляє на ША адресу ПВВ і сигналом Введення вказує на тип виконуваної операції. За сигналом Введення контролер ПВВ, який адресується, зчитує байт або слово даних з ПП, виставляє на лініях шини даних значення розрядів зчитаного слова та сигналом Готовність сповіщає про це МП. Прийнявши дане через контролер по ШД процесор знімає сигнали з ША та Введення. При виконанні операції виведення МП виставляє на ША адресу

пристрою виведення, на ШД значення розрядів байта або слова, що виводиться, та сигналом Готовність сповіщає процесор, що дані прийняті і можна зняти інформацію з ША та сигнал Виведення.




а) б)

Рисунок 6.1 – Порядок використання керувальних сигналів при введенні-виведенні

При реалізації обміну за аналогією зі зверненням до пам'яті (рис. 6.1, б) використовуються тільки керувальні сигнали Запис у пам'ять та Читання з пам'яті, які використовуються при обміні МП з пам'яттю. Для адрес ПВВ та ПВИВ відводиться визначена частина адресного простору, де можна адресувати необхідну кількість пристроїв.

Спрощена схема організації інтерфейсу з пристроями введення-виведення, яка описана вище, справедлива для багатьох типів МП, на яких побудовані мікропроцесорні системи.

Слід зазначити, що при обміні інформацією МП з периферійним пристроєм слід перевіряти апаратно або програмно готовність його до обміну. Якщо у МПС є кілька ПП, то будують блок формування сигналу Готовність з мультиплексуванням цього сигналу від кількох ПП.

**Контрольні питання:**

1 Яку функцію виконують контролери переривань та ПДП у МПС?

2 Які дані треба повідомити контролеру ПДП при його програмуванні?

3 Яку функцію виконують керувальні сигнали МП при взаємодії з іншими підсистемами МПС?

**Контрольні питання підвищеної складності:**

1 Керувальний пристрій знаходиться на відстані 15 м від МПС; якого типу інтерфейс Ви рекомендуєте використати для його підключення?

2 Датчик параметрів стану процесу знаходиться на відстані 30 см від МПС; якого типу інтерфейс Ви рекомендуєте використати для його підключення?

### 6.2 Асинхронний послідовний адаптер *RS*-232-*C*

**Вхідний контроль:**

1 Які цифрові автомати називаються синхронними, а які асинхронними?

2 Які пристрої можуть використовуватись як джерела синхроімпульсів для цифрових автоматів?

При підключенні ПК до мережі, МПС до периферійних пристроїв використовується послідовне передавання даних, яке не створює жорстких обмежень на довжину лінії. В той самий час шина даних ПК та МПС є паралельною. Для перетворення паралельно поданих даних у послідовності в ВІС послідовних інтерфейсів використовуються регістри зсуву. При передаванні даних вони паралельно записуються у регістр зсуву і з кожним тактовим імпульсом зсуваються праворуч, виходячи із регістра в лінію молодшими розрядами вперед. При прийманні дані з лінії вводяться в регістр зсуву праворуч також молодшими бітами вперед і, після заповнення регістра, дані з нього у паралельному поданні передаються в МПС.

Порт послідовного передавання *RS*-232-*C*, який називається також **асинхронним адаптером**, або послідовним інтерфейсом, має багатоцільове призначення:

- підключення миші;
- підключення графобудувачів, сканерів, принтерів, диджитайзерів;
- реалізація зв’язку між двома комп’ютерами з використанням спеціального кабелю та оболонки *NORTON COMMANDER*;
- підключення модемів для передавання даних телефонними лініями;
- підключення до мережі персональних комп’ютерів.

У складі кожного комп’ютера є хоча б один послідовний порт для обміну даними.

Протокол обміну складається з опису передаваного сигналу та призначення його складових частин.

Послідовне передавання даних означає, що дані передаються з використанням однієї лінії. На рис. 6.2 наведено форму електричних сигналів та формат даних при передаванні *ASCII* коду літери *A* (41*H*).

Рівень логічного нуля становить +15 В, логічної одиниці –15 В. Початковий стан лінії – одиничний. Стартовий біт сигналізує про початок передавання даних; далі передаються біти даних у такому порядку: *D*0–*D*1–*D*2–*D*3–*D*4–*D*5… Якщо використовується біт парності *P*, то передається й він. Біт парності має таке значення, щоб у пакеті бітів загальна кількість одиниць була парною або непарною залежно від перевірки на парність або непарність. У кінці передаються один чи два стопових біти, які завершують передавання, після чого рівень лінії знову установлюється одиничним до появи наступного

стартового біта. Використання парності, стартових та стопових бітів визначає протокол обміну даними.

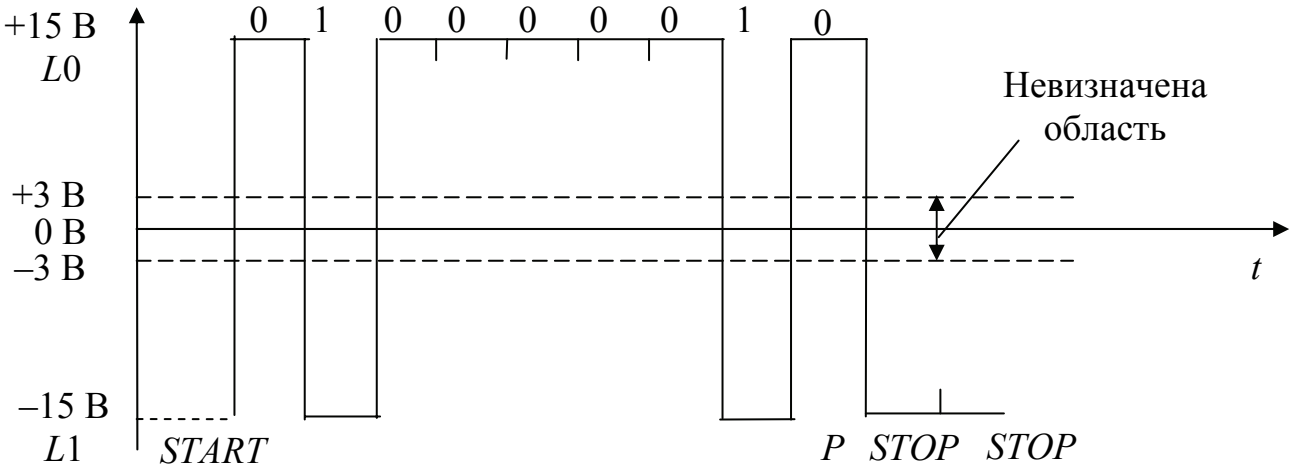



Рисунок 6.2 – Часова діаграма електричних сигналів та формату даних для даного 41*H*

Іншою важливою характеристикою є швидкість передавання даних. Вона повинна бути однаковою для передавача та приймача. Швидкість передавання даних вимірюється у Бодах (Бод – це кількість бітів, які передаються за секунду). При цьому враховуються і стартстопні біти і біт парності. Іноді використовується інший термін – біти за секунду (*bps*). Він означає ефективну швидкість передавання даних без урахування службових бітів.

Комп'ютер має від одного до восьми портів послідовного передавання даних, які зреалізовуються на мікросхемі *Intel* 8250. Це універсальний асинхронний приймач/передавач (*UART – Universal Asynchronous Receiver Transmitter*). Мікросхема вміщує кілька внутрішніх регістрів, доступних через команди введення-виведення. При передаванні байт записується у буферний регістр передавача, звідки потім переписується у зсувовий регістр передавача. Байт „висувається" з регістра по бітах, молодшими бітами вперед. Аналогічно, приймач теж має зсувовий та буферний регістри. Зовнішні пристрої підключаються до порту введення-виведення через з'єднувач типу *DB*25*P* або *DB*9*P*, які мають відповідно 25 та 9 виводів. У табл. 6.1 подається призначення контактів з'єднувача.

На етапі ініціалізації комп'ютера модуль *POST BIOS* тестує наявні асинхронні адаптери та ініціалізує перші два з них. Їхні базові адреси розташовано в області даних *BIOS*, починаючи з адреси 0000:0400*H*. Перший адаптер *COM*1 має базову адресу 3*F*8*H* і займає діапазон адрес з 3*F*8*H* до 3*FFH*. Другий адаптер *COM*2 має базову адресу 2*F*8*H* і займає адреси 2*F*8*H*…2*FFH*. Асинхронні адаптери можуть викликати переривання: *COM*1 – *IRQ*4 (*INT* 0*CH*), *COM*2 – *IRQ*3 (*INT* 0*BH*).

Таблиця 6.1 – Призначення контактів з’єднувача

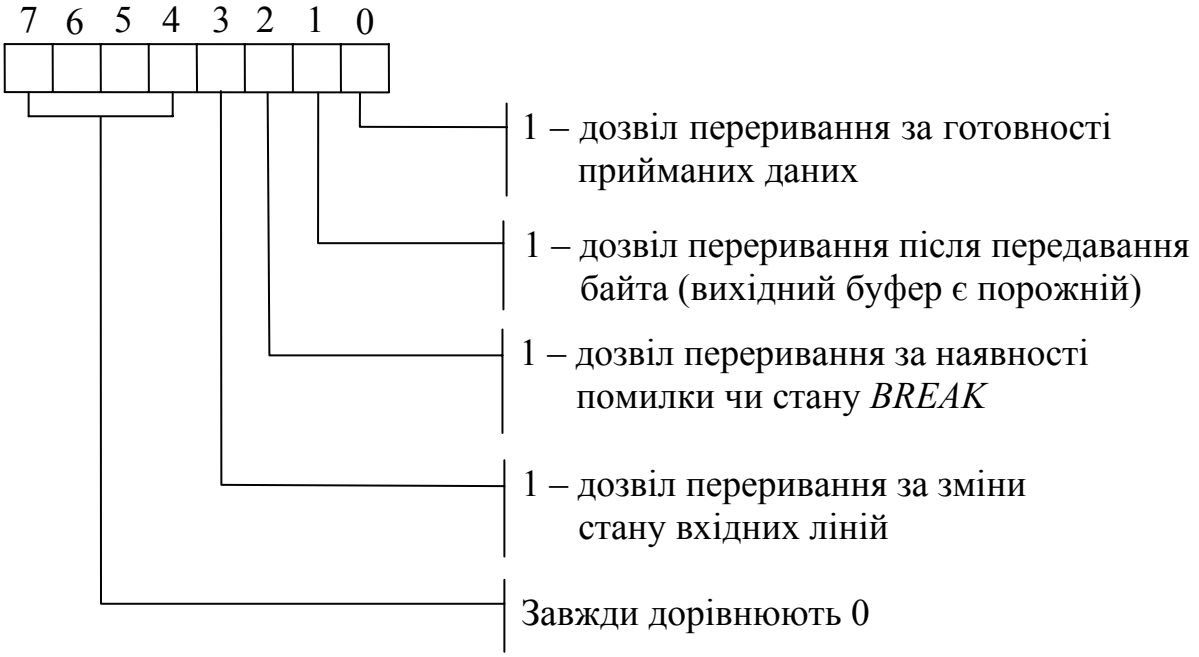
| Номер контакту | Призначення контакту | Вхід чи вихід |
|---|---|---|
| 1 | Детектор сигналу з лінії *DCD* | Вхід |
| 2 | Приймані дані | Вхід |
| 3 | Передавані дані | Вихід |
| 4 | Готовність вихідних даних *DTR* | Вихід |
| 5 | Земля за сигналом | Вихід |
| 6 | Готовність даних *DSR* | Вхід |
| 7 | Запит для передавання *RTS* | Вихід |
| 8 | Скидання для передавання *CTS* | Вхід |
| 9 | Індикатор виклику *RI* | Вхід |

Порт 3*F*8*H* відповідає регістру даних, які передаються чи приймаються. Для передавання дані треба записати в цей порт. Після прийняття даних від зовнішнього пристрою вони можуть зчитуватись з цього порту.

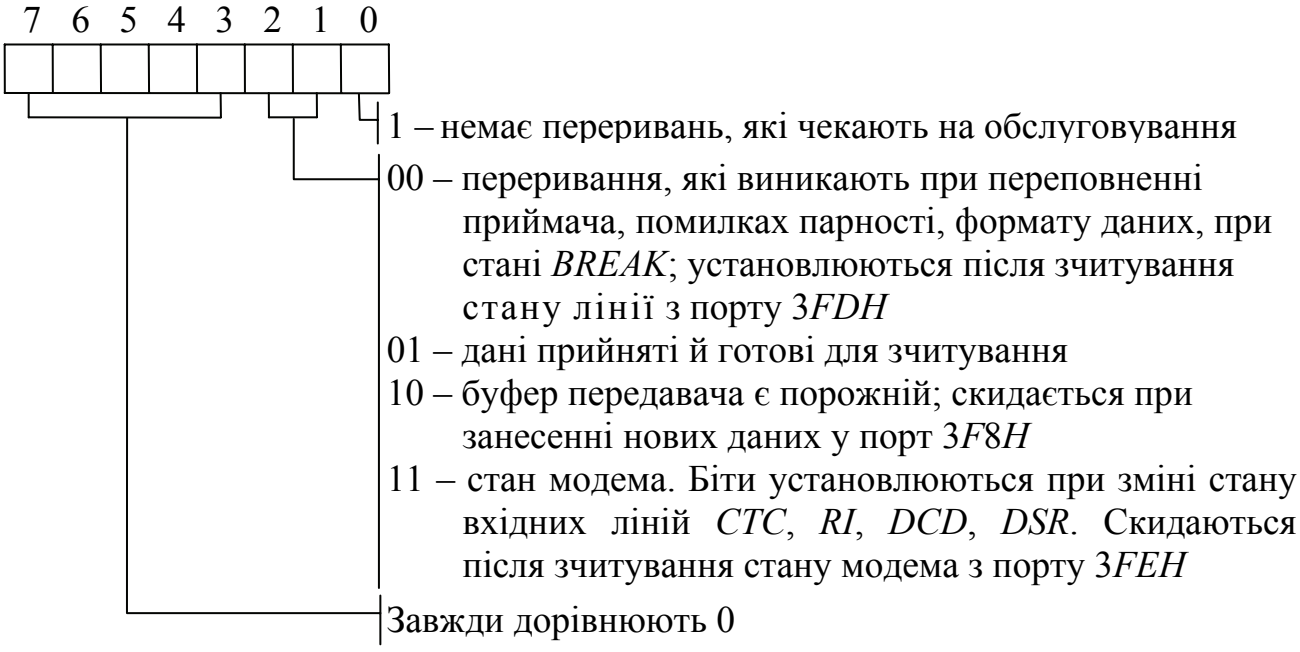
Залежно від стану старшого біта керувального слова, яке виводиться в керувальний регістр з адресою 3*FBH*, призначення порту 3*F*8*H* може змінюватись. Якщо цей біт дорівнює 0, то порт 3*F*8*H* використовується для запису передаваних даних. Якщо ж він дорівнює 1, то порт 3*F*8*H* використовується для виведення молодшого байта подільника частоти тактового генератора. Змінюючи значення подільника, можна змінювати швидкість передавання даних. Старший байт подільника записується у порт 3*F*9*H*. Залежність швидкості передавання даних від значення подільника частоти подано нижче:

| Подільник | Швидкість (Бод) |
|---|---|
| 1040 | 110 |
| 768 | 150 |
| 384 | 300 |
| 192 | 600 |
| 96 | 1200 |
| 48 | 2400 |
| 24 | 4800 |
| 12 | 9600 |
| 6 | 19200 |
| 3 | 38400 |
| 2 | 57600 |
| 1 | 115200 |

Порт 3*F*9*H* використовується як регістр керування перериваннями від асинхронного адаптера або (після виведення в порт 3*FBH* байта з установленим в 1 старшим бітом) для виведення значення старшого байта подільника частоти. В режимі керування перериваннями порт має такий формат:

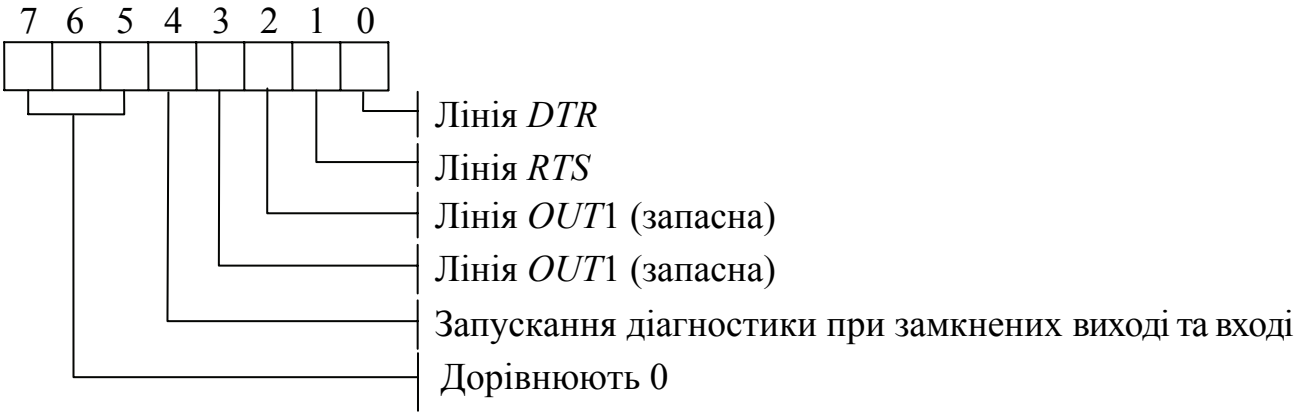



Порт 3*FAH* призначено для ідентифікації переривань. Його вміст визначає причину переривань. Регістр має такий формат:



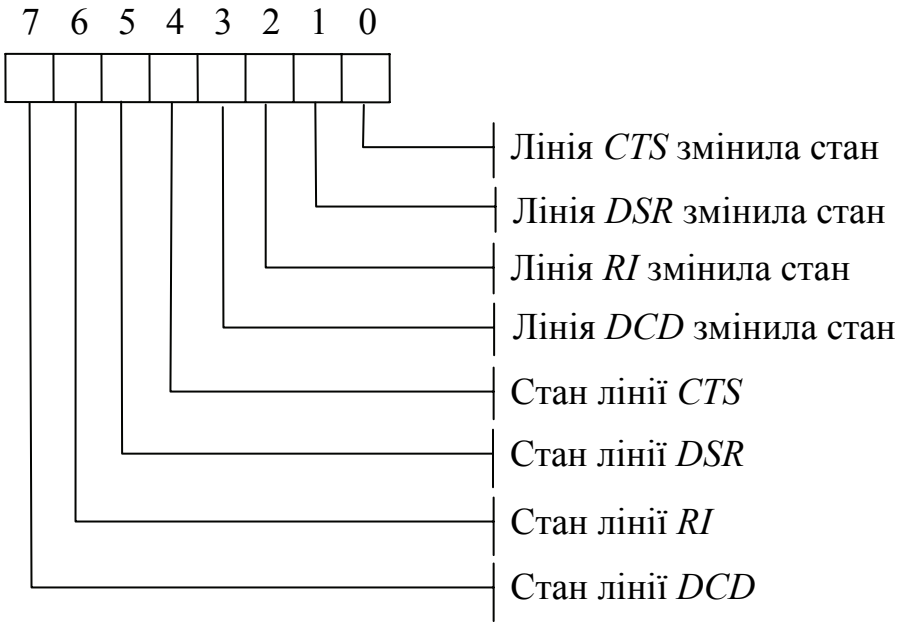
Керувальний регістр 3*FBH*, доступний для запису та зчитування:

Регістр 3*FC* керує модемом: станом вихідних ліній *DTR*, *RTS*, ліній, специфічних для модемів *OUT*1 та *OUT*2, для запускання діагностики при вході асинхронного адаптера, замкненого на його вихід. Формат порту:



Регістр стану лінії 3*FDH*:

Регістр стану модема 3*FEH*:



**Контрольні питання:**

1 Які функції виконує асинхронний послідовний адаптер?

2 Які пристрої, що входять до складу комп'ютера, підключаються за допомогою адаптера *RS*-232-*C*?

3 Який пріоритет мають порти комп'ютера *COM*1 та *COM*2?

4 Який режим роботи адаптера задається при завантаженні операційної системи?

5 Задайте керувальне слово, яке забезпечить передавання даного 82*H* з контролем на непарність та двома стоповими бітами. Наведіть часову діаграму електричних сигналів та формату даних для даного 82*H*.

6 Яку інформацію вміщує керувальне слово, яке задає режим роботи адаптера?

7 З якою метою зчитується слово стану лінії при прийманні та слово стану буфера при передаванні?

**Контрольні питання підвищеної складності:**

1 Назвіть послідовність дій при задаванні бажаної швидкості обміну між RS-232-C та зовнішнім пристроєм?

2 Як заборонити переривання під час обміну даними через *RS*-232-*C*?

3 Як перевіряється готовність периферійного пристрою до роботи при обміні даними через *RS*-232-*C*?

# 7 МІКРОПРОЦЕСОРИ

## 7.1 Архітектура мікропроцесорів

**Вхідний контроль:**

1 Які мікрооперації може виконувати універсальний регістр?

2 Наведіть приклади пристроїв пам’яті з послідовним доступом та з довільним доступом.

Під архітектурою мікропроцесорів розуміють структурну схему самого МП, програмну (регістрову) модель МП, організацію пам’яті, яку забезпечує МП у складі мікропроцесорної системи, спосіб організації введення-виведення та мову Асемблера, яка керує цим процесором.

З цієї точки зору існують два основні типи архітектури – **фоннеймановська** та **гарвардська**.

Фоннеймановська архітектура показана на рис. 7.1.

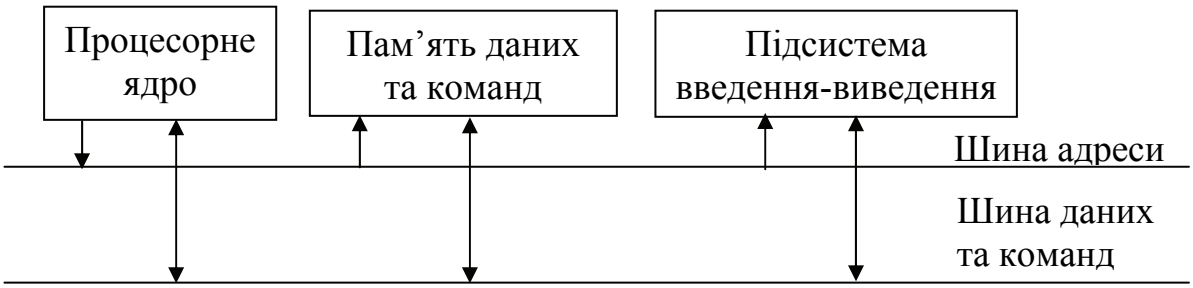



Рисунок 7.1 – Фоннеймановська архітектура

До загальних архітектурних властивостей та принципів побудови фоннеймановської архітектури можна віднести такі особливості:

1 Принцип програми, що зберігається, – це означає, що код програми та її дані знаходяться в єдиному адресному просторі в оперативній пам’яті, доступ до якої здійснюється по одній шині даних та команд.

2 Принцип мікропрограмування – машинна мова, коди, не керують апаратною частиною МП прямо. Кожна команда може бути виконана як результат дії набору сигналів, які треба згенерувати для її фізичної реалізації під керуванням блока мікропрограмного керування. Принцип мікропрограмного керування полягає в тому, що певна комбінація мікрокоманд (зсуву, пересилання інформації, логічних операцій) може створювати набір команд МП.

3 Лінійний простір пам’яті, яку адресує МП, – сукупність комірок пам’яті з послідовним адресуванням.

4 Послідовне виконання команд програми – на послідовних ділянках програми МП вибирає з пам’яті команди строго послідовно. Розгалуження програм виконується за використанням спеціальних команд умовного та безумовного переходів та при зверненні до підпрограм.

5 Дані та команди розміщуються в одному просторі пам’яті у вигляді послідовності нулів та одиниць; процесор не бачить принципової різниці між даними та командами і намагається трактувати вміст деяких послідовних комірок пам’яті як коди машинної команди, а якщо це не так, то програма завершується аварійно. Тому важливо у програмі чітко розподіляти простір даних та команд.

6 Мікропроцесору всеодно, яке логічне навантаження несуть дані, що він їх обробляє.

Класична фоннейманівська архітектура процесора з одним банком пам’яті не дозволяє виконувати багаторазовий доступ до запам’ятовувального пристрою під час виконання одної команди.

Для прискорення оброблення потоків цифрових сигналів у спеціалізованих процесорах цифрового оброблення сигналів використовують так звану гарвардську архітектуру, відповідно до якої процесорне ядро взаємодіє з двома банками пам’яті, як показано на рис. 7.2.

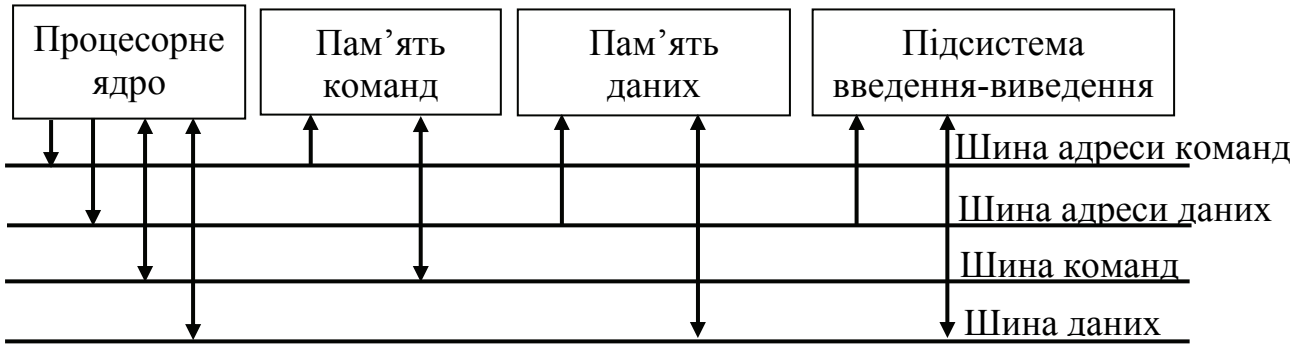



Рисунок 7.2 – Гарвардська архітектура

Звернення до кожного з банків пам’яті виконується за допомогою двох незалежних шин адреси та даних. У гарвардській архітектурі один з банків пам’яті використовується для зберігання програм, а другий для зберігання даних. Частіше використовується модифікована гарвардська архітектура. У цьому випадку один банк зберігає як програми, так і дані, а другий – тільки дані. Гарвардська архітектура дозволяє процесорному ядру за один цикл команди паралельно звертатись до пам’яті даних та пам’яті програм. Деякі мікропроцесори використовують три банки пам’яті з трьома незалежними парами шин адреси та даних (супергарвардська архітектура). Наявність трьох банків дозволяє за один командний цикл виконувати три паралельних звернення до пам’яті: вибирати команду та два операнди.

**Контрольні питання:**

1 Які особливості фоннейманівської архітектури мікропроцесорів гальмують підвищення їхньої потужності?

2 При програмуванні фоннейманівських процесорів хто повинен слідкувати за чітким розподілом адресного простору даних та команд?

3 Чи є пряме керування кодами програм апаратною частиною МП?

4 Чи можна передбачити напрямок розгалуження при виконанні команд умовного переходу?

5 Які особливості гарвардської архітектури дозволяють прискорити оброблення цифрових сигналів?

**Контрольні питання підвищеної складності:**

1 Що таке модифікована гарвардська архітектура?

2 Що таке супергарвардська архітектура?

## 7.2 МП фірми *Intel*

**Вхідний контроль:**

1 Які особливості має *n*-МДП-технологія виготовлення ВІС?

2 Чи може працювати МП поза мікропроцесорною системою?

### *7.2.1 Історична довідка про розвиток мікропроцесорів фірми Intel (Для самостійного вивчення)*

Напівпровідникова промисловість почала розвиватися в 50-ті роки XX століття у лабораторіях фірми *Bell Telephone Laboratories* (*Bell Labs*) у штаті Каліфорнія, США, в долині Санта Клара, яка сьогодні відома як “Селіконова долина”. Компанія *Intel* (*Integrated Electronics*) була заснована Робертом Нойсом та Гордоном Муром у 1968 році, а у 1972 році фірма *Intel* створила свій перший 8-розрядний мікропроцесор 8008, який мав фоннеймановську архітектуру і багато ознак усього сімейства мікропроцесорів: НОЗП, засоби організації стека, систему переривань і міг слугувати як центральний процесор мікроЕОМ, а також виконувати задачі керування в складі контролерів. Мікропроцесор 8008 був виготовлений за *n*-канальною МОП-технологією, виконував 30000 операцій за секунду та адресував 16 кбайт пам’яті, що було недостатньо для використання його щодо вирішення широкого кола задач. У 1974 році фірма випустила МП *І*8080, який мав розширену систему команд мови Асемблер і був сумісний з МП 8008. Вперше був виконаний принцип сумісності знизу вверх, тобто користувачі програмного забезпечення МП 8008 могли виконувати його на МП *І*8080. Швидкість обчислень МП І8080 становила 200000 операцій за секунду, він адресував 64 кбайт пам’яті.

Розробка МП *І*8080 поставила фірму *Intel* з річним обсягом понад 1 мільярд доларів у ряд найбільших компаній з виробництва надвеликих інтегральних мікросхем у США.

До 1978 року розвиток МП фірми *Intel* відбувався шляхом удосконалення та розширення функцій МП *І*8080.

Аналогом МП *І*8080 російського виробництва був МП К580ВМ8А, який випускався з метою побудови керувальних МПС, зокрема контролерів жорстких дисків, на ньому також виконувались простіші мікроЕОМ. Частота роботи МП становила 2 МГц.

Структурна схема 8-розрядного мікропроцесора КР 580ВМ80А показана на рис. 7.3.

Для зберігання операндів, які беруть участь в операціях АЛП, передбачено сім 8-розрядних регістрів. Регістр *A*, який називається **акумулятором**, призначений для обміну інформацією з зовнішніми пристроями; при виконанні арифметичних, логічних операцій та операцій зсувів він є джерелом операнда, в нього пересилається результат виконаної операції.

Шість інших 8-розрядних регістрів *B, C, D, E, H, L* утворюють блок регістрів загального призначення (РЗП). Ці регістри можуть використовуватися як одиночні або об'єднуватись у регістрові пари *BC, DE, HL*.

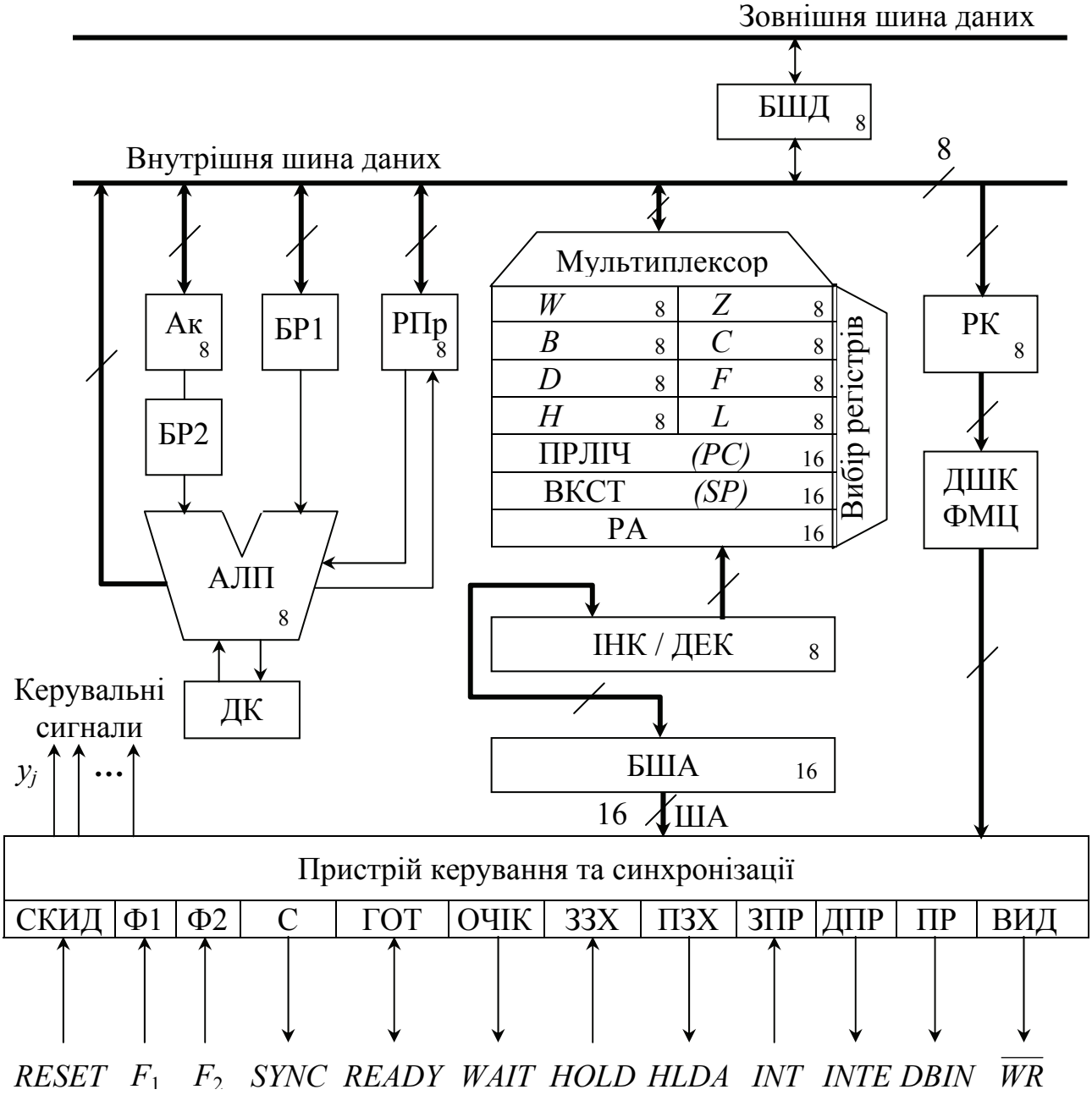



Рисунок 7.3 – Структурна схема 8-розрядного мікропроцесора КР 580ВМ80А

Регістри БР1, БР2, *W, Z* використовуються як буферні, програмно-недоступні регістри.

Вказівник стека – ВКСТ – (*SP*) – 16-розрядний регістр, слугує для адресування особливо організованої пам'яті, яка називається стеком.

Програмний лічильник – ПРЛІЧ – (*PC*) або лічильник команд призначений для зберігання адрес команд; після вибирання з оперативної пам'яті поточної команди вміст лічильника команд збільшується на кількість байтів цієї команди і, таким чином, за відсутності умовних та безумовних переходів у програмах формується адреса наступної команди. Блок регістрів вміщує також спеціальну схему інкрементування/декрементування (ІНК/ДЕК), яка забезпечує без АЛП у процесі передавання даних між регістрами модифікацію вмісту регістрів на 1.

При зверненні до пам'яті з метою вибирання даних, але не команд, як адресу можна використовувати вміст будь-якої регістрової пари з РЗП.

Арифметично-логічний пристрій – АЛП (*ALU*) виконує арифметичні операції складання з передаванням перенесення у молодший розряд та без урахування цього перенесення, логічні операції кон'юнкції, диз'юнкції, складання за модулем 2, порівняння, а також чотири види циклічних зсувів над 8-розрядними операндами.

АЛП виконано на основі комбінаційної схеми суматора з власними регістрами тимчасового зберігання даних БР1 та БР2.

При виконанні арифметичних та логічних операцій одним з операндів слугує вміст акумулятора і результат операції розміщується в акумуляторі. Циклічний зсув виконується тільки над вмістом акумулятора.

Схема десяткової корекції (ДК) дозволяє виконувати операції над даними, які подані у двійково-десятковій системі числення. При зберіганні операнда, трактованого як десяткове число, розряди кожного регістра, в якому воно вміщене, поділяються на дві групи по чотири розряди кожна і в кожній групі розрядів зберігається одна десяткова цифра, подана у коді *BCD* (8421). Таким чином, у регістрі можна зберігати 2-розрядне десяткове число. При виконанні арифметичних операцій над двійково-десятковими числами може знадобитися корекція результату (додавання до нього числа $0110_2$). Така корекція у кожній 4-розрядній групі результату виконується схемою ДК при виконанні окремої команди мови програмування.

Регістр прапорців РПр, або регістр ознак, складається з п'яти тригерів, які призначені для зберігання певних ознак, якими можна охарактеризувати результат виконання операції АЛП.

Ознака *CY* (ознака перенесення, від *Carry*) – перенесення із старшого розряду при виконанні арифметичних операцій та вміст висуваного з акумулятора розряду при виконанні операцій зсувів.

Ознака *Z* (ознака нуля, від *Zero*) – установлюється в стан 1, якщо результат операції дорівнює нулю.

Ознака *S* (ознака знаку, від *Sign*) установлюється в стан, який визначається значенням старшого розряду результату; $S = 1$ означає, що результат позитивний, $S = 0$ – негативний.

Ознака *P* (ознака парності або непарності кількості одиниць у результаті, від *Parity*) установлюється в 0, якщо кількість одиниць у результаті непарна, і в 1 – якщо парна; ознака *P* використовується у багатьох випадках, коли треба перевірити правильність передавання даних за парністю або непарністю кількості одиниць.

Ознака *AC* (ознака додаткового перенесення від *Auxiliary Carry*) установлюється в 0, якщо перенесення з молодшої тетради у старшу відсутні, і в 1, якщо має місце; ознака *AC* використовується при виконанні операцій над двійково-десятковими числами, за наявністю *AC* за необхідності підключається схема десяткової корекції.

Усі регістри та регістрові пари можуть бути скомутовані за допомогою мультиплексора (*MUX*) та схеми вибору регістрів як з внутрішньою шиною даних, так і з ША. Регістри акумулятора (А) та регістр прапорців (*F*) також можуть утворювати регістрову пару *AF*, яка називається **словом стану програми** (ССП або *PSW – PROGRAM STATUS WORD*). ССП зазвичай використовується для зберігання в стеку стану програми при зверненні до підпрограм обробки переривань.

При передаванні адреси вміст регістрових пар зберігається у 16-розрядному регістрі адреси (РА), з якого далі через буфер шини адреси (БША) надходить на 16-розрядну шину адреси (ША) і далі в оперативну пам’ять. Число кодових комбінацій 16-розрядної адреси дорівнює $2^{16}$, кожна з них може визначати адресу (номер) одної з комірок оперативної пам’яті. Таким чином, забезпечується можливість звернення до пам’яті, яка вміщує до 64 кбайт, тобто 65536 8-розрядних слів.

Буферні регістри даних (БР) та буфер шини адреси (БША) забезпечують зв’язок процесора з зовнішніми шинами даних та адрес. Особливістю буферів є те, що у кожному розряді вони мають логічні пристрої з трьома стабільними станами: крім станів *L*0 та *L*1 передбачено третій стан, в якому вони мають безкінечний вихідний опір і є відімкнені від відповідних шин.

Регістр команд (РК) приймає з шини даних код операції команди, який потім декодується у дешифраторі коду (ДШК). Формувач машинних циклів (ФМЦ) організовує взаємодію між усіма блоками і схемами МП та пристроями МПС.

Пристрій керування та синхронізації реалізований апаратно з використанням програмованої логічної матриці. Вихідні сигнали пристрою надходять, як до вузлів мікропроцесора, наприклад, вказуючи код операції АЛП, так і для виконання мікрооперацій в МПС.

З шини керування на пристрій керування та синхронізації надходять такі сигнали:

- *RESET* – скидання або початкового установлення;
- $F_1$ та $F_2$ – дві послідовності імпульсів синхронізації з неоднаковими фазами;
- *READY* – сигнал готовності зовнішнього пристрою до обміну; використовується для організації обміну даними з пристроями, які мають низьку швидкодію порівняно з МП;

− *HOLD* – сигнал запиту прямого доступу до пам’яті (ПДП) або запиту на захоплення шин; використовується для організації обміну даними з пристроями, швидкодія яких більша ніж у МП;

− *INT* – сигнал запиту передавання від зовнішнього пристрою, коли він є готовий до роботи.

На шину керування з пристрою керування та синхронізації надходять такі сигнали:

− *SYNC* – вихід сигналу синхронізації, високий рівень цього сигналу свідчить про те, що по шині даних передається байт стану, який використовується для формування керувальних сигналів для зовнішніх пристроїв;

− *WAIT* – сигнал підтвердження очікування; його високий рівень свідчить про те, що процесор знаходиться у режимі очікування і виконує такти очікування (холості такти);

− *HLDA* – сигнал підтвердження прямого доступу до пам’яті або підтвердження захоплення шин; високий рівень цього сигналу свідчить про те, що процесор перевів свої шини адреси, даних та керування у стан високого опору;

− *INTE* – сигнал дозволу на переривання;

− *DBIN* – сигнал читання; його високий рівень свідчить про те, що двоспрямована шина даних знаходиться у режимі прийому інформації;

− $\overline{WR}$ – сигнал запису; низький рівень цього сигналу свідчить про те, що двонаправлена шина даних знаходиться у режимі видачі інформації, високий – її прийому.

До групи синхронізуючих сигналів відносяться $F_1$, $F_2$**,** *SYNC***.** До групи сигналів, які інформують МП про стан зовнішніх пристроїв, можна віднести сигнали *RESET* та *READY*. Сигнали запитів на дозвіл роботи від зовнішніх пристроїв та відповідні сигнали від МП – це сигнали *HOLD, HLDA, INT, INTE*. Сигнали *WAIT, DBIN*, $\overline{WR}$ свідчать про стан процесора та шини даних.

**Програмна модель МП К580ВМ80А**

Програмно доступні вузли та елементи архітектури МПС показані на рис. 7.4.

У МП до програмно доступних вузлів відносяться 8-розрядні РЗП *B, C, D, E, H, L* та A (акумулятор), регістр ознак результату *F*, 16-розрядний лічильник команд *PC* (*PROGRAM COUNTER*), 16-розрядний вказівник стека *SP* (*STACK POINTER*), тригер дозволу переривань *I* (*INTERRUPT*).

Програмно доступною є пам’ять МПС *M* (*MEMORY*): ППЗП, ОЗП та організований на його базі стек.

МПС має ефективну підсистему введення-виведення *I–O* (*INPUT–OUTPUT*), яка може складатися з 256 пристроїв введення та 256 пристроїв виведення.

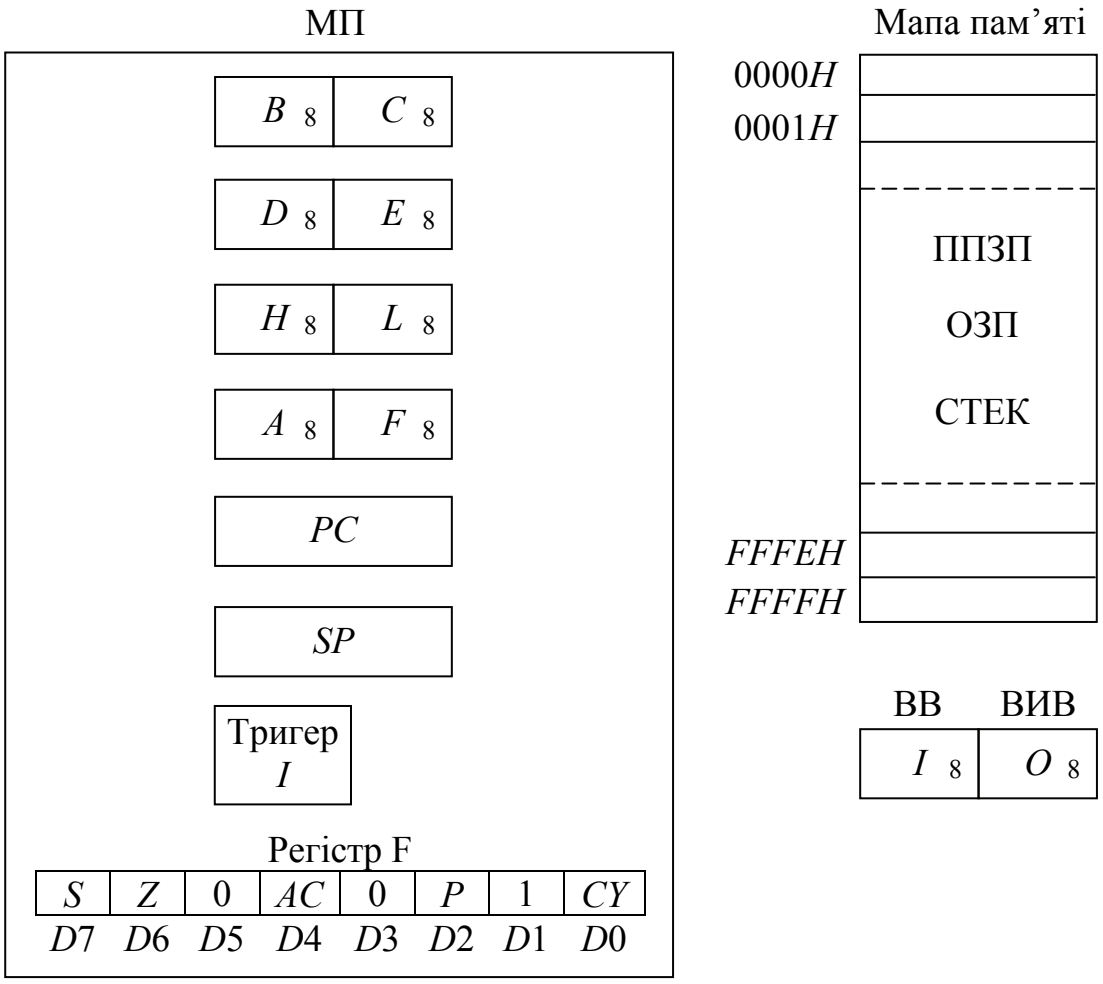



Рисунок 7.4 – Програмно доступні елементи архітектури МПС

Розглянута далі МПС на базі МПК КР580 також є цікавою тільки з точки зору принципів її побудови, взаємодії підсистем, які її складають, та прикладів можливого застосування у телекомунікаціях. Доцільно також на простих прикладах показати виконання окремих команд у МПС.

**Мікропроцесорна система КР580**

Наведена на рис. 7.5 структурна схема МПС складається з трьох підсистем – центрального процесорного елемента, підсистеми пам'яті та підсистеми введення-виведення і побудована на мікропроцесорному комплекті КР580.

Підсистема центрального процесорного елемента складається з МП типу КР580ВМ80А, генератора тактових імпульсів (ГТІ) типу КР580ГФ24 та системного контролера-формувача (СКФ) типу КР580ВК28. ГТІ виробляє кварцовану двофазову послідовність синхроімпульсів *F*1 та *F*2 та СТБС (строб синхронізації в інверсній формі), а також формує вхідні сигнали керування МП – скидання та готовності ГОТ. СКФ формує основні керувальні сигнали ШК.

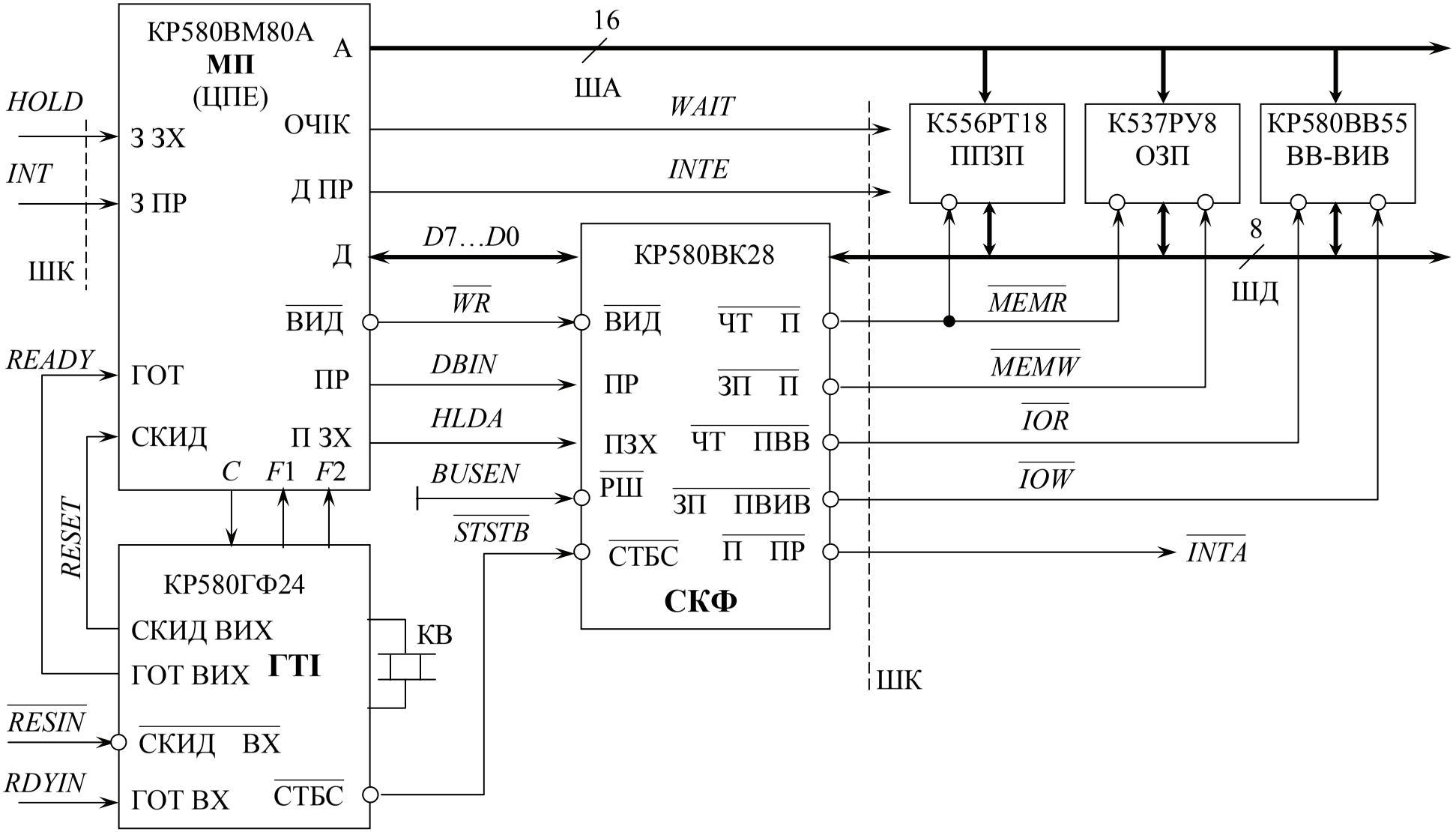



Рисунок 7.5 – Структурна схема МПС КР580

Сигнали видачі даних від МП на ШД для інших підсистем $\overline{\text{ВИД}}$($\overline{WR}$ – *WRITE*), прийому даних з ША у МП ПР (*DBIN – DATA BUS INPUT*) та підтвердження захоплення шин для пристроїв, які працюють у режимі ПДП, ПЗХ (*HLDA – HOLD ACKNOWLEDGE*) видаються безпосередньо з виводів МП на СКФ, а інші – короткочасно з ШД у вигляді 8-розрядного слова стану процесора ССПЦ (*PCSW – PROCESSOR STATUS WORD*), яке фіксується у СКФ. ССПЦ має такий порозрядний склад:

*D*7 – ЧТ П (*MEMR – MEMORY READ*), читання даних з пам'яті.

*D*6 – ЧТ ПВВ (*INP – INPUT CYCLE*), читання даних з пристроїв введення.

*D*5 – *M*1, робота МПС у циклі читання та виконання першого байта команди.

*D*4 – ЗП ПВИВ (*OUT – OUTPUT CYCLE*), запис/виведення даних через пристрої виведення.

*D*3 – П ЗУП (*HLTA – HALT ACKNOWLEDGE*), підтвердження виконання операції зупину МПС.

*D*2 – СТ (*ST – STACK*), використання стекової пам'яті.

*D*1 – ЗП-ВИВ ($\overline{WO}$ – *WRITE-OUTPUT*), запис даних у пам'ять або виведення через пристрій виведення.

*D*0 – П ПР (*INTA – INTERRUPT ACKNOWLEDGE*), підтвердження переривання МПС.

СКФ синхронізується сигналом $\overline{\text{СТБС}}$ (*STSTB – STATE STROBE*) – строб синхронізації. За необхідності СКФ можна відключити від ШК установленням його виходів у високоімпедансний стан; для цього на його вхід ДШ (*BUSEN* – дозвіл шин) треба подати сигнал, який дорівнює 1.

Таким чином на ШК з СКФ подаються такі сигнали:

– $\overline{\text{ЧТ П}}$($\overline{MEMR}$) – читання даних з пам'яті;

– $\overline{\text{ЗП П}}$ ($\overline{MEMW}$ – *MEMORY WRITE*) – запис даних у пам'ять;

– $\overline{\text{ЧТ ПВВ}}$ ($\overline{IOR}$ – *INPUT-OUTPUT READ*) – читання даних з пристроїв введення;

– ЗП ПВИВ ($\overline{IOW}$ – *INPUT-OUTPUT WRITE*) – запис даних у пристрій виведення;

– $\overline{\text{П ПР}}$ (*INTA*) – підтвердження переривання;

– $\overline{\text{ОЧІК}}$ (*WAIT*) – очікування, робота МПС призупиняється;

– $\overline{\text{Д ПР}}$ (*INTE – INTERRUPT ENABLE*) – дозвіл переривання.

МП може отримувати від різних зовнішніх пристроїв керувальні сигнали:

– З ЗХ (*HOLD*) – запит на захоплення шин пристроями, які працюють у режимі ПДП;

– З ПР (*INT – INTERRUPT*) – запит на переривання роботи МПС;

– $\overline{\text{СКИД ВХ}}$ ($\overline{RESIN}$ – *RESET IN*) – СКИД (*RESET*) – скидання МП, установлення в нуль програмного лічильника *PC*;

– ГОТ ВХ (*RDYIN – READY IN*) – ГОТ (*READY*) – сигнал готовності зовнішніх пристроїв до обміну даними.

СКФ організує буферування ШД та її двонаправленість. Для спрощення на схемі не показані також буфери ША та ШК.

ППЗП призначене для зберігання програм і підпрограм та ОЗП, призначене для тимчасового зберігання проміжних результатів та організації стека, складає підсистему пам’яті. Підсистема введення-виведення може складатися, наприклад, з паралельного або послідовного інтерфейсів, контролерів ПДП або переривань.

**Функціонування МПС**

Для виконання команди програми необхідна взаємодія якнайменш двох підсистем МПС: підсистеми центрального процесорного елемента та підсистеми пам’яті. Команда зчитується з ППЗП, декодується та виконується. На всі процедури, залежно від типу команди, витрачається від одного до п’яти машинних циклів М1...М5.

У кожному циклі МП один раз звертається до пам’яті або пристрою введення-виведення.

Кожний з машинних циклів складається з трьох-п’яти машинних тактів Т1...Т5, тривалість такту визначається періодом тактових імпульсів ГТІ (рис. 7.6).

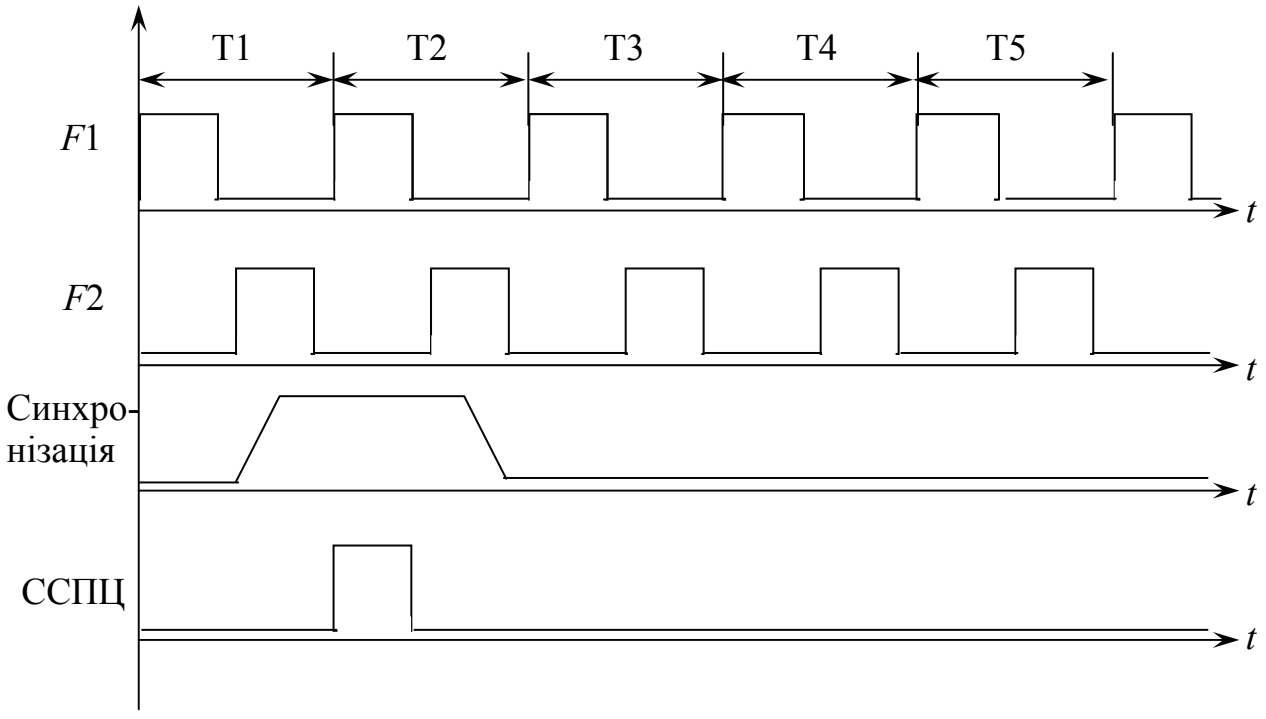



Рисунок 7.6 – Часова діаграма циклу з п'яти тактів

Такти відраховуються від фронтів імпульсів *F*1. Якщо МП К580ВМ80А працює на частоті 2 МГц, то тривалість одного такту складає 0,5 мкс. За цей час у системі виконується одна або водночас декілька елементарних дій – мікрооперацій. Набір керувальних сигналів, які керують самим МП, змінюються кожного такту, а в МПС сигнали керування оновлюються тільки на

початку кожного машинного циклу команди, тобто за кілька тактів – від 4 до 18 залежно від команди.

У МПС може вироблятися до 10 різних ССПЦ, тобто до 10 різних типів машинних циклів, які показані в табл. 7.1.

Таблиця 7.1 – Типи машинних циклів

| № пп. | Тип циклу | Код ССПЦ |
|---|---|---|
| 1 | Вибірка коду операції команди | 01000000 |
| 2 | Читання пам’яті | 01000001 |
| 3 | Запис у пам’ять | 00000000 |
| 4 | Читання із стека | 01100001 |
| 5 | Запис у стек | 00100000 |
| 6 | Читання з пристрою введення | 01000010 |
| 7 | Запис у пристрій виведення | 00001000 |
| 8 | Дозвіл переривань | 11000100 |
| 9 | Дозвіл зупину | 01010001 |
| 10 | Дозвіл переривань під час зупину | 11010100 |

Виконання будь-якої команди завжди починається з основного машинного циклу М1, який складається з чотирьох або п’яти тактів:

Т1 – з програмного лічильника (РС) через буфер адреси (БА) на ША МП видає адресу комірки ППЗП з кодом операції виконуваної команди, після чого вміст *PC* збільшується на 1 (інкрементується); в інших циклах М2...М5 на такті Т1 на ША видається адреса комірки пам’яті для зчитування або запису даних або номер (адреса) пристрою введення-виведення. Одночасно у Т1 кожного циклу в СКФ по ШД надходить слово стану процесора (ССПЦ).

Т2 – ССПЦ фіксується у регістрі СКФ, після чого формуються сигнали ПР (*DBIN*) та $\overline{\text{ЧТ}}$ $\overline{\text{П}}$ ($\overline{MEMR}$) для керування подальшим зчитуванням у МП коду операції та його декодування. Одночасно в Т2 аналізуються сигнали готовності ГОТ (*READY*), запиту захоплення шин ЗЗХ (*HOLD*) та ЗУП (*HALT*). При виконанні Т2 у машинних циклах М2...М5 ССПЦ також фіксується в СКФ, але далі виконуються дії залежні від команди.

Т3 – за адресою пам’яті, поданої на ША, зчитується з пам’яті код операції команди та надходить із ШД на регістр коду операції МП (РКОП) для декодування. При виконанні інших циклів М2...М5 у такті Т3 вибираються дані з пам’яті, пристроїв введення, стека або дані виводяться в пам’ять, пристрої виведення та стек.

Т4, Т5 – у МП розшифровується прийнятий код операції команди, визначається її формат, виконуються однобайтові команди або формуються машинні цикли М2...М5 для виконання дво- та трибайтових команд. При виконанні циклів М2...М5 у Т4 виконуються дії залежно від команди, а Т5 може бути відсутній.

На рис. 7.7 показані спрощені часові діаграми циклів вибірки та читання пам’яті, а на рис. 7.8 – часові діаграми циклу запису в пам’ять.

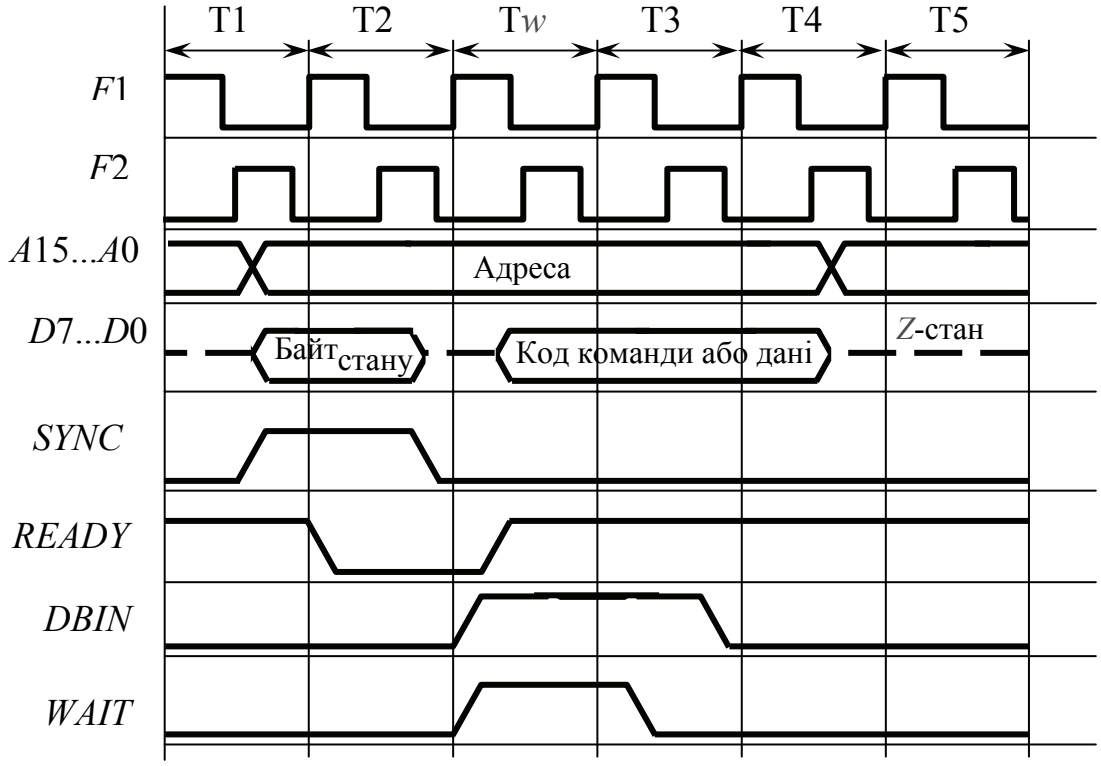



Рисунок 7.7 – Часові діаграми циклів вибірки (читання пам’яті)

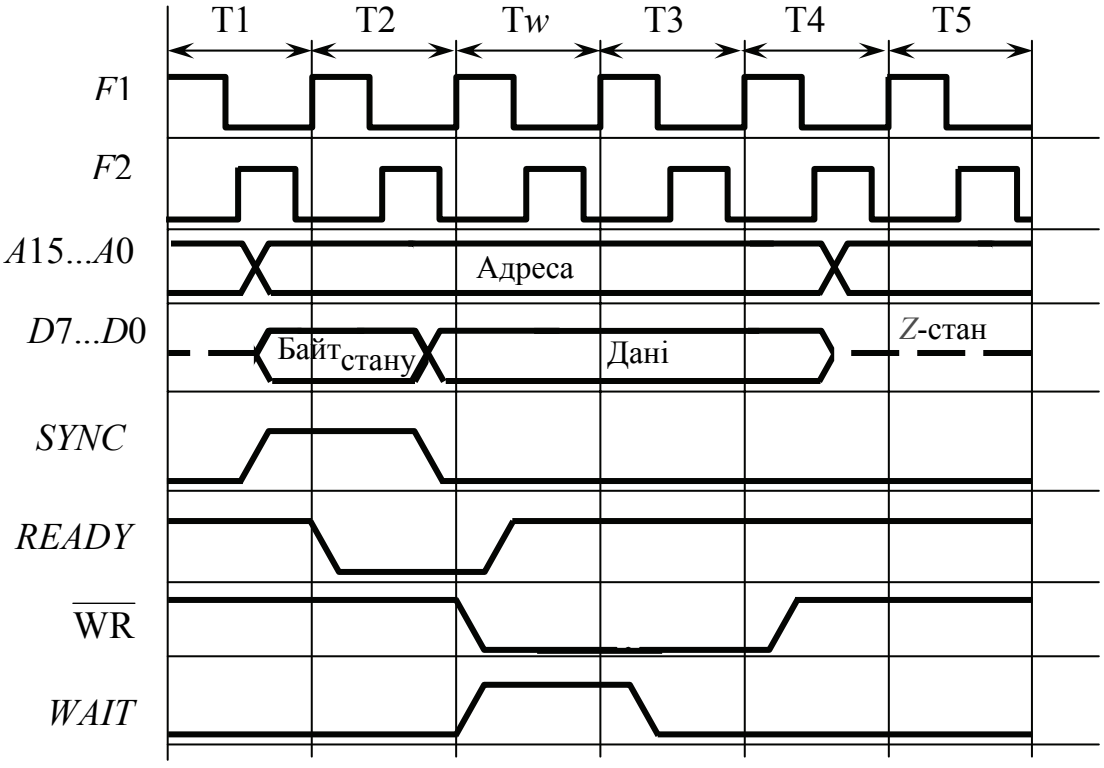



Рисунок 7.8 – Часові діаграмі циклу запису в пам’ять

На діаграмах показано також реакцію МП на наявність сигналу неготовності, який перевіряється в Т2. Якщо зовнішні пристрої неготові до

обміну або надійшов сигнал ЗЗХ, сигнал ГОТ = 0 й обмін даними здійснюватись не може і після такту Т2 уставляється один, як показано на рис. 7.7 і 7.8, або кілька тактів очікування Tw залежно від готовності зовнішніх пристроїв або пам'яті.

**Робота МПС у режимі переривань**

При готовності до роботи асинхронний зовнішній пристрій, підключений до контролера переривань, надсилає у МПС сигнал запиту переривання програми ЗПР (*INT*). Цей запит може бути задовільнено лише при дозволі переривань, який задається в основній програмі командою *EI* (*ENABLE INTERRUPT*) – дозволити переривання. Вона задає у своєму четвертому такті Т4 внутрішній тригер МП дозволу переривань *I* (*INTERRUPT*). Команда *DI* (*DESABLE INTERRUPT*) забороняє переривання; у цьому разі запит ЗПР ігнорується. За наявності дозволу переривань МП установлює керувальний сигнал дозволу переривань ДПР (*INTE*). За запитом переривання поточна команда основної програми повністю завершується, після чого починається виконання машинного циклу оброблення переривань *MI*:

*MI*1

Т1 – на ША через РА установлюється вміст лічильника команд *PC* – адреси наступної команди основної програми, а на ШД видається ССПЦ дозволу переривань.

Т2 – ССПЦ фіксується в СКФ, після чого на ШК посилається сигнал підтвердження переривань $\overline{\text{П ПР}}$ ($\overline{INTA}$), який скидає в нуль тригер дозволу переривань *I* у МП, і знімається сигнал дозволу переривань ДПР (*INTE*). У результаті блокуються всі подальші запити переривань до установлення тригера *I* командою *EI* в основній програмі або підпрограмі обробки переривань. У цьому такті вміст лічильника команд не інкрементується.

Т3 – з шини даних у РКОП МП завантажується спеціальна однобайтова команда *RSTV* (*RESTART*), сформована апаратним способом з кодом 11*AAA*111, де *AAA* – будь-яка комбінація одиниць та нулів, забезпечуючи перехід МПС до одної з восьми підпрограм обслуговування переривань, кожна з яких займає по вісім комірок ППЗП.

Т4, Т5 – у цих тактах команда *RSTV* дешифрується і формуються два наступних тритактних машинних цикли *MI*2 та *MI*3.

Після декодування команди *RSTV* визначаються початкові адреси одної з восьми підпрограм, які обслуговують переривання.

У машинних циклах оброблення переривань *MI*2 та *MI*3 вміст *PC* для забезпечення можливості повернення до основної програми після завершення підпрограми обслуговування переривань запам'ятовується у стековій пам'яті, після чого у *PC* завантажується адреса першої команди одної з підпрограм, які обслуговують переривання. Останньою командою підпрограми завжди є команда повернення *RET* (*RETURN*), при виконанні якої у *PC* повертається зі

стека адреса повернення на чергову команду основної програми для її продовження.

На рис. 7.9 та 7.10 показані спрощені часові діаграми циклу переривань відповідно *MI*1 та *MI*2 та *MI*3.

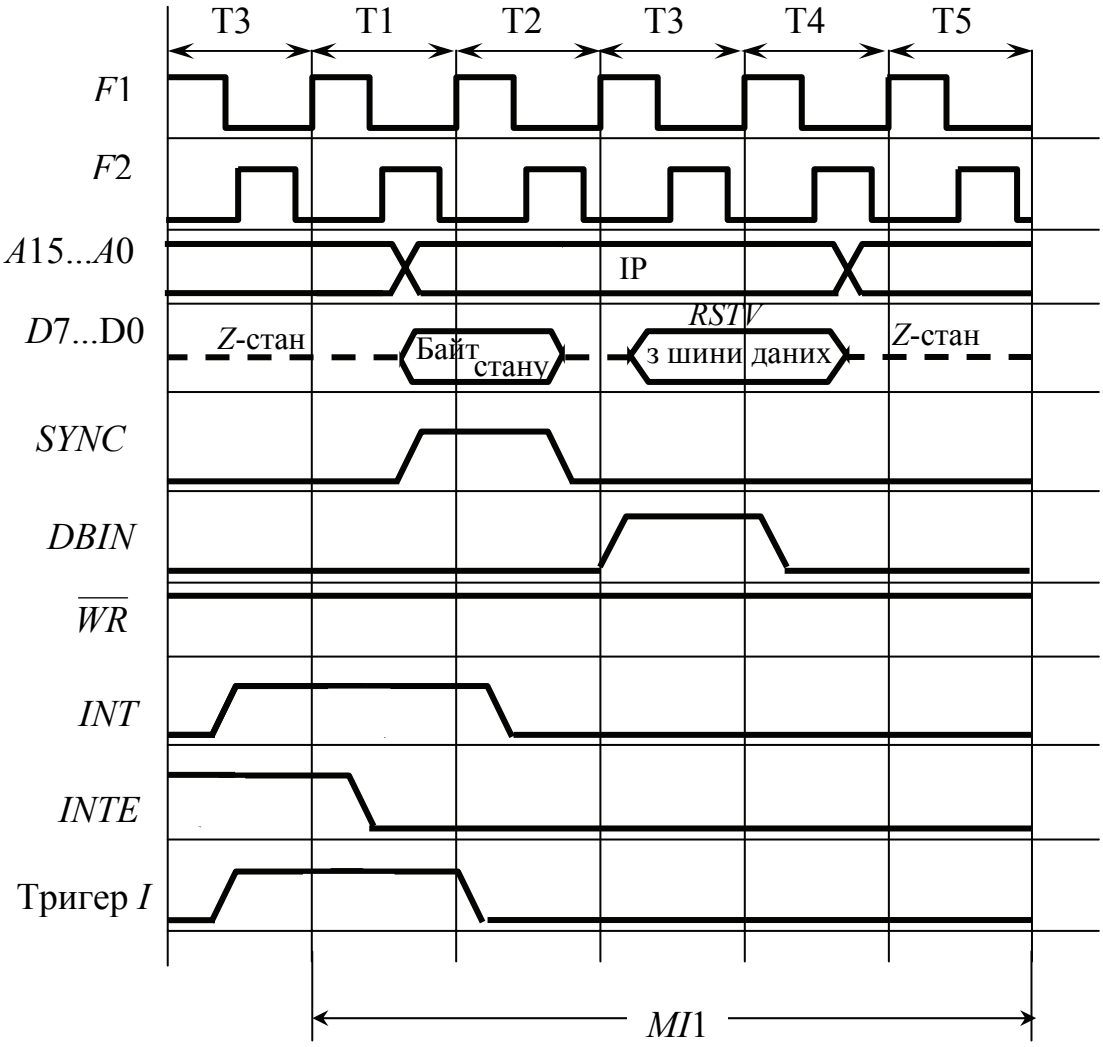



Рисунок 7.9 – Часові діаграми циклу переривань *MI*1

Можливості режиму переривань можуть бути розширені, якщо замість коду команди *RSTV* зовнішній пристрій буде формувати код стандартної команди виклику процедур *CALL ADDR*, де *ADDR* – початкова адреса підпрограми оброблення переривань.

Розглянуті способи організації переривань називаються **векторними**: ЗП, який запросив переривання вказує початкову адресу першої команди підпрограми оброблення переривань. Є ще багато різних можливостей організації переривань, наприклад невекторні, де задається адреса комірки пам’яті вектора переривань тощо.

Якщо виникають кілька запитів на переривання від декількох зовнішніх пристроїв, то виникають конфлікти, які можна розв’язати у такий спосіб. По-перше, можна послідовно опитувати зовнішні пристрої на наявність від них запитів на переривання, але це призводить до зниження швидкодії МПС і деякі

запити можна перепустити. По-друге, і цей спосіб виявився більш перспективним, можна організувати багаторівневі переривання з наданням кожному з пристроїв свого пріоритету. Запити на переривання виконуються за старшинством пріоритетів. При виникненні запиту від пристрою з більш високим пріоритетом обробка переривання з менш високим пріоритетом переривається і може продовжуватись тільки після повного обслуговування більш пріоритетного пристрою. Таких ієрархічних вкладень обслуговуючих підпрограм може бути декілька. Для організації переривань зазвичай використовуються програмовані контролери переривань.

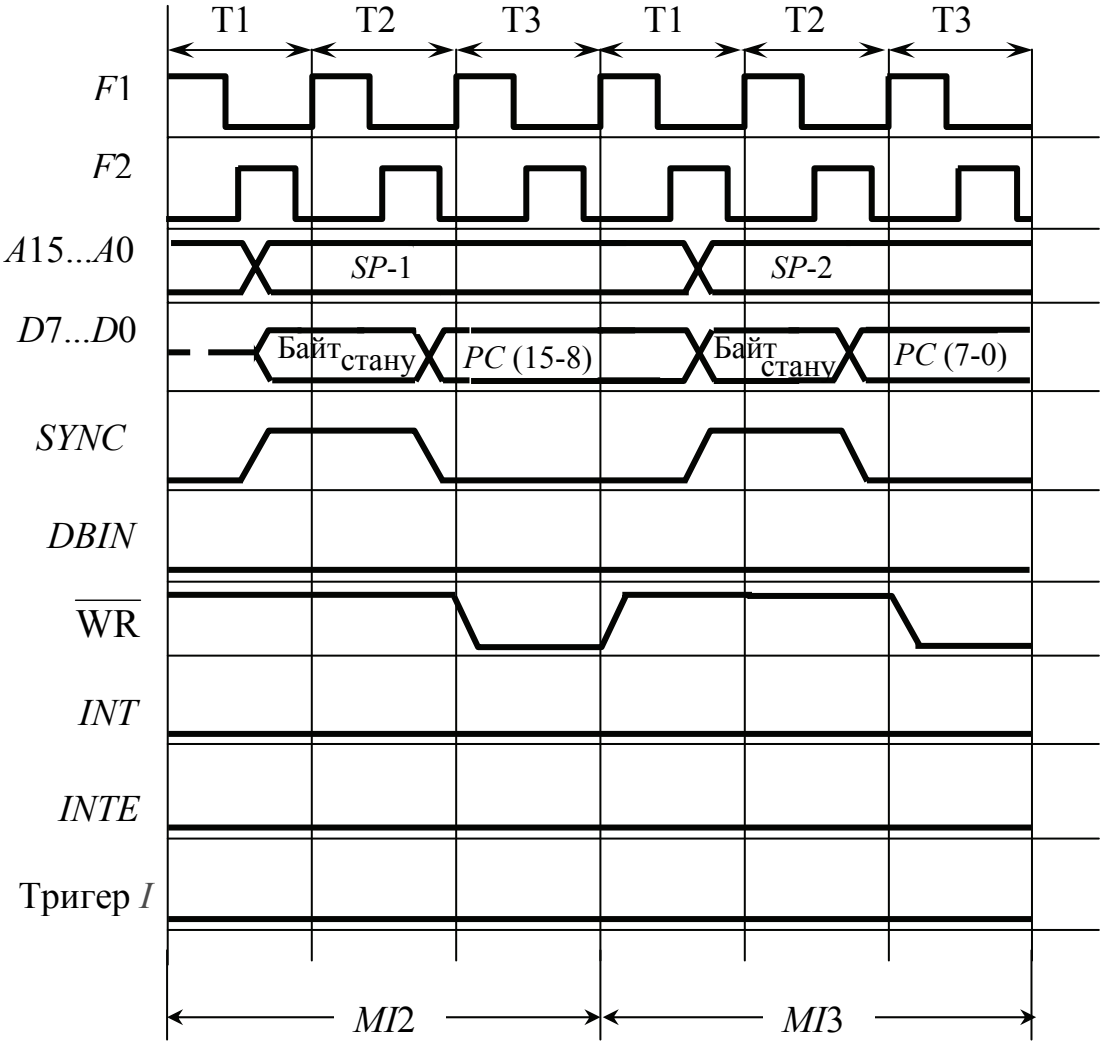



Рисунок 7.10 – Часові діаграми циклу переривань *MI*2 та *MI*3

**Контрольні питання:**

1 В яких засобах обчислювальної техніки використовуються 8-розрядні МП?

2 Яку розрядність має інтерфейс обміну даними з зовнішніми пристроями 8-розрядний МП?

3 Як інтерпретуються дані у 8-розрядних МП фірми *Intel*?

4 Які є програмно доступні елементи архітектури МПС на МП К580ВМ80А (*I*8080)?

5 З яких підсистем складається МПС КР580?

6 На прикладі виконання команди пересилання *MOV B, C* покажіть функціонування МПС.

7 Використовуючи часові діаграми циклів вибірки (читання пам'яті), покажіть реакцію МПС на сигнал неготовності ГОТ = 0.

**Контрольні питання підвищеної складності:**

1 Поясніть, як можна за допомогою пріоритетного шифратора підключити до МП зовнішній пристрій з максимальним пріоритетом?

2 В якому режимі працює клавіатура у складі ПК?

3 В якому режимі працює жорсткий диск у складі ПК?

### *7.2.2 Організація 16-розрядних мікропроцесорів*

**Вхідний контроль:**

1 Як операційна система ПК розміщує коди виконуваної програми в пам'яті?

2 Як у МПС з фоннеймановською архітектурою МП можна розрізняти, чи є код у пам'яті кодом команд, чи кодом даних?

Класичною реалізацією фоннеймановської архітектури є адресування пам'яті за плоскою моделлю, коли вся пам'ять є єдиною лінійною послідовністю байтів. У цій пам'яті зберігаються і дані, і коди програм. Відповідальність за коректне використання пам'яті цілком покладається на прикладного програміста, який повинен забезпечувати цілісність кодів та даних: дані не повинні зіпсувати коди, а стек – перекритися з областю кодів. У багатозадачному режимі, коли кільком десяткам задач надається можливість по черзі виконуватись на віртуальній машині, розподілом ресурсів займається операційна система. Віртуальна (надавана) машина складається з віртуального процесора, віртуальної пам'яті та віртуальної підсистеми введення-виведення. Найбільш незахищеною у багатозадачному режимі є пам'ять. Першим кроком до її захисту було створення у МП *I*8086, який випускався з 1978 р., моделі пам'яті реального режиму. Ця модель пам'яті організовувала пам'ять у вигляді сегментів коду, стека та даних, які мали різне призначення і різні права доступу. Ця модель була вимушеною і використовувалась з метою можливості адресування обсягу пам'яті 1 Мбайт за допомогою 16-розрядних регістрів, в яких формується адреса комірок пам'яті. Таку модель до цього часу використовують додатки, написані для операційних систем реального режиму типу *MS DOS*. МП *I*8086 створювався як призначений для роботи у багатопроцесорних системах з розподіленою пам'яттю та у складі персональних комп'ютерів, тобто вже під керуванням операційної системи.

Для зазначення початку сегмента використовуються спеціальні сегментні регістри: у 16-розрядних МП – *CS, DS, SS,ES*.

Переміщення програм та даних означає, що вони можуть бути у різні моменти часу в різних областях пам'яті, не вимагаючи від операційної системи, або додатків своєї модифікації.

Для визначення фізичної адреси адресованої комірки пам’яті у межах $2^{20}$ адресного простору вміст сегментних регістрів помножується на 4 (зсувається вліво на 4 двійкових розряди) і до нього додається значення зміщення адреси комірки від початку сегмента.

Сегментування пам’яті визначило архітектуру і програмні моделі мікропроцесорів фірми *Intel*.

Структурна схема МП *I*8086 наведена на рис. 7.11.

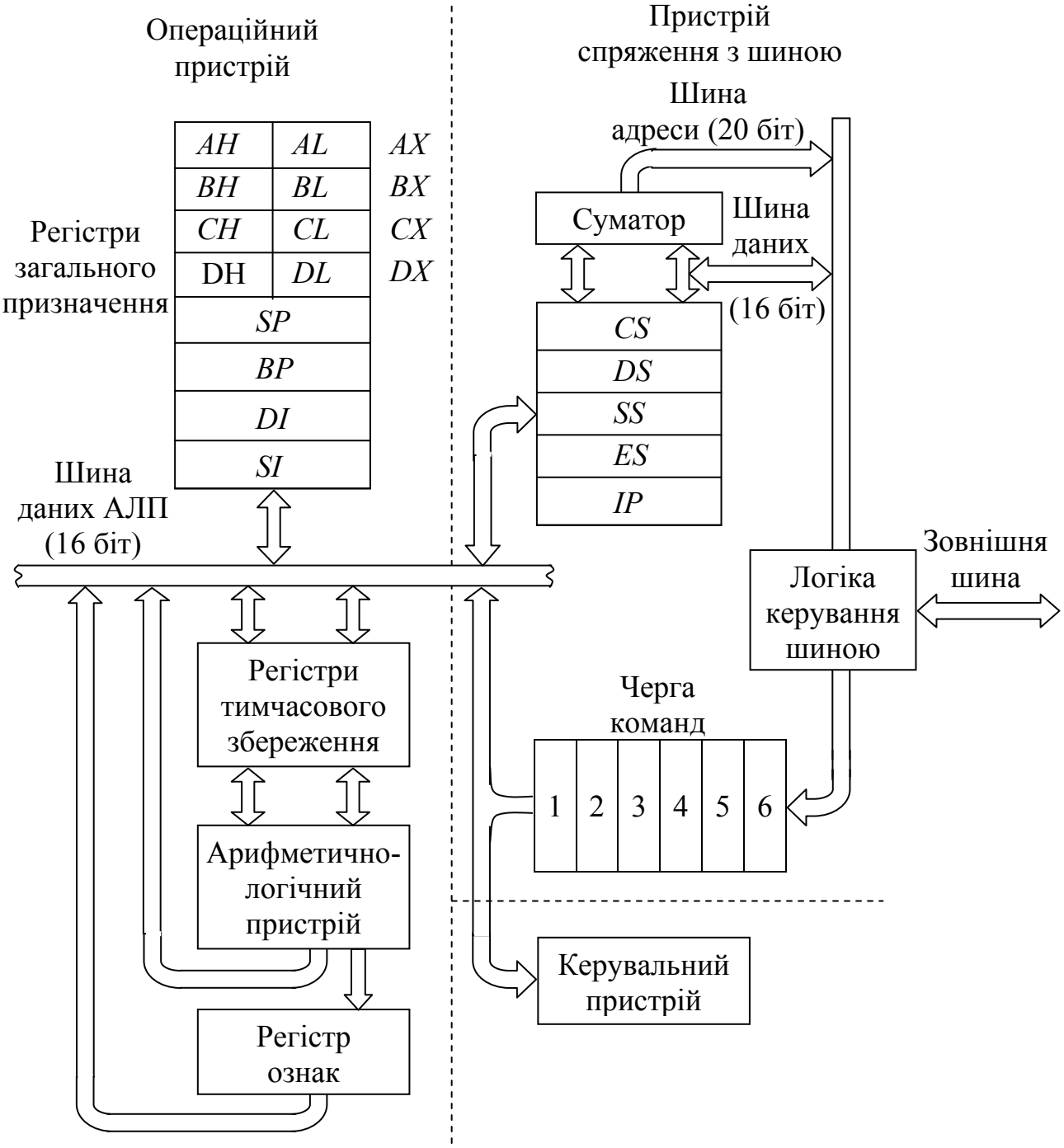



Рисунок 7.11 – Структурна схема МП *I*8086

За функціональним призначенням вузли схеми можна розбити на операційний пристрій, пристрій спряження з шиною та керувальний пристрій. В операційному пристрої виконуються обчислення, пов'язані з виконанням команд з оброблення даних; у пристрої з'єднання з шиною – обчислення, які забезпечують формування адрес команд та операндів, виклик команд та їх зберігання до початку виконання; керувальний пристрій, який дешифрує коди команд та генерує керувальні сигнали та сигнали синхронізації для МП та МПС. До операційного пристрою входять арифметично-логічний пристрій з регістрами тимчасового збереження і регістром ознак, блока регістрів загального призначення, який складається з чотирьох 16-розрядних регістрів загального призначення *AX, BX, CX, DX*, кожний з них може використовуватись як два 8-розрядних, регістрів-вказівників *SP, BP* та індексних регістрів *SI, DI*, які використовуються при формуванні адрес.

Пристрій з'єднання з шиною складається з шестибайтової регістрової пам'яті, яка називається **чергою команд**, сегментних регістрів *CS, DS, SS, ES* та вказівника команд *IP* (*Instruction Pointer*), суматора та логіки керування шиною.

Черга команд працює за принципом *FIFO* (*First Input – First Output*, перший увійшов – перший вийшов). У черзі команд можуть накопичуватись послідовні байти команд обсягом до 6 байт. Це дозволяє подавати команди з випередженням з черги в операційний пристрій, де вони виконуються, і дає можливість сумістити за часом виконання процеси, пов'язані з вибиранням команд із пам'яті, і процеси, пов'язані з їх виконанням. Черга команд називається також **конвеєром команд**.

Швидкодія процесора збільшується за рахунок конвеєризації на лінійних ділянках програм, але коли у чергу попадають команди переходів, викликів підпрограм, у тому числі підпрограм оброблення переривань та повернення з них, черга скидається і вибирання починається з нової адреси програмної пам'яті.

Суматор бере участь у формуванні виконавчої адреси (*Executive Address*) операндів, які розміщуються у пам'яті, та фізичної адреси операндів і команд.

Процесор має мультиплексовану двоспрямовану 16-розрядну шину даних/адреси *AD*15...*AD*0, яка використовується у мінімальному, однопроцесорному режимі. У максимальному режимі при роботі у складі багатопроцесорних систем МП може адресувати 1 Мбайт пам'яті за рахунок додаткових чотирьох ліній адреси/стану *A*19/*S*6…*A*16/*S*3.

Мікропроцесор *I*8086 (російський аналог К1810ВМ86) є розвитком МП *I*8080, їхня архітектура подібна. Програмно-доступні вузли і система команд *I*8080 є підмножиною вузлів і системи команд МП *I*8086. Хоча програми, написані машинною мовою *I*8080, не можуть виконуватись безпосередньо на МП *I*8086, але можна говорити про сумісність МП фірми *Intel* знизу уверх; програми достатньо просто переводяться з мови Асемблера *I*8080 на мову Асемблера *I*8086 за допомогою їхньої повторної трансляції.

МП *I*80186 є наступною моделлю МП з архітектурою *I*8086 і має додатково логічний блок вибирання мікросхеми, два незалежних швидкісних канали ПДП, три програмованих таймери, програмований контролер

переривань та вбудований генератор тактових імпульсів. Система команд порівняно з МП *І*8086 є розширеною і вміщує всі команди попередніх моделей, починаючи з *І*8080.

У 1983 р. фірма *Intel* випустила МП *І*80286, який став основою для розробки ПК *IBM PC/AT*. МП *І*80286 може працювати у двох режимах: реальному та віртуальному. У реальному режимі МП *І*80286 функціонує як МП *І*8086 з підвищеною швидкодією. У віртуальному режимі фізична адреса комірки пам'яті знаходиться за допомогою таблиць, які індексуються за допомогою сегментних регістрів, тобто реалізується так званий захищений режим роботи МП, який розглядається у підрозділі 7.2.5. При включенні у МП *І*80286 установлюється реальний режим, ініціалізуються різні регістри і прапорці і виконується команда завантаження слова стану процесора, в якому установлено біт дозволу захисту. Після переходу у віртуальний режим повернути його до реального режиму можна тільки апаратним сигналом скидання.

МП *І*80286 забезпечує засоби захисту різних областей пам'яті при роботі у віртуальному режимі за рахунок наявності рівнів привілеїв виконуваних сегментів кодів. Якщо рівень виконуваної програми є менший за рівень, який надано сегменту, МП відключає засоби захисту.

МП *І*80286 адресує до 2 Мбайт реальної пам'яті та 4 Гбайт віртуальної. Система команд вміщує всі команди більш ранніх моделей та кілька нових команд, пов'язаних з реалізацією реального та віртуального режимів.

МП *І*80286 припускає спряження з співпроцесором *І*80287, призначеним для виконання операцій з плаваючою точкою. Обидві мікросхеми можуть виконувати команди одночасно.

МП *І*80286 підтримує багатозадачний режим шляхом перемикання з однієї виконуваної задачі на іншу. МП створює для кожної задачі сегмент пам'яті, який вміщує всю інформацію, небхідну для запуску та зупину задачі (сегмент стану задачі). Основне його призначення – збереження вмісту регістрів задачі на той час, коли задача не виконується.

**Контрольні питання:**

1 З якою метою у 16-розрядних процесорах фірми *Intel* реалізується сегментування пам'яті?

2 Яку роль у структурній схемі МП *І*8086 (К1810) відіграє пристрій спряження з шиною?

3 Які операції виконує АЛП у складі МП *І*8086?

4 Які прапорці виставляє АЛП *І*8086?

**Контрольні питання підвищеної складності:**

1 З якою метою МП *І*8086 має мультиплексовану двоспрямовану шину даних/адреси?

2 Як, на Ваш погляд, можна зберегти цілісність даних, не сегментуючи пам'ять?

### *7.2.3 Програмна модель МП І8086*

**Вхідний контроль:**

1 Які вузли МП входять до програмної моделі МП?

2 З якою метою кожний регістр МП *І*8086 має своє функціональне призначення?

Програмна модель МП І8086 показана на рис. 7.12.

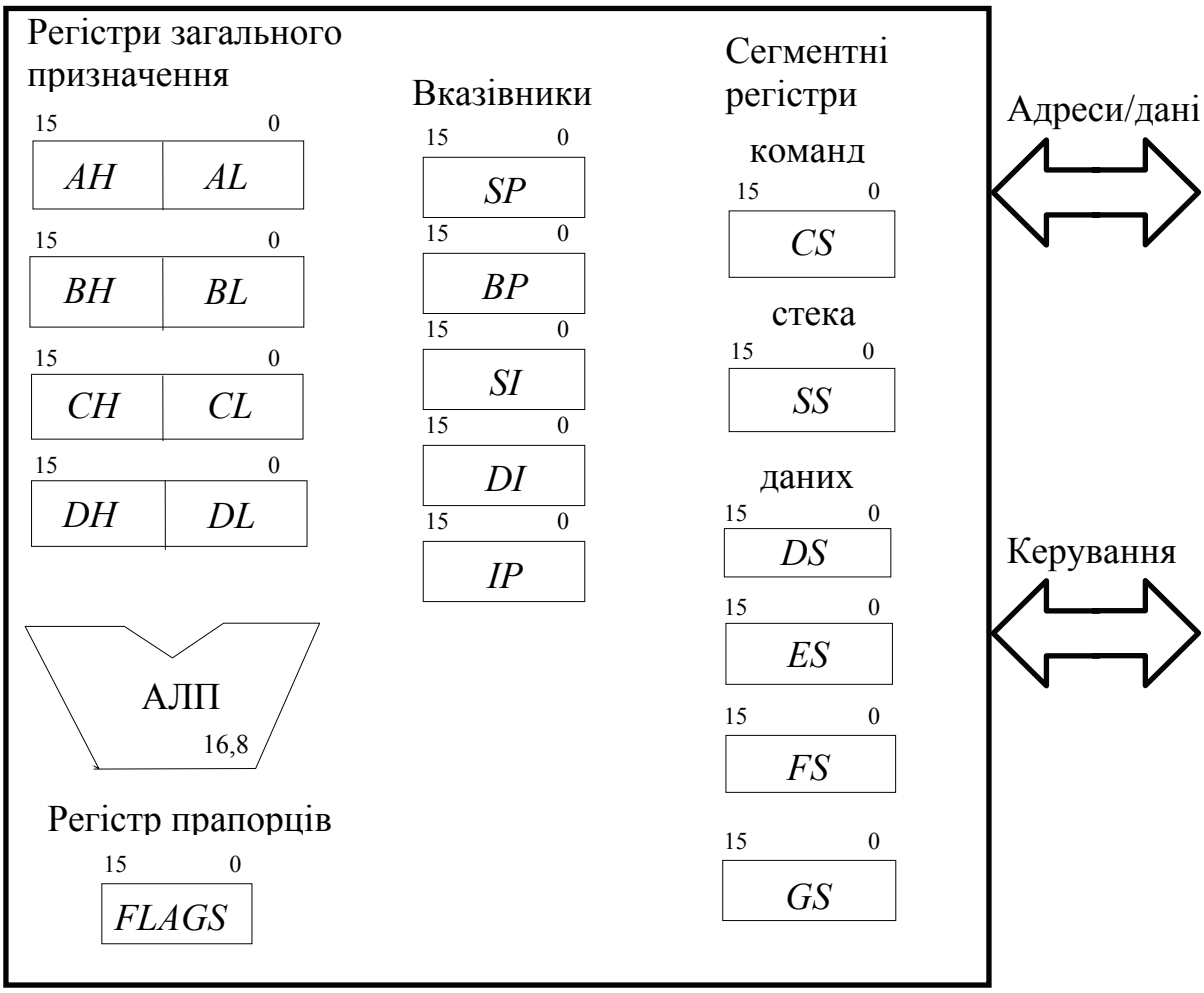



Рисунок 7.12 – Програмна модель МП *І*8086

Регістри загального призначення (РЗП) *AX, BX, CX, DX* допускають окреме використання їх молодших байтів *AL, BL, CL, DL* і старших байтів *AH, BH, CH, DH*. Тим самим забезпечується можливість операцій з байтами та словами і створюються умови для програмної сумісності МП *І*8086 та МП *І*8080.

Решта регістрів є неділимою, вони оперують 16-бітовими словами навіть у разі використання тільки їх молодшого або старшого байтів. Регістри-вказівники *SP* і *BP* та індексні регістри *SI* та *DI* зберігають зміщення адреси у межах поточного сегмента пам’яті.

Регістри загального призначення, або регістри даних, призначені для тимчасового зберігання змінних або проміжних результатів:

– *AX* – регістр-акумулятор – є регістр, в якому зберігається один із операндів перед деякими операціями, що виконуються в АЛП, та куди передається результат операції;

– *BX* – базовий регістр, який може використовуватися для формування базової адреси будь-яких змінних у пам’яті;

– *DX* – регістр додаткових даних у деяких операціях, в яких результат перевищує довжину розрядної сітки процесора; крім того, у командах введення-виведення цей регістр вміщує адресу порту;

– *CX* – регістр-лічильник; у командах організації циклів за умовчанням використовується як лічильник циклів.

Група індексних регістрів та вказівників, яку утворюють регістри:

– *BP* – вказівник бази;

– *SI* – регістр індексу джерела;

– *DI* – регістр індексу приймача;

– *SP* – вказівник стека;

– *IP* – вказівник команд.

Регістри *BP*, *SI*, *DI* використовуються при формуванні адрес змінних або для зберігання проміжних даних або результатів.

Індексні регістри *SI* і *DI* у режимі автоінкрементування та автодекрементування використовуються, здебільшого, для адресування елементів масивів. Вони можуть використовуватись також для зберігання проміжних результатів.

Вказівник стека *SP* адресує вершину стека, особливо організованої області (сегмента) пам‘яті.

Регістри сегментів *CS, SS, DS, ES* використовуються для збереження інформації про початкові адреси сегментів пам‘яті:

– *CS* – регістр сегмента команд;

– *SS* – регістр сегмента стека;

– *DS* – регістр сегмента даних;

– *ES* – регістр додаткового сегмента даних.

Регістр *CS* разом з регістром вказівника команд *IP* використовується для обчислення адреси наступної виконуваної команди.

Регістр *SS* разом з вказівником стека *SP* використовується для визначення адрес у сегменті стека. При роботі з підпрограмами адреси стекової пам’яті формуються з використанням регістра *SS* та вказівника бази *BP*.

Регістр *DS* визначає область пам’яті, де зберігаються змінні, що використовуються у програмі. Адреси цих змінних визначаються з використанням регістрів *BX, SI, DI.*

Регістр *ES* адресує сегмент пам’яті під час операцій з рядками.

Довжина сегментних регістрів становить 16 розрядів, тому операції з сегментними регістрами є операціями зі словами, які називаються **селекторами**.

Регістр вказівника (лічильника) команд *IP*, призначений для адресування в середині поточного сегмента коду. Вказівник команди прямо у командах не вказується, але бере участь в усіх командах передавання управління (умовних та безумовних переходів, виклику підпрограм, повернення з підпрограм тощо). У програміста немає можливості безпосередньо змінювати вміст регістра *IP*.

Регістр прапорців *FLAGS* зберігає інформацію про ознаки результату.

АЛП призначений для виконання арифметичних та логічних операцій над 16-розрядними або 8-розрядними числами.

Регістр прапорців *FLAGS* вміщує 16 бітів, але не всі з них зайняті ознаками результату. АЛП виставляє 9 прапорців:

– *CF*(0) – прапорець перенесення, дорівнює 1, коли результат операції виходить за межі розрядної сітки;

– *PF*(2) – прапорець парності кількості одиничних бітів у молодшому байті результату, установлюється в 1, коли кількість одиничних бітів парна;

– *AF*(4) – прапорець додаткового перенесення, установлюється в 1, коли є перенесення або позика для третього біта результату;

– *ZF*(6) – прапорець нульового результату, установлюється в 1, коли результат операції дорівнює 0;

– *SF*(7) – прапорець знака, дублює стан найстаршого біта результату, при роботі з числами зі знаками *SF* визначає знак числа – для додатних чисел *SF* = 0, для від'ємних *S* = 1;

– *TF*(8) – прапорець трасування, що використовується при налагодженні програм у покроковому режимі, в який переводиться МП при *TF* = 1;

– *IF*(9) – прапорець переривань, якщо *IF* = 1, переривання дозволені;

– *DF*(10) – прапорець напряму, використовується у командах роботи з рядками, якщо *DF* = 0, вміст регістрів *SI* та *DI* збільшується і рядок обробляється зліва направо, при *DF* = 1 – навпаки;

– *OF*(11) – прапорець переповнення, *OF* = 1 установлюється при перебільшенні результату операції над числами зі знаком допустимого діапазону.

Первісне завантаження регістрів загального призначення *AX, BX, CX, DX* та їхніх частин, а також вказівників та індексних регістрів *SP, BP, SI, DI* можливе за допомогою безпосереднього адресування. Завантаження сегментних регістрів *SS, DS, ES* може реалізуватися через акумулятор *AX*. Регістр командного сегмента *CS* є програмно недоступний і завантажується системою програмування, наприклад, *TASM*.

Приклади завантаження регістрів:

```
MOV AX,1234H   ;   Завантаження 16-розрядного регістра AX числом
               ;   1234H
MOV AX,1234H   ;   Завантаження 16-розрядного акумулятора числом
               ;   1234H
MOV AH,34H     ;   Завантаження старших восьми розрядів
               ;   акумулятора числом 34H
MOV AL,12H     ;   Завантаження молодших восьми розрядів
               ;   акумулятора числом 12H
```

```
MOV BP,400H     ;   Завантаження вказівника бази числом 400H
MOV SP,200H     ;   Завантаження вказівника стека зміщенням 200H
MOV SS,AX       ;   Завантаження сегментного регістра SS початковою
                ;   адресою сегмента 3412H з AX
MOV DS,AX       ;   Завантаження сегментного регістра DS початковою
                ;   адресою сегмента 3412H з AX
MOV ES,AX       ;   Завантаження сегментного регістра ES початковою
                ;   адресою сегмента 3412H
MOV SI,400H     ;   Завантаження індексного регістра SI початковим
                ;   зміщенням 400H
PUSH BX         ;   Завантаження вершини стека з BX
POP CX          ;   Перевантаження стека до CX
MOV AX,CS       ;   Запам'ятовування сегментного регістра кодів у AX
PUSH AX         ;   Завантаження AX до стека
POP BX          ;   Перевантаження стека до BX
MOV SS,BX       ;   Завантаження SS з BX
MOV AX,0000H    ;   Обнулення регістра AX
PUSH DS         ;   Ініціалізація стека початковою адресою
PUSH AX         ;   Ініціалізація стека нульовою адресою
```

**Контрольні питання:**

1 Яке функціональне навантаження несуть регістри загального призначення у МП *I*8086?

2 Як обчислюється виконавча адреса у МП *I*8086?

3 Як реалізується первинне завантаження сегментних регістрів?

**Контрольні питання підвищеної складності:**

1 Напишіть фрагмент програми, який би обмінював вміст регістрів *BX* та *DX* за допомогою стека.

2 Як можна завантажити, на Ваш погляд, сегментний регістр кодів *CS*?

### *7.2.4 Режим переривань МП I8086*

**Вхідний контроль:**

1 Які режими роботи МП Ви знаєте?

2 З якого типу периферійними пристроями повинен працювати МП у режимі переривань?

**Переривання** – це режим мікропроцесора, коли він тимчасово перериває виконання поточної програми і переходить до підпрограми обробки переривань, яка є більш терміновою або важливою. Після повернення з підпрограми процесор продовжує виконувати поточну програму. Для цього у стеку запам'ятовується адреса повернення (*CS* та *IP*), а також вміст регістра прапорців *FLAGS*, які потрібні для виконання також підпрограми обробки переривань. Значення IP перед завантаженням у стек коригується до адреси

команди, перед якою МП почав обслуговувати переривання. Це пов'язано з наявністю у процесорі конвеєра команд, завдяки якому *IP* адресує команди з випередженням. Слід зазначити, що вміст регістрів *CS, IP* та *FLAGS* запам'ятовується у стеку автоматично, а вміст усіх інших регістрів, які будуть задіяні потім у підпрограмі, треба запам'ятати на її початку, а потім наприкінці її вивантажити зі стека.

Переривання за видами можуть ідентифікуватись як зовнішні і внутрішні, апаратні та програмні, масковані (ті, що можна забороняти) і немасковані. Всього МП підтримує 256 типів переривань. Можливі джерела переривань показані на рис. 7.13.

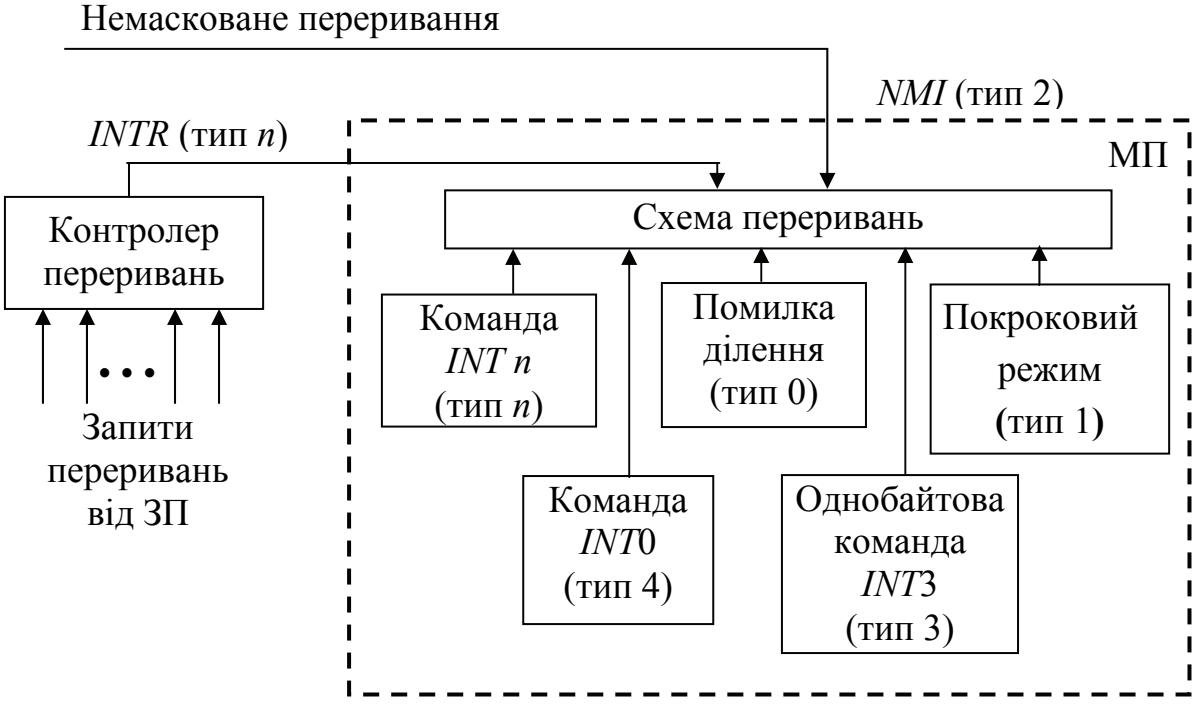



Рисунок 7.13 – Джерела переривань

Зовнішні переривання можуть виникати як реакція на запит переривань від зовнішніх пристроїв, вони є апаратні. Запити на масковані переривання надходять від периферійних пристроїв на контролер переривань, який формує вхідний сигнал МП *INTR*. У відповідь МП, якщо переривання дозволені програмно, видає сигнал підтвердження переривань *INTA*, після чого контролер посилає у МП по шині даних байт, який визначає тип переривання і МП його обробляє. Якщо програмно переривання заборонені, МП ігнорує запит, і продовжує виконання програми.

Запити на немасковане переривання надходять по входу NMI і використовуються для переривання роботи МП у таких випадках, як аварійне вимкнення живлення, помилка пам'яті тощо. Реакція на немасковані переривання не залежить від їх програмного дозволу. У більшості випадків МП реагує на запит будь-якого переривання після закінчення поточної команди. Винятком є строкові команди та команда *WAIT*, які можуть бути перервані під час виконання.

Пріоритети переривань наведені у табл. 7.2.

Таблиця 7.2 – Пріоритети переривань

| Вид переривання | Тип переривання | Пріоритет | Час виклику підпрограми переривань |
|---|---|---|---|
| За командою “помилка ділення” | 0 | 1 | 50 |
| За командою *INTn* | 5-31 | 1 | 51 |
| За командою *INT*0 | 4 | 1 | 53 |
| За командою *INT*3 | 3 | 1 | 52 |
| По входу *NMI* | 2 | 2 | 50 |
| По входу *INTR* | 32-255 | 3 | 61 |
| За прапорцем *TF* | 1 | 4 | 50 |

Оскільки зовнішні пріоритети можна маскувати, це може призвести до перерозподілу призначених різним перериванням пріоритетів. Немасковані переривання, як такі, що мають найвищий пріоритет, можуть маскувати зовнішні до закінчення обслуговування немаскованого переривання. Обслуговування запиту маскованого переривання може бути дозволено програмно шляхом установлення прапорця *IF*.

Внутрішні переривання можуть бути апаратними та програмними.

Переривання через помилку ділення (тип 0) генеруються МП відразу після виконання команд ділення, якщо формат частки перевищує формат приймача, або при діленні на нуль.

Покрокове переривання (тип 1) виробляється автоматично при установленні прапорця *TF* після виконання кожної команди, яка змінює вміст сегментного регістра, або у режимі налагодження. Підпрограма обробки покрокового переривання здійснює індикацію внутрішніх регістрів МП та деяких комірок пам’яті.

Переривання за командою *INT*3 реалізуються у вказаних точках програми.

Переривання за переповненням (тип 4) генерується після виконання команди *INTO*, якщо установлено прапорець *OF*.

Переривання, яке визначається користувачем при складанні програм, здійснюється двобайтовою командою *INTn*.

Програмні переривання дозволяють процедурам викликати одна одну через таблицю вказівників векторів (адрес) переривань, яка розміщена у пам’яті фіксовано, починаючи з адреси 00000*H* до адреси 003*FFH*, і займає 1 кбайт пам’яті (рис. 7.14).

Кожний елемент таблиці вміщує два слова, які визначають початкову логічну адресу підпрограми. Слово з більшою адресою вміщує базову адресу сегмента, а слово з меншою адресою – зміщення підпрограми від початку кодового сегмента. При переході на підпрограму адреса сегмента завантажується у регістр *CS*, а зміщення – у регістр *IP*. Розмір кожного елемента таблиці складає 4 байти, МП обчислює адресу кожного елемента шляхом множення потрібного *n* на 4. При адресуванні елементів таблиці

значення жодного сегментного регістра не використовується. На початку при виконанні підпрограм за командою *INTn* зовнішні переривання забороняються, але сама підпрограма може їх дозволити. Немасковані та внутрішні переривання можуть перервати виконання підпрограми, але у ній не припустимі переривання того типу, який вона обслуговує.

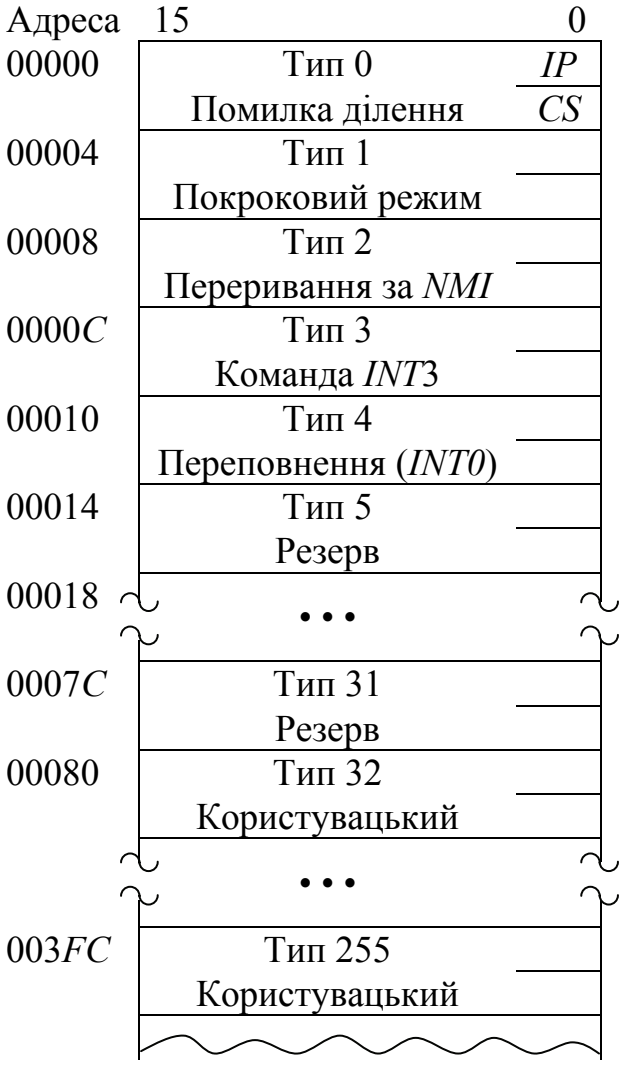



Рисунок 7.14 – Таблиця вказівників векторів переривань

Внутрішні апаратні переривання у більшості моделей мікропроцесорів обумовлені перш за все роботою системного таймера.

**Контрольні питання:**

1 Які типи переривань підтримує МП *I*8086?

2 Де зберігаються вказівники векторів переривань у МП *I*8086?

3 До якого типу переривань призведе вимкнення живлення або помилка пам’яті?

**Контрольні питання підвищеної складності:**

1 Визначіть фізичну адресу сегмента таблиці переривань типу 20.

2 З якою метою забороняються зовнішні переривання на початку виконання підпрограм оброблення переривань?

### *7.2.5 Організація 32-розрядних мікропроцесорів (Для самостійного вивчення)*

**Вхідний контроль:**

1 Прослідкуйте розвиток МП фірми *Intel* та поясніть, завдяки яким рисам архітектури МП цієї фірми вийшли на перше місце з продажу на ринку.

2 З якою метою у МПС на МП фірми *Intel* використовуються співпроцесори?

Перший повністю 32-розрядний МП фірми *Intel* МП *I*80386 був створений у 1985 р., для нього було придатне існуюче у 80-ті роки минулого століття прикладне програмне забезпечення вартістю 6,5 млрд. дол., написане для МП попередніх моделей від *I*8086 до *I*80286, тобто зберігалась сумісність знизу вверх. МП *I*80386 підтримує багатозадачність, вбудоване керування пам'яттю, віртуальну пам'ять, захищений режим. МП може виконувати до 4 млрд. операцій за секунду, працює під керуванням операційних систем *MS-DOS* та *UNIX* і припускає роботу з співпроцесорами *I*80287 та *I*80387. Пам'ять сегментується, розмір сегмента може становити від 1 байта до 4 Гбайт.

Представником другого покоління 32-розрядних мікропроцесорів є МП *I*80486, який з'явився у 1989 р. Порівняно з МП *I*80386 він має такі ознаки:

– на його кристалі додатково розміщені кеш-пам'ять даних та команд, процесор обробки даних з плаваючою точкою. Найбільш часто використовувані команди виконуються за один цикл. МП *I*80486 має апаратну та командну підтримку роботи у складі багатопроцесорних систем і зовнішню кеш-пам'ять другого рівня. Продуктивність мікропроцесора забезпечується на рівні МП архітектури *RISC*. Модель МП *I*80486*DX*2 працює на частоті 66 МГц.

Слід зазначити, що деякі засоби телекомунікацій, зокрема, ЦАТС Квант-Е використовують 32-розрядні МП другого покоління як керувальні пристрої для реалізації комутаційних полів.

МП *I*80486 може працювати з операційними системами *MS-DOS*, *Microsoft Windows*, *Unix System V*/386, операційною системою реального часу iRMX.

З появою 32-розрядних процесорів найбільш перспективною стає сегментована захищена модель, в якій пам'ять складається з захищених незалежних сегментів, які можна цілком переміщати з жорсткого диску в ОЗП і навпаки. Кожній з програм у багатозадачному режимі надаються сегменти стека, коду та до 4-х сегментів даних. Некоректні звернення додатків до пам'яті блокуються системою захисту, а для запису та читання кодів як даних потрібні деякі штучні засоби – перевизначення сегментів.

Мікропроцесор *I*80386 призначений для виконання 32-розрядних операцій, але він може функціонувати як швидкі МП *I*8086 та МП *I*80286. МП *I*80386 може працювати у реальному, захищеному режимах та режимі віртуального *I*8086. Різниця між цими режимами полягає у способах адресування та обсязі адресованої пам'яті. У реальному режимі МП *I*80386 працює як 16-розрядний *I*8086 і може адресувати пам'ять обсягом 1 Мбайт. У захищеному режимі МП працює як 16-розрядний *I*80286 та адресує пам'ять 1 Гбайт або як 32-розрядний з можливістю адресування пам'яті 4 Гбайти; можлива також організація пам'яті зі сторінковою підтримкою та апаратна підтримка захисту пам'яті від несанкціонованого доступу, реалізуються принципи захищеного режиму адресування пам'яті.

Коли виконуються команди з даними 32-розрядної довжини, МП *I*80386 використовує 8- або 32-розрядні зміщення і будь-який регістр, крім ESP, може бути використаний як базовий або індексний. При виконанні команд з даними 16-розрядної довжини як базові регістри використовуться регістри *BX, BP*, індексні – *SI, DI*, зміщення може становити 0, 8, 16 біт.

Програмна модель МП *I*80386 вміщує регістри загального призначення *EAX, EDX, ECX, EBX*, регістри-вказівники *ESP, EBP, ESI, EDI*, сегментні регістри *CS, SS, DS, ES, FS, GS*, вказівник команд *EIP*, регістр прапорців *EFLAGS*.

РЗП вміщують підмножину регістрів МП *I*8086, мають довжину 32 розряди і можуть вміщувати адреси та дані. В них зберігаються дані довжиною 1, 8, 16, 32 або при використанні двох регістрів 64 біти, або розрядні поля від 1 до 32 біт, адреси довжиною 16 або 32 біт. Програмна модель МП показана на рис. 7.15.

| | 31 16 | 15 7 | 0 7 | 0 | |
|---|---|---|---|---|---|
| *EAX* | | *AH* | | *AL* | акумулятор в командах |
| *EDX* | | *DH* | | *DL* | множення, ділення, вводу-виводу |
| *ECX* | | *CH* | | *CL* | лічильник |
| *EBX* | | *BH* | | *BL* | регістр бази |
| *EPB* | | *BP* | | | вказівник бази |
| *ESI* | | *SI* | | | індексні регістри, вказівники в |
| *EDI* | | *DI* | | | строкових командах |
| *ESP* | | *SP* | | | вказівник стека |

Рисунок 7.15 – Програмна модель МП *I*8086

РЗП називаються *EAX, EBX, ECX, EDX, ES*; регістри-вказівники – *EBP, ESP*; індексні регістри – *ESI, EDI*. Доступ до молодших 16 розрядів цих регістрів виконується при використанні імен 16-розрядних регістрів *AX, BX, CX, DX, SI, DI, BP* та *SP*. Розширений вказівник команд (*EIP*) є 32-розрядний регістр. Він вміщує відносну адресу наступної виконуваної команди. Доступ до

молодших 16 розрядів регістра *EIP* здійснюється при використанні 16-розрядного імені регістра *IP*. Це може знадобитися при виконанні програм для мікропроцесорів *I*8086 та *I*80286, які мають тільки регістр *IP*. Регістр прапорців *EFLAGS* керує введенням-виведенням, перериваннями, які можна маскувати, налагодженням програм та систем, перемиканням задач, включенням та виконанням задач МП *I*8086 у віртуальному режимі у захищеному багатозадачному середовищі.

Молодші 16 розрядів регістра *EFLAGS* є 16-розрядний регістр прапорців з назвою *FLAGS*. Регістр *EFLAGS* показано на рис. 7.16.

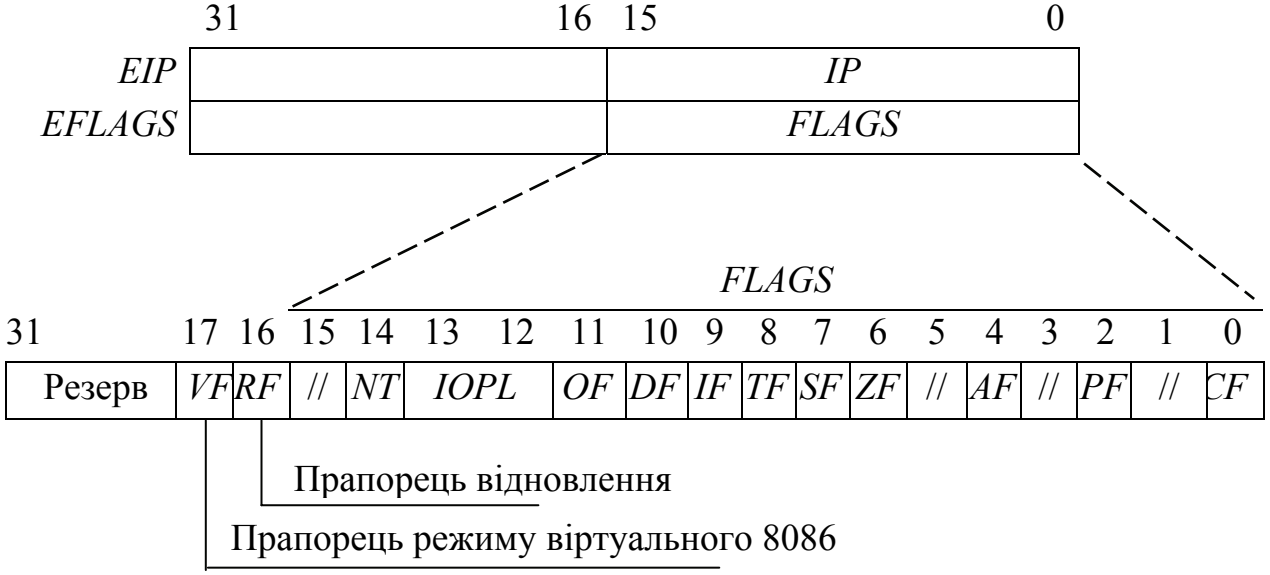



Рисунок 7.16 – Регістр вказівника команд та прапорці

Прапорці результату, які вміщуються у регістрі *FLAGS*:

*CF*(0) – прапорець перенесення зі старшого розряду. *CF* установлюється, якщо у результаті виконання операції є перенесення (при складанні) або зайом (при відніманні) відповідно зі старшого або у старший розряд. Для 8-, 16-, 32-розрядних операцій *CF* установлюється відповідно при перенесенні з розрядів 7, 15, 31.

*AF*(4) – прапорець допоміжного перенесення. *AF* установлюється, якщо у результаті виконання операції додавання чисел в упакованому форматі *BCD* виникає перенесення з розряду 3 незалежно від довжини операндів – 8, 16 або 32 розряди.

*OF*(11) – прапорець переповнення, установлюється, якщо у результаті виконання операції виникає знакове переповнення.

*ZF*(6) – прапорець нуля, установлюється, якщо результат операції дорівнює 0.

*SF*(7) – прапорець знака, установлюється, якщо установлено старший розряд результату, для 8-, 16- та 32-розрядних операндів *SF* співпадає зі значенням розрядів 8, 16, 31.

*PF*(2) – прапорець паритету (парності), установлюється, якщо молодший байт результату операції вміщує парну кількість одиниць.

Прапорці керування:

*DF*(10) – прапорець напряму, показує напрям проходження операндів при виконанні операцій з рядками (0 – у напрямі збільшення адрес, 1 – у напрямі зменшення адрес, при адресуванні за участі індексних регістрів *SI* та *DI*).

*IF*(9) – прапорець переривань, дозволяє (1) сприйняття переривань, які надходять з входу *INTR*.

*IOPL*(12, 13) – прапорець привілеїв, вказує на максимальне значення поточного рівня привілеїв (*CPL*).

*TF*(8) – прапорець трасування, при *TF* = 1 МП переводиться у покроковий режим для налагодження програми.

*NT*(14) – прапорець вкладності задач, установлюється в 1, якщо вирішувана задача вкладена у іншу задачу; значення *NT* у *EFLAGS* перевіряється при виконанні команди *IRET* для визначення, звідки треба вибрати адресу повернення:

при *NT* = 0 *IRET* вибирає адресу з поточного стека;

при *NT* = 1 *IRET* вибирає адресу із сегмента стану поточної задачі (*TSS*).

*VM*(17) – прапорець режиму, *VM* установлюється в 1 тільки у захищеному режимі при бажанні включення віртуального *I*8086;

*RF*(16) – прапорець відновлення, при *RF* = 1 використовується режим налагодження з точками зупину.

Сегментні регістри:

Шість 16-розрядних сегментних регістрів вміщують значення селекторів сегментів, які у реальному режимі адресування вказують на поточні сегменти адрес пам’яті: *CS* – сегментний регістр кодів команд програми, регістр сегмента стека *SS* і чотири регістри сегментів даних *DS, ES, FS, GS* (рис. 7.17).

Лінійна адреса операнда або команди є сума базової адреси сегмента та 32-розрядної відносної адреси.

Для адресування пам’яті у захищеному режимі в МП *I*80386 є “невидимі” для програм користувача регістри дескрипторів (описувачів) сегментів, в яких зберігається інформація, яка керує пам’яттю. До 16-розрядних регістрів МП *I*80286 додані 64-розрядні дескриптори сегментів. МП *I*80386 вибирає дескриптори сегментів з таблиць дескрипторів, використовуючи сегментний регістр як індекс при зверненні до цієї таблиці.

МП *I*80386 перетворює логічні адреси (використані програмістом у програмі) у фізичні адреси двома способами:

– перетворенням сегмента, коли логічна адреса (вміст селектора сегмента та зміщення сегмента) перетворюється у лінійну адресу, яка і є фізичною;

– перетворенням сторінки, коли лінійна адреса перетворюється у фізичну адресу.

Обидва способи є скриті від програміста і підтримуються апаратно.

Рисунок 7.17 – Сегментні регістри

На рис. 7.18 показано узагальнену схему перетворення адрес у МП *I*80386.

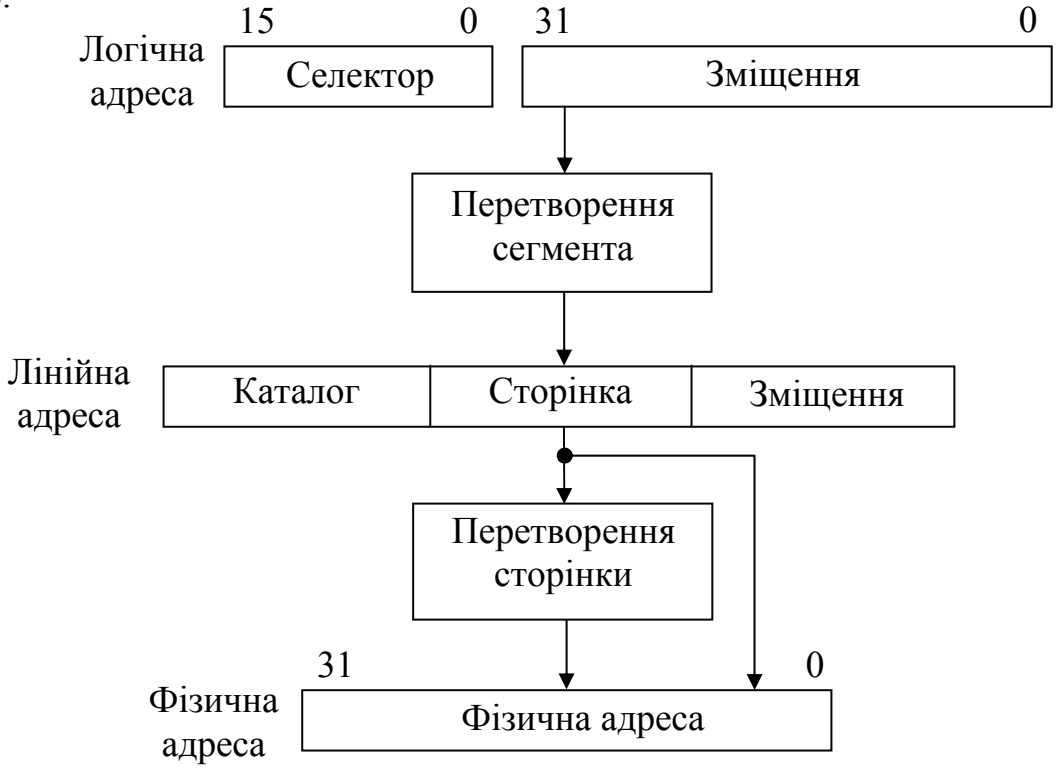



Рисунок 7.18 – Процес перетворення адреси в МП 80386

На рис. 7.19 показано, як перетворюється вміст сегментного регістра та зміщення (логічна адреса) на лінійну адресу – перша фаза.

При сегментній організації пам’яті цієї фази достатньо для отримання фізичної адреси комірки пам’яті.

При сторінковому адресуванні потрібно виконати ще одну фазу перетворення – лінійна адреса перетворюється у фізичну.

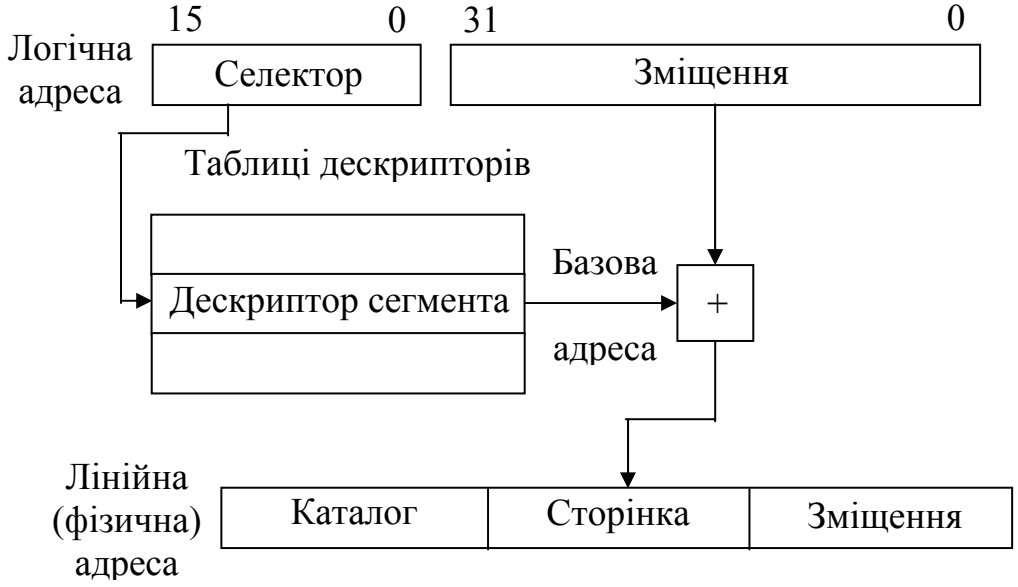



Рисунок 7.19 – Сегментна організація пам’яті

При сторінковій організації пам’яті (рис. 7.20) лінійну адресу можна трактувати як складену з трьох полів: каталогу, сторінки та зміщення. На рис. 7.20 показано дворівневу систему. Поле “Каталог” вказує на таблицю сторінок, яка відноситься до конкретного елемента каталогу, а поле “Сторінка” використовується як індекс у таблиці сторінок, які визначаються каталогом; поле “Зміщення” використовується як адреса байта всередині сторінки, вказаної

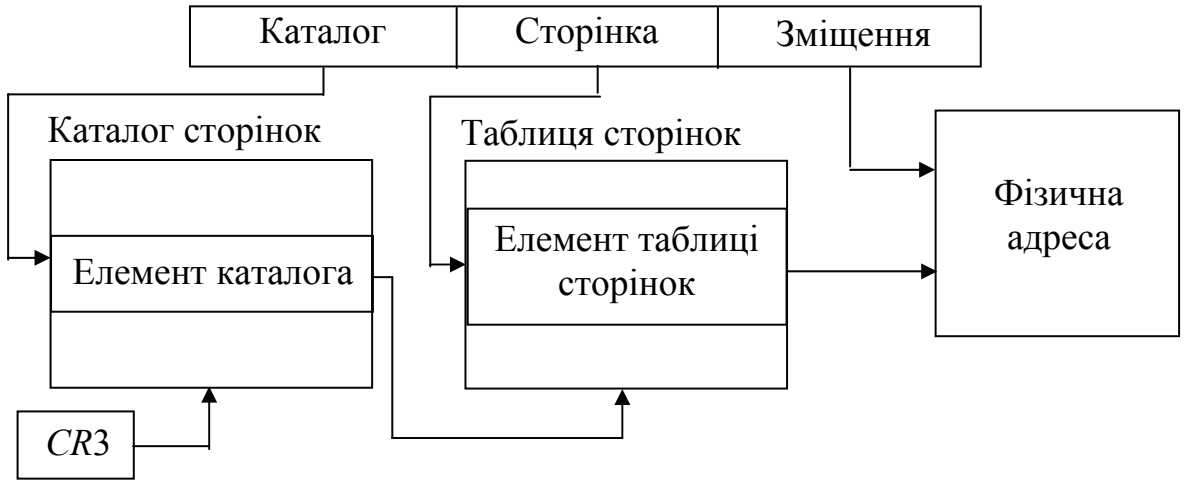



Рисунок 7.20 – Перетворення сторінок

таблицею сторінок, яка є область 1К 32-розрядних елементів. Для адресування пам’яті використовуються два рівня таблиць. На вищому рівні знаходиться каталог сторінок, який адресує до 1К сторінок таблиць другого рівня. Таблиця сторінок другого рівня адресує до 1 кбайта сторінок другого рівня. Вся таблиця

адресується одною сторінкою каталогу, тому у сукупності можна адресувати 1 Мбайт сторінок. Розмір однієї сторінки має 4 кбайти, таким чином, таблиці одного каталогу сторінок можуть адресувати всю область фізичних адрес МП *I*80386. Фізична адреса поточного каталогу сторінок зберігається у регістрі *CR*3 МП. Програми керування пам'яттю можуть задати один каталог для всіх задач, по одному каталогу сторінок на кожну задачу або комбінації з кількох каталогів. Для підвищення продуктивності при перетворюванні адрес МП зберігає найбільш часто використовувані адреси таблиці сторінок у внутрішній кеш-пам'яті, яка має назву буфер швидкого переадресування. Якщо у кеш-пам'яті немає сторінкової інформації, треба звертатися до обох рівнів таблиць сторінок. Керувати кеш-пам'яттю може тільки системний програміст.

Слід зазначити, що при сегментному адресуванні пам'яті сегмент може або цілком знаходитись у пам'яті, або ні; при значних розмірах сегментів це призводить до перевантаження МП; сторінковий спосіб адресування є більш ефективним через невеликий розмір сторінки (4 кбайти) та можливість часткового знаходження на жорсткому диску та у пам'яті.

МП *I*80386 має чотири рівні захисту привілеїв – для ізолювання програм користувача одна від одної і кожної з них від операційної системи і навпаки у багатозадачному режимі. Правила доступу до даних такі:

– дані, які зберігаються у сегменті з рівнем привілеїв *p* можуть бути доступними тільки при виконанні команд з такими самими або меншими привілеями, ніж *p*;

– кодовий сегмент з рівнем привілеїв p може бути викликаний задачею, виконуваною на тому самому рівні або більш привілейованому, ніж p.

МП *I*80386 здійснює контроль типу дескриптора сегмента, який виконується для пошуку програмних помилок при спробі використати сегменти у випадках, коли це не передбачено системним програмістом. МП перевіряє тип сегмента у двох випадках: коли селектор дескриптора завантажується у сегментний регістр і коли команда звертається до сегментного регістра, причому жодна команда не може нічого записувати у виконуваний сегмент, неможливе читання кодового сегмента, який має доступ тільки на виконання, неможливий запис у сегмент даних, який має доступ тільки на читання, контролюється межею сегмента. При виконанні команд керування *JMP, CALL* та *RET* передача управління здійснюється всередині поточного кодового сегмента і тому контролюються тільки межею сегментів.

При міжсегментних переходах команди *JMP* і *CALL* можуть звертатись до інших сегментів, тому МП здійснює контроль привілеїв сегментів і дозволяє перехід безпосередньо між сегментами, або через шлюзи при виконанні певних умов.

При виконанні команди *RET*, якщо керування на повернення з процедури є внутрішньосегментним, межі поточного сегмента не контролюються. Міжсегментна форма команди *RET* відновлює вказівник стека, і МП перевіряє привілеї сегментів, тому що є вірогідність пошкодження або зміни поточного стека.

**Співпроцесори МП *I*80386**

МП *I*80386 може працювати з арифметичними співпроцесорами *I*80287 і *I*80387 та співпроцесором локальної обчислювальної мережі *I*82586.

*Я*кщо МП *I*80386 працює з співпроцесором *I*80387, то він виконує 32-розрядні передачі. Робота співпроцесора з процесором реалізована як синхронна або псевдосинхронна.

Співпроцесор локальної обчислювальної мережі (ЛОМ) є високопродуктивним інтелектуальним контролером зв'язку, здатним вирішувати задачі керування доступом до локальної обчислювальної мережі. На рис. 7.21 показано структурну схему робочої станції локальної обчислювальної мережі *Ethernet*.

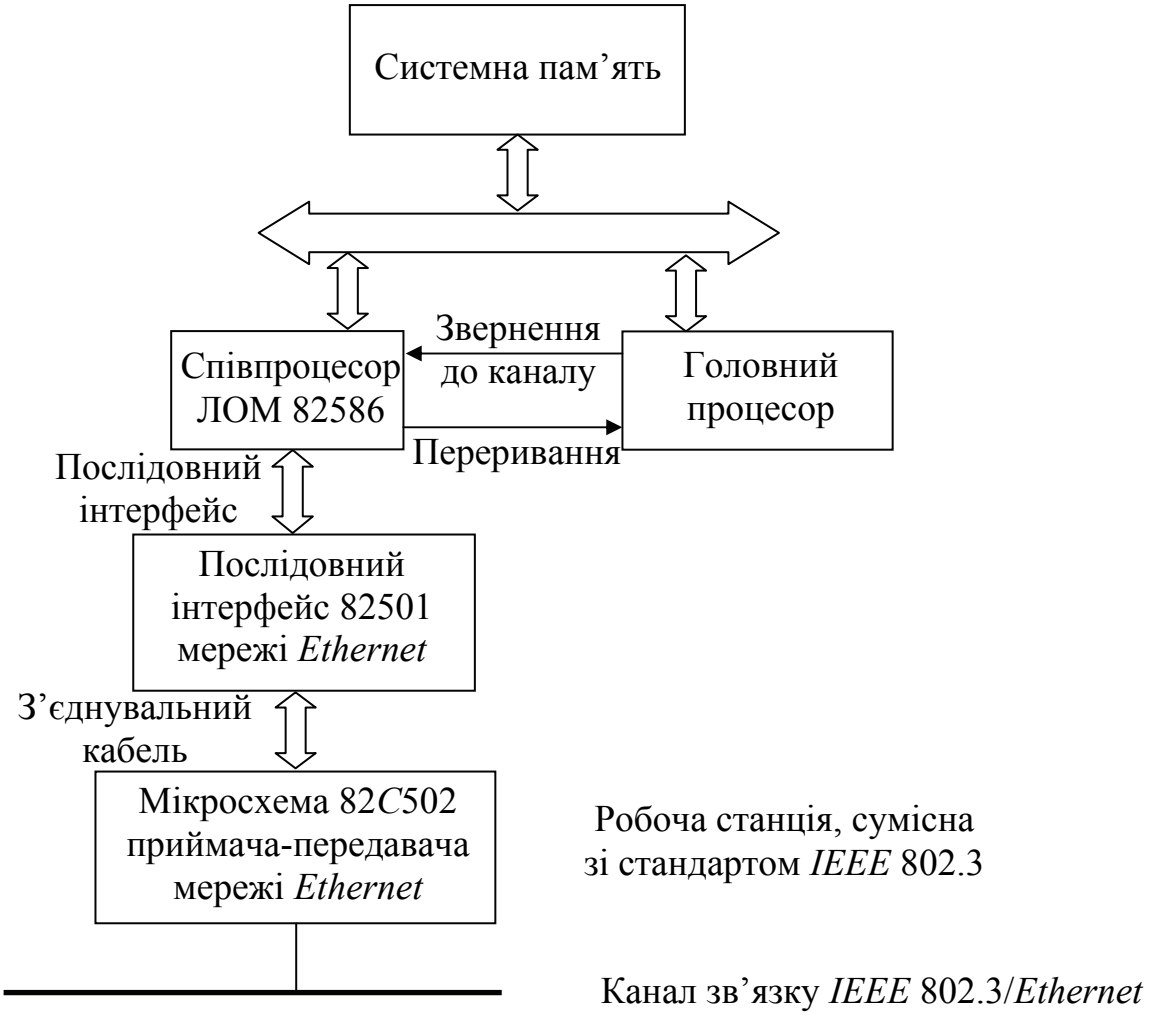



Рисунок 7.21 – Структурна схема робочої станції ЛОМ

Співпроцесор виконує усі функції, які відносяться до передавання даних між розподілюваною пам'яттю та каналом мережі, а саме: фільтрацію адреси, керування каналом, розбиття даних на кадри, виявлення помилок, кодування даних, керування мережею, прямий доступ до пам'яті, організація буферної пам'яті.

Робоча станція вміщує центральний процесор *I*80386, послідовний інтерфейс, розподілювану пам’ять, приймач-передавач та канал *Ethernet*.

Кеш-пам'ять МП *I*80386 розташована між основною пам’яттю і процесором для збільшення швидкодії системи. Механізм кеш-пам’яті є прозорим для програміста, а саме слово “кеш” є тайник. Швидкодія кеш порівняна зі швидкодією основної пам’яті.

**Кеш-пам'ять МП *I*80386**

Концепція кеш-пам’яті основана на передбаченні найбільш вірогідного використання процесором даних з основної пам’яті шляхом копіювання їх у кеш. Ця концепція розповсюджується на дані, сусідні з поточними даними. Для цього зазвичай здійснюється передавання з основної пам’яті у кеш-пам'ять блока з кількох слів, навіть якщо у даний момент потрібне тільки одне слово. Якщо потрібне слово є частиною потоку послідовних команд, то вибирання наступних команд виконується на першому етапі роботи, оскільки завдяки цьому зникає необхідність повторного звернення до основної пам’яті. Коли процесору потрібно зчитати або записати дані у пам'ять, то спочатку він перевіряє їх наявність у кеш. Якщо дані знаходяться у кеш-пам’яті (кеш-попадання), то МП *I*80386 може їх швидко отримати. У разі їх відсутності (кеш-промах) ці дані зчитуються з основної пам’яті та переписуються у кеш. МП *I*80386 працює з зовнішньою кеш-пам’яттю даних та команд.

Передбачення адреси наступного звернення до пам’яті можливе тому, що програми звертаються до пам’яті за адресою, близькою до адреси попереднього звертання. Цей принцип називається **локальністю програми**. Для збільшення відсотка кеш-попадань процесор використовує блочну вибірку. Контролер кеш-пам’яті поділяє на блоки основну пам’ять довжиною 2, 4, 8 або 16 байт. Зазвичай 32-розрядний процесор вміщує два або чотири слова на блок. У разі промаху контролер пересилає увесь блок, який вміщує потрібне слово, з основної пам’яті у кеш. Блочна вибірка дозволяє записати у кеш дані, розташовані перед потрібним байтом (перегляд назад), або дані, розташовані після нього (перегляд вперед) або обидва випадки.

Кеш-пам'ять у МП *I*80386 має ємність 64 кбайт з 16384 входами. Кожний вхід вміщує 32 біти даних разом з 16 бітами адресної інформації. Довжина рядка (одиниця обміну між МП та кеш) складає 32 біти. Для керування пам’яттю використовуються два компаратори та дві програмовані логічні матриці. Сама ж пам’ять складається з восьми статичних ОЗП даних з організацією 8*K* x 8 і два динамічних ОЗП тегового поля з організацією 8*K* x 8. Кеш-пам’ять побудована за принципом прямого відображення, кожний елемент в основній пам’яті відображається в один елемент усередині кеш. 30-розрядна фізична адреса процесора поділяється на дві частини: 16-розрядний тег та 14-розрядний індекс. Тег відповідає старшій адресі рядків (*A*16...*A*31), а індекс – молодшій адресі рядків (*A*2...*A*15). Поля тега та індексу разом визначають місце подвійного слова в основній пам’яті. Додатково індекс вибирає один із 16384 рядків у кеш. Наймолодші два біти адреси кодуються так, що доступними є всі

чотири байти подвійного слова. Кеш-пам'ять використовує буферизований наскрізний запис для оновлення динамічного ОЗП. Під час циклів запису нові дані записуються у кеш-пам’ять і одночасно в ОЗП. При використанні буфера цикл запису в ОЗП перекривається за часом з наступним циклом магістралі, що дозволяє виконувати запис за три такти. Буферизований наскрізний запис реалізується шляхом відділення режиму роботи ПК з кеш від режиму роботи з ДОЗУ. Це можливо тому, що кеш та динамічний ОЗП мають кожний свої власні контролери, що забезпечує можливість операцій з перекриттям цих функцій. Керувальна програмована логічна матриця визначає, коли у режимі роботи з кеш необхідний цикл звернення до пам’яті, а у режимі роботи з динамічним ОЗП запит перевіряється тільки у разі готовності до запуску нового циклу.

При оновленні динамічного ОЗП будь-який запис буферизується, тобто інформація затримується у кеш-пам’яті перед записом в основну пам’ять, а схеми кеш керують процесом доступу до основної пам’яті асинхронно по відношенню до роботи процесора. Процесор починає новий цикл до завершення циклу запису в основну пам’ять. Якщо за записом відбувається зчитування, то обов’язково буде кеш-попадання, тому що читання виконується у той час, коли контролер кеш зайнятий оновленням основної пам’яті. Буферизація підвищує продуктивність кеш. На рис. 7.22 наведено приклад буферизованої наскрізної пам’яті. Припустимо, що МП *І*80386 закінчив запис “нових даних 1”. Контролер кеш розмістив у буфер дані для запису й оновлення кеш. Процесор зчитує “старі дані 2” з кеш. Виникає ситуація кеш-попадання. У той час, поки відбувається читання “старих даних 2”, “нові дані 1” переписуються в основну пам’ять замість “старих даних 2”.

Недоліком буферизованого наскрізного запису є буферизація поодинокого запису, через що два останніх записи в основну пам’ять потребують циклу чекання процесора. Запис перепущеним наступним читанням потребує стану чекання процесора, який використовується для синхронізації його з повільно діючою пам’яттю. Це знижує продуктивність системи.

Останні моделі МП *І*80486 та подальші моделі використовують більш продуктивний режим зворотного запису, за наявності якого дані записуються в ОЗП тільки тоді, коли вони змінюються.

**Контрольні питання:**

1 Які системні функції підтримують 32-розрядні МП *І*80386 та *І*80486?

2 Під якими операційними системами можуть працювати ці МП?

3 В яких режимах працюють МП *І*80386 та *І*80486?

4 З якою метою використовується захищений режим?

5 Які переваги забезпечують сегментний та сторінковий способи перетворення віртуальних адрес?

6 Опишіть концепцію кеш-пам’яті.

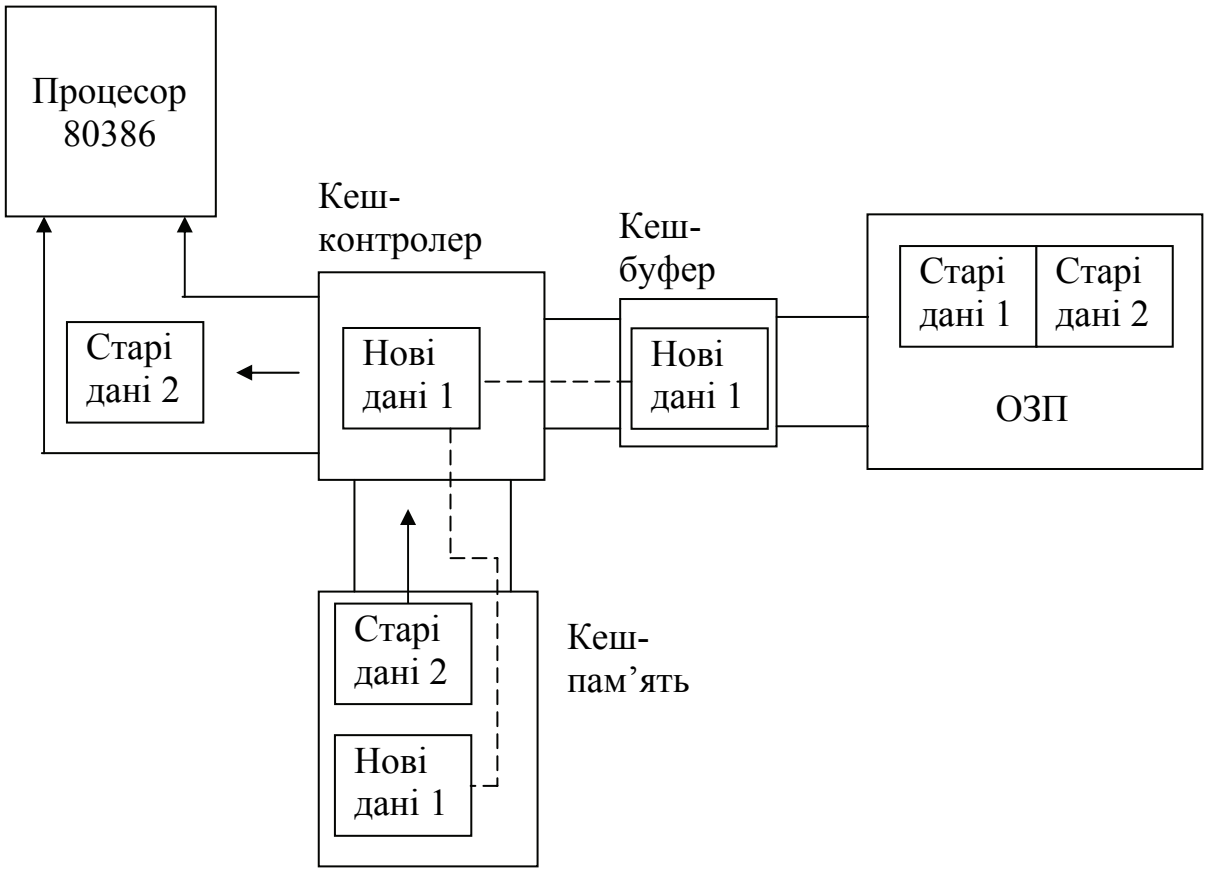



Рисунок 7.22 – Приклад буферизованої наскрізної пам’яті

**Контрольні питання підвищеної складності:**

1 Як адресується кеш-пам’ять?

2 Опишіть режим зворотного запису у кеш-пам’ять.

## 7.3 Архітектура сучасних мікропроцесорів

### *7.3.1 Тенденції розвитку архітектури сучасних мікропроцесорів*

**Вхідний контроль:**

1 З якою метою у МП вбудовуються конвеєри даних та команд?

2 Чи можна у програмі передбачити виконувану гілку перед виконанням команди умовного переходу?

Сучасний підхід до організації обчислювального процесу на мікропроцесорах має дві тенденції. Перша базується на припущенні, що система команд не вміщує вказівок на паралельну обробку даних всередині процесора, але є апаратна підтримка виконання кількох команд за один такт. Такі процесори відносяться до суперскалярних. Друга передбачає включення до формату команд спеціальних полів, які містять вказівки для кожного з пристроїв, які паралельно обробляють дані. Такі процесори називаються процесорами з **довгим командним словом** (*VLIW*) і потребують також

наявності відповідних компіляторів з мов високого рівня, які подають програми у машинних кодах для завантаження їх у процесори.

Розвиток суперскалярних мікропроцесорів відбувається шляхом якомога більшої кількості паралельних структур (конвеєрів) при традиційно послідовних програмах. Без втручання програміста апаратні та програмні засоби процесора забезпечують завантаження паралельно працюючих функціональних пристроїв процесора.

З метою підвищення продуктивності апаратно-програмні засоби процесора паралельно виконують кілька команд іноді у порядку, відмінному від їх розташування у програмі. І суперскалярні, і *VLIW* процесори використовують паралелізм на рівні команд.

Основною перешкодою для паралельного виконання програм є залежність по керуванню (розгалуженню), її треба виявити раніше, ніж будуть виконані усі наступні команди.

Процес виконання програми з конкретними наборами даних може бути представлений динамічною структурою програми, тобто у тому порядку виконання команд, як його оптимізує процесор разом з транслятором. Обмеженнями при оптимізації є залежність і по командах, і по даних. При виконанні програми процесор просувається по ній за допомогою вікна виконання. Якщо команди, які попали у вікно, є незалежні, то вони можуть виконуватись паралельно. Для усунення залежностей, викликаних командами переходів, використовується метод передбачення, який може викликати у конвеєр команд та умовно виконувати команди передбачених переходів. Якщо передбачення зроблено вірно, то результати виконуваних команд враховуються, якщо ж ні, то стан процесора відновлюється на момент прийняття рішення про виконання команд.

Команди у вікні виконання можуть також бути залежними за даними, що обумовлено використанням тих самих регістрів, комірок пам’яті або результатів попередніх команд. Деякі з цих залежностей можна усунути, використовуючи додаткові ресурси процесора, після чого команди можна виконувати паралельно. Основні блоки суперскалярного процесора – це блок вибірки команд і передбачення переходів, блок декодування команд, аналізу залежності між командами, перейменування регістрів, диспетчеризація , блоки регістрів та обробляючих пристроїв з плаваючою та фіксованою точками, блок керування пам’яттю, а також блок упорядкування виконаних команд.

Роздільні багаторівневі кеш-пам’яті даних та команд забезпечують одночасне введення у паралельно працюючі функціональні пристрої процесора кількох команд за один такт; у кеш-пам’яті введені засоби передбачення переходів. Одні засоби передбачення застосовують інформацію з двійкового коду команд або вироблену компілятором. Так, певні коди операцій частіше викликають розгалуження ніж інші, або розгалуження більш вірогідне, наприклад, у циклах, або компілятор у процесі перетворення програми у машинні коди виставляє прапорець, який установлює напрямок переходу. Може використовуватись також статистична інформація, отримана під час трасування програми. Інші засоби передбачення використовують інформацію

щодо історії розгалуження, яка запам’ятовується у таблицях розгалужень та передбачень. Якщо передбачення виявилось невірним, результати команд, які були умовно (спекулятивно) виконані, анулюються.

Після декодування команд створюються групи даних щодо виконуваної операції, адрес операндів та адреси розміщення результату. На наступному кроці відбувається відображення логічних ресурсів, які потребує програма, на фізичні ресурси мікропроцесора. Якщо один логічний ресурс відображається на кілька фізичних ресурсів, кожний з яких відповідає значенню логічної величини в один з моментів послідовного виконання програми, команда створює нове значення для логічного ресурсу; фізичний ресурс, в якому розміщується це значення, отримує ім’я. Наступні команди, які потім це значення використовують, сповіщаються відносно імені фізичного ресурсу; частіше за все це стосується перейменування регістрів. Цей прийом усуває залежність різних команд від даних.

У суперскалярних процесорах послідовні програми розбиваються на фрагменти, в яких команди виконуються паралельно. Після заповнення конвеєра командами між ними установлюються тільки необхідні залежності за даними. Ефективне використання суперскалярних процесорів обмежується двома обставинами. По-перше, через наявність умовних переходів паралелізм обчислювань на рівні команд є обмежений. Розмір вікна виконання також обмежує можливий притаманний програмі паралелізм, тому що зовсім не розглядається паралельне виконання команд, які знаходяться за межами вікна. По-друге, складність суперскалярного процесора зростає швидше, ніж кількість паралельно виконуваних команд.

Процесори *VLIW* використовують задання у командному слові сукупності паралельно виконуваних команд. Підготовка таких програм виконується компілятором. Перевагою VLIW є наявність програмних засобів, які більш ефективно, ніж апаратні, обмежені розміром вікна виконання, можуть аналізувати залежності між командами і вибирати паралельно виконувані команди. Крім того, що є важливим, *VLIW*-процесор має більш простий пристрій керування і потенційно може мати більш високу тактову частоту. Недоліком *VLIW* є також обмеження його продуктивності за рахунок обмеженості вікна виконання, вже реалізованого програмно, та проблеми з умовними переходами. Представниками процесорів типу *VLIW* є процесори фірми *Transmeta* (МП типу *Crusoe* моделей TM3120, TM5400, TM5600 з тактовою частотою 100 МГц), *Intel* (МП *Itanium*, 800 МГц) і *Heulett-Packard* – модель *McKinley*.

Перспективними є також мультискалярні процесори, які розбивають послідовний потік команд на задачі, і забезпечують більшу глибину передбачення та високу вірогідність вибору правильного напряму обчислень, а також більш широке вікно виконання. Мультискалярний процесор схожий на багатопроцесорну систему з розподілюваною пам’яттю і не вимагає ніяких апріорних знань щодо зв’язків команд з керуванням та даними. Розпаралелювання задач потребує використання компіляторів з мов високого рівня, які їх розпаралелюють. На ринку мікропроцесорів суперскалярні

мікропроцесори є лідерами. При виборі цих процесорів для розв'язання задач у проблемній області, для якої створюється обчислювальна система, слід перевіряти адекватність прийомів підвищення продуктивності розв'язуваним задачам.

Архітектура 64-розрядних процесорів не є ані 64-розрядним розширенням архітектури *CISC*, ані переробленням архітектури *RISC*. Це нова архітектура, яка для забезпечення більшого паралелізму виконання команд використовує довгі слова команд (*LIW*), предикати команд, попереднє завантаження даних, сумісність на етапі декодування команд *VLIW* та *CISC*. Для цього операційні системи повинні мати підтримку, як 64-розрядних додатків, так і 32-розрядних.

**Контрольні питання:**

1 Які тенденції розвитку архітектури сучасних МП можна назвати?

2 З якою метою процесори можуть виконувати команди у порядку, відмінному до їх написання?

3 Залежність між якими складовими програми заважає оптимізації порядку виконання команд у програмі?

4 Які методи усунення залежностей, викликаних командами переходів, Вам є відомі?

5 З якою метою у сучасних процесорах використовується вікно виконання?

6 Скільки регістрів загального призначення вміщують регістрові файли сучасних процесорів?

7 Які переваги мають суперскалярні процесори?

8 Які переваги мають процесори *VLIW*?

**Контрольні питання підвищеної складності:**

1 З точки зору програміста, для яких процесорів, суперскалярних або VLIW, складніше створювати програмне забезпечення і чому?

2 Що таке потоковий процесор?

3 Що таке мультискалярний процесор?

4 З якими операційними системами можуть працювати 64-розрядні процесори?

### *7.3.2 Мікропроцесори Pentium*

**Вхідний контроль:**

1 Скільки інструкцій виконував МП *I*80486 за один такт?

2 Скільки конвеєрів мав МП *I*80486?

Мікропроцесори п'ятого покоління *Pentium* принципово відрізняються від *I*80486 своєю суперскалярною архітектурою – здатністю виконувати на своїх конвеєрах до двох інструкцій при частоті 200 МГц. МП *Pentium* повністю програмно сумісний з попередніми МП *Intel* і дозволяє використовувати раніш розроблене програмне забезпечення для ПК.

Технічні новації, притаманні мікропроцесору, є також такі:

– окремі кеш-пам’яті для команд та даних;
– передбачення переходів;
– високопродуктивні операції з плаваючою точкою;
– удосконалена 64-розрядна шина даних;
– засоби забезпечення цілісності даних;
– засоби керування енергоживленням;
– підтримка багатопроцесорності;
– моніторинг продуктивності;
– підтримка сторінкової пам’яті різних розмірів.

Структурна схема МП *Pentium* показана на рис. 7.23.

Два конвеєри команд *U* та *V* паралельно можуть виконувати дві різні команди. Конвеєр *U* виконує усі команди, а *V* – їх обмежений набір більш простих команд.

Кожна кеш-пам'ять процесора Pentium має розмір 8 кбайт. Буфер трансляції адрес (*TLB*) перетворює адресу комірки зовнішньої пам’яті у відповідну адресу даних у кеш. У МП *Pentium* використовується метод зворотного запису, коли дані записуються в ОЗП тільки при їх видаленні з кеш. Це дозволяє модифікувати дані у кеш без збереження в ОЗП. При роботі МП *Pentium* у багатопроцесорній системі завдяки протоколу *MEST* (*Modified, Exclusive, Shared, Invalid*) забезпечується узгодженість даних в усіх кеш мікропроцесорів системи і в основній пам’яті.

Передбачення переходів у програмах реалізується завдяки буферу *BTB* (*Branch Target Buffer*) і двом буферам попередньої вибірки. Один із них використовується для попередньої вибірки команди у припущенні, що переходу не буде, а інший виконує передвибірку інструкцій у буфер, використовуючи вміст *BTB* (запам’ятовуване при першому виконанні переходу). Такий алгоритм не тільки прогнозує вибір простих віток, але й виконує складне передбачення в укладених циклах, завдяки тому, що в буфері *BTB* можуть зберігатись кілька (до 256) адрес переходів.

У МП *Pentium* застосовується блок обчислень з плаваючою точкою, який використовує складні багатоступеневі конвеєри та внутрішні функції. Майже всі команди з плаваючою точкою починають виконуватись у конвеєрі *U* або *V*, а потім передаються на конвеєр з плаваючою точкою, тому цілочислові конвеєри не є незалежні – при зупинці одного з них для роботи з числами з плаваючою точкою другий теж зупиняється. Такі функції, як множення, ділення, додавання реалізовані як внутрішні у просторі операцій з плаваючою точкою.

Всі ці нововведення призводять до перевищення швидкості виконання операцій з плаваючою точкою у 10 разів порівняно з МП *I*80486 *DX* 33 МГц. Завдяки 64-розрядній шині даних МП *Pentium* може обмінюватись даними з пам’яттю зі швидкістю 528 Мбайт/с. МП *Pentium* реалізує конвеєризацію циклів шини, що дозволяє почати другий цикл ще до завершення першого. Для підвищення швидкості виконання послідовних операцій запису в пам’ять МП Pentium має по одному буферу запису на кожний конвеєр, завдяки чому

мікропроцесор може продовжувати роботу, виконуючи наступні команди до запису результату попередньої команди в пам’ять.

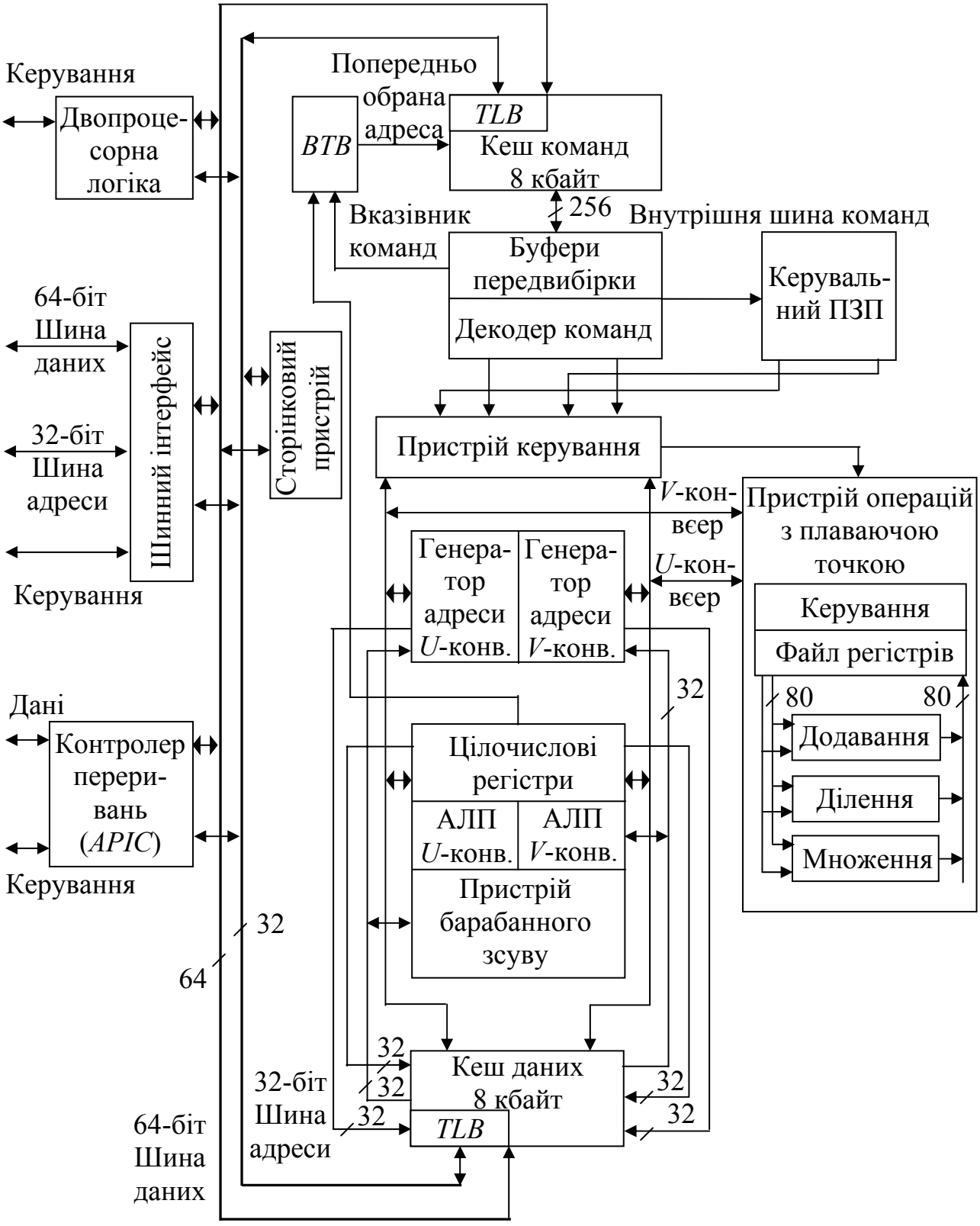



Рисунок 7.23 – Структурна схема МП *Pentium*

Цілісність даних перевіряється за ознакою парності одиниць у внутрішніх буферах процесора.

У МП *Pentium* засоби енергозбереження застосовуються як на рівні мікропроцесора, так і на рівні системи. При виконанні задач, які не потребують інтенсивних обчислень, процесор може бути переведений у режим зі зниженою тактовою частотою та зниженим енергоживленням.

Підтримка багатопроцесорності забезпечується наявністю внутрішнього контролера багатопроцесорних переривань *APIC*, який підтримує до 60 процесорів у системі, та двовходового контролера кеш-пам'яті другого рівня, який дозволяє двом процесорам сумісно використовувати один кеш другого рівня.

Мікропроцесор *Pentium* підтримує розміри сторінки пам'яті 4 кбайти та 4 Мбайти. Це дає можливість регулювати частоту переключення сторінок у ядрі операційної системи або у графічних додатках.

МП *Pentium MMX* характеризується апаратною підтримкою багатьох операцій, характерних для процесорів цифрової обробки сигналів: одна інструкція виконує дії над кількома (2, 4, 8) комплектами операндів (принцип *SIMD*), операції над векторами, згортка, перетворення Фур'є тощо. Це стало можливим завдяки розширенню системи команд на 57 команд, орієнтованих на ефективне виконання типових мультимедійних алгоритмів. Процесор вирішує задачі синтезу звука та музики, розпізнавання мови, обробки відео- та графічної інформації, виконання комунікаційних функцій.

*SIMD*-обробка потоків даних значно прискорює виконання мультимедійних алгоритмів, для яких є характерне виконання ідентичних операцій над великими масивами однотипних даних – 16-бітними відліками цифрового мовного сигналу, 8-бітні коди кольору пікселя.

Процесор підтримує операцію *MAC* – множення з накопиченням, у ньому вперше був застосований новий вид арифметики з насиченням: якщо результат операції не вміщується у розрядній сітці, то замість переповнення або втрачання порядку установлюється відповідно максимально або мінімально можливе значення числа.

*Pentium MMX* має дві окремі кеш-пам'яті – кеш команд та кеш даних, обсяг кожної з них 16 кбайт.

Ефективність за швидкістю виконання мультимедійних додатків у *Pentium MMX* на 60% вища ніж у процесора *Pentium* за однакових тактових частот. Регістри *MMX Pentium MMX* суміщені з 64-розрядними регістрами з плаваючою точкою, що забезпечує сумісність з архітектурою *Pentium* і дозволяє використовувати раніше розроблене програмне забезпечення. *Pentium MMX* за 50 тактів може переключатися в режим обчислень з плаваючою точкою і використовуватися як універсальний.

Процесор *Pentium Pro* (*P*6) відноситься до шостого покоління; він має кеш другого рівня на 512 кбайт, розміщений у тому ж корпусі, який працює на частоті ядра – 200 МГц. Архітектура процесора спрямована на збільшення паралельно виконуваних гілок програми, причому команди можуть виконуватись не в тому порядку, в якому вони розташовані у програмі, але

послідовність запису результатів у пам'ять або виведення їх у порти зберігається відповідно до програми, це так зване динамічне виконання інструкцій. Передбачено спекулятивне виконання команд з випередженням, переупорядкування команд по конвеєрах з метою вирівнювання часу їх виконання, що також підвищує продуктивність процесора. Процесор має різні незалежні шини для з'єднання процесорного ядра з кеш-пам'яттю та основною пам'яттю. Перша шина працює на тактовій частоті процесора, а друга – з частотою системи. Слід зазначити, що 16-розрядні додатки виконуються на *Pentium Pro* не швидше ніж на *Pentium* з такою ж тактовою частотою. Це пояснюється частим перезавантаженням виконавчого багатоступінчатого конвеєра з короткими даними.

До МП шостого покоління відносяться також МП *Pentium* II, *Pentium* III, *Celeron* та *Xeon*. Загальними рисами цих процесорів є те, що ядро процесора вміщує кілька конвеєрів, до яких підключаються виконуючі пристрої для операцій над цілими числами, звернень до пам'яті, передбачення переходів та обчислень з плаваючою точкою. Кілька різних виконавчих пристроїв можуть об'єднуватися на одному конвеєрі.

*Pentium* II є більш швидкий ніж *Pentium Pro* з підтримкою *MMX*, але не пристосований до роботи у багатопроцесорних системах. *Pentium* III – це подальша розробка *Pentium* II, його основною відміною від попередніх моделей є розширення *SIMD*-інструкцій, засноване на новому блоці 128-розрядних регістрів *XMM*. Блок дозволяє одній інструкції виконувати операції одночасно над чотирма комплектами 32-розрядних операндів з плаваючою точкою. При виконанні повних інструкцій обладнання для виконання традиційних операцій *FPU/MMX* не використовується, це дозволяє одночасно виконувати інструкції *MMX* з інструкціями над операндами з плаваючою точкою, а також логічні інструкції.

Сімейство Xeon – *Pentium* II *Xeon* та *Pentium* III *Xeon* призначені для роботи у складі серверів, які можуть бути об'єднані у системи з *FRC*-режимом, в якому процесор працює як перевірний з надлишковим контролем функціональності у двопроцесорній, а також у симетричних 1-, 2-, 4- та 8-процесорних системах. Процесори *Pentium* II *Xeon* мають частоту шини 100 МГц і частоту ядра 500 МГц, вторинний кеш 1 Мбайт, виконують стандартний набір інструкцій *Pentium* II та *MMX*. Процесори *Pentium* III *Xeon* мають частоту шини 133 МГц при частоті ядра до 1000 МГц. Вторинний кеш сягає 2 Мбайти.

Процесори *Celeron* призначені для однопроцесорних конфігурацій, мають частоту ядра до 3,2 МГц, для *Celeron D* частота шини пам'яті складає 533 МГц, обсяг кеш *L*2 дорівнює 256 кбайт.

До сьомого покоління процесорів *Intel* належить *Pentium* 4. З програмної точки зору він є процесор з архітектурою x86 з розширенням системи команд *SIMD – SSE*2, який виник у 2001 році. У 2005 році був представлений процесор *Pentium* 4 з 64-бітним розширенням (*EM*64*T*) і додатковими командами *SSE*3. Цей процесор можна віднести вже до восьмого покоління. Процесор має так звану архітектуру *NetBurst* і працює на високих частотах ядра до 3,4 ГГц і шини

пам’яті 800 МГц; кеш L2 становить 512 кбайт, L3 – 2048 кбайт. Процесор спрямовано на використання у мультимедійних потокових інтернет-додатках: обробка відеоінформації у реальному часі, задачі кодування-декодування сигналів, *IP*-телефонія, шифрування даних, телебачення високої чіткості тощо. *Pentium* 4 побудований за технологією гіперпотоковості (*hyperthucading*), яка розміщує на одному кристалі два логічних процесори, які одночасно виконують два потоки інструкцій.

Завдяки тому, що два самостійних процесори реалізовані на одному кристалі, можна об’єднати їх кеш та зменшити час передавання даних поміж ними. Двоядерні процесори дають значне підвищення продуктивності, тільки якщо програмне забезпечення підтримує паралельні обчислення, але не у два рази відносно до одного ядра через те, що потрібний час на перерозподіл навантаження між ядрами, розподіл пам’яті тощо.

Мультиядерні процесори мають кілька функціонально завершених процесорів на одній шині. Кожне ядро має власні кеші *L*1 та *L*2 та кеш трас. Розмір кожного кеша *L*2 сягає 2 Мбайт. Інтерфейс системної шини може бути спільним або роздільним.

Двоядерний процесор *Core* 2 *DUO E*6850 має штатну частоту ядра 3 ГГц і частоту шини 1333 МГц. Чотириядерний процесор *Core DUO Q*6600 має штатну частоту ядра 2,4 ГГц і частоту шини 1067 МГц. Процесори мають архітектуру *Intel*® 64®, що дозволяє підтримувати 64-розрядні обчислення, збільшувати обсяг пам’яті, яка адресується, підвищувати рівень безпеки у віртуальних та спеціалізованих обчислювальних середовищах.

Слід зазначити., що всі сучасні процесори у своїй архітектурі мають елементи, які призначені для зменшення енергоспоживання. Вони також не вимагають особливих умов для охолодження.

У табл. 7.3 наведені основні технічні характеристики чотириядерних процесорів *Intel*® *Xeon*® серії 5300.

Таблиця 7.3 – Основні технічні характеристики чотириядерних процесорів *Intel*® *Xeon*® серії 5300

| Процесор | Ємність кеш-пам’яті, Мбайт | Частота системної шини, МГц | Тактова частота, ГГц |
|---|---|---|---|
| *Intel*® *Xeon*® *X*5355 | 8 | 1333 | 2,66 |
| *Intel*® *Xeon*® *X*5345 | 8 | 1333 | 2,33 |
| *Intel*® *Xeon*® *X*5320 | 8 | 1066 | 1,86 |
| *Intel*® *Xeon*® *X*5310 | 8 | 1066 | 1,6 |
| *Intel*® *Xeon*® *X*5335 | 8 | 1333 | 2 |

**Контрольні питання:**

1 Чи можна сказати, що процесори *Pentium* усіх моделей мають фон-нейманівську архітектуру?

2 Які технічні новації притаманні процесору *Pentium*?

3 Яку структуру має кеш-пам’ять процесора *Pentium*?

4 Скільки розрядів має шина даних процесора *Pentium*?

5 Які засоби енергозбереження має процесор *Pentium*?

6 Які загальні риси мають процесори шостого покоління Pentium II, *Pentium* III, *Celeron* та *Xeon*?

**Контрольні питання підвищеної складності:**

1 В яких процесорах використовуються архітектури багатопроцесорних систем *SIMD, MISD, MIMD*?

2 Поясніть, які особливості має архітектура двоядерних процесорів?

3 Які особливості має архітектура мультиядерних процесорів?

### *7.3.3 Процесори фірми AMD*

**Вхідний контроль:**

1 Який з відомих Вам процесорів фірми *Intel* має найвищу тактову частоту і яка вона є?

2 В яких областях використовують мультиядерні процесори?

Процесор *Athlon* (*K*7) фірми *AMD* є високопродуктивним за рахунок високої тактової частоти (з 1 ГГц), а також особливої суперконвеєрної, суперскалярної архітектури. Процесор має триканальний декодер інструкцій x86, який вибирає їх з пам'яті через кеш, та потужний блок передбачення розгалужень. Інструкції мови Асемблер 86, які можуть мати довжину від 1 до 15 байт, перетворюються в уніфіковані макрооперації. За кожний такт блок керування інструкціями може отримати до трьох інструкцій від декодера. Сам блок є буфер, який перевпорядковує інструкції та розподілює макрооперації по виконавчих пристроях процесора, перейменовує регістри та приймає результати обчислень з виконавчих конвеєрів. Процесор має три незалежних конвеєри цілочислових обчислень, три конвеєри для обчислення адрес операндів та триканальний пристрій для обчислень з плаваючою точкою. У процесорі вперше застосована повністю конвеєризована суперскалярна архітектура зі зміненням порядку виконання інструкцій для обчислень з плаваючою точкою. Процесор виконує усі інструкції традиційного *FPU*(*x*87), *MMX* та 3*D Now!*. Продуктивність процесора сягає 4 Гфлопс при обчисленнях з одинарною точністю.

Система команд вміщує стандартний набір інструкцій *Intel* шостого покоління, *MMX* і розширену технологію 3*D Now!*. 12 нових цілочислових *SIMD*-інструкцій призначені для виконання операцій, пов'язаних з розпізнаванням мови, відеокодуванням, 7 інструкцій пов'язані з прискоренням передавання даних, 5 інструкцій реалізують функції сигнальних процесорів, вони дозволяють підвищувати продуктивність таких додатків, як програмні моделі (*ADLS*), процесори об'ємного звучання тощо.

Первинний кеш процесора становить 128 кбайт (64 для даних і 64 для команд) і високопродуктивний інтерфейс розрядністю даних 72 біти для

вторинного кеш обсягом 8 Мбайт. Процесори *Athlon* можуть працювати у багатопроцесорних системах.

Для підтримки процесора *Athlon* фірма *AMD* випустила чипсет, який складається з двох мікросхем: системного контролера та периферійного контролера. Системний контролер забезпечує зв'язок процесора, динамічної пам'яті, порту *AGP* та шини персонального комп'ютера *PCI*. Чипсет допускає установлення до трьох модулів *DIMM SDRAM*, з сумарним обсягом 768 Мбайт. Шина *PCI* може обслуговувати до 6 контролерів. Системний контролер має буфери, які забезпечують можливість одночасного виконання обміну даними між парами "користувачів" (процесор, пам'ять, порт *AGP* і шина *PCI*). Периферійний контролер має міст шин *PCI-ISA*, який підключає усі системні пристрої *PC*-сумісного комп'ютера (контролери переривань, прямого доступу до пам'яті, клавіатури та миші, таймер, інтерфейс флеш-*BIOS*). Контролер *USB* виконує функції центрального концентратора і має 4 порти для підключення зовнішніх пристроїв.

Процесор *Athlon*-64 та *Opteron* мають власний контролер пам'яті з окремою шиною, до якої безпосередньо підключаються модулі пам'яті. У *Athlon*-64 шина даних пам'яті є 64-бітна, а у *Opteron* – 128-бітна. Завдяки цьому затримка доступу до пам'яті становить 45 нс. Процесори *Opteron* призначені для серверів (1–8-процесорних систем) та робочих станцій (1–4-процесорних систем) мають три 16-бітних інтерфейси *Hyper-Transport* з пропускною здатністю 19,2 Гбайт/с. Кожний процесор входить у багатопроцесорну систему і має також доступ до пам'яті інших процесорів.

Якщо надати кожному процесору свій адресний простір, можна реалізувати багатокомп'ютерну систему, в якій за сумісної паралельної роботи вони можуть обмінюватись лише повідомленнями через інтерфейс *Hyper-Transport.*

Фірма *AMD* випускає двоядерні процесори *Athlon*-64x2 для робочих станцій, а також процесори *Opteron* (64-бітний) і *Athlon MP* (32-бітний) для мультипроцесорних серверів і робочих станцій.

Для настільних комп'ютерів призначені 64-бітні процесори *Athlon* 64 *FX* і *Athlon* 64, а також 32-бітні *Semptron* і *Athlon XP*.

**Контрольні питання:**

1 Які особливості має архітектура процесорів фірми *AMD*?

2 Який з відомих Вам процесорів фірми *AMD* має найвищу продуктивність?

**Контрольні питання підвищеної складності:**

1 Що таке додатки *MMX*?

2 Що таке технологія 3*D Now!*?

3 Чи підвищує сьогодні якість зображень в ігрових програмах застосування у ПК мультиядерних процесорів?

### *7.3.4 Продуктивність мікропроцесорів та її оцінювання*

**Вхідний контроль:**

1 Як Ви пов'язуєте продуктивність МП з тактовою частотою?

2 З якою метою Ви обираєте свій домашній ПК з максимальною продуктивністю?

Технічна пікова продуктивність МП – це теоретичний максимум швидкодії комп'ютера або МПС за ідеальних умов. Вона визначається як кількість обчислювальних операцій, які виконуються за секунду усіма арифметико-логічними пристроями, які є у процесорі. Максимальна швидкодія досягається при обробленні нескінченної послідовності не зв'язаних між собою за даними команд, які також не конфліктують при доступі до пам'яті. Практично жодна система не може довго працювати з максимальною продуктивністю, але майже всі комп'ютери досягають 0,8…0,95 максимуму.

Для оцінки пікової продуктивності потрібно знати тактову частоту процесора, розрядність оброблюваних даних, пропускну здатність та кількість внутрішніх шин, перелік функціональних пристроїв.

Відповідність між одиницями вимірювання тактової частоти та продуктивності процесора установлюється для одного конвеєра такою: 1 МГц відповідає 1 *MFLOPS* або 1 *MIPS* пікової залежно від типу операції – з плаваючою або фіксованою точками. Для суперскалярних процесорів пікова продуктивність обчислюється множенням значення тактової частоти на кількість паралельно виконуваних операцій.

Складна архітектура сучасних високопродуктивних процесорів – суперскалярна та суперконвеєрне оброблення даних, багаторівнева пам'ять тощо призводять до того, що характеристики швидкодії на рівні внутрішніх пристроїв суттєво залежать від програми та даних.

У світовій практиці набуло широкого розповсюдження використання наборів задач (тестів), характерних для визначеної області застосування обчислювальної техніки, для оцінки продуктивності комп'ютерів. Час, необхідний для вирішення кожної задачі з набору, складає основу для обчислення індексу продуктивності, який є відносною оцінкою.

Першу групу тестів складають компанії-виробники для попередньої оцінки. Головна їх особливість полягає в тому, що вони орієнтовані на порівняння обмеженої кількості комп'ютерів, які часто відносяться до одного сімейства. Прикладом такої оцінки для МП з архітектурою *x*86 компанія *Intel* запропонувала індекс продуктивності *iCOMP* (*Intel Corporative Microprocessor Performance*), а в якості еталонного прийнятий процесор *I*80486 *SX*-25, для якого індекс дорівнює 100. Індекс *iCOMP* визначається при виконанні суміші операцій, яка складається з 67% операцій над 16-розрядними цілими, 3% операцій над 16-розрядними числами з плаваючою точкою, 25% – над 32-розрядними цілими та 5% – над 32-розрядними числами з плаваючою точкою й оцінює тільки продуктивність мікропроцесора, а не системи.

Стандартні тести, орієнтовані на порівняння широкого спектра комп'ютерів, складаються незалежними експертами або групами, об'єднуючими потужних виробників комп'ютерів. Це виключає орієнтацію тестів на конкретного виробника. Так, для оцінки серверів, які оброблюють трансакції у реальному часі, використовується набір тестів *TPP–C* (*Transaction Processing Performance Council*), а продуктивність оцінюється кількістю трансакцій, які виконуються за хвилину.

Існують також тести, які орієнтовані на вибір комп'ютерів та програмного забезпечення, найбільш придатних для вирішення певного кола задач. Такі тести дають найбільш точні оцінки продуктивності для конкретного класу додатків.

Найбільшої популярності набули пакети тестових програм компанії *SPEC* (*Standard Performance Evalution Corporation*) – *SPEC* 89, *SPEC* 92 тощо. Задачі, які входять у тестові пакети, вражають своєю різноманітністю і торкаються широких областей наукової та прикладної діяльності: задача з теорії мереж, інтерпретатор мови *Lisp*, *Unix* – утиліта пакування тестового файла 1 Мбайт, який стискається у 20 разів, моделювання керування рухом робота з використанням відеосистеми, моделювання вуха людини тощо. В останні версії тестів були включені задачі для оцінки продуктивності процесора у багатозадачному режимі для однорідного навантаження (комп'ютер виконував багато копій одної програми), а результатом вимірювань був нормований загальний час виконання всіх копій.

Тестовий пакет *SYS Mark* 2007 вимірює продуктивність процесорів у складі ПК у реальних додатках. Тест вміщує сценарій, в якому моделюється підготовка веб-сайта, який навчає. Медіаконтент використовує *Adobe Illustrator CS*2, *Adobe Photoshop CS*2, *Macromedia Flash* та *Microsoft Power Point* 2003. За сценарієм виготовляється відеоролик з використанням нелінійного монтажу та різних ефектів. Ролики монтуються з різних джерел і вміщують статичні зображення.

Тестовий пакет *SYS Mark* 2007, *Productivity* модулює типову офісну роботу: використання *e-mail*, обробку даних, керування проектом і роботу з документами. Тест працює з додатками: *Microsoft Excel* 2003, *Microsoft Outlook* 2003, *Microsoft Power Point* 2003, *Microsoft Word* 2003, *Microsoft Project* 2003 та *WinZip* 10.0.

Тестовий пакет *SYS Mark* 2007, 3*D* присвячено створенню архітектурної презентації, яка вміщує фотореалістичне зображення об'єкта і ролик, де проектований будинок облітається літаком. У сценарії використовуються додатки *AutoDesk* 3*ds Max* 8 та *Sketch Up* 5.

**Контрольні питання:**

1 Що таке технічна продуктивність МП?
2 Як вона оцінюється?
3 Що таке реальна продуктивність МП?
4 Які Ви знаєте способи оцінки реальної продуктивності?
5 Які шляхи підвищення продуктивності МП Ви знаєте?

6 Які задачі входять до пакетів тестових програм оцінювання продуктивності МП?

**Контрольні питання підвищеної складності:**

1 Чому в пакети тестових програм залучають задачі оброблення зображень?

2 Які ще задачі Ви могли б запропонувати для використання у пакетах тестових програм?

# 8 ВИКОРИСТАННЯ СУЧАСНИХ МІКРОПРОЦЕСОРІВ У ТЕЛЕКОМУНІКАЦІЙНОМУ ОБЛАДНАННІ (Для поглибленого вивчення)

**Вхідний контроль:**

1 Які системи телекомунікацій Ви знаєте?

2 Які з них працюють на основі мікропроцесорів та мікропроцесорних систем?

Понад 50% сучасного мережного обладнання використовують персональні комп'ютери на основі процесорів фірми *Intel* та сумісних з ними або вбудовують їх у пристрої шлюзів, маршрутизаторів, інтегрованих платформ, систем комутації.

В усіх засобах зв'язку, будь-то квазіелектронні АТС, електронні АТС, цифрові системи комутації, шлюзи, маршрутизатори та інші мережні пристрої, інтегровані платформи тощо, використовуються багатопроцесорні системи як менш двопроцесорні, або такі, що складаються з двох комп'ютерів. Можливий розподіл між кількома різними процесорами в одній системі функцій керування та цифрового оброблення сигналів. Тоді це універсальний процесор та до 20 – 30 процесорів цифрового оброблення сигналів, як правило, фірм-виробників *Motorola, Analog Devices, Texas Instruments*.

Нижче наведені приклади застосування МП фірми *Intel* у телекомунікаційному обладнанні.

Централізовані керувальні системи складаються з центрального пристрою керування, периферійних пристроїв керування. Керування усіма процесами на вузлі комутації квазіелектронних АТС виконує двомашинний керувальний комплекс. Наприклад, система комутації "Квант-Е" використовує два персональних комп'ютери на МП *Intel* 486*DX*, один комп'ютер виконує усі функції керування в автоматичному режимі, а другий знаходиться у гарячому резерві. У синхронному режимі обидва комп'ютери синхронно виконують одні й ті ж самі функції і на певних станах оброблення порівнюють результати з метою підвищення надійності та достовірності оброблення викликів. У режимі розподілу навантаження обидва комп'ютери обслуговують виклики паралельно, що підвищує продуктивність приблизно в два рази.

Система комутації "Квант-Е" використовує двомашинну систему на базі системних плат *IBM PC* "*PCA*-6143 *P*" із процесором *Intel* 486*DX* з тактовою частотою не менше ніж 133 МГц, а останні моделі використовують процесори *Pentium*.

Багато фірм, які випускають шлюзи, використовують як керувальний та оброблюючий пристрій персональні комп'ютери на мікропроцесорах фірми *Intel* та *Intel*-сумісних. Так, фірма *Clarent* використовує ПК з двома процесорами *Pentium Pro* для оброблення, передавання та приймання телефонних дзвінків та факсів по мережі *TCP/IP*.

Фірма *Comdial* випускає шлюз *IP*-телефонії на базі кількох серверних пристроїв на 2-х або 4-х *Pentium Pro*, а кодування/декодування мовного сигналу здійснюється на *DSP*.

Шлюзи фірми *Franclin Telecom* використовують для кожного каналу *IP*-телефонії один кодек *G*.729 для стиснення мовних сигналів у 8 разів на *DSP*. У цілому шлюзи працюють під ОС *UNIX* на універсальному процесорі і мають 24 голосових канали.

*WebPhone Exchange (WGX) Server* компанії *Netspeak* переводить дзвінки зі звичайних телефонів у мережу *IP* та дзвінки з додатка *Webphone* у телефонну мережу. *WGX* має процесор *Pentium* (200 МГц) та працює з ОС *NT*. Дзвінок за допомогою *WGX* може бути спрямований з *WebPhone* на звичайний телефон і навпаки, з телефону на телефон, зі сторінки Web на звичайний телефон. Це потребує не менше двох серверів *WGX*, які підтримують кодек *Microsoft GSM*. *WGX* мають доступ до інформації про маршрути *WGX* через *Connection Server*. Система здатна реєструвати події у реальному часі завдяки серверу *NetSpeak Database Services* (*DBS*) і підтримувати управління викликами у реальному часі.

*Internet Gateway* компанії *Rockwell* дозволяє спілкуватися користувачам *Web* і *Internet* та співробітникам центру обслуговування без телефону або додаткової лінії при збереженні з'єднання у вузлах *Web*. Сервер *Internet Gateway* поставляється разом з додатком *WebPhone* і має апаратну платформу мультимедійного ПК на базі ОС *Windows*.

Фірма *VocalTec* випускає шлюзи *IP*-телефонії – *VocalTec Telephony Gateway Release* 3.1. Ця система на базі *NT* для організації моста між телефонною мережею та *Intranet*, *Internet* з підтримкою дзвінків з телефону на ПК та з *Web* на телефон. Після з'єднання зі шлюзом автоматичний секретар запитує в абонента телефонний номер адресата, після чого номер вводиться з клавіатури. Місцевий шлюз автоматично визначає, що виклик треба адресувати віддаленому шлюзу. Система надає також послугу інтерактивній голосовій відповіді, відправки факсів у реальному часі або з проміжним зберіганням. Шлюз використовує механізм автоматичного визначення для запобігання роз'єднання в результаті довгих періодів мовчання. Модуль *Surf&Call* для броузера *Web* дозволяє дзвонити з сервера *Web* на звичайний телефон. Користувачі можуть звертатися до голосової пошти за технологією *DTMF*, а віддалені користувачі ПК отримують зручний доступ до мережі через *VocalTec Internet Phone*. Ця технологія буде корисною корпораціям з кількома офісами, якщо вони надають додаткові платні послуги провайдерам *Internet*. Версія шлюзу 3.2 замінює магістральні лінії (одноступеневий набір номера). Шлюз забезпечує 8 ліній.

Шлюз Інтернет-телефонії фірми *Intertel* виконано у вигляді автономного сервера з окремими платами обробки голосового сигналу за протоколами *H*.323 та *G*.723.1. Він підтримує сторінки *Web* з кнопками з функцією „натисни, щоб поговорити". Система побудована на ПК, який працює під ОС *NT*, і вимагає процесора, який підтримує цю ОС, тобто не нижче *Pentium*.

Обладнання *IP*-телефонії фірми *Cisco* має архітектуру *AVVID* (*Architecture for Voice, Video and Integer Data*) і пропонує широку номенклатуру

обладнання для *IP*-телефонії, яке призначене для застосування у корпоративних мережах, створення серверів багатофункціональних систем цифрової телефонії, підключення її до мереж загального користування, надання сучасних сервісів для абонентів. Усі пропоновані системи можуть масштабуватись від кількох десятків до сотень тисяч абонентів, у тому числі віддалених один від одного географічно.

До базових пропозицій компанії можна віднести такі: *IP*-телефони з власною *IP*-адресою, які можна підключати до будь-якого *H*.323-сумісного пристрою та користуватись типовим ПЗ, наприклад, *Microsoft NetMeeting*.

Сервер *Call Manager* реалізує керування телефонними з'єднаннями, сервісами в системі адміністрування тощо. Сервери можуть об'єднуватись у кластери.

ПЗ *Call Manager* установлюється на ПК не нижче *Pentium* під *Windows NT*, підключений до *IP*-мережі, *IP*-телефон або програмно реалізований віртуальний телефон, і забезпечує такі можливості, як утримання дзвінка, переспрямування дзвінка, перехоплення дзвінка, ідентифікацію абонента, який викликає. Інтерфейс *SMDI* ПЗ *Call Manager* забезпечує інтеграцію з голосовою поштою, інтерактивними системами мовного зв'язку для складання звітності про телефонні послуги, ПЗ для білінга.

За обсягом продажу обладнання *IP PBX* на світовому ринку лідирує компанія *CISCO*. Вона проводить політику переходу до єдиної системи комунікацій за протоколом *IP*. ПЗ *CISCO Call Manager* реалізують функції телефонної офісної УАТС і працюють на відокремленому сервері під *Windows* 2000 з процесором не нижче *Pentium* III *Xeon*. Система може бути розподіленою або централізованою, масштабується, дозволяє швидко нарощувати мережу для передавання голосу, підмикати нові офіси, підтримувати систему уніфікованої обробки повідомлень *Unity* (інтегровану з *Microsoft Exchange* та *Lotus Notes*) тощо. *Версія Call Manager* 3.3.2 здатна обслуговувати до 30 тис. *IP*-телефонів, має розширену сигналізацію *Q.SIG* (протокол *SIP* поки не підтримується). *Call Manager* може використовуватись у поєднанні з традиційною УАТС або як окреме рішення. Кілька серверів під *Call Manager* можуть об'єднуватись у кластери (до 8 вузлів), сервер може знаходитись у клієнта, в оператора, здійснювати все керування *IP*-телефонією корпорації, адміністрування та налаштування мережі. Нова модульна *IP PBX ICS* 7750 з функціями підтримки адміністратора викликів, функцій голосової пошти та *IVR* на основі архітектури *AVVID CISCO* пропонує кілька десятків продуктів. Завдяки ОС *IOS CISCO* маршрутизатори можуть виконувати функції малих АТС та підтримують сотні *IP*-телефонів, адаптери *ATA*186, призначені для підключення до *Ethernet* аналогових телефонів, модемів, факсів.

Фірма *Siemens* представляє *IP BPX HiPath* серії 3000 та 5000 на ОС *Windows NT*, процесори не нижче *Pentium* на 250-300 користувачів та УАТС *Hico* з підтримкою IP.

Комунікаційна платформа 3300 *ICP* канадської компанії *MITEL NETWORKS Ltd*, основана на технології *IP*, призначена для середнього та корпоративного бізнесу з приватними офісами. Усі компоненти платформи

керуються через *Web*-браузери. Одна платформа 3300 *ICP* підтримує 100 *IP*-телефонних номерів, забезпечує голосовий зв'язок, сигналізацію, централізоване оброблення даних та системні телекомунікаційні ресурси. Контролер підключається до клієнтської локальної мережі та до *IP*-пристроїв за допомогою чотирьох портів комутатора 10/100 *Base T Ethernet*. Універсальний блок аналогових послуг 3300 *Mitel Network* вміщує 16 внутрішніх аналогових телефонних ліній та 4 зовнішніх міських телефонних лінії. Універсальний блок мережних послуг 3300 *Mitel Network* забезпечує *T*1 або *E*1 з'єднання і підтримує до двох інтерфейсів *T*1 або *E*1 на модуль. Протоколи, підтримувані *T*1 інтерфейсами: *T*1*CAS – Digital E TAM, Digital CO, Digital DID T*1 *CCS*. Первинна швидкість *ISDN* (4*ESS*, 5*ESS*, *DHS* 100, *DSM* 250, *N* 12, *N* 13), *XNET* через *PRI*, *DASSII, MSDN/DPNSS*. Протоколи, підтримувані *E*1 інтерфейсом: *Q.Sig, ISDN, XNET* через *PRI, DASSII, MSDN/DPNSS*. Блок мережних послуг *R*2*NSU* забезпечує з'єднання з міжстанційними трактами *R*2, використовуючими цифрову сигналізацію *MF*-*R*2. Один блок *R*2 *NSU* підтримує до двох трактів, додаткові тракти можуть додаватись до 3300 *ICP* при з'єднанні двох *NSU*. Блок *BRI NSU* забезпечує з'єднання для передачі голосу та даних через інтерфейси основного доступу (*BRI*) для *ISDN*. Один блок підтримує 15 *BRI U*-інтерфейсів. *IP*-консоль 5550 основана на базі ПК має високоякісний графічний інтерфейс (*GUI*). Вимоги до ПК, на якому інсталюється консоль, такі: процесор *Pentium*, ОС *Microsoft Windows* 2000 *Professional*, 128 MB доступної *RAM* 4 GB жорсткого диску тощо.

АТС системи Квант-Е використовує як центральний пристрій управління двомашинний комплекс на базі системних плат з процесором *Intel* 386*DX*-40 (або сумісних). Фірма *Micom* (відділення *Nortel*) розробила плату *Vo/IP Phone/Fax IP Gateway* – шлюз *IP*-телефонії. Базовими блоками шлюзу є голосові плати та комплект програмного забезпечення для конфігурації системи, керування викликами, налагодження, протоколів пріоритезації для маршрутизаторів тощо. *Micom* розробила ефективні методи подавлення пауз, завдяки яким пропускна здатність знижується до 7 кбіт/с для однієї розмови за рахунок зупину передавання, коли абонент робить паузи між словами. Досить розвинене програмне забезпечення *Micom* потребує дуже малих процесорних ресурсів: йому достатньо процесора *I*80486.

При виборі типу мікропроцесора або мікроконтролера для мережного пристрою певного призначення визначальним є наявність апаратної підтримки комунікаційних функцій та висока продуктивність. За рахунок наявності вбудованих функцій необхідна частота процесора може бути знижена. Для обробки пакетів у сучасних мережах важлива наявність у процесорі кешу 1- та 2-го рівнів, що забезпечить високу швидкість мережі.

Фірми *AMD* та *SUN* випускають сервери *Sun Fire* для центрів обробки даних на основі МП *AMD Opteron*, які є 64-розрядними. У табл. 8.1 наведені основні характеристики таких серверів.

У табл. 8.2 та 8.3 наводяться характеристики робочих станцій та серверних систем *Sun Fire*.

Таблиця 8.1 – Сервери *Sun Fire* для центрів обробки даних

| | *Sun Fire X*2100 | *Sun Fire X*2200 | *Sun Fire X*4100 | *Sun Fire X*4200 | *Sun Fire X*4500 | *Sun Fire X*4600 | *Sun Blade X*8400 | Sun Blade *X*8420 |
|---|---|---|---|---|---|---|---|---|
| Процесори *AMD* Opteron | 1 одноядерний або двоядерний (серії 100 або 1000) | До 2-х двоядерних | До 2-х одноядерних або двоядерних | До 2-х одноядерних або двоядерних | До 2-х двоядерних | До 8-ми одноядерних або двоядерних | До 4-х двоядерних | 4 двоядерних (серії 8000) |
| Тактова частота | До 3,0 ГГц | До 2,6 ГГц | До 3,0 ГГц | До 3,0 ГГц | До 3,0 ГГц | До 3,0 ГГц | До 2,6 ГГц | 2,4 ГГц, 2,6 ГГц, 2,8 ГГц |
| Кеш-пам'ять 2-го рівня | По 1 МБ на ядро | По 1 МБ на ядро | По 1 МБ на ядро | По 1 МБ на ядро | По 1 МБ на ядро | По 1 МБ на ядро | По 1 МБ на ядро | 1 МБ на ядро, кеш *L*2 |
| Оперативна пам'ять | До 8 ГБ | До 64 ГБ | До 16 ГБ | До 16 ГБ | До 16 ГБ | До 64 ГБ | До 64 ГБ | До 64 ГБ |
| Внутрішні диски | До 2-х дисків | До 2-х дисків | До 2-х дисків з *DVD-ROM* або до 4-х дисків без *DVD-ROM* | До 4-х дисків з *DVD-ROM* | До 48-ми дисків | До 4-х дисків | До 2-х дисків | До 2-х дисків |

Продовження табл. 8.1

| | *Sun Fire X*2100 | *Sun Fire X*2200 | *Sun Fire X*4100 | *Sun Fire X*4200 | *Sun Fire X*4500 | *Sun Fire X*4600 | *Sun Blade X*8400 | *Sun Blade X*8420 |
|---|---|---|---|---|---|---|---|---|
| Типи внутріш-ніх дисків | 80/250/500 ГБ 7200 об/хв, *SATA* | 250/500 ГБ 7200 об/хв, *SATA* | 36/73 ГБ 10000 об/хв, *SAS* | 36/73 ГБ 10000 об/хв, *SAS* | 250/500 ГБ 7200 об/хв, *SATA*, макс. 24 ТБ | 10000 об/хв, *SAS* | 73 ГБ 10000 об/хв., *SAS*; 80 ГБ 5400 об/хв, *SATA* | 73 ГБ/146 ГБ *SAS* або 80 ГБ *SATA*, “гаряча заміна” |
| З’єдну-вачі введення-виведення | До 2-х слотів *PCIe x*8 | До 2-х слотів *PCIe x*8; 2 слоти *PCI-X* | 2 слоти *PCI-X* | 9 слотів *PCI-X* | 2 64-розряд-них слоти *PCI-X* | 8 слотів *PCI-X*; 4 слоти *x*8 *PCIe*; 2 64-розряд-них слоти *PCI-X* 100 МГц | 6 інтерфейсів *PCI-Express* | 6 інтерфейсів *PCI-Express* |
| Віддалене керування | Стандартний вбудований сервісний процесор з підтримкою *IPMI* 2,0 | Вбудований *Sun Integrated Light Out* з повною підтримкою *KVMS* | Вбудований *Sun Integrated Light Out* з повною підтримкою *KVMS* | Вбудований *Sun Integrated Light Out* з повною підтримкою *KVMS* | *Sun Integrated Lights Out Manager, N*1 *System Manager* | Вбудований сервісний процесор *Sun Integrated Lights Out Manager* з повною підтримкою *KVMS* | *Sun Integrated Lights Out Manager* з повною підтримкою *KVMS* | *Sun Integrated Lights Out Manager* в стандартній конфігурації |

Закінчення табл. 8.1

| | *Sun Fire X*2100 | *Sun Fire X*2200 | *Sun Fire X*4100 | *Sun Fire X*4200 | *Sun Fire X*4500 | *Sun Fire X*4600 | *Sun Blade X*8400 | *Sun Blade X*8420 |
|---|---|---|---|---|---|---|---|---|
| Операційна система | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux Enterprise Server* та *Microsoft Windows* | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Microsoft Windows* та *VMware ESX Server* | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Microsoft Windows* | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Microsoft Windows* та *VMware ESX Server* | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux Enterprise Server* | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Microsoft Windows ma VMware GSX* та *Virtual Server* | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux Enterprise Server* та *Microsoft Windows* | *OS Solaris* 10, *Red Hat Enterprise Linux* 4, *SUSE Linux Enterprise Server* 9, *Windows Server* 2003 *EE* та *SE VMware ESX Server* 3.0.1 |

Таблиця 8.2 – Робочі станції *Sun Fire*

| | Робочі станції *Sun Ultra* 20 | Робоча станція *Sun Ultra* 40 |
|---|---|---|
| Процесори *AMD Opteron* | 1 одноядерний або двоядерний (серії 100 або 1000) | До 2-х одноядерних або двоядерних |
| Тактова частота | До 3,0 ГГц | До 3,0 ГГц |
| Кеш-пам'ять 2-го рівня | По 1 МБ на ядро | По 1 МБ на ядро |
| Оперативна пам'ять | До 8 ГБ | До 8 ГБ |
| Внутрішні диски | До 2-х дисків | До 4-х дисків |
| Типи внутрішніх дисків | 80/250/500 ГБ, макс. 1 ТБ, *SATA* | 80/250/500 ГБ, макс. 2 ТБ, *SATA* |
| З'єднувачі введення-виведення | До 2-х слотів *PCIe x*16; 2 слоти *PCIe x*1; до 4-х слотів *PCI* | 2 слоти PCI-X x16; 2 слоти *PCIe x*4; 2 стандартних слоти *PCI* |
| Графічна система | *PCI: ATI RageXL, ATI ES* 1000; *PCIe: NVIDIA Quadro NVS* 285; *FX* 540, *FX* 560, *FX* 1400, *FX* 1500, *FX* 3450, *FX* 3500 | *PCIe: NVIDIA Quadro NVS* 285 DDR2, *FX* 560, *PCIe/SLI-Ready: NVIDIA Quadro FX* 1500, *FX* 3500, *FX* 5500 |
| Програмне забезпечення | Попередньо установлені *Sun Studio, Sun Java Studio Enterprise, Sun Java Studio Creator* та *NetBeans IDE* | Має безкоштовну ліцензію на *Sun N*1 *Grid Engine* 6. Попередньо установлені *Sun Studio, Sun Java Studio Enterprise, Sun Java Studio Creator* та *NetBeans IDE* |
| Операційна система | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux, Microsoft Windows* та *Microsoft Windows Vista* | *Solaris* 10, *Red Hat Enterprise Linux, SUSE Linux, Microsoft Windows* |

Таблиця 8.3 – Серверні системи *Sun Fire*

| | Модульна система *Sun Blade* 8000 | Модульна система *Sun Blade* 8000 P |
|---|---|---|
| Серверні модулі | 10 серверних модулів *Sun Blade* 84x0 в один корпус | 10 серверних модулів *Sun Blade* 84x0 на шасі |
| Розміри | Висота 10 *RU*, 2 модуля *PCIe Express Module*, до 20 з'єднувачів на шасі. До 4-х модулів *PCIe Network Express* на шасі | 14 *U*, 2 модуля *Network Express* стандарту *PCIe*, загальною ємністю 40 портів введення-виведення на шасі або 120 портів введення-виведення на стійку |
| Віддалене керування | Модуль моніторингу корпусу (*Chassis Monitoring Module, CMM*) забезпечує можливості моніторингу корпусу та установлених у ньому компонентів; є можливість установлення модуля з надмірністю (подвійний *CMM*) для моніторингу з високою доступністю | Кожен сервер має вбудований сервісний процесор *ILOM* з підтримкою: *DMTF CLI* через *SSH*, графічний інтерфейс користувача на основі браузера по *HTTPS/HTTP*, *IPMI* 2.0, *SNMP v*1, *v*2 та *v*3, віддалений *KVM* по *Ethernet*, віддалені носії інформації по *Ethernet*, сумісні з ПЗ керування *Sun N*1 *System Manager* |
| Інтерфейси | Інтерфейс *DB*9 RS-232*C* (включаючи електронний ключ *DB*9 на *RJ*45), два порти *Gigabit Ethernet* для віддаленого керування | Інтерфейс *DB*9 *RS*-232*C* (включаючи електронний ключ *DB*9 на *RJ*45), два порти *Gigabit Ethernet* для віддаленого керування |
| Підтримувані протоколи | *SSH, SNMP v*1, *v*2 та *v*3, *IPMI*, *HTTPS*, віддалена клавіатура, відео, миша та система зберігання даних (*RKVMS*), інтерфейс командного рядка *DMTF/SMASH* для керування по мережі або з використанням послідовного порту | *SSH, SNMP v*1, *v*2 та *v*3, *IPMI, HTTPS*, віддалена клавіатура, відео, миша та система зберігання даних (*RKVMS*), інтерфейс командного рядка *DMTF/SMASH* для керування по мережі або з використанням послідовного порту |

**Контрольні питання:**

1 Які архітектурні особливості сучасних 32- та 64-розрядних мікропроцесорів фірми *Intel* зумовили їх широке застосування у телекомунікаціях?

2 З якою метою у шлюзах використовують МП фірми *Intel*?

3 Під якою операційною системою працюють шлюзи, які використовують МП фірми *Intel*?

4 Як використовуються ПК на МП фірми *Intel* у АТС системи Квант-Е?

5 Як використовує фірма *CISCO* МП *Pentium* III *Xeon*?

6 Як використовуються МП фірми *Intel* в обладнанні *IP*-телефонії?

**Контрольні питання підвищеної складності:**

1 В якому обладнанні використовуються двоядерні процесори *Opteron* фірми *AMD*?

2 Під якими операційними системами працюють робочі станції *Sun Fire*?

3 Які вимоги пред’являють до операційних систем, під якими працюють мультиядерні процесори?

## СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ДО Частини І 1-го МОДУЛЯ


1 Брэй Б. Микропроцессоры *Intel*: 8086/8088, 80186/80188, 80286, 80386, 80486, *Pentium, Pentium Pro Processor, Pentium* II, *Pentium* III, *Pentium* 4. Архитектура, программирование и интерфейсы. – [6-е изд.]; пер. с англ. – С.Пб.: БХВ – Петербург, 2005. – 1328 с.: ил.

2 Вычислительные системы, сети и телекомуникации / В. Л. Бройдо. – С.Пб.: Питер, 2002. – 688 с.: ил.

3 Мікропроцесорна техніка: підручник / [Якименко Ю. І. . Терещенко Т. О., Сокол Є. І. та ін.] за ред. Т. О. Терещенко. – [2-е вид. пер. і доп.]/. – ІВЦ “Вид–во “Політехніка”; “Кондор”, 2004. – 440 с.: іл.

4 Схемотехника электронных систем. Микропроцессоры и микроконтроллеры / [В. И. Бойко, А. Н. Гуржий, В. Я. Жуйкою и др.] – С.Пб.: БХВ-Петербург, 2004. – 464 с.: ил.

5 Корнеев В. В., Киселев А. В. Современные микропроцесоры. – М.: НОЛИДЖ, 1998. – 240 с.: ил.

6 Корнеев В. В., Киселев А. В. Современные микропроцесоры. – М.: НОЛИДЖ, 2000. – 320 с.: ил.

7 Ричард Григонис. Шлюзы *IP*-телефонии // *LAN*. – 1998. № 07–08. (*www.osp.ru*).

8 Руководство по архитектуре *IBM PC AT* / [Ж. К. Голенкова, А. В. Заблоцкий, М. Л. Марсахин и др.]; под общ. ред. М. Л. Марсахина. – Минск.: ООО “Консул”, 1992. – 949 с.: ил.

9 Каган Б. М. Электронные и вычислительные машины и системы: [учеб. пособие для вузов]. – [3-е изд., пер. и доп.]. – М.: Энергоатомиздат, 1991. – 592 с.: ил.

10 Цифровая и вычислительная техника: [учебник для вузов] / Э. В. Евреинов, Ю. Т. Бутыльский, И. А. Мамзелев и др.; Под ред. Э. В. Евреинова. – М.: Радио и связь. 1991. – 464 с.: ил.

11 Проектирование импульсных и цифровых радиотехнических систем: учеб. пособие для радиотехн. спец. вузов / [Ю. П. Гришин, Ю. М. Казаринов, В. М. Катипов и др.]; пд ред. Ю. М. Казаринова. – М: Высшая школа, 1985. – 340 с.: ил.

12 Брамм П., Брамм Д. Микропроцессор 80386 и его программирование; пер. с англ. – М.: Мир, 1990. – 448 с.: ил.

13 Калабеков Б. А. Микропроцессоры и их применение в системах передачи и обработки сигналов: учеб. пособие для вузов. – М.: Радио и связь, 1988. – 368 с.: ил.

14 Литовкин В. Ф., Митрофанов Ю. Н., Номоконов В. Н. Микропроцесорные системы с фиксированной разрядностью: учеб. пособие. – Одеса: ОЭИС им. А.С. Попова, 1988. – 72 с.: ил.

15 Литовкин В. Ф., Митрофанов Ю. Н. Цифровая и вычислительная техника: учеб. пособие. – Одеса, 1989. – 65 с.: ил.

16 Трой Д. Программирование на языке Си для персонального компьютера *IBM PC*; пер. с англ.– М.: Радио и связь, 1991. – 432 с.: ил.
17 Прикладная теория прикладных автоматов. [Валуйский В. Н., Каневский Ю. С., Пиневич М. М. и др.].– К.: Вища школа, 1987. – 375 с.: ил.
18 Мещерякова Э. В., Митрофанов Ю. Н. Комплексное задание и методические указания к его выполнению по дисциплине «Цифровая и вычислительная техника» для студентов 3-го курса специальности 2305, 2306 и 2307 (часть 1). – Одеса, 1990. – 27 с.: ил.
19 Лебедев О. Н. Микросхемы памяти и их применение. – М.: Радио и связь, 1990. – 234 с.: ил.
20 Потемкин И. С. Функциональные узлы цифровой автоматики. – М.: Энергоатомиздат, 1988. – 320 с.: ил.
21 Гук М. Ю. Аппаратные средства IBM PC. Энциклопедия. – [3-е изд.] – С.Пб.: Питер, 2008. – 1072 с.: ил.
22 *http://www.virt.ru/news/news.htm*. «Характеристики микропроцессоров».
23 *http://www.infocity.kiev.ua* . Курмаз М. «*Pentium* 4: результаты тестов».
24 *http://www.rim*2000.*com* . Толопунский С. «Обзор *AMD Duron* с ядром *Morgan*».
25 *http://www.rim*2000.*com* . «Процессор *Pentium*® II *Xeon*™».
26 *ttp://www.infocity.kiev.ua* . Холмогорок В. «Процессоры – краткий обзор».
27 *http://www.infocity.kiev.ua* . Самотин С. «Новые процессоры для мобильных компьютеров».
28 *http://www.infocity.kiev.ua* . Джок А. «Многопроцессорная обработка на кристалле».
29 *http://www.infocity.kiev.ua* . Дереза Д. «*Pentium* 3 - *Pentium* 4. Новый ход *Intel*».

# Частина ІІ ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРІВ ФІРМИ *INTEL*

# 9 ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРІВ ФІРМИ *INTEL*

## 9.1 Сегментування пам'яті мікропроцесорами

**Вхідний контроль:**

1 Чи використовують сегментування пам'яті 8-розрядні МП фірми *Intel*?
2 Поясніть, що є адреса комірки пам'яті?
3 Поясніть, що є вміст комірки пам'яті?

Мікропроцесори фірми *Intel* призначені для роботи, перш за все, у складі комп'ютерів, які працюють у мультизадачному режимі, та у складі багатопроцесорних систем, де усі процесори можуть звертатися до спільного пристрою пам'яті. Найбільш незахищеною підсистемою МПС у такому застосуванні є пам'ять, це і призвело до способу організації пам'яті з кількох сегментів, кожний з яких має своє функціональне призначення: сегмент кодів, сегмент даних, сегмент стека, кілька додаткових сегментів даних залежно від моделі МП. МП *I*8086 та *I*8088 можуть працювати тільки у так званому реальному режимі, адресуючи 1 Мбайт пам'яті. Мікропроцесорам моделей від *I*80286 до *Pentium* 4 є доступні два режими функціонування − реальний та захищений. Реальний режим дозволяє процесору, навіть якщо це *Pentium* 4, адресувати також тільки 1 Мбайт пам'яті. При включенні живлення комп'ютера та при його повторному пуску всі моделі мікропроцесорів фірми Intel починають працювати саме у реальному режимі.

Таким чином, МП 8086 адресує пам'ять 1 Мбайт, яка являє собою масив з $2^{20}$ 8-розрядних комірок. У пам'яті можуть зберігатись як байти, так і двобайтові слова. Слова розміщуються у двох сусідніх комірках пам'яті за принципом *little endian* − старший байт розміщується за старшою адресою, а молодший − за молодшою. Адресою слова вважається адреса його молодшого байта. Організація пам'яті, коли кожній адресі відповідає вміст однієї комірки пам'яті, називається **лінійною**. У реальному режимі адреса пам'яті є комбінацією сегментної адреси та зміщення − двох логічних адрес. Сегментна адреса, задана одним із сегментних регістрів, визначає початок сегмента пам'яті обсягом 64 кбайт. Зміщення задає положення комірки пам'яті усередині сегмента у байтах відносно до початку сегмента, тобто адресування усередині сегмента є лінійним. Складові лінійної адреси − сегмент та зміщення − часто записують парами через двокрапку: *CS:IP, SS:SP*.

Розмір сегмента становить 64 кбайти. Сегменти можуть бути суміжними, розділеними, перекриватися повністю або частково.

Початкова адреса, тобто найменша адреса у цьому сегменті, вміщується у відповідні сегментні регістри, може установлюватися робочою програмою та завжди починається з 16-байтової межі, наприклад, 0000*H*, 7000*H*.

Сегментні регістри мають 16 розрядів і доступні програмно через команди пересилань типу *MOV* та *POP*. Вони мають цільове призначення і не

можуть виконувати функції регістрів загального призначення в арифметичних і логічних операціях.

Сегментний регістр команд *CS* – керувальний регістр, що вказує сегмент, який вміщує адресу поточно виконуваної команди. Його вміст може бути змінений тільки при виконанні команд переходів, зверненні до підпрограм, поверненні з підпрограм або перериваннях із зазначенням іншого сегмента команд.

Сегментний регістр даних *DS* – керувальний регістр, що вказує сегмент, який вміщує програмно змінювані таблиці, масиви даних та константи. Така організація, за якої дані групуються разом в окремому сегменті і входять до сегмента, що зберігає коди команд, полегшує програмування. Усі прямі та непрямі засоби адресування даних, що використовують регістри *BX, SI* або *DI*, визначаються відносно регістра *DS*.

Сегментний регістр стека *SS* – керувальний регістр, вміст якого вказує початок стекової пам’яті (вершину). Вміст регістрів *SP* або *BP* може використовуватись для зазначення відносних адрес переходів, які формуються за участю цих регістрів, або адресу відносно регістра *SS*.

Сегментний регістр додаткового сегмента даних *ES*, а також сегментні регістри додаткових сегментів у МП *I*80386 та старших моделях *FS* та *GS*, – керувальні регістри. Вміст цих регістрів вказує на початок області пам’яті, яка зазвичай використовується для запам’ятовування проміжних результатів, тобто робочої області пам’яті. За необхідності використання додаткових сегментних регістрів використовуються команди з префіксом заміни сегмента, наприклад, команда

```
MOV AX,ES:[100]
```

звертається до додаткового сегмента даних, початкова адреса якого знаходиться у регістрі *ES*.

Вказівник команд *IP* – регістр, що виконує роль лічильника команд. Його вміст вказує адресу наступного байта команди у сегменті пам’яті, що визначається вмістом регістра сегмента команд *CS*. Вміст регістрів *IP* та *CS* однозначно визначає адресу байта в усьому адресному просторі. Вміст регістра IP можна змінити тільки при виконанні команд переходів, виклику та поверненні з підпрограм та перериваннях.

Мікропроцесор *I*8086 є 16-розрядний, з 16-розрядними внутрішніми шинами адрес та даних, 16-розрядними регістрами та АЛП. З метою підвищення частоти функціонування МП, яка обмежується також кількістю виводів ВІС, зовнішня шина адреси *AD*0…*AD*15 даних є мультиплексована і також 16-розрядна. Окремо є 4 виводи ВІС *A*16...*A*19, які у багатопроцесорному максимальному режимі дозволяють виводити адресні розряди *A*16...*A*19 або інформацію про використовуваний у програмі на цей час сегментний регістр (*ES, SS, CS, DS*) та стан прапорця *IF* дозволу переривань.

Сегментні регістри мають довжину 16 розрядів, зміщення адреси даного у сегменті відносно початкової адреси сегмента (так звана ефективна або

виконавча адреса) має також 16 розрядів, а фізична адреса, тобто адреса комірки пам'яті в єдиному адресному просторі МП системи у реальному режимі, що видається на шину адреси, вміщує 20 розрядів, бо треба адресувати 1 Мбайт пам'яті. Фізична адреса усіх МП сім'ї *INTEL* у реальному режимі складається з суми вмісту одного з сегментних регістрів, зсунутого вліво на чотири розряди, та ефективної адреси або *IP, SP, BP* (рис. 9.1)

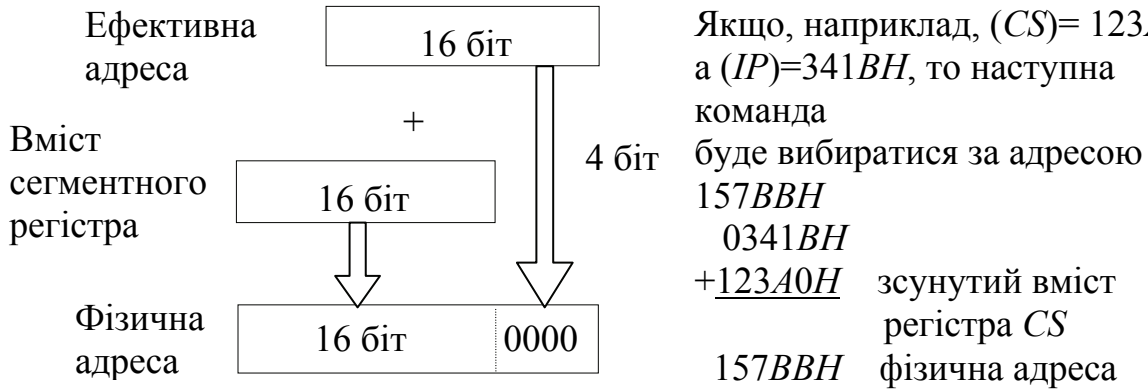



Рисунок 9.1 – Принцип формування 20-розрядної фізичної адреси

Формування фізичної адреси в усіх моделях до *I*80386 здійснюється за схемою, що подана на рис. 9.2.

Ефективна адреса є сумою вмісту вказаного у команді регістра *BX, BP, DI, SI* та поданого безпосередньо у формі числа зміщення, або сумою деякої бази, що зберігається у базовому регістрі *BX* або *BP*, вмісту індексного регістра *DI* або *SI* та зміщення.

Перенесення зі старшого біта, яке може виникнути при обчисленні фізичної адреси комірки пам'яті, ігнорується. Це призводить до кільцевої організації пам'яті: за коміркою з максимальною адресою *FFFFF* йде комірка з нульовою адресою. Аналогічну кільцеву структуру має і кожний сегмент. Таким чином, фізична адреса комірки пам'яті однозначно визначається 20-бітовим числом у діапазоні 0 − *FFFFF* у просторі пам'яті 1 Мбайт. Коди команд та даних можна вільно розміщувати за будь-якою адресою, що дозволяє економити пам'ять завдяки її ущільненню. Але для підвищення швидкості виконання програми доцільно розміщувати слова даних за парними адресами, тому що МП передає такі слова за один цикл шини. Для передачі слова з непарною адресою потрібні два цикли шини, що знижує продуктивність МП. Шинний інтерфейс ініціює необхідну для вибирання слова кількість звернень до пам'яті автоматично, тобто дворазове звернення до пам'яті не потребує спеціальних вказівок у програмі. Вказівник стека *SP* треба завжди ініціалізовувати на парну адресу, тому що обмін з пам'яттю відбувається тільки словами. Команди завжди вибираються словами за парними адресами. Виняток становить перша вибірка після передачі керування за непарною адресою, коли вибирається один байт. Вирівнювання команд за парною адресою не впливає на продуктивність МП, тому що при заповненні конвеєра (черги команд) вони поділяються на байти.

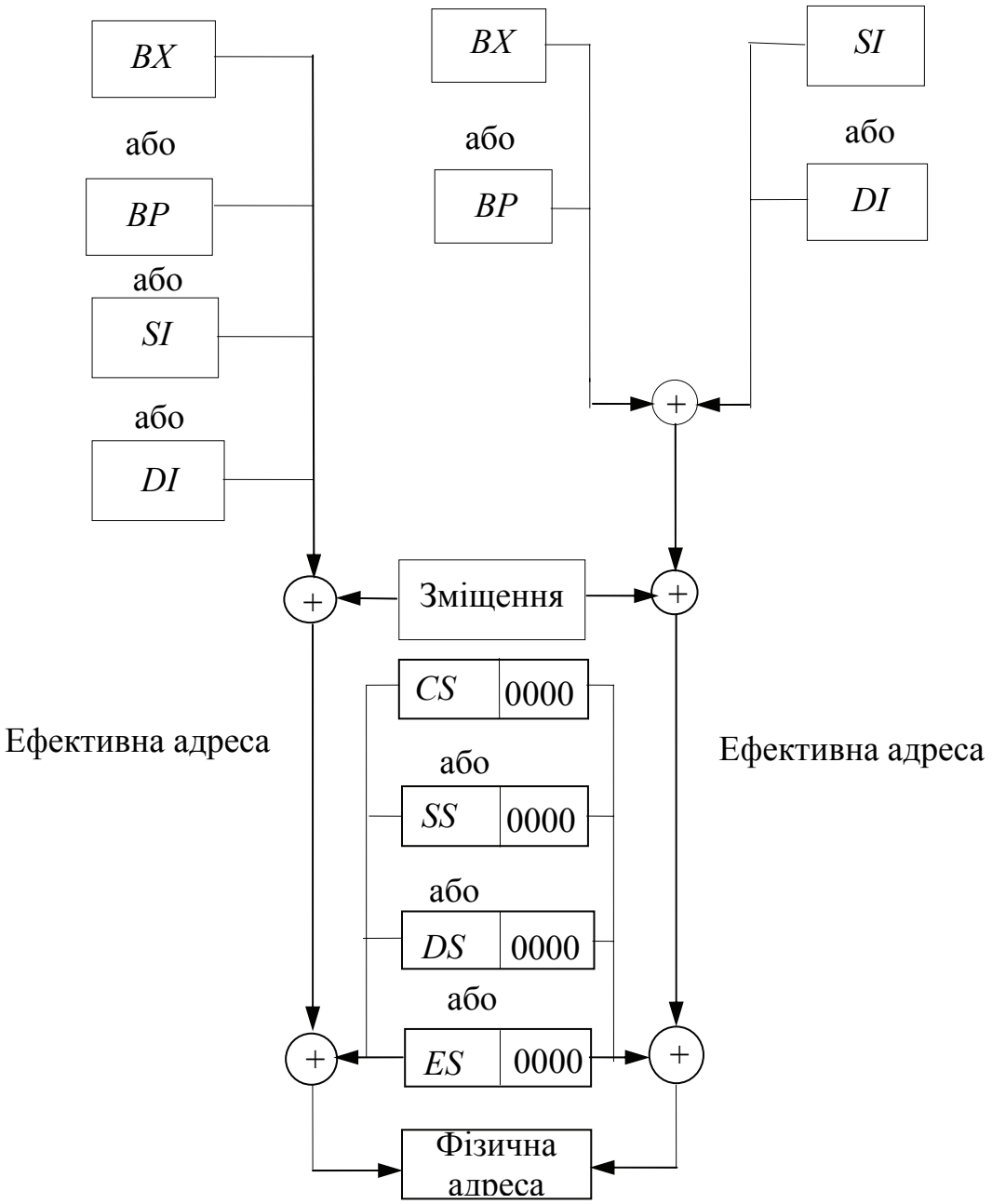



Рисунок 9.2 – Способи формування фізичної адреси

Сегментна організація пам’яті характеризується тим, що програмно доступною є не вся пам’ять, а лише організовані сегменти.

У сучасних 32-розрядних МП розмір сегмента може сягати 4 Гбайт.

У сегментах команд розміщено коди команд і при виконанні програми з цього сегмента інформацію можна тільки зчитувати, що є одним зі способів захисту кодів команд. У решті сегментів розміщуються дані, які у процесі виконання програми можна як зчитувати, так і записувати, тобто сегменти мають різні права доступу, а також різне змістовне призначення. При адресуванні операндів у командах з різними способами адресування не треба вказувати кожного разу вміст відповідного сегментного регістра, а оперувати тільки ефективною адресою, що скорочує команду та економить пам’ять.

Механізм сегментування пам’яті, незважаючи на складність, спрощує розподіл пам’яті при завантаженні програм. Програми, написані для роботи у реальному режимі, можуть працювати також у захищеному режимі старших моделей МП фірми *Intel*. Код та дані при роботі ПК або МПС з багатозадачною операційною системою можуть бути розміщені у довільних областях пам’яті без змін у програмі, що дозволяє використовувати програмне забезпечення на ПК з різною конфігурацією пам’яті.

Оскільки у межах сегмента дані адресуються зміщенням, сегмент можна пересувати у будь-яку область адресного простору без корекції зміщень: програма копіюється як єдиний блок пам’яті у нову область, після чого коректуються значення сегментних регістрів.

Кожна програма може складатись з будь-якої кількості сегментів, але безпосередній доступ без перевантаження сегментних регістрів вона має тільки до сегментів коду, даних, стека та одного додаткового сегмента даних для I8086. Програма ніколи не знає, за якими фізичними адресами будуть розміщені її сегменти, це вирішує операційна система.

Механізм керування пам’яттю є апаратний, тобто у реальному режимі програма не може сама сформувати фізичну адресу пам’яті, яка видається на шину адреси. Результат використання цього механізму є такий:

– компактність зберігання адреси у машинній команді;
– гнучкість способів адресування;
– захист адресних просторів задач у багатозадачній системі;
– підтримка віртуальної пам’яті при роботі у захищеному режимі.

Старші моделі МП фірми *Intel*, починаючи з *I*80386, підтримують апаратно кілька моделей використання оперативної пам’яті:

– сегментовану модель;
– сторінкову модель, яку можна розглядати як розташовану поверх сегментованої моделі; у рамках цієї моделі оперативна пам’ять поділяється на блоки фіксованого розміру 4 кбайт; сторінкова модель використовується при організації віртуальної (надаваної) пам’яті, що дозволяє операційній системі використовувати для роботи програм простір пам’яті більший, ніж обсяг фізичної оперативної пам’яті (до 4 Тбайт для МП *Pentium*).

Недоліками сегментної організації пам’яті є:

– сегменти безконтрольно розміщуються з будь-якої адреси, кратної 16 (вміст сегментного регістра апаратно зміщується ліворуч на 4 двійкових розряди);
– максимальний розмір сегмента для МП *I*8086 становить 64 кбайт;
– сегменти можуть перекриватись з дозволу програміста.

Ці недоліки зумовили застосування захищеного режиму у старших моделях МП.

## 9.2 Способи адресування операндів МП фірми *Intel*

### Регістрове адресування операндів

Найбільш швидко виконуваним є регістрове адресування, за якого обидва операнди знаходяться у регістрах, які вказуються у команді. Вміст регістра-джерела передається (копіюється) у регістр-приймач, який у команді вказується першим. Наприклад:

```
MOV AX,CX  ; Копіювання 16-розрядного вмісту регістра CX в
           ; акумулятор AX
```

Вміст регістра-приймача завжди змінюється на вміст регістра-джерела, який не змінюється.

### Безпосереднє адресування операндів

При безпосередньому адресуванні операнд-джерело вказується безпосередньо у команді у вигляді 8- або 16-розрядного даного. Операнд-приймач може бути регістром, або коміркою пам'яті, адресованою будь-яким способом, але не може бути числом. Наприклад:

```
MOV BL,11H  ; 8-розрядне дане записується у регістр BL
```

При безпосередньому адресуванні знак константи поширюється до 8- або 16-розрядного регістра, а від'ємні числа записуються у регістр або комірку пам'яті у доповнювальному коді.

Припустимо, що до виконання команди

```
MOV CL,12H  ; Завантаження у CL константи 12H
```

у регістрі *CL* було записане число *FFH* (11111111*B*). Після виконання цієї команди у регістр *CL* буде записано число 12*H* (00010010*B*).

Результатом виконання команди

```
MOV CH,-30  ; Завантаження у CH константи -30D
```

буде запис у регістр *CH* числа *E*2*H* (11100010B) – числа -30*D* у доповнювальному коді.

Покажемо на прикладі команди з безпосереднім адресуванням як розміщуються байти команди у сегменті кодів. При уведенні команди

```
MOV AX,7000H
```

наприклад, у сегмент кодів, який починається з адреси 7000H (CS) та зміщення 0100H три байти команди у машинних кодах будуть розміщені у трьох послідовних комірках пам’яті з адресами

| Адреса | Машинні коди | |
|---|---|---|
| 7000:0100 | *B*8 | ; Код операції “пересилати до *AX*” |
| 7000:0101 | 00 | ; Молодший байт |
| 7000:0102 | 70 | ; Старший байт операнда безпосередньо |
| 7000:0103 | | ; Вільна комірка пам’яті для прийому коду |
| | | ; операції наступної команди |

Запис команди у сегмент кодів можна також проілюструвати за допомогою фрагмента пам’яті (рис. 9.3):

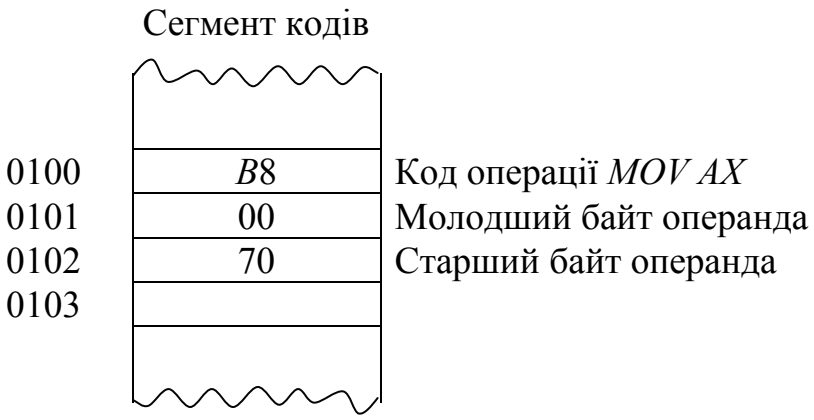



Рисунок 9.3 – Розташування байтів команди *MOV AX*,7000*H* у сегменті кодів

Таким чином, обсяг пам’яті, яку займає програма, є сумою байтів, займаних усіма командами цієї програми.

**Пряме адресування**

При прямому адресуванні ефективна адреса операнда у сегментах даних вказується прямо у команді і розміщується у квадратних дужках. Початкова адреса сегмента даних, яка заздалегідь має бути розміщена у відповідному сегментному регістрі даних *DS* або *ES*, участь у формуванні ефективної адреси не бере. На рис. 9.4 показано фрагмент сегмента даних, в якому у комірках пам’яті з ефективними адресами 0010*H* та 0011*H* зберігаються відповідно дані *BBH* та *AAH*, а з адресами 0020*H* та 0021*H* – дані *CCH* та *DDH*.

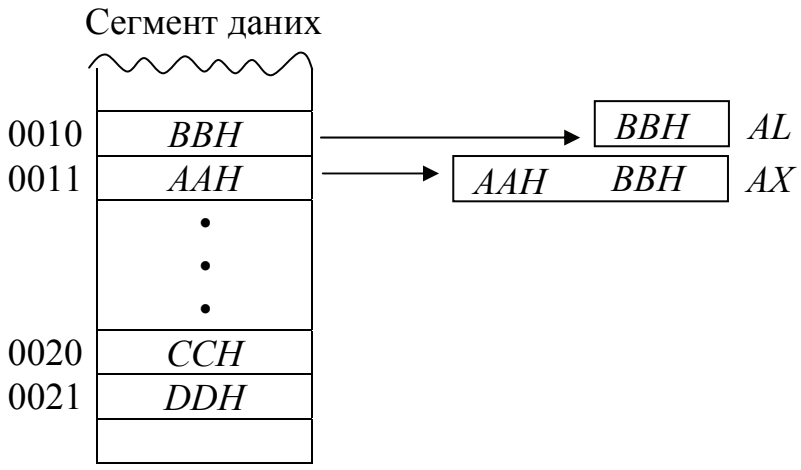



Рисунок 9.4 – Пряме адресування

Після виконання команди

```
MOV AL,[0010H] ; Завантаження в AL даного з комірки пам'яті з адресою
               ; 0010H
```

у регістр *AL* буде завантажене дане *BBH*.

Після виконання команди

```
MOV AX,[0010H] ; Завантаження в AX слова з комірок пам'яті з адресами
               ; 0010H та 00011H
```

у регістр *AX* буде завантажено слово *AABBH* відповідно до принципу зберігання у пам'яті слів – little endian.

Після виконання фрагмента програми

```
MOV AX,[0010H]  ;
MOV [0020H],AX  ; Завантаження комірок пам'яті з адресами 0020H та
                ; 0021H з акумулятора AX
```

у комірках пам'яті з адресами 0020*H* та 0021*H* будуть записані нові дані: *BBH* та *AAH* відповідно.

**Непряме регістрове адресування**

При непрямому регістровому адресуванні ефективна адреса вміщується у будь-якому базовому або індексному регістрі: *BX, BP, SI, DI*. Непряме регістрове адресування стає у нагоді при обробленні елементів масивів. На рис. 9.5 показані два масиви, які розташовані у сегменті даних, починаючи

відповідно з адрес 0010*H* – перший та 0020*H* – другий, елементи яких становлять відповідно числа 12*H,* 34*H*, 56*H*, 78*H* та *A*1*H*, *A*2*H*, *A*3*H*, *A*4*H*.

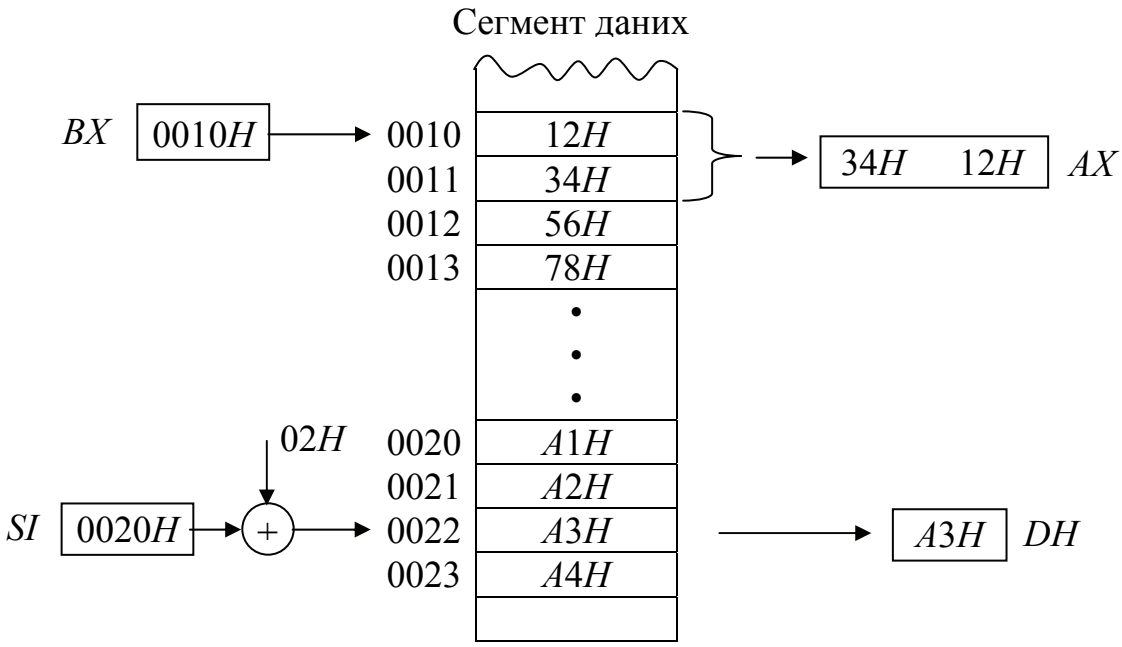



Рисунок 9.5 – Непряме регістрове адресування

Після виконання фрагмента програми

```
MOV BX,0010H
MOV AX,[BX]
```

у регістр *AX* з двох комірок пам’яті з адресами (*BX*) та (*BX*+1) будуть передані два байти: 12*H* у регістр *AL* та 34*H* у регістр *AH*.

Після виконання фрагмента програми

```
MOV DI,0022H
MOV DH,[DI]
```

у регістр *DH* буде записане дане *A*3*H*.

Спосіб непрямого регістрового адресування використовується для послідовного доступу до елементів масивів при нарощуванні або зменшенні вмісту базових або індексних регістрів. Якщо елементи масиву трактуються програмістом як двобайтові (перший фрагмент), то вміст базового або індексного регістра треба змінювати на 2, а якщо як однобайтові (другий фрагмент), то на 1.

При непрямому регістровому адресуванні при зазначенні адреси комірки пам’яті до вмісту базового або індексного регістра може додаватись зміщення.

Ефективна адреса підраховується як алгебраїчна сума вмісту вказаного у команді регістра та зміщення. Результатом виконання фрагмента програми

```
MOV SI,0020H
MOV DH,[SI+0002H]
```

буде запис у регістр *DH* однобайтового елемента другого масиву з комірки пам’яті з ефективною адресою

$$EA = 0020H + 0002H = 0022H,$$

тобто даного *A3H*.

Такий спосіб адресування називається ще **непрямим регістровим адресуванням зі зміщенням**.

**Адресування за базою**

За цим способом ефективна адреса операнда обчислюється шляхом підсумовування вмісту базових регістрів *BX* або *BP* та зміщення – 8- або 16-розрядного числа. Регістри бази використовуються для доступу до структурованих записів даних (масивів), розташованих у різних областях пам'яті. Базова адреса масиву, тобто його початкова адреса у сегменті, розміщується у базовому регістрі, а доступ до її окремих елементів здійснюється за їх зміщенням відносно базової адреси. Для доступу до елементів різних масивів, розташованих на однаковій відстані від бази, достатньо змінити вміст базового регістра. Припустимо, що у сегменті даних, починаючи з ефективних адрес 0010 та 0020 розташовані відповідно два одновимірних масиви з елементами (рис. 9.6): 12*H*, 34*H*, 56*H*, 67*H*, 89*H* та *A*1*H*, *A*2*H*, *A*3*H*, *A*4*H*, *A*5*H*, *A*6*H*. За необхідності переслати у регістр *AL* четвертий елемент першого масиву, а у регістр *AH* – четвертий елемент другого масиву треба виконати такий фрагмент програми:

```
MOV BX,0010H       ; Запис у BX базової адреси першого масиву
MOV AL,[BX+04H] ; Пересилання до AL четвертого елемента першого
                   ; масиву (89H)
MOV BX,0020H       ; Запис у BX базової адреси другого масиву
MOV AL,[BX+04H] ; Пересилання до AH четвертого елементу другого
                    ; масиву (A5H)
```

Ефективна адреса четвертого елемента першого масиву становить 14*H*, а четвертого елемента другого масиву – 24*H*. Слід зазначити, що наступні команди:

```
MOV AX,[BX]+4
MOV AX,4[BX]
MOV AX,[BX+4]
```

з точки зору підрахування ефективної адреси операнда у сегменті даних є ідентичними і відрізняються тільки формою завдання адреси.

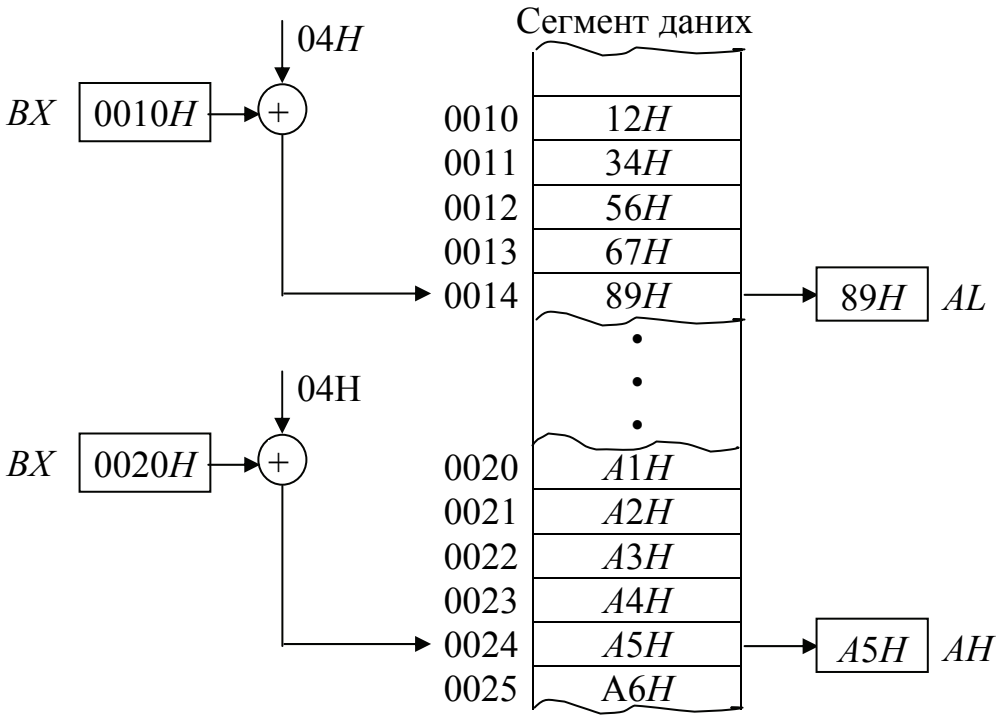



Рисунок 9.6 – Адресування за базою

Коли елементи масиву трактуються як двобайтові, як у наведених командах, то у регістр *AX* буде завантажено при значенні вмісту *BX*, який дорівнює 0020*H*, другий елемент другого масиву, тобто *A6A5H*.

**Пряме адресування з індексуванням**

При прямому адресуванні з індексуванням ефективна адреса обчислюється як сума зміщення (прямої адреси) та вмісту індексного регістра. Цей тип адресування зручний для доступу до елементів масиву у сегменті даних, коли прямо адресується початок масиву, а вміст індексного регістра вказує на потрібний елемент (рис. 9.7). Фрагмент програми:

```
MOV SI,0002H
MOV AX,[SI+20H]
```

призведе до пересилання з сегмента даних у регістр *AX* двобайтового даного *ACBAH* з комірок пам'яті з адресами [20*H*+2*H*] та [20*H*+2*H*+1*H*].

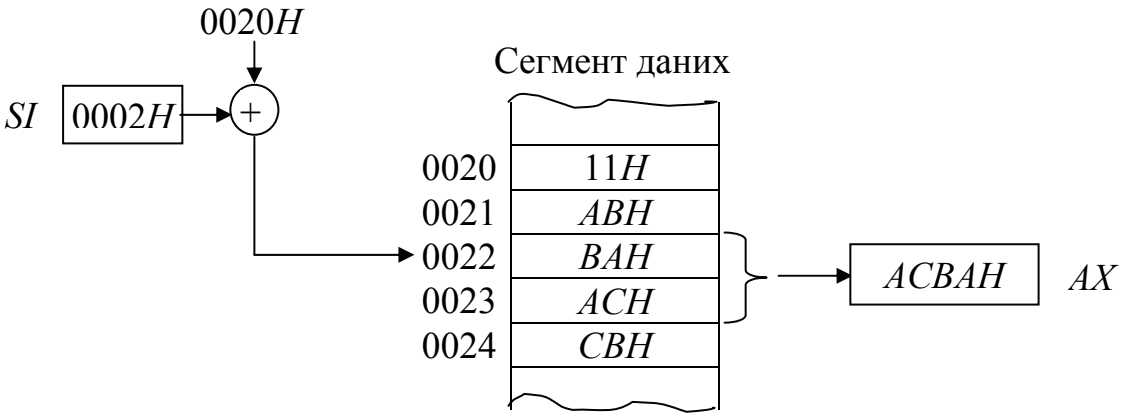



Рисунок 9.7 – Пряме адресування з індексуванням

**Адресування за базою з індексуванням**

При адресуванні за базою з індексуванням ефективна адреса обчислюється як сума значень базового регістра, індексного регістра і, можливо, зміщення. Такий спосіб адресування зручно використовувати для адресування двовимірних масивів, коли базовий регістр вміщує початкову адресу масиву, а значення зміщення та вміст індексного регістра вибирають елементи у рядку та стовпчику. Наступні команди є ідентичні:

```
MOV AX,[BX+2+DI]  ; Пересилання у регістр AX
                  ; елемента, адреса якого
MOV AX,[DI+BX+2]  ; визначається вмістом регістра DI
                  ; з третього
MOV AX,[BX+2][DI] ; стовпчика масиву, початкова адреса
                  ; якого
MOV AX,[BX][DI+2] ; знаходиться у регістрі BX
```

Команди з таким типом адресування виконуються найдовше.

**Непряме адресування з масштабуванням**

У старших моделях мікропроцесорів фірми *Intel* (*I*80386 та вищих) використовується також непряме адресування з масштабуванням. Зміщення задається двома 32-розрядними регістрами (базовим та індексним). Вміст індексного регістра множиться на масштабний множник 1, 2, 4 або 8. Масштабний множник використовується для адресування елементів масивів слів, множник 4 – для обробки подвійних слів, множник 8 – для доступу до елемента масиву 8-байтових даних. Наприклад, команда

```
MOV EAX,[EBX+4*ECX]
```

копіює в *EAX* подвійне слово з сегмента даних, розташоване, починаючи з адреси 4**ECX*+*EBX*.

**Контрольні питання:**

1 З якою метою організовується сегментування основної пам'яті МП сім'ї *Intel*?

2 Який обсяг пам'яті можуть адресувати МП сім'ї *Intel* у реальному режимі?

3 Як адресується сегмент основної пам'яті МП сім'ї *Intel*?

4 Яке призначення за умовчанням має кожний із сегментних регістрів мікропроцесорів фірми *Intel*?

5 Як обчислюється фізична адреса комірок основної пам'яті МП сім'ї *Intel*?

6 Який спосіб адресування у командах реалізується найшвидше?

7 Які особливості має безпосереднє адресування операндів?

8 В яких випадках операнд у командах адресується прямо?

9 Де вміщується ефективна адреса операнда при непрямому регістровому адресуванні?

10 Які способи адресування використовуються для звернення до елементів масиву?

11 Які особливості має непряме адресування з масштабуванням?

12 Які переваги має сегментування пам'яті порівняно з лінійною організацією?

13 Які недоліки має сегментування пам'яті?

14 Виконати фрагмент програми:

1) Задати у пам'яті, починаючи з ефективних адрес 0010*H* та 0020*H*, два масиви по 10*H* елементів кожний.
2) Завантажити сегментний регістр даних початковою адресою 7000*H*.
3) Завантажити індексний регістр *SI* константою 04*H*.
4) Використовуючи адресування з індексуванням, завантажити регістри *DH* та *CX* третім елементом першого масиву. Вказати вміст регістрів після завантаження.
5) Використовуючи адресування з індексуванням, завантажити регістри *CL* та *DX* другим елементом другого масиву. Вказати вміст регістрів після завантаження.

**Контрольні питання підвищеної складності:**

1 У пам'яті розташований двовимірний масив 3 х 3 елементи у вигляді квадратної матриці; використовуючи непряме регістрове адресування зі зміщенням та індексуванням, вкажіть адресу центрального елемента масиву?

2 Покажіть, як будуть змінюватись адреси елементів цього масиву під час руху вдовж головної діагоналі матриці.

## 9.3 Мова програмування Асемблер-86

**Вхідний контроль:**

1 Що входить до поняття “архітектура мікропроцесорів”?

2 В чому полягає різниця між мовами програмування високого і низького рівнів?

3 Як можуть будуть подані дробові числа в обчислювальній системі?

4 В яких кодах подаються числа зі знаком в обчислювальній системі?

5 Наведіть регістрову модель 16-розрядного МП фірми *Intel*.

6 Які сегменти існують в основній пам’яті МПС, побудованої на МП фірми *Intel*?

7 Які способи адресування операндів МП фірми *Intel* Ви знаєте?

8 Що називається стеком і як його організовано?

Кожна мікропроцесорна система на МП певної моделі має свою власну мову програмування – мову машинних команд (машинну мову) і функціонувати може безпосередньо лише під керуванням програми, яка написана цією мовою. Машинна мова є сукупністю двійкових кодів усіх операцій, які може виконувати певний процесор. Машинна мова є незручною для широкого використання, тому що потребує досить великих витрат часу для написання і налагодження таких програм. Більш широке поширення находять мови програмування, які не співпадають з машинною мовою і є зручнішими за неї при використанні. Усі мови програмування можливо поділити на дві групи: мови високого і низького рівнів (машинно-орієнтовані мови), що відбиває ступінь близькості цих мов до машинної.

Мова Асемблер – це машинно-орієнтована мова, яка дозволяє використовувати усі структурні особливості мікропроцесорної системи, що пов’язані з апаратними можливостями, набором машинних команд, складом периферійного обладнання тощо. Мова Асемблер – це символічне подання машинної команди у вигляді її мнемонічного зображення. Програмування на Асемблері вимагає знання способів подання й оброблення даних на рівні машинних команд, що забезпечується знанням різних систем числення та архітектури МПС. Мова Асемблер поєднує у собі переваги машинної мови і деякі особливості мов високого рівня. Асемблер є найбільш ефективною мовою програмування при розв’язанні задач розробки системного програмного забезпечення, розробки програм для обміну даними з нестандартним периферійним обладнанням (драйвери), програмуванням систем реального часу для керування технологічними процесами й обладнанням, а також для забезпечення роботи інформаційно-обчислювальних систем та ефективного використання можливостей (робота у захищеному режимі) і ресурсів МПС.

Мова Асемблер мікропроцесорів фірми *Intel* є досить розвиненою і гнучкою. До складу мови Асемблер-86 входять понад 100 базових команд, відповідно до яких генерується понад 3800 машинних команд; близько 20 директив, призначених для розподілення пам’яті, ініціалізації змінних, умовного асемблювання тощо. Також у ній передбачено використання засобів

структурування даних, що є характерним для мов програмування високого рівня.

Всі команди мови Асемблер можливо поділити за групами, відповідно їх функціональному призначенню: пересилання даних, арифметичні операції, логічні операції і зсуви, передачі керування, оброблення рядків даних і команди керування станом центрального процесора.

Сукупність команд і директив, що представляють текст програми, називаються **початковим модулем**. Початкові модулі створюються за допомогою текстового редактора і можуть зберігатися на будь-якому носії у вигляді початкового файла. Цей файл може мати будь-яку назву і розширення .asm. Асемблер перетворює початковий модуль в **об'єктний модуль**. Ця операція називається **трансляцією**, а програма, яка її виконує – **транслятором**. На етапі створення об'єктного модуля виконується перетворення команд Асемблера на машинні команди. У результаті роботи транслятора створюються: об'єктний модуль – файл, який має розширення *.obj*, файл лістинга, який має розширення *.lst* і файл перехресних посилань з розширенням *.crf*. Файл лістинга вміщує код Асемблера початкової програми, для кожної команди якої вказано машинний код і зміщення у кодовому сегменті, крім того, до цього файла входять таблиці з інформацією про мітки і сегменти, які використовуються у програмі і повідомлення про синтаксичні помилки. Імена файлів призначає програміст, вони можуть співпадати, а можуть бути різними.

Після створення об'єктного модуля він оброблюється за допомогою програми-укладача, яка генерує завантажувальний модуль. Це файл, який має розширення *.exe* і який може виконуватися МПС. Програма-укладач у завантажувальному модулі об'єднує об'єктні модулі, що входять до програми і підключає, за необхідності, бібліотечні модулі і замінює відносні адреси комірок пам'яті на абсолютні, з урахуванням області пам'яті, в яку модуль буде завантажено для виконання. Бібліотечні модулі – це об'єктні файли, які містять найбільш поширені програми. Ім'я завантажувального модуля також призначає програміст. Блок-схему алгоритму підготовки програми на мові Асемблер для її виконання показано на рис. 9.8.

Текст програми на мові Асемблер складається з операторів (речень) чотирьох типів:

- команди, які є символічними аналогами машинних команд МПС;
- макрокоманди – конструкції, які при трансляції програми замінюються на послідовність відповідних команд;
- директиви – інструкції транслятору з виконання певних дій;
- коментарі – пояснення з виконання команди або фрагмента програми, використовуються для документування і кращого розуміння змісту програми; коментар ігнорується Асемблером, тому він може бути написаний будь-якою мовою, і відокремлюється символом ; від іншої частини речення.

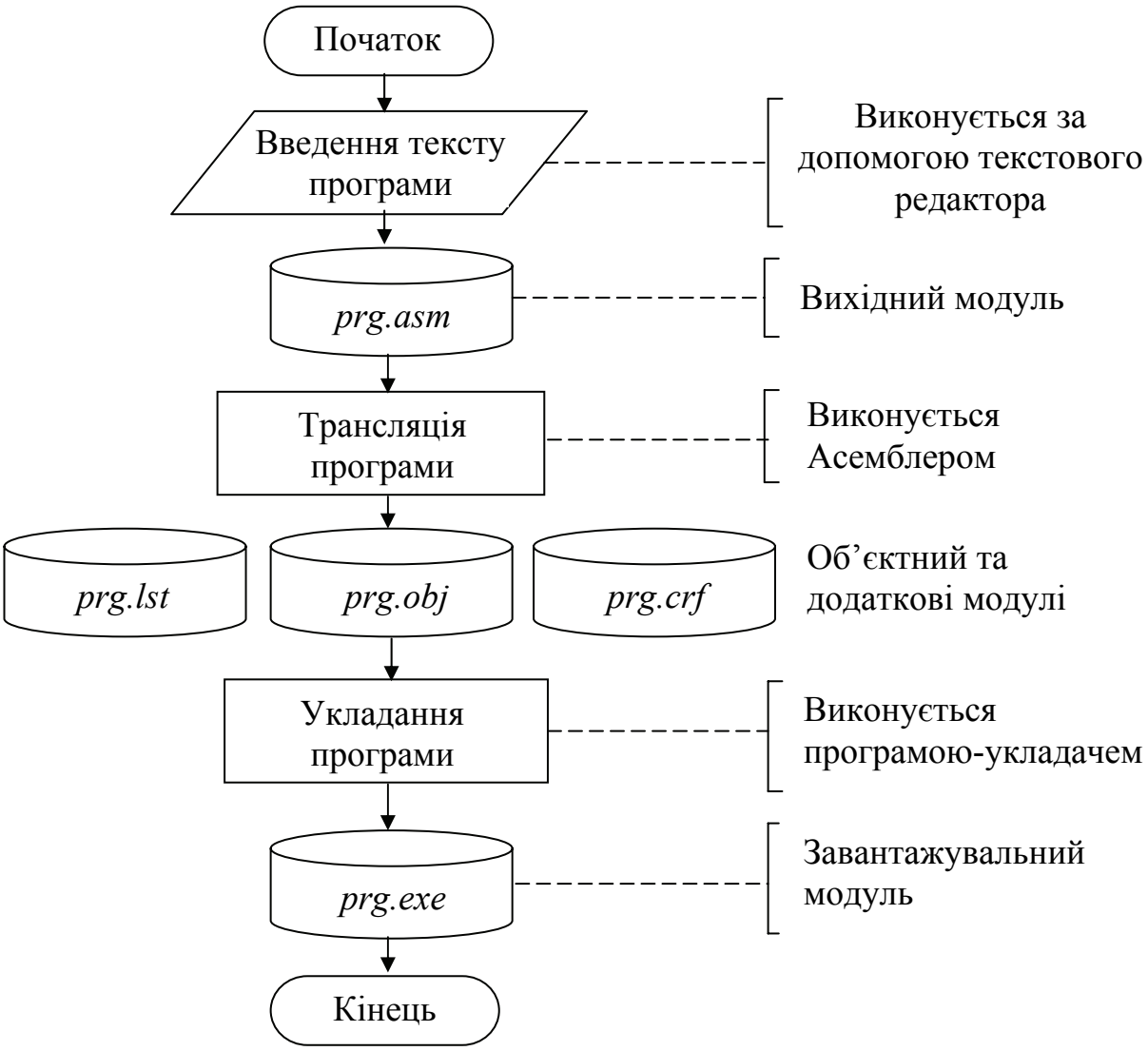



Рисунок 9.8 – Блок-схема підготовки програми для її виконання

Довжина речення становить 131 символ, усі інші ігноруються. Речення повинно розташовуватися у межах одного рядка, переносити речення на інший рядок не дозволяється. Речення Асемблера формуються із найпростіших конструкцій мови – **лексем**, синтаксично нероздільних послідовностей допустимих символів алфавіту Асемблера, які мають значення для транслятора. Як символи алфавіту Асемблера використовуються:

– усі літери латинського алфавіту, при цьому великі і малі літери є еквівалентні;

– цифри від 0 до 9;

– спеціальні і розподілювальні символи, до яких відносяться: ?, @, $, _, &, [ ], ( ), < >, { }, +, –, /, *, %, !, “ “, \, =, #,^ і деякі недруковані символи.

До лексем відносяться: ідентифікатори, рядки символів, цілі числа двійкової, шістнадцяткової або десяткової систем числення.

**Ідентифікатор** – це послідовність символів, яка визначена програмістом і використовується для іменування об’єктів програми (міток, змінних, назв операцій тощо). Ідентифікатор може складатися з одного або кількох символів, але починатися може лише літерою. Крапка може бути лише першим символом

ідентифікатора. Інші спеціальні символи використовувати в ідентифікаторах не дозволяється. Довжина ідентифікатора становить до 255 символів, але транслятор сприймає лише 32 перших символи, інші ігноруються.

Ідентифікатори поділяються на **службові слова** й **імена**. До службових слів відносяться ідентифікатори, значення яких однозначно визначено (назви регістрів, команд і таке інше). Імена – це символічні назви, які користувач призначає певним об'єктам програми (змінним, коміркам пам'яті тощо).

**Рядком символів** може бути будь-яка послідовність символів, в якості яких можливо використовувати літери і спеціальні символи, крім символа повернення каретки та перенесення рядка. Рядок символів обов'язково брати у лапки.

Через те, що мікропроцесор може працювати з різними сегментами, то для завантаження і роботи програми необхідно задати сегменти й особливості їх використання. Це робиться за допомогою директиви *SEGMENT*, яка має вигляд:

{Ім'я сегмента} *SEGMENT* {Тип вирівнювання} {Тип комбінування} {Клас сегмента} {Тип розміру}

Програма {Ім'я сегмента} *END*

Розглянемо окремі поля цієї директиви.

**Тип вирівнювання** – повідомлення програмі-укладачу про адресу початку сегмента. При правильному розміщенні програма виконується швидше. Можливі значення:

– *BYTE* – вирівнювання не виконується. Початкова адреса сегмента може бути будь-якою;

– *WORD* – сегмент починається з адреси, яка є кратна 2, при цьому молодший значний біт фізичної адреси дорівнює 0. Виконується вирівнювання по межі слова;

– *DWORD* – сегмент починається з адреси, яка є кратна 4. Виконується вирівнювання по межі подвійного слова;

– *PARA* – сегмент починається з адреси, яка є кратна 16, при цьому остання шістнадцяткова цифра фізичної адреси дорівнює 0H. Виконується вирівнювання по межі параграфа;

– *PAGE* – сегмент починається з адреси, яка є кратна 256. Виконується вирівнювання по межі сторінки;

– *MEMPAGE* – сегмент починається з адреси, яка є кратна 4 кбайтам.

За умовчанням тип вирівнювання має значення *PARA*.

**Тип комбінування –** повідомлення програмі-укладачу про об'єднання при виконанні сегментів різних модулів програми, які мають однакові імена. Можливі значення:

– *PRIVATE* – сегмент не об'єднується з іншими сегментами;

– *PUBLIC* – об’єднуються усі сегменти з однаковими іменами. Сегмент, що отримаємо, буде цілий і безперервний, а всі адреси будуть формуватися від його початку;

– *COMMON* – усі сегменти з однаковими іменами розміщуються за однією адресою. Таким чином, усі сегменти будуть перекриватися і сумісно використовувати пам’ять;

– *AT xxxx* – розміщує сегмент за абсолютною адресою параграфа, адреса якого задається виразом *xxxx*;

– *STACK* – визначення сегмента стека. Усі сегменти, що мають однакові імена, об’єднуються таким чином, що їх адреси розраховуються відносно вмісту регістра *SS*.

**Клас сегмента** – повідомлення про відповідний порядок включення сегментів при складанні програми із сегментів різних модулів. Програма-укладач з’єднує усі сегменти, що мають однаковий клас. Рекомендується об’єднувати сегменти з однаковим функціональним призначенням (наприклад, сегменти коду програми). Клас сегмента – це рядок, який береться у лапки, наприклад, сегмент коду «*CODE*».

**Тип розміру сегмента** – задає розмір сегмента і порядок формування фізичної адреси в ньому, тому що дані можуть бути 16- або 32-розрядними. Може приймати такі значення:

1 *USE*16 – формування фізичної адреси у такому сегменті виконується тільки з 16-розрядним зміщенням. Розмір такого сегмента 64 кбайт.

2 *USE*32 – формування фізичної адреси у такому сегменті виконується тільки з 32-розрядним зміщенням. Розмір такого сегмента 4 Гбайт.

Для призначення сегментам конкретного функціонального призначення і закріплення їх за сегментними регістрами використовується директива *ASSUME*, яка має вид:

*ASSUME* Назва сегментного регістра; Ім’я сегмента

Для простих програм, що мають по одному сегменту для коду, даних і стека, у найбільш поширених трансляторах *TASM* і *MASM* для спрощення директив сегментування уведено директиву зазначення моделі пам’яті *MODEL*, яка частково керує розміщенням сегментів. Ця директива зв’язує сегменти, які при використанні спрощених директив сегментування мають наперед визначені імена з сегментними регістрами. Обов’язковим параметром цієї директиви є модель пам’яті. У табл. 9.1 наведено назви і значення параметрів директиви *MODEL*.

У цій таблиці поняття *near* і *far* характеризують віддаленість об’єкта від місця використання. Якщо об’єкт розташовано у сегменті разом з кодом і звернутися до нього можливо за допомогою двобайтового зміщення, то це відповідає типу near (близький перехід). При цьому команда модифікує лише вміст вказівника команд *IP*. В іншому випадку об’єкт розміщується в іншому сегменті (тип *far*) і для звернення до нього необхідно надати чотирибайтний

вказівник – сегмент: зміщення. При цьому буде модифіковано вміст двох регістрів.

Таблиця 9.1 – Назви і параметри моделей пам’яті

| Модель | Тип коду | Тип даних | Призначення моделі |
|---|---|---|---|
| *TINY* | *near* | *near* | Код і дані поєднано в одну групу з ім’ям *DGROUP*. Використовується для створення програм з розширенням .com |
| *SMALL* | *near* | *near* | Код займає один сегмент, дані об’єднано у одну групу з ім’ям *DGROUP*. Ця модель, звичайно, використовується для створення програм мовою Асемблера |
| *MEDIUM* | *far* | *near* | Код займає декілька сегментів, по одному на кожен програмний модуль, що об’єднується. Усі посилання на передачу керування – типу *far*. Дані поєднано в одній групі. Усі посилання на них – типу *near* |
| *COMPACT* | *near* | *far* | Код знаходиться в одному сегменті. Дані можуть бути у різних сегментах. Посилання на них типу *far* |
| *LARGE* | *far* | *far* | Код займає декілька сегментів, по одному на кожен програмний модуль, що об’єднується. Усі посилання типу – *far* |

**Опис простих типів даних.** Для правильного описування і оброблення даних їх необхідно описати, для чого використовуються спеціальні директиви резервування й ініціалізації даних, які є вказівками транслятору на виділення певного обсягу пам’яті для розміщення даних певної довжини. Ці директиви мають вид:

- *db* – резервування пам’яті для даних довжиною 1 байт;
- *dw* – резервування пам’яті для даних довжиною 2 байта (слово);
- *dd* – резервування пам’яті для даних довжиною 4 байта (подвійне слово);
- *df* – резервування пам’яті для даних довжиною 6 байт;
- *dq* – резервування пам’яті для даних довжиною 8 байт;
- *dt* – резервування пам’яті для даних довжиною 10 байт.

Нижче наведено приклад використання директив для керування програмою на мові Асемблер.

```
MODEL SMALL                    ; Тип моделі пам’яті, що буде
                               ; використовуватися
STACK 256                      ; Визначення сегмента стека
```

```
DATA SEGMENT                        ; Визначення сегмента даних
DATA ENDS
CODE SEGMENT                        ; Визначення сегмента коду
ASSUME CS:CODE,DS:DATA,ES:DATA      ; Закріплення сегментних
                                    ; регістрів
      .                             ;
      .                             ; Програма
      .                             ;
CODE ENDS                           ; Кінець коду програми
END                                 ; Закінчення
```

**Контрольні питання:**

1 Які особливості мови Асемблер забезпечують її використання?

2 Які файли формуються у результаті роботи програми-транслятора?

3 Чому виконання асемблерної програми можливо лише після роботи програми-укладача?

4 Які файли використовує програма-укладач для формування завантажувального модуля?

5 Які оператори (речення) можуть входити до тексту програми на мові Асемблера?

6 Які моделі пам’яті Ви знаєте? Чим відрізняються поняття *near* і *far*?

7 Які директиви використовуються для опису простих типів даних?

**Контрольні питання підвищеної складності:**

1 Для чого використовуються різні типи комбінування?

2 Яким чином програмувач задає типи сегментів і особливості їх використання?

3 Які директиви резервування й ініціалізації даних Ви знаєте?

### *9.3.1 Формат команди*

Формат рядка команди складається з кількох полів, деякі з них не є обов’язковими. Формат рядка команди показано на рис. 9.9. У фігурних дужках показано необов’язкові елементи.

| Поле мітки | Поле команди | Поле операндів | Поле коментарів |
|---|---|---|---|
| {Мітка:} | {Префікс}Команда | {Операнди} | {; Коментар} |

Рисунок 9.9 – Формат команди Асемблера

**Мітка** – це символічне ім’я, що відповідає значенню поточного вмісту лічильника адреси в поточному сегменті коду. Вміст лічильника адреси відповідає значенню регістра-вказівника команд (*IP*) при виконанні програми, тому мітка – це адреса команди, до якої необхідно зробити умовний або безумовний перехід. Мітка відокремлюється від інших частин рядка двокрапкою. Максимальна довжина мітки – 31 символ. Кожна мітка у програмі повинна бути унікальною. Як мітки не допускається використовувати службові слова.

**Префікс** – це елемент команди, який уточнює або модифікує її дію у випадках:

– зміна сегмента, якщо нас не задовольняє значення сегмента за умовчанням;
– зміна розрядності адреси;
– зміна розрядності операнда;
– вказівка на повторення цієї команди.

**Команда (мнемокод)** – службове слово, яке відповідає типу машинної команди, що генерується. Як команди використовуються скорочені (повні) англійські слова або абревіатури англійських слів, що передають значення основної функції команди. В залежності від подання операндів, для виконання однієї команди може генеруватися декілька кодів.

**Операнди** – імена, які надаються об’єктам, над якими виконується операція (команда). У команді можуть бути один або два операнди, які відокремлюються один від одного комою. Є команди, в яких операндів немає. Поле операндів відокремлюється від поля команди пропуском. У команді можливі лише такі сполучення операндів:

– регістр – регістр;
– регістр – пам’ять;
– пам’ять – регістр;
– безпосередній операнд (число) – регістр;
– безпосередній операнд (число) – пам’ять.

Як операнди, що можуть оброблюватися транслятором, використовуються:

– постійні або безпосередні операнди – числа, рядки, імена або вирази, які мають фіксоване значення. Імена, які використовуються як безпосередні операнди, задаються операторами **=:** або *equ*;

– адресні операнди – адреси розміщення операндів у пам’яті. Задаються за допомогою двох складових адреси – сегмента і зміщення;

– переміщувані операнди – символьні імена, які є певними адресами пам’яті. Такі адреси вказують на розташування у пам’яті певних інструкцій (операнд – мітка) або даних (операнд – ім’я області пам’яті у сегменті даних). Переміщувані операнди відрізняються від адресних тим, що вони не зв’язані з конкретною адресою фізичної пам’яті, а формуються після завантаження програми для її виконання;

– лічильник адреси – це специфічний вид операнда, який позначається символом **$.** Транслятор замість цього символа ставить поточне значення вказівника команд і модифікує його значення відповідно до зміщення;

– регістровий операнд – ім'я відповідного регістра, вміст якого використовується у команді;

– базові і індексні операнди – операнди, які використовуються при адресуванні за базою, індексного адресування або їх комбінацій.

Загальний формат команди показано на рис. 9.10.

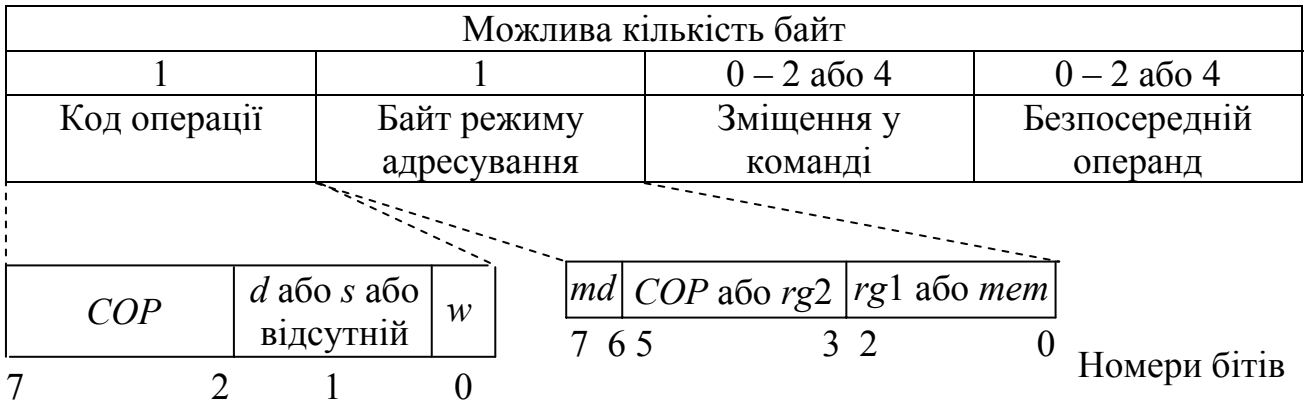



Рисунок 9.10 – Загальний формат команди Асемблера МП фірми *Intel*

Байт коду операції – це обов'язковий елемент, що описує операцію, яка буде виконуватися. Складається з поля коду (*COP*) операції і двох однобітових полів:

– *w* – відповідає типу операндів, які використовуються у команді:
  *w* = 0 – команда працює з байтами;
  *w* = 1 – команда працює зі словами;

– *d* – адресує приймач результату:
  *d* = 1 – відбувається передавання операнда або результату операції у регістр *rg*2, який описано у другому байті команди;
  *d* = 0 – відбувається передавання із цього регістра;

– *s* – біт, який є ознакою використання одного байта безпосереднього операнду при роботі зі словами. Використовується сумісно зі значенням *w*.

$$sw = \begin{cases} x0 – \text{один байт даних, який розміщується в молодшому байті операнду } L; \\ 01 – \text{два байта даних у двох байтах операнду } H \text{ і } L; \\ 11 – \text{один байт даних, який поширюється зі знаком до слова.} \end{cases}$$

Байт режиму адресування визначає форму подання адреси операндів. Складається з трьох полів:

– *md* – визначає довжину адреси операнда (у байтах), яка подана у наступних байтах – зміщення у команді:

$md =$ { 00 – зміщення у команді відсутнє й адреса операнда визначається вмістом регістрів *BX, BP, DI, SI*, які використовуються для обчислення фізичної адреси операнда;
01 – команда містить 8-розрядне зміщення у наступному байті, яке поширюється зі знаком до 16 розрядів;
10 – команда містить 16-розрядне зміщення у двох наступних байтах;
11 – операнди знаходяться лише у регістрах;

– *COP* або *rg*2 – значення *COP* використовується для уточнення операції, що виконується; *rg*2 визначає регістр, в якому знаходиться операнд, який умовно вважається другим;

– *rg*1 або mem – визначає операнд, який може бути розміщено у комірці пам'яті або у регістрі. Цей операнд умовно вважається першим. Кодування полів *rg*2 і *rg*1 в залежності від значення біта s подано у табл. 9.2.

Таблиця 9.2 – Кодування полів *rg*2 і *rg*1

| *rg*2 / *rg*1 | *w* = 0 | *w* = 1 | *rg*2 / *rg*1 | *w* = 0 | *w* = 1 |
|---|---|---|---|---|---|
| 000 | *AL* | *AX* | 100 | *AH* | *SP* |
| 001 | *CL* | *CX* | 101 | *CH* | *BP* |
| 010 | *DL* | *DX* | 110 | *DH* | *SI* |
| 011 | *BL* | *BX* | 111 | *BH* | *DI* |

**Коментар –** будь-який текст, який використовується для документування і кращого розуміння змісту програми. При трансляції програми коментарі ігноруються, тому наявність коментаря не впливає на ефективність виконання програми.

**Контрольні питання:**

1 Який формат має рядок команди мови Асемблер?
2 Що називається операндом у рядку команди мови Асемблер?
3 Які поєднання операндів є можливими?
4 Який формат має команда Асемблера МП фірми *Intel*?
5 В яких байтах може розміщуватися поле коду операції?
6 За допомогою яких бітів кодується тип операндів, що використовуються у команді?

**Контрольні питання підвищеної складності:**

1 Яким чином визначається тип режиму адресування у команді?
2 За допомогою яких полів визначаються регістри, які використовуються у команді?
3 Доведіть, що операнди команди *MOV AX,BX* розташовано у регістрах, якщо формат (код) цієї команди становить 89*D*8*H*.

### *9.3.2 Команди пересилань*

**Вхідний контроль:**

1 Які програмно-доступні вузли МПС Ви знаєте?
2 Яким чином вказується адреса регістрів загального призначення?
3 Які регістри входять до складу центрального процесора?
4 Яким чином може вказуватися адреса комірок пам'яті?
5 Що таке сегментування пам'яті мікропроцесорів фірми *Intel*?

Усі команди, що входять до цієї групи, в свою чергу, зручно поділити на п'ять груп і розглядати кожну окремо. До групи команд пересилань входять:

– загальні команди пересилань;
– команди передавання даних з використанням стека (стекові команди);
– команди введення-виведення;
– команди передавання рядків даних;
– команди перетворення типів даних.

Усі команди пересилань не змінюють стан регістра прапорців, його вміст зберігається незалежно від виконання команди пересилань і кількості пересилань між будь-якими пристроями. Виключення становлять лише команди POPF і SAHF, які у подальшому будуть розглянуті.

**Загальні команди пересилань –** це команди, що здійснюють пересилання операндів: регістр – регістр, регістр – пам'ять, пам'ять – регістр, регістр – безпосередній операнд (число), пам'ять – безпосередній операнд (число).

Найбільш поширеною командою цієї групи є команда ***MOV (MOVe operand)***, яка має вигляд

*MOV dst,src.*

Команда здійснює пересилання операнда *src* (джерело) до *dst* (приймач), причому вміст операнда *src* (джерело) не змінюється. У цій команді можливі усі способи адресування операндів, чим пояснюється її універсальність і розповсюдженість. При пересиланні даних між регістрами лише один з них може бути сегментним регістром. Типи даних, що пересилаються, повинні співпадати. В залежності від подання операндів команда може мати різний формат і займати від 3 до 6 байт.

Приклади цієї команди при використанні різних способів адресування:

```
MOV AL,BH      ; Пересилання байта з регістра BH до регістра AL
MOV BX,CX      ; Пересилання слова з регістра CX до регістра BX
MOV AL,[BX]    ; Пересилання байта з комірки пам'яті, адреса якої
               ; знаходиться у регістрі BX, до регістра AL
MOV AX,[BX]    ; Пересилання слова з двох сусідніх комірок
               ; пам'яті, молодша адреса котрих знаходиться
```

; у регістрі BX, до регістра AX

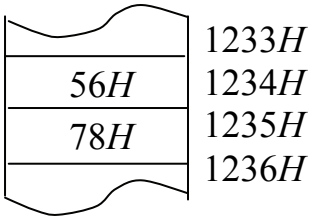
Виконання команди *MOV AX,[BX]* можливо пояснити таким чином: вважаємо, що до початку виконання цієї команди регістри і комірки пам'яті мали такий вміст:

Вміст регістра (*BX*) = 1234*H* – базова адреса комірки пам'яті у поточному сегменті даних.

Вміст регістра (*DS*) = 7000*H* – адреса поточного сегмента даних.

Вміст регістра (*AX*) – не визначений.

У комірках пам'яті поточного сегмента розміщені байти даних відповідно




Під час виконання операції відбувається копіювання вмісту комірки пам'яті з адресою, що знаходиться у регістрі *BX*, у молодший байт регістра *AX* – *AL* і комірки з наступною адресою (1235*H*) у старший байт регістра *AX* – *AH*. Після виконання цієї команди вміст регістрів буде:

Вміст регістра (*BX*) = 1234*H* – не зміниться.

Вміст регістра (*DS*) = 7000*H* – не зміниться.

Вміст регістра (*AX*) = 7856*H* – дані, які переслано з пам'яті.

Вміст комірок пам'яті не змінився.

Команда ***MOVSX (MOVe and Sign eXtension)*** – пересилання зі знаковим поширенням. Синтаксис команди:

*MOVSX dst,src.*

Використовується для перетворення операндів зі знаком, які мають малий розмір в еквівалентні елементи більшої розмірності.

Як операнди *dst* (приймач) використовуються 16- або 32-розрядні регістри, а операндом *src* (джерело) можуть бути 8- або 16-розрядні регістри або комірки пам'яті.

При виконанні команди вміст операнда "джерело" записується, починаючи з молодших розрядів в операнд "приймач", і знаковий розряд операнда "джерело" поширюється на вільні старші розряди операнда "приймач".

Наприклад, виконання команди

*MOVSX AX,CL,*

якщо до початку її виконання регістри мали такий вміст: (*CL*) = 82*H*, а (*AX*) – не визначено, полягає в копіюванні вмісту регістра *CL* у регістр *AL* і поширенні знакового розряду у регістрі *AH*. Після виконання команди у регістрі *AX* буде записано операнд *FF*82*H*.

Команда ***MOVZX (MOVe and Zero eXtension*)** – пересилання з нульовим поширенням. Синтаксис команди:

*MOVZX dst,src.*

Використовується для перетворення беззнакових операндів, які мають малий розмір, в еквівалентні елементи більшої розмірності.

Як операнди *dst* (приймач) використовуються 16- або 32-розрядні регістри, а операндом *src* (джерело) можуть бути 8- або 16-розрядні регістри або комірки пам'яті.

При виконанні команди, вміст операнда "джерело" записується, починаючи з молодших розрядів в операнд "приймач", а у старші розряди операнда "приймач" поширюються нулі.

Наприклад, виконання команди

*MOVZX AX,CL,*

якщо до її виконання регістри мали такий вміст: (*CL*) = 82*H*, а (*AX*) – не визначено, полягає в копіюванні вмісту регістра *CL* у регістр *AL* і поширенню нулів у регістрі *AH*. Після виконання команди у регістрі *AX* буде записано операнд 0082*H*.

Команда ***XCHG (eXCHanGe)*** – обмін вмісту операндів. Синтаксис команди:

*XCHG dst,src.*

Використовується для обміну вмісту 8-, 16-, 32-розрядних регістрів між собою, або для обміну байтами, словами і довгими словами між регістрами і пам'яттю. Команда не працює з сегментними регістрами. Адресування пам'яті може виконуватися будь-яким способом.

Приклади цієї команди при використанні різних способів адресування:

```
XCHG AL,BH    ; Обмін вмісту (байта) регістра BH на вміст регістра AL
XCHG BX,CX    ; Обмін вмісту (слова) регістра CX на вміст регістра BX
XCHG AL,[BX]  ; Обмін байтом з комірки пам'яті, адреса якої
              ; знаходиться у регістрі BX, і вмістом регістра AL
XCHG AX,[SI]  ; Обмін словом з комірок пам'яті, адреса яких задана
              ; вмістом регістра SI і словом з регістра AX.
```

Команда ***XLAT (transLATe byte from table)*** – перетворення байта. Синтаксис команди:

*XLAT.*

Використовується для завантаження у регістр *AL* байта з 256-байтової таблиці, яка розміщена у пам'яті і початкова адреса якої знаходиться у регістрі *BX*. Вміст регістра *AL* у цій команді використовується як індекс у таблиці.

Команди ***LDS/LES/LFS/LGS/LSS (Load pointer into DS/ES/FS/GS/SS segment register)*** завантаження одного із сегментних регістрів вказівником з пам'яті. Синтаксис команд:

*LDS/LES/LFS/LGS/LSS dst,src.*

Як операнди *dst* (приймач) можуть використовуватись 16- або 32-розрядні регістри загального призначення або індексні, а як операнди *scr* (джерело) – 4 або 8 комірок пам'яті, адресування яких здійснюється довільним способом. Усі команди виконуються однаково, зміни обумовлені регістром, назва якого входить у команду.

Використовується для завантаження повного вказівника у вигляді сегментної складової та зміщення. Повний вказівник завантажується у регістр – приймач і сегментний регістр, який указаний в команді.

Виконання однієї з команд, наприклад

*LES DI*,[*SI*+2],

за умови попереднього стану:
(*DS*) = 7000*H*; (*SI*) = 1234; *DX* – не визначено; у комірках пам'яті з адресами 7000:1234 66 77 88 99 *AA BB CC* …
відбувається таким чином:

– вміст комірок пам'яті з адресами 1234+2 і (1234+2)+1 завантажуються у регістр *DX*. Його вміст стане – 9988*H*;

– вміст комірок пам'яті з адресами (1234+2)+3 і (1234+2)+2 завантажуються у регістр *ES*. Його вміст стане – *BBAAH*.

Команда ***LEA (Load Effective Address)*** – завантаження ефективної адреси. Синтаксис команди:

*LEA dst,src.*

Як операнди *dst* (приймач) можуть використовуватись 16- або 32-розрядні регістри загального призначення або індексні, а як операнд *src* (джерело) – ефективна адреса, яка обчислюється відповідно заданому режиму адресування.

В операнд завантажується значення операнда *src* (джерело), а не вміст комірки пам'яті, яку він адресує.

Команда використовується для завантаження ефективної адреси у регістр, вміст якого можливо вважати базовим при подальшій роботі, наприклад, при роботі з рядками даних.

Команди *LDS/LES/LFS/LGS/LSS* і *LEA* використовуються для ініціалізації регістрів МП перед виконанням обробки рядків даних за допомогою команд обробки рядків. Використання цих команд спрощує комутацію сегментів даних, тому що одночасно завантажується базовий або індексний регістр та сегментний регістр.

Приклад команди завантаження ефективної адреси:

```
LEA DX,[BX+SI+2]   ; Завантаження у регістр DX ефективної адреси,
                   ; значення якої становить суму вмісту регістрів
                   ; BX і SI та числа 2. Якщо вміст регістрів
                   ; (BX) = 0100H, (SI) = 0020H, то у регістрі DX
                   ; після виконання команди буде завантажено
                   ; число 0122H = 0100H+0020H+0002H
```

Команда ***LAHF (Load AH register from register Flags)*** **–** завантаження регістра *AH* ознаками результату із регістра *FLAGS*. Синтаксис команди:

*LAHF*.

Команда завантажує копію вмісту молодшого байта регістра Flags, в якому зберігаються 5 ознак – *CF, PF, AF, ZF, SF*, до 7, 6, 4, 2 і 0 бітів регістра *AH*.

Команда ***SAHF (Store AH register into register Flags)*** **–** завантаження регістра *FLAGS* із регістра *AH*. Синтаксис команди:

*SAHF*.

Команда відновлює вміст молодшого байта регістра Flags з регістра *AH*, в якому зберігалися 5 ознак результату – *CF, PF, AF, ZF, SF* у бітах 7, 6, 4, 2 і 0 відповідно.

Команди *LAHF* і *SAHF* використовуються для забезпечення програмної сумісності 8-розрядних МП зі старшими моделями, а також для аналізу стану прапорців і їх зміни за необхідності.

**Команди передавання даних з використанням стека (стекові команди)** – це команди звернення до стека для завантаження або вивантаження даних. Головна особливість роботи зі стеком – одночасне завантаження/вивантаження двох байтів і автоматична модифікація вмісту

регістра вказівника стека *SP*. До виконання стекових команд необхідно ініціювати регістри *SS* (регістр сегмента стека) і *SP* (регістр – вказівник стека).

Команда ***PUSH (PUSH operand onto stack)*** – завантаження операнда у стек. Синтаксис команди:

*PUSH src.*

Як операнд *src* (джерело) у цій команді можуть бути регістри загального призначення, регістри вказівники, сегментні регістри, комірки пам'яті і безпосередні дані. Довжина операнда становить 2 або 4 байти.

Виконання команди полягає у завантаженні вмісту регістра/комірки пам'яті у дві комірки пам'яті з адресами, що визначаються вмістом регістра *SP* і становлять, для операндів довжиною 2 байти – (*SP*)–1 і (*SP*)–2, причому вміст старшого байта операнда записується у комірку з адресою (*SP*)–1, а молодшого – у комірку з адресою (*SP*)–2. Якщо використовуються 4-байтові операнди, значення вказівника стека становлять (*SP*)–1, (*SP*)–2, (*SP*)–3 і (*SP*)–4. Команду *PUSH* необхідно використовувати у парі з командою вивантаження *POP*.

Команда ***POP (POP operand from the stack)*** – вивантаження операнда зі стека. Синтаксис команди:

*POP dst.*

Команда завантажує в операнд *dst* (приймач) слово або подвійне слово зі стека. Як операнд *dst* (приймач) у цій команді можуть бути регістри загального призначення, регістри вказівники, сегментні регістри і комірки пам'яті. Довжина операнда становить 2 або 4 байти. Відповідно довжині операнда модифікується вміст регістра SP, який після виконання команди набуває значення (*SP*)+2 при довжині операнда 2 байти, і становить (*SP*)+4, якщо довжина операнда становить подвійне слово.

Команда ***PUSHA (PUSH All general registers onto stack)*** – завантаження вмісту всіх регістрів загального призначення у стек. Синтаксис команди:

*PUSHA.*

При виконанні команда розміщує у стеку вміст усіх регістрів загального призначення у наступному порядку: *AX, CX, DX, BX, SP, BP, SI, DI*. Вміст *DI* буде розташовано у новій вершині стека (у комірці пам'яті з найменшою адресою). У стеку буде зберігатися вміст регістра *SP*, що був до початку виконання цієї операції. Значення вказівника стека *SP*, після виконання команди, буде дорівнювати (*SP*)–16.

Для роботи з 32-розрядними регістрами загального призначення є команда ***PUSHAD***.

Команда ***POPA (POP All general registers from the stack)*** – вивантаження вмісту всіх регістрів загального призначення у стек. Синтаксис команди:

*POPA.*

Команда вивантажує вміст усіх регістрів загального призначення у порядку, зворотному тому, який було описано у команді *PUSHA*. При цьому вміст регістра SP вивантажується, але не відновлюється. Значення вказівника стека SP після виконання команди збільшується на 16.

Для роботи з 32-розрядними регістрами загального призначення є команда ***POPAD.***

Команда ***PUSHF (PUSH Flags registers onto stack) –*** завантаження вмісту регістра прапорців у стек. Синтаксис команди:

*PUSHF.*

Команда ***POPF (POP Flags registers from the stack)*** – вивантаження вмісту регістра прапорців у стек. Синтаксис команди:

*POPF.*

Команди *PUSHF* і *POPF* можуть використовуватися для звернення до регістра прапорців як до звичайного регістра у програмах обробки переривань й інших випадках, коли необхідно зберегти локальні ознаки обчислень.

**Команди введення-виведення** – це команди, за допомогою яких можливо обмінюватися даними з периферійними пристроями.

Існують дві команди:

– команда введення ***IN (INput operand from port)***. Синтаксис команди:

*IN dst,src,*

де як операнд *dst* (приймач) використовується вміст акумулятора *AL, AX* або *EAX*, а як операнд *src* (джерело) – адреса (номер) порту у вигляді безпосереднього числа або як вміст регістра *DX*;

– команда виведення ***OUT (OUT operand to port).*** Синтаксис команди

*OUT dst,src,*

де операнд *dst* (приймач) – адреса (номер) порту у вигляді безпосереднього числа або як вміст регістра *DX*, а операнд *src* (джерело) – акумулятор *AL, AX* або *EAX.*

Якщо номер порту задається безпосередньо, то можливо адресувати до 255 портів, а якщо задається вмістом регістра *DX*, то допускається формування і звернення до 65536 адрес різних портів. Непряме адресування через регістр *DX* зручно використовувати, якщо адреса порту обчислюється у ході виконання програми.

Команди не змінюють значення регістра прапорців.

Приклади команд введення- виведення:

```
IN AX,22H    ; Ведення у регістр AX слова з порту, номер якого 22H
IN AL,DX     ; Введення у регістр AL байта з порту, номер якого
             ; завантажено до регістра DX
OUT DX,AX    ; Виведення слова через порт, номер якого завантажено
             ; до регістра DX
```

**Команди передавання рядків даних –** це команди, що дозволяють обмінюватися даними у вигляді рядків байтів, слів або довгих слів. Рядок – це контекстно-зв'язана послідовність даних, що містяться в суміжних комірках пам'яті. Загалом у системі команд є п'ять операцій, кожна з яких зреалізована трьома командами.

Пересилання елементів рядок-джерело у рядок-приймач відбувається в залежності від довжини елемента рядка за допомогою команд:

***MOVSB (MOVe String Byte) –*** пересилання байтів;

***MOVSW (MOVe String Word)*** – пересилання слів;

***MOVSD (MOVe String Double word)*** – пересилання подвійних слів.

Ці команди копіюють елемент рядок-джерело довжиною байт, слово або подвійне слово, що знаходиться за адресою *DS:SI* в елемент рядок-приймач, що визначається адресою *ES:DI*. Адреса операнда-приймача може визначатися тільки регістром *ES*, заміна адреси операнда-джерела дозволяється, за умовчанням визначено *DS*.

Після пересилання вміст регістрів *SI* і *DI* автоматично змінюється відповідно значення прапорця *DF*. Якщо прапорець *DF* дорівнює 0 (була виконана команда *CLD*), то вміст регістрів *SI* і *DI* збільшується, якщо *DF* має значення 1 (була виконана команда *STD*), то вміст регістрів *SI* і *DI* зменшується. Збільшення/зменшення відбувається на 1, якщо пересилається байт, на 2, якщо пересилається слово, і на 4, якщо пересилається подвійне слово.

Команді може передувати префікс *REP*, який примушує виконувати пересилання поки вміст регістра *CX* не стане дорівнювати 0.

Команди завантаження елемента з рядка в акумулятор AL/AX/EAX мають вигляд:

***LODSB (LOad String Byte operands) –*** завантаження байта;

***LODSW (LOad String Word operands)*** – завантаження слова;

***LODSD (LOad String Double word operands)*** – завантаження подвійного слова.

Ці команди завантажують елемент рядок-джерело довжиною байт, слово або подвійне слово, що знаходиться за адресою *DS(ES/GS/FS):SI* в акумулятор. Після пересилання вміст регістра *SI* автоматично змінюється відповідно значення прапорця *DF*. Якщо прапорець *DF* дорівнює 0 (була виконана команда *CLD*), то вміст регістра *SI* збільшується, якщо *DF* має значення 1 (була виконана команда *STD*), то вміст регістра *SI* зменшується. Збільшення/зменшення відбувається на 1, якщо пересилається байт, на 2, якщо пересилається слово, і на 4, якщо пересилається подвійне слово.

Команді може передувати префікс *REP*, однак більш доцільним є використання цієї команди у *LOOP*-конструкціях, тому що зазвичай необхідне подальше оброблення кожного елемента.

Команди збереження вмісту акумулятора *AL/AX/EAX* у рядку мають вигляд:

***STOSB (Store String Byte operands) –*** завантаження байта;

***STOSW (Store String Word operands)*** – завантаження слова;

***STOSD (Store String Double word operands)*** – завантаження подвійного слова.

Ці команди завантажують дані з акумулятора *AL/AX/EAX* довжиною байт, слово або подвійне слово у рядок, елемент якого знаходиться за адресою *ES:DI*. Після пересилання вміст регістра *DI* автоматично змінюється відповідно значення прапорця *DF*. Якщо прапорець *DF* дорівнює 0 (була виконана команда *CLD*), то вміст регістра *DI* збільшується, якщо *DF* має значення 1 (була виконана команда *STD*), то вміст регістра *DI* зменшується. Збільшення/зменшення відбувається на 1, якщо пересилається байт, на 2, якщо пересилається слово, і на 4, якщо пересилається подвійне слово.

Команді може передувати префікс *REP*, що дає можливість формувати блоки пам’яті, кількість елементів у яких буде визначатися вмістом регістра *CX*.

Команди введення рядка даних через порт мають вигляд:

***INSB (INput String Byte operands)*** – введення рядка байт з порту;

***INSW (INput String Word operands) –*** введення рядка слів з порту;

***INSD (INput String Double word operands) –*** введення рядка подвійних слів з порту.

Ці команди завантажують у пам’ять, адреса якої визначається вмістом регістрів *ES:DI*, байт, слово або подвійне слово з порту, номер якого задано у регістрі *DX*. Використовувати у цій команді інші регістри для адресування не дозволяється. Після пересилання вміст регістра *DI* автоматично змінюється відповідно значення прапорця *DF*. Якщо прапорець *DF* дорівнює 0 (була виконана команда *CLD*), то вміст регістра *DI* збільшується, якщо *DF* має значення 1 (була виконана команда *STD*), то вміст регістра *DI* зменшується. Збільшення/зменшення відбувається на 1, якщо пересилається байт, на 2, якщо пересилається слово, і на 4, якщо пересилається подвійне слово.

Команді може передувати префікс *REP*, що дає можливість приймати рядки, кількість елементів у яких буде визначатися вмістом регістра *CX*.

Команди виведення рядка даних через порт мають вигляд:
***OUTSB (INput String Byte operands)*** – виведення рядка байт через порт;
***OUTSW (INput String Word operands)*** – виведення рядка слів через порт;
***OUTSD (INput String Double word operands)*** – виведення рядка подвійних слів через порт.

Ці команди виводять через порт, номер якого задано вмістом регістра *DX*, дані з пам'яті, яку адресовано регістрами *ES:SI*, байт, слово або подвійне слово. Дозволяється заміна сегментного регістра для адресування пам'яті. Номер порту може задаватися лише регістром *DX*. Після пересилання вміст регістра *SI* автоматично змінюється відповідно до значення прапорця *DF*. Якщо прапорець *DF* дорівнює 0 (була виконана команда *CLD*), то вміст регістра *SI* збільшується, якщо *DF* має значення 1 (була виконана команда *STD*), то вміст регістра *SI* зменшується. Збільшення/зменшення відбувається на 1, якщо пересилається байт, на 2, якщо пересилається слово, і на 4, якщо пересилається подвійне слово.

Команді може передувати префікс *REP*, що дає можливість виводити рядки, кількість елементів у яких буде визначатися вмістом регістра *CX*.

**Команди перетворення типів даних** використовуються для формування даних однієї розмірності (кількості розрядів) перед виконанням арифметичних операцій. Особливе значення ці команди набувають, якщо в арифметичних командах використовуються числа зі знаком. Їх виконання ґрунтується на поширенні знакового біта на всі старші розряди числа. Таким чином, формуються числа з тим самим значенням і знаком, але з більшою кількістю розрядів. Команди цієї групи виконуються з фіксованими регістрами, тому операндів у них немає. Усі команди цієї групи не змінюють значення регістра прапорців.

Команда ***CBW (Convert Byte to Word)*** – перетворення байта (у регістрі *AL*) у слово (у регістрі *AX*). При виконанні знаковий розряд числа з регістра *AL* поширюється на всі розряди регістра *AH*. Наприклад, якщо до виконання команди вміст регістра *AL* у регістрі *AX* був 85*H* (число –05 у доповнювальному коді), то після її виконання вміст регістра *AX* буде *FF*85*H*. Якщо вміст регістра *AL* був 74*H* (число +74*H*), то після виконання команди вміст регістра *AX* стане 0075*H*.

Команда ***CWD (Convert Word to Double Word)*** – перетворення слова (у регістрі *AX*) у подвійне слово (у регістрах *DX:AX*). Виконання команди полягає у поширенні знакового розряду з регістра *AX* на усі розряди регістра *DX*.

Операцію можливо використовувати при підготовці до виконання операції ділення, в якій розмір діленого повинен бути у два рази більший ніж

розмір дільника. Наприклад, якщо необхідно у програмі поділити число, що знаходиться у регістрі *AX* на число з регістра *BX*. Використання цієї команди показано у прикладі виконання команди *DIV*.

Команда ***CWDE (Convert Word to Double Word Extended)*** – перетворення слова (у регістрі *AX*) у подвійне слово (у регістрі *EAX*). При виконанні знаковий розряд числа з регістра *AX* поширюється на всі старші розряди регістра *EAX*.

До цієї групи також можливо віднести команди *MOVSX* і *MOVZX*, які були розглянуті раніше.

**Контрольні питання:**

1 Які типи команд входять до групи команд пересилань?

2 Чим відрізняється використання команд *MOV, MOVSX* і *MOVZX*?

3 Покажіть вміст регістрів *DS* і *BX* після виконання команди *LDS BX*,[1234], якщо комірки пам’яті у поточному сегменті мають вміст: 7000:1234 10 20 30 40 50?

4 Які команди введення-виведення Ви знаєте?

5 Напишіть коментарі з поясненням виконання кожної команди до наступної програми:

```
MOV CX,0004H ; Завантаження регістра-лічильника
CLD
MOV SI,1234H
REP LODSB
```

Назвіть вміст регістрів *SI* і *AL* після виконання цієї програми, якщо вміст комірок пам’яті у поточному сегменті був такий самий, як у запитанні 3.

6 Для чого використовуються команди перетворення типів даних?

**Контрольні питання підвищеної складності:**

1 Яким чином виконується перетворення типів даних зі знаком?

2 Для чого використовується префікс *REP*?

3 Наведіть вміст регістра – вказівника стека *SP* і вміст комірок пам’яті з адресами 3456*H* і 3455*H* після виконання команди *PUSH CX*, якщо до виконання команди регістри мали вміст: (*SP*) = 3457*H* і (*CX*) = 1234*H*?

### *9.3.3 Команди перетворення даних мови Асемблер-86*

**Вхідний контроль:**

1 Які математичні та логічні операції виконуються в обчислювальній техніці?

2 Який цифровий пристрій виконує математичні та логічні операції?

3 Які ознаки результату формуються пристроєм АЛП при виконанні математичних та логічних операцій?

4 У яких системах числення можуть бути подані дані, які оброблюються МПС?

5 Як подаються десяткові числа у двійково-десятковій кодованій системі числення?

6 Як формуються ознаки результатів при виконанні арифметичних і логічних операцій?

До цієї групи можливо віднести арифметичні і логічні команди, а також команди, які у своїй роботі використовують арифметичні або логічні принципи виконання.

Арифметичні команди мови Асемблер-86 виконуються над цілими числами чотирьох типів: двійкові числа без знака; двійкові числа зі знаком; розпаковані двійково-десяткові числа й упаковані двійково-десяткові числа. Числа зі знаком подаються у МП у вигляді доповнювального коду. Для виконання арифметичних операцій над числами з плаваючою точкою використовується математичний співпроцесор, команди якого становлять окрему групу команд.

**Команди двійкової арифметики**

До групи арифметичних команд над двійковими числами входять команди: додавання, віднімання, множення, ділення і зміни знака.

**Команди додавання** представлені трьома командами:

***ADD (ADDition) –*** команда додавання операндів. Команда має вид

*ADD dst,src*

і виконує додавання операндів *src* (джерело) і *dst* (приймач), які можуть бути байтами, словами, подвійними словами. Результат операції розміщується в операнді *dst* (приймач), при цьому втрачається один із доданків. Як операнд *dst* (приймач) можуть використовуватися усі регістри загального призначення, крім сегментних, і комірки пам’яті, які можливо адресувати будь-яким способом адресування. У будь-якому випадку адресується лише одна комірка, що відповідає молодшому байту результату, адреси інших для розміщення старших байтів формуються автоматично. В разі використання як операнд *dst* (приймач) акумулятора, довжина команди стає меншою і виконується вона швидше.

Операнд *src* (джерело) може розміщуватись у регістрах загального призначення, комірках пам’яті і бути безпосереднім числом. Розмір обох операндів обов’язково повинен співпадати.

Команда установлює прапорці *OF, SF, ZF, AF, PF, CF* регістра ознак (*FLAGS*), які можливо аналізувати у подальших командах.

***ADC (Addition with Carry)*** – команда додавання операндів з урахуванням вхідного перенесення з біта *CF* регістра ознак, значення якого додається до результату.

Вид і виконання команди повністю співпадає з командою *ADD*.

Використовується при додаванні двійкових 64-розрядних чисел.

Приклади використання команд додавання:

```
ADD AX,SI          ; Додавання вмісту регістра AX до вмісту SI і
                   ; збереження результату у регістрі AX
ADD [1234],BX      ; Додавання вмісту комірок пам'яті з адресами
                   ; 1234H і 1235H у поточному сегменті даних до
                   ; вмісту регістра BX і завантаження результату до
                   ; тих самих комірок. Наприклад, вміст регістра
                   ; (BX) = 1111H; у комірках пам'яті з адресами
                   ; 1234H і 1235H зберігаються відповідно числа 22H
                   ; і 33H. Додавання буде виконуватися таким чином
                   ;       (BX)        1111H
                   ;       [1234]      0022H
                   ;       [1235]      3300H
                   ;                   4433H
                   ; Результат буде розташовано у тих самих комірках –
                   ; молодший байт у комірці з адресою 1234H, а
                   ; старший – у комірці 1235H
ADC AL,DH          ; Додавання до вмісту регістра DH вмісту AL і
                   ; поточного значення біту CF, завантаження
                   ; результату до регістра AL
                   ;       (AL)        1FH
                   ;       (DH)        04H
                   ;       CF          01H
                   ;                   24H
```

***INC (INCrement operand by 1)*** – команда додавання 1 до значення операнда. Синтаксис команди:

*INC dst.*

Операндом *dst* (приймач) може бути будь-який регістр загального призначення (8- , 16- або 32-розрядний) або операнд (8- , 16- або 32-розрядний), розташований у пам'яті. При виконанні команди операнд вважається беззнаковим числом.

Команда установлює прапорці *OF, SF, ZF, AF, PF* регістра ознак.

**Команди віднімання** також представлені трьома командами:
***SUB (SUBtract)*** – команда віднімання цілих чисел. Команда має вид

*SUB dst,src*

і виконує віднімання від вмісту операнда *dst* (приймач) вмісту операнда *src* (джерело). Результат зберігається в операнді *dst* (приймач), при цьому втрачається попередній вміст цього операнда. Як операнди *dst* (приймач) можуть використовуватися усі регістри загального призначення, крім сегментних, і комірки пам'яті, які можливо адресувати будь-яким способом адресування. У будь-якому випадку адресується лише одна комірка, що відповідає молодшому байту результату, адреси інших для розміщення старших байтів формуються автоматично. В разі використання як операнд *dst* (приймач) акумулятора, довжина команди стає меншою і виконується вона швидше.

Операнд *src* (джерело) може розміщуватись у регістрах загального призначення, комірках пам'яті і бути безпосереднім числом. Розмір обох операндів обов'язково повинен співпадати.

Команда змінює прапорці *OF, SF, ZF, AF, PF, CF* регістра ознак, які можливо аналізувати у подальших командах.

***SBB (SuBtract with Borrow)*** – віднімання з урахуванням позики зі старшого розряду. Використовується для виконання віднімання старших частин значень багатобайтових операндів з урахуванням можливої попередньої позики під час віднімання молодших частин операндів.

При її виконанні, від значення операнда *dst* (приймач) віднімається сума значення операнда *src* (джерело) і значення біта *CF*; результат завантажується на місце операнда *dst* (приймач). Вид команди збігається з командою *SUB*.

Команда змінює прапорці *OF, SF, ZF, AF, PF, CF* регістра ознак.

Приклад використання цієї команди покажемо за допомогою наступної програми:

```
STC              ; Установлення біта CF в 1
MOV AX,4567H  ; Завантаження до регістра AX числа 4567H
MOV BX,1111H  ; Завантаження до регістра BX числа 1111H
SBB AX,BX        ; Виконання цієї команди відбувається у два етапи:
                 ; – до вмісту регістра BX додається 1
                 ;          1111H + 0001H = 1112H;
                 ; – із вмісту регістра AX віднімається число, яке
                 ; отримано на попередньому етапі:
                 ;          _ 4567H
                 ;            1112H
                 ;            3455H
                 ; при виконанні прапорець CF скидається в 0
                 ; (позики немає)
MOV CX,1234H  ; Завантаження до регістра CX числа 1234H
```

```
MOV DX,9111H  ; Завантаження до регістра DX числа 9111H
SBB CX,DX     ; Виконання команди відбувається аналогічно
              ; описаному вище:
              ;       9111H + 0000H = 9111H (прапорець CF
              ;   скинуто);
              ;          1234H
              ;        − 9111H
              ;          8123H  (з урахуванням позики зі
              ;     старшого розряду; прапорець CF
              ;     установлюється в 1)
```

***DEC (DECrement operand by 1)*** – віднімання від значення операнда 1. Синтаксис команди:

*DEC dst.*

Операндом *dst* (приймач) може бути будь-який регістр загального призначення (8-, 16- або 32-розрядний) або операнд (8-, 16- або 32-розрядний), розташований у пам'яті. При виконанні команди операнд вважається беззнаковим числом.

Команда змінює прапорці *OF, SF, ZF, AF, PF* регістра ознак.

**Команди множення** у системі команд представлені двома командами: для цілих беззнакових чисел і для цілих чисел зі знаком.

***MUL (MULtiply)*** – команда множення двох цілих беззнакових чисел. Команда має узагальнений вид

*MUL src.*

Алгоритм виконання залежить від формату операнда у команді і потребує визначити лише операнд *src* (джерело), розмір якого байт, слово або подвійне слово і який може розміщатися у пам'яті або у регістрі. Розміщення другого операнда фіксовано і залежить від розміру операнда *src* (джерело):

- якщо операнд *src* (джерело) – байт, то другий операнд повинен розміщуватися у регістрі *AL*;
- якщо операнд *src* (джерело) – слово, то другий операнд повинен розміщуватися у регістрі *AX*;
- якщо операнд *src* (джерело) – подвійне слово, то другий операнд повинен розміщуватися у регістрі *EAX*.

Розміщення результату також є фіксованим і визначається розмірами співмножників:

- якщо перемножуються байти, то результат розміщується у регістрі *AX*;

– якщо перемножуються слова, то результат розміщується у регістрах *DX:AX*;

– якщо перемножуються подвійні слова, то результат розміщується у регістрах *EDX:EAX*.

У результаті виконання цієї команди формуються лише прапорці *OF* і *CF*. Їх значення, якщо старша половина добутку не дорівнює нулю, становить 1, що показує наявність розрядів результату у регістрах *AH* і *DX*. В іншому випадку їх значення дорівнює 0. На стан інших прапорців виконання команди *MUL* не впливає.

Приклад використання команди:

```
MOV AL,02H  ; Завантаження першого співмножника (02H = 02D)
            ; до регістра AL
MOV BL,80H  ; Завантаження другого співмножника (80H = 128D)
            ; до регістра BL
MUL BL      ; Множення операндів, результат розташовано у
            ; регістрі AX (0100H = 256D)
```

***IMUL (Integer MULtiply)*** – команда множення двох цілих чисел зі знаком. Команда має узагальнений вид

*IMUL src.*

Команда виконується майже так само, як і команда *MUL*, проте у цій команді операнди подаються як числа зі знаком у доповнювальному коді.

Прапорці *OF* і *CF* установлюються залежно від розміщення знакових розрядів: якщо старша половина добутку, яка знаходиться у регістрах *AH* або *DX*, не містить поширення знаку молодшої частини результату, то прапорці *OF* і *CF* установлюються в 1, що показує наявність у старшій половині результату значущих цифр. В іншому випадку – установлюються у 0. Стан інших прапорців є невизначеним.

**Команди ділення** у системі команд представлено двома командами: для цілих беззнакових чисел і для цілих чисел зі знаком.

***DIV (DIVide unsigned)*** – команда ділення цілих беззнакових чисел. Узагальнена форма цієї команди має вигляд:

*DIV src.*

Для виконання команди необхідно задавати лише один операнд – *src* (джерело), розмір якого може становити байт, слово і подвійне слово, розміщення другого операнду є фіксоване і залежить від розміру дільника, який визначено у команді:

– якщо дільник це байт, то ділене повинне бути розміщене у регістрі *AX*. Частку буде розташовано у регістрі *AL*, а залишок – у регістрі *AH*;

– якщо дільник це слово, то ділене розміщується у регістрах *DX:AX* і старша частина діленого розміщується у *DX*. Частка розміщується в *AX*, а залишок у регістрі *DX*;

– якщо дільник це подвійне слово, то ділене розміщується у регістрах *EDX:EAX* і старша частина діленого розміщується у *DX*. Частка розміщується в *EAX*, а залишок – у регістрі *EDX*.

Якщо при діленні формується дробна частка, то вона округлюється до цілого числа у результаті відкидання дробної частини.

Виконання команди не впливає на прапорці регістра ознак.

Якщо частка за розміром більша за акумулятор або дільник дорівнює 0, то генерується переривання типу 0 і програма виконує підпрограму переривання.

Приклад виконання такої команди:

```
MOV AX,1111H  ; Завантаження діленого (1111H = 4369D) до
              ; регістра AX
MOV BX,20H    ; Завантаження дільника (20H = 32D) до регістра BL
CWD           ; Формування необхідного розміру діленого
DIV BX        ; Ділення, частка у регістрі AX дорівнює 0088H,
              ; залишок у регістрі DX дорівнює 0011H
```

***IDIV (Integer DIVide)*** – команда ділення цілих чисел зі знаком. Узагальнена форма цієї команди має вигляд:

*IDIV src.*

Команда виконується майже так само як і команда *DIV*, проте у цій команді операнди подаються як числа зі знаком у доповнювальному коді.

Стан прапорців у регістрі ознак є невизначеним.

***NEG (NEGate operand)*** – команда, яка змінює знак операнда. Команда має вигляд:

*NEG src.*

Команда змінює знак операнда, який може бути вмістом регістра загального призначення (8-, 16- або 32-розрядним) або вмістом комірок пам’яті (8, 16 або 32 розряди). Операція двійкового доповнення виконується як інвертування всіх розрядів операнда і додавання до результату 1. Команда змінює лише прапорець *CF*.

**Команди десяткової арифметики**

До цієї групи входять команди, що дозволяють працювати з десятковими числами, які можуть бути поданими у двох формах: як упаковані двійково-десяткові числа (*BCD*-формат), і як розпаковані двійково-десяткові числа (*ASCII*-формат). У кожному з форматів багаторозрядні десяткові числа представлено послідовностями байтів, з якими і працює МП. Основним робочим регістром в усіх командах десяткової арифметики є регістр *AL*. Слід зазначити, що команд, спеціально налаштованих на виконання операцій складання, віднімання, множення і ділення десяткових чисел немає, тому що розмірність таких чисел може бути довільною, крім того, виконання множення і ділення десяткових чисел є можливе лише в розпакованому форматі. Тому виконання арифметичних операцій виконується за допомогою команд двійкової арифметики, а потім проводиться корекція результату.

**Корекція результату додавання десяткових чисел** виконується після виконання команди *ADD* або *ADC*. Як було показано раніше, неправильний результат може бути обумовлений двома причинами:

– у результаті додавання отримали тетраду, двійковий еквівалент якої є більший ніж 9;

– отримано перенесення до старшої тетради з вагою 16.

Корекція результату проводиться за результатами аналізу прапорців *AF* і *CF* регістра ознак, які визначаються вмістом регістра *AL*, в якому розміщено суму.

***DAA (Decimal Adjust for Addition)*** – команда десяткової корекції додавання BCD-чисел, які подано в упакованому виді.

Операндів у команди немає, тому що вона працює лише з регістром *AL*. Алгоритм корекції результату виконання попередньої команди додавання виконується за таким алгоритмом:

– якщо прапорець $AF = 1$ або молодша тетрада регістра *AL* має значення, більше ніж 9, то до вмісту *AL* додається число 06;

– якщо прапорець $CF = 1$ або старша тетрада регістра *AL* має значення, більше ніж 9, то до вмісту *AL* додається число 60;

– якщо мають місце обидві попередні ситуації, то корекція починається з молодшого розряду.

Після виконання команди *DAA* необхідно перевіряти значення прапорця *CF* для його урахування при роботі зі старшими розрядами десяткових цифр числа.

***AAA (Ascii Adjust for Addition)*** – команда десяткової корекції додавання *BCD*-чисел, які подано у розпакованому виді. Команда також працює лише з регістром *AL*. Алгоритм роботи:

– якщо молодша тетрада регістра *AL* є припустима і значення $AF = 0$, то необхідно обнулити старшу тетраду *AL*;

– якщо молодша тетрада регістра *AL* більша 9 або значення *AF* = 1, то необхідно додати 06 до вмісту регістра *AL* і додати 1 до вмісту регістра *AH*;
– обнулити старшу тетраду регістра *AL*;
– установити прапорець *CF* у такий стан, в якому знаходиться *AF*.

**Корекція результату віднімання десяткових чисел** виконується після виконання команди *SUB* або *SBB*. Корекція результату проводиться за результатами аналізу прапорців *AF* і *CF* регістра ознак, які визначаються вмістом регістра *AL*, в якому розміщено віднімання.

***DAS (Decimal Adjust for Subtraction)*** – команда десяткової корекції після віднімання. Алгоритм роботи команди:
– якщо прапорець *AF* = 1 або молодша тетрада *AL* має вміст, більший ніж 9, то з вмісту *AL* віднімається 06;
– якщо *CF* = 1 або старша тетрада *AL* має вміст, більший ніж 9, то з вмісту *AL* віднімається 60.

***AAS (Ascii Adjust after Subtraction)*** – команда десяткової корекції після віднімання BCD-чисел, які подано у розпакованому виді. Алгоритм роботи команди:
– якщо молодша тетрада регістра *AL* припустима і *AF* = 0, то обнулити старшу тетраду регістра *AL*;
– якщо вміст молодшої тетради регістра *AL* не є припустимий або *AF* = 1, то необхідно відняти число 06 із вмісту *AL*;
– обнулити старшу тетраду регістра *AL*;
– надати *CF* такий же самий стан як і *CF*.

**Корекція результатів множення** виконується командою
***AAM (Ascii Adjust after Multiply)*** – команда корекції результату множення двох розпакованих *BCD*-чисел і перетворення результату у вигляді двійкового числа, меншого 63*H* (99*D*), у його розпакований *BCD*-еквівалент.

Команда здійснює ділення вмісту регістра *AL* на десять (0*AH*) і завантажує частку у регістр *AH*, а залишок – у регістр *AL*. Стан прапорців *SF*, *ZF, PF* визначається вмістом *AL*, а стан прапорців *OF AF CF* не є визначений.

**Корекція результатів ділення** проводиться до виконання ділення командою
***AAD (Ascii Adjust before Division)*** – команда підготовки двох розпакованих *BCD*-чисел для ділення і перетворення двозначного розпакованого числа, меншого 63*H* (99*D*), у двійковий еквівалент. Алгоритм роботи команди:
– вміст регістра *AH* множиться на 10*D* і додається до вмісту регістра *AL*;
– регістр *AH* обнулюється.

Виконується ділення числа у регістрі AX за допомогою команди *DIV*.

До групи арифметичних команд також входять такі команди:

***BSWAP (Byte SWAP)*** – команда перестановки байтів. Синтаксис команди

*BSWAP src.*

Операндом *src* (джерело) може бути лише 32-розрядний регістр загального призначення. Команда використовується для зміни порядку слідування байтів і переходу від однієї форми адресування до іншої (від форми «молодший байт за молодшою адресою» на звернену). Команда не змінює вміст регістра прапорців. Алгоритм роботи можливо пояснити за допомогою рис. 9.11.

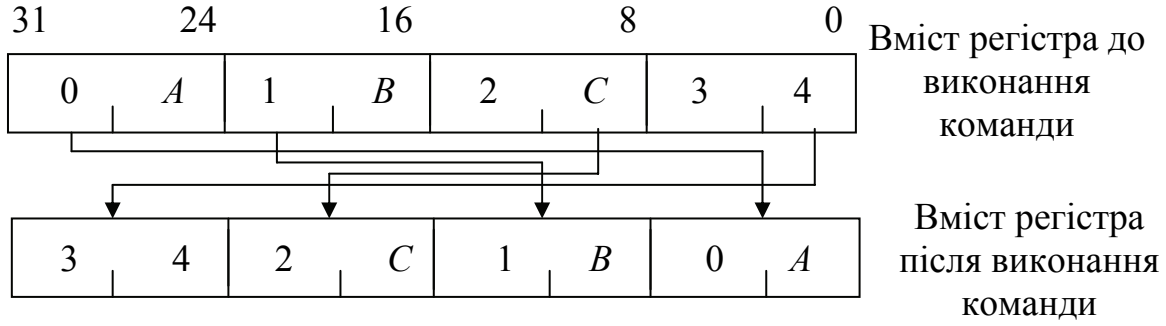



Рисунок 9.11 – Алгоритм виконання команди *BSWAP*

***XADD (eXchange and ADD)*** – обмін і додавання значень двох операндів. Команда має вигляд

*XADD dst,src.*

Операнд *dst* (приймач) може бути 8-, 16- або 32-розрядним регістром загального призначення або коміркою пам’яті з такою ж кількістю розрядів. Операнд *src* (джерело) може бути лише пам’яттю відповідної розмірності.

Команда виконується у три етапи:

– вміст операнда *dst* (приймач) копіюється в операнд *src* (джерело);

– виконується додавання вмісту операнда *dst* (приймач) до вмісту операнда *src* (джерело);

– сума завантажується в операнд *dst* (приймач). Команда змінює усі прапорці результату.

***CMP (CoMPare operands)*** – команда для порівняння двох операндів. Команда має вигляд

*CMP dst,src.*

Операнд *dst* (приймач) розміщується у регістрі загального призначення або комірці пам’яті, а операнд *src* (джерело) також може бути вмістом регістра загального призначення або комірки пам’яті, але одночасно не можуть бути вмістом комірок пам’яті. Крім того, операндом *src* (джерело) може бути безпосереднє число.

Алгоритм виконання команди полягає у відніманні від вмісту операнда *dst* (приймач) вмісту операнда *src* (джерело). Результат не формується, лише установлюються ознаки результату, які можливо аналізувати і робити висновки про співвідношення операндів між собою.

Порівняння елементів рядків байтів, слів, подвійних слів відбувається в залежності від довжини елемента рядків за допомогою команд:

***CMPS (CoMPare String)*** – команда для порівняння елементів двох рядків;

***CMPSB (CoMPare String Byte)*** – команда для порівняння двох рядків байтів;

***CMPSW (CoMPare String Word)*** – команда для порівняння двох рядків слів;

***CMPSD (CoMPare String Double word operands)*** – команда для порівняння двох рядків подвійних слів.

Команда порівняння рядків має узагальнений вид

*CMPS dst,src.*

Команди *CMPSB, CMPSW, CMPSD* опереандів не мають.

Ці команди порівнюють елементи рядка-джерела довжиною байт, слово або подвійне слово, що знаходиться за адресою *DS:SI* з елементами рядка-приймача, адреса яких визначаються вмістом регістра *ES:DI.* Зміна регістрів, які визначають адреси оперендів не дозволена.

Алгоритм виконання команди полягає у відніманні від вмісту операнда *dst* (приймач) вмісту операнда *src* (джерело). Результат не формується, лише установлюються ознаки результату, які можливо аналізувати і робити висновки про співвідношення операндів між собою.

Після віднімання вміст регістрів *SI* і *DI* автоматично змінюється відповідно значення прапорця *DF*. Якщо прапорець *DF* дорівнює 0 (була виконана команда *CLD*), то вміст регістрів *SI* і *DI* збільшується, якщо *DF* має значення 1 (була виконана команда *STD*), то вміст регістрів *SI* і *DI* зменшується. Збільшення/зменшення відбувається на 1, якщо пересилається байт, на 2, якщо пересилається слово, і на 4, якщо пересилається подвійне слово.

Команді може передувати префікс *REP*, який примушує виконувати порівняння, до поки вміст регістра *CX* не стане дорівнювати 0.

***CMPXCHG (CoMPare and eXCHanGe)*** **–** команда для порівняння двох операндів і обміну даними між ними. Синтаксис команди

*CMPXCHG dst,src.*

Операнд *dst* (приймач) розміщується у регістрі загального призначення або комірці пам'яті, а операнд *src* (джерело) також може бути вмістом регістра загального призначення або комірки пам'яті, але одночасно не можуть бути вмістом комірок пам'яті.

Команда порівнює значення операндів, установлює значення прапорця $ZF = 1$, якщо операнди рівні або $ZF = 0$ в іншому випадку, після чого обмінює вміст операндів між собою.

***SETcc (byte SET on condition)*** – команда установлення вмісту байта при виконанні певної умови. Команда має вигляд

*SETcc src.*

Операнд у цій команді 8-розрядний регістр або комірка пам'яті.

Команда виконує установлення вмісту операнда $01H$, якщо умова виконується, або у $00H$, якщо не виконується.

Значення модифікатора кода операції *cc* аналогічне значенню модифікатора команд умовних переходів, де вони будуть розглянуті.

Сканування елементів рядка байтів, слів, подвійних слів відбувається в залежності від довжини елемента рядка за допомогою команд:

***SCAS (SCAn String)*** – команда для сканування елементів рядка;

***SCASB (SCAn String Byte)*** – команда для сканування елементів рядка байтів;

***SCASW (SCAn String Word)*** – команда для сканування елементів рядка слів;

***SCASD (SCAn String Double word)*** – команда для сканування елементів рядка подвійних слів;

Команда сканування елементів рядка має узагальнений вид

*SCAS dst.*

Команди *CMPSB, CMPSW, CMPSD* опереандів не мають.

Алгоритм виконання команди полягає у відніманні елемента рядка – операнда *dst* (приймач), який адресується тільки регістром *ES:DI*, від вмісту акумулятора *AL* (байт), *AX* (слово), *EAX* (подвійне слово). Результат не формується, лише установлюються ознаки результату, які можливо аналізувати і робити висновки про співвідношення операндів між собою.

Після віднімання вміст регістра *DI* автоматично змінюється відповідно значення прапорця *DF*. Якщо прапорець *DF* дорівнює 0 (була виконана команда

*CLD*), то вміст регістру *DI* збільшується, якщо *DF* має значення 1 (була виконана команда *STD*), то зменшується. Збільшення/зменшення відбувається на 1, якщо пересилається байт, на 2, якщо пересилається слово, і на 4, якщо пересилається подвійне слово.

Команді може передувати префікс *REP*, який примушує виконувати порівняння поки вміст регістра *CX* не стане дорівнювати 0.

**Команди логічних операцій**

До цієї групи команд відносяться команди, які при роботі використовують правила алгебри логіки. До таких команд відносяться команди, яки виконують логічні операції над операндами, команди обробки біт і команди зсувів.

Логічні операції мови Асемблер-86:

***AND (logical AND)*** – команда логічного множення операндів (кон'юнкція).

***OR (logical OR)*** – команда логічного додавання операндів (диз'юнкція).

***XOR (logical eXclusive OR)*** – логічне виключне АБО.

***TEST (TEST operand)*** – логічне ТА. Використовується для логічного порівняння операндів.

***NOT (NOT operand)*** – інвертування операнда.

Узагальнене представлення логічних команд має вигляд:

*AND dst,src.*
*OR dst,src.*
*XOR dst,src.*
*TEST dst,src.*
*NOT src.*

В усіх командах операндами можуть бути 8-, 16-, 32-розрядні регістри загального призначення і пам'ять з відповідною кількістю комірок. Операнд *src* (джерело) також може бути представлено безпосереднім числом (крім команди *NOT*).

Команди *AND, OR, XOR, TEST* змінюють прапорці таким чином:

*OF* і *CF* – завжди установлюються у нульовий стан;

*SF, ZF, PF* – установлюються відповідно до результату за тими самими правилами, що і для арифметичних операцій;

*AF* – не визначається.

**Команди зсувів** поділяються на арифметичні і логічні. Для логічних зсувів характерно, що біт, який виходить за межі регістра, втрачається, а на місце біта, який зсунувся, у регістр записується 0. При виконанні арифметичних зсувів праворуч знаковий біт не втрачається, а зберігається у сусідньому, тим самим зберігаючи знак числа.

Асемблер включає такі команди зсувів:

***SHL (Shift logical Left)*** – зсунути логічно ліворуч;

***SAL (Shift Arithmetic operand Left)*** – зсунути арифметично ліворуч;

***SHR (Shift logical Right)*** – зсунути логічно праворуч;

***SAR (Shift Arithmetic operand Right)*** – зсунути арифметично праворуч;

***RCL (Rotate operand through Carry flag Left)*** – зсунути циклічно ліворуч через перенесення;

***RCR (Rotate operand through Carry flag Right)*** – зсунути циклічно праворуч через перенесення;

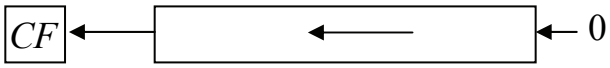
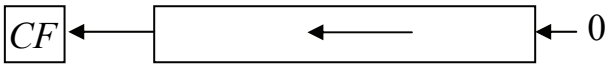
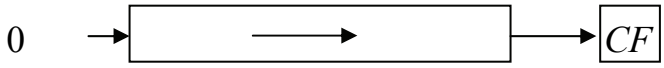
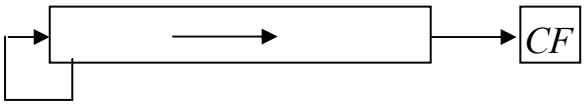
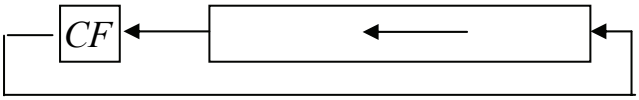
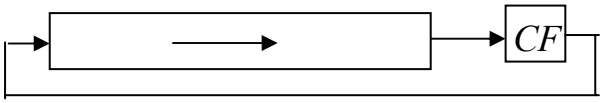
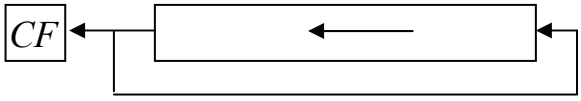
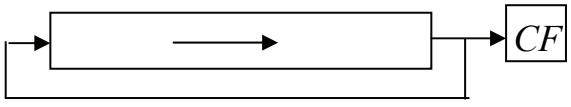
***ROL (Rotate operand Left)*** – зсунути циклічно ліворуч;

***ROR (Rotate operand Right)*** – зсунути циклічно праворуч.

Пояснення алгоритму роботи і синтаксис кожної з команд зсувів подано у табл. 9.3.

У таблиці прийнято умовні позначення: *opr* – операнд (регістр або комірка пам'яті, вміст яких зсувається); *cnt*– кількість зсувів (може приймати значення 1 або задаватися вмістом регістра-лічильника *CL*).

Таблиця 9.3 – Команди зсувів

| Мнемоніка та формат команди | Опис виконання |
|---|---|
| *SHL opr,cnt* |   |
| *SAL opr,cnt* |   |
| *SHR opr,cnt* |   |
| *SAR opr,cnt* |   |
| *RCL opr,cnt* |   |
| *RCR opr,cnt* |   |
| *ROL opr,cnt* |   |
| *ROR opr,cnt* |   |
| | *opr* |

**Контрольні питання:**

1 Які команди додавання операндів у мові Асемблер Ви знаєте? Чим вони відрізняються одна від одної?

2 Чим різняться команди *SUB* і *SBB*? Поясніть на прикладі.

3 Як виконується команда *DEC CX*?

4 Як виконується команда *MUL* і в яких регістрах може розміщуватися операнд *src* (джерело) і операнд *dst* (приймач)?

5 Як виконується команда *DIV* і в яких регістрах може розміщуватися операнд *src* (джерело) і операнд *dst* (приймач)?

6 Для чого у командах ділення використовується команда *CWD*?

7 Для чого використовується команда *CMP*? Де розміщується результат виконання цієї команди?

8 Які команди логічних операцій Ви знаєте?

9 Який результат буде отримано при виконанні фрагмента програми:

```
MOV AX,AFH
SHL AX,1
```

10 Який результат буде отримано при виконанні фрагмента програми:

```
MOV AX,AFH
ROR AX,1
```

11 Який результат буде отримано при виконанні фрагмента програми:

```
MOV AX,AFH
RCL AX,1
```

**Контрольні питання підвищеної складності:**

1 Чому при виконанні арифметичних команд над двійково-десятковими числами виникає потреба у корекції результату?

2 Як виконується корекція результату при виконанні арифметичних команд над двійково-десятковими числами?

3 Які прапорці формуються при виконанні логічних операцій?

4 Які команди зсувів можливо використовувати як команди множення і ділення на 2?

### *9.3.4 Команди умовних та безумовних переходів*

**Вхідний контроль:**

1 Які ознаки результату, що формуються у МПС фірми *Intel*, Ви знаєте?

2 В якому випадку формується ознака *ZF*?

3 Яким чином формується ознака знаку результату *SF*?

4 Чим відрізняється формування ознак перенесення *CF* і переповнення *OF*?

5 Наведіть приклад реалізації умовного переходу будь-якою мовою високого рівня.

6 Наведіть приклад реалізації безумовного переходу будь-якою мовою високого рівня.

До групи таких команд відносяться команди, які можуть зробити перехід не до наступної за даною командою, а до команди, яка знаходиться в іншій комірці програмної пам’яті, адреса якої визначається адресою переходу.

Команди передачі керування підрозділяються на команди безумовних переходів, умовних переходів, викликів і повернення з підпрограм, команди переривань.

**Команди безумовних переходів**

При виконанні таких команд виконується зміна вмісту регістра *PC* або одночасно *PC* і *CS*. Команди такого типу мають довжину 2...5 байтів у залежності від типу переходу і довжини зміщення у команді, яким задається адреса переходу.

**JMP (JuMP)** – команда безумовного переходу

Команда має вигляд

*JMP мітка.*

Мітка може відповідати 8-, 16- або 32-розрядному зміщенню відносно наступної за *JMP* команди. При цьому змінюється лише вміст *EIP/IP*. Якщо мітка у команді – символічний ідентифікатор комірки пам'яті (16-, 32- або 48-розрядна адреса, то Асемблер визначає його як адресу, по якій необхідно зробити перехід. Така адреса може відповідати, як близькому, так і далекому переходу.

**Команди умовних переходів**

При виконанні цих команд мова Асемблера робить аналіз певних ознак результатів і за результатами аналізу здійснює або не здійснює передачу керування.

***Jcc (Jump if condition)*** **–** команда умовного переходу.

Перелік команд умовного переходу й умов, які перевіряються, подано у табл. 9.4.

Таблиця 9.4 – Команди умовних переходів

| № пп. | Дія | Мнемоніка та формат | Альтернативна мнемоніка | Умова, що перевіряється |
|---|---|---|---|---|
| 1 | Перейти, якщо нуль або дорівнює | *JZ ADDR* | *JE ADDR* | *ZF*=1 |
| 2 | Перейти, якщо не нуль або не дорівнює | *JNZ ADDR* | *JNE ADDR* | *ZF*=0 |
| 3 | Перейти, якщо знак установлено | *JS ADDR* | | *SF*=1 |
| 4 | Перейти, якщо знак скинуто | *JNS ADDR* | | *SF*=0 |
| 5 | Перейти, якщо є переповнення | *JO ADDR* | | *OF*=1 |

Продовження табл. 9.4

| № п.п | Дія | Мнемоніка та формат | Альтернативна мнемоніка | Умова, що перевіряється |
|---|---|---|---|---|
| 6 | Перейти, якщо немає переповнення | *JNO ADDR* | | *OF*=0 |
| 7 | Перейти, якщо паритет установлений | *JP ADDR* | *JPE ADDR* | *PF*=1 |
| 8 | Перейти, якщо паритет скинуто | *JNP ADDR* | *JPO ADDR* | *PF*=0 |
| 9 | Перейти, якщо вище (без знака) | *JA ADDR* | | *CF*=0, *ZF*=0 |
| 10 | Перейти, якщо нижче (без знака) | *JNA ADDR* | | *CF*=1, *ZF*=1 |
| 11 | Перейти, якщо вище/ або дорівнює (без знака) | *JAE ADDR* | | *CF*=0 |
| 12 | Перейти, якщо нижче/ або дорівнює (без знака) | *JBE ADDR* | | *CF*=1, *ZF*=1 |
| 13 | Перейти, якщо нижче/не вище або дорівнює (без знака) | *JB ADDR* | *JNAE ADDR* *JC ADDR* | *CF*=1 |
| 14 | Перейти, якщо не нижче/ вище або дорівнює (без знака) | *JNBE ADDR* | *JNB ADDR* *JNC ADDR* | *CF*=0 |
| 15 | Перейти, якщо є перенесення/позика | *JC ADDR* | | *CF*=1 |
| 16 | Перейти, якщо перенесення/позики немає | *JNC ADDR* | | *CF*=0 |
| 17 | Перейти, якщо менше/ не більше або дорівнює ( зі знаком) | *JL ADDR* | *JNGE ADDR* | ((*CF*) *XOR*(*OF*))=1 |
| 18 | Перейти, якщо не менше/ більше або дорівнює ( зі знаком) | *JNL ADDR* | *JGE ADDR* | ((*CF*) *XOR*(*OF*))=0 |

Закінчення табл. 9.4

| № п.п | Дія | Мнемоніка та формат | Альтернативна мнемоніка | Умова, що перевіряється |
|---|---|---|---|---|
| 19 | Перейти, якщо менше або дорівнює/не більше (зі знаком) | *JLE ADDR* | | ((*SF*) *XOR*(*OF*)*OR*(*ZF*))=1 |
| 20 | Перейти, якщо не менше або дорівнює/більше (зі знаком) | *JNLE ADDR* | *JG ADDR* | ((*SF*) *XOR*(*OF*)*OR*(*ZF*))=0 |

Використання і пояснення роботи деяких команд наведено у наступній програмі:

```
M1:     NOP          ; Команда NOP (No Operation) – немає
                     ; операції, рядок помічено міткою M1
M2:     JMP M4       ; Операції немає, рядок помічено міткою M2
M3:     JMP M5       ; Команда безумовного переходу на мітку M4
M4:     MOV AX,5678H ; Завантаження до регістра AX числа 5678H
        MOV BX,1234H ; Завантаження до регістра BX числа 1234H
        CMP AX,BX    ; Порівняння вмісту регістра AX з вмістом
                     ; регістра BX
        JZ M1        ; Умовний перехід на мітку M1, якщо результат
                     ; порівняння дорівнює 0 (операнди рівні).
                     ; У нашому випадку умовний перехід не
                     ; відбувається, тому що операнди не рівні і
                     ; прапорець ZF скинуто в 0
        JNS M2       ; Умовний  перехід на мітку M2, якщо результат
                     ; порівняння додатний (прапорець SF = 0). У
                     ; нашому випадку не відбувається, тому що
                     ;          _ 1234H
                     ;            5678H
                     ;            BBBCH   (прапорець SF = 1)
        JLE M3       ; Умовний перехід на мітку M3, якщо вміст
                     ; операнда dst (приймач) менший за операнд
                     ; src (джерело). В нашій програмі відбувається
M5:     NOP
```

**Команди виклику і повернення з підпрограми**

***CALL (CALL)*** – команда виклику підпрограми (процедури). Може відповідати близькому або далекому переходу до підпрограми, в залежності від початкової адреси розміщення підпрограми. При зверненні до підпрограми в стеку зберігається адреса повернення до основної програми (адреса комірки пам’яті в якій розміщується команда *CALL*).

Команда має вигляд

*CALL ADDR.*

Адреса переходу до підпрограми може задаватися міткою, вмістом регістра або комірки пам'яті.

Мітка (вміст регістра або комірки пам'яті) може відповідати зміщенню в поточному сегменті команд, куди передається управління. При цьому змінюється лише вміст *EIP/IP*. Якщо здійснюється далекий перехід, то змінюється вміст *EIP/IP* і вміст регістра *CS*.

***RET (RETurn)* –** команда повернення з підпрограми до основної програми.

Команда має вигляд

*RET.*

Команда відновлює вміст регістра *EIP/IP*, шляхом завантаження із стека, значення збереженого при виконанні команди *CALL*.

Команди виклику і повернення з підпрограми не змінюють значення регістра ознак.

### *9.3.5 Команди організації циклів*

**Вхідний контроль:**

1 Які види циклічних програм Ви знаєте?

2 В чому полягає різниця між арифметичним та ітераційним циклами?

3 В якому регістрі організується програмний лічильник циклів?

4 Наведіть приклади організації циклів будь-якою мовою високого рівня.

5 Покажіть на прикладах, як задається кількість повторень циклів будь-якою мовою високого рівня.

До команд управління циклом відносяться:

– команди організації циклу з лічильником *ECX/CX*;

– команди організації циклу з лічильником *ECX/CX* з можливістю дострокового виходу з циклу за додатковою умовою.

До першої групи відносяться команди:

***JCXZ (Jump if CX=Zero) –*** перехід, якщо *CX* дорівнює 0;

***JECXZ (Jump if ECX=Zero) –*** перехід, якщо *ECX* дорівнює 0.

Операндом в усіх командах слугує зміщення (мітка), за допомогою якого визначається адреса переходу.

Команди перевіряють вміст відповідного лічильника і якщо його вміст дорівнює 0, відбувається перехід на вказане у команді зміщення (мітку), а якщо не дорівнює, то виконується наступна команда.

До команд другої групи відносяться команди:

***LOOP (LOOP control by register CX)*** – управління циклом за вмістом регістра *CX*;

***LOOPE/LOOPZ (LOOP control by register CX not equal 0 and ZF=1)*** – управління циклом за вмістом регістра CX з урахуванням значення прапорця *ZF*;

***LOOPNE/LOOPNZ (LOOP control by register CX not equal 0 and ZF=0)*** – управління циклом за вмістом регістра CX з урахуванням значення прапорця *ZF*.

Команда *LOOP* <адреса> забезпечує умовний перехід для циклічного виконання ділянки програми. Кількість повторень циклу визначається вмістом регістра *ECX/CX*. Усі різновиди команди *LOOP* автоматично виконують декремент вмісту *ECX/CX* і зупиняють виконання циклу, якщо вміст лічильника дорівнює 0.

Команди *LOOPE/LOOPZ* є різновидами однієї команди, так само як і команди *LOOPNE/LOOPNZ*. Алгоритм виконання цих команд однаковий. Команди декрементують вміст *ECX/CX* й аналізують його вміст і значення прапорця *ZF*, якщо вміст *ECX/CX* дорівнює 0, то виконується наступна за *LOOPxx* команда, якщо вміст *ECX/CX* дорівнює 1, то виконується перехід до початку циклу. Якщо значення *ZF* = 0, то команди *LOOPE/LOOPZ* виконують вихід з циклу, а команди *LOOPNE/LOOPNZ* повертаються до початку циклу. Для значення *ZF* = 1 команди виконується навпаки. Команди *LOOPNE/LOOPNZ* можливо використовувати для пошуку першого нульового елементу у рядку даних, якщо безпосередньо перед цією командою виконати порівнювання елемента з 0, а команди *LOOPE/LOOPZ* для пошуку першого ненульового елемента.

**Контрольні питання:**

1 Які групи команд управління циклом Ви знаєте?

2 Що слугує операндом у командах управління циклом?

3 Як виконується декремент вмісту регістра-лічильника *CX* у програмі при використанні команд *JCXZ* і *LOOP*?

4 Які прапорці перевіряє команда *LOOPE*?

**Контрольні питання підвищеної складності:**

1 Для чого використовується мітка в командах передачі управління?

2 Яка команда буде виконуватися після виконання команди *LOOPNZ M*1

```
M1: MOV AX,DX
    …
    LOOPNZ M1
    NOP,
```

якщо до її виконання в регістрі *CX* було записано число 0001*H*?

3 Як можливо використовувати команди *LOOPE/LOOPZ* при обробленні рядків даних?

4 Скільки разів буде виконуватись команда *LOOPE M*1, якщо до початку циклу в регістр *CX* був записаний нуль?

5 Наведіть фрагмент програми, в якому вихід з циклу здійснювався б за умовою *JPO M*1.

## 9.4 Створення програм на мові Асемблер-86

### *9.4.1 Лінійні програми*

**Вхідний контроль:**

1 Які ділянки програм називаються лінійними?

2 Команда з якою адресою буде виконуватись після команди

7000:0100 MOV AX,7000H;?

У сучасних додатках рідко використовується програмне забезпечення, цілком написане мовою Асемблера. Частіше мова Асемблера використовується разом з мовою високого рівня, наприклад, *C/C*$^{++}$. Частина програми, написана мовою Асемблера, зазвичай призначена для керування периферійними пристроями, оскільки рішення таких задач на мовах високого рівня є більш складним і менш ефективним. Крім того, програми на мові Асемблер виконуються значно швидше, ніж написані будь-якою мовою високого рівня.

Основним принципом отримання ефективних програм є ефективний розподіл та використання ресурсів процесора. Оптимізація програми реалізується за рахунок мінімізації пересилань, використання вказівників на дані замість самих даних при роботі зі складними структурами, розміщення структури даних в одній локальній області пам'яті, розподіл регістрів для локальної ділянки програми тощо. Так, перш за все розподіляються спеціалізовані регістри, а решта виділяється для зберігання найбільш часто використовуваних даних. Дані, які використовуються одноразово, слід зберігати у пам’яті. Акумулятор виділяється для зберігання оперативних результатів.

Потрібна міра оптимальності програми визначається умовами її експлуатації і залежить від багатьох факторів. Для рідко використовуваних програм високий рівень оптимізації не є обов’язковий. Простіше за все оптимізація програми реалізується на лінійних послідовностях команд. У середині лінійних ділянок робиться присвоювання, виконуються операції над даними, вилучаються зайві операції за рахунок одноразового програмування загальних виразів з подальшим включенням їх у циклічні програми. Більш складною задачею є виконання еквівалентних адекватних перетворювань на лінійних ділянках з метою спрощення аналітичних виразів при збереженні достатньої точності та надійності обчислень.

*Приклад 9.4.1* Ввести з портів з адресами 3*F*8*H* та 2*F*8*H* дані та запам’ятати їх у суміжних комірках пам’яті, починаючи з адреси 0020H у

сегменті даних. Округлити їх до 4-х розрядів, упакувати в один байт і зберегти у пам'яті за адресою 0030*H* у сегменті даних.

Під упакуванням розуміють розміщення у старшому півбайті першого числа, а в молодшому півбайті – другого числа.

Задача вирішується за виконанням наступного фрагмента програми:

```
MOV DX,3F8H    ; Завантаження у регістр DX адреси порту 3F8H
MOV SI,0020H   ; Завантаження в індексний регістр SI адреси комірки
               ; пам'яті для першого даного
MOV DI,0030H   ; Завантаження у індексний регістр DI адреси комірки
               ; пам'яті для упакованого даного
MOV CL,04H     ; Завантаження у CL кількості операції зсувів
XOR CH,CH      ; Обнулення регістра CH
IN AL,DX       ; Введення в акумулятор першого даного з порту
MOV CH,AL      ; Запам'ятовування першого даного у регістрі CH
MOV [SI],AL    ; Запам'ятовування першого даного у пам'яті
MOV DX,2F8H    ; Завантаження у регістр DX адреси порту 2F8H
IN AL,DX       ; Введення в акумулятор другого даного з порту
INC SI         ; Нарощування адреси комірки пам'яті для другого
               ; даного
MOV [SI],AL    ; Запам'ятовування другого даного у пам'яті
AND CH,F0H     ; Округлення першого даного
AND AL,F0H     ; Округлення другого даного
SHR AL,CL      ; Переміщення округленого другого даного на місце
               ; другого півбайта
OR CH,AL       ; Упаковування даних в один байт
MOV [DI],CH    ; Запам'ятовування упакованих даних
NOP
```

Припустимо, що перше дане *D*1 становить 12*H* (00010010*B*), а друге *D*2 – 34*H* (00110100*B*). Округлення першого даного дасть півбайт, який дорівнює 0001*B*, а другого – 0011*B*. Упаковане дане складатиме 00010011*B* або 13*H*.

Програма складена з урахуванням подальшої можливості введення із портів масивів даних, їх упакування та запам'ятовування.

*Приклад 9.4.2* Число -28*H* записати у регістри *DL* та *DH* у прямому коді двома способами. Наступні фрагменти програми виконують ці записи:

```
а) MOV DL,-28H        б) MOV DH,-28H
   NEG DL                NOT DH
   OR DL,80H             ADD DH,81H
```

Запис будь-якого від'ємного числа здійснюється у регістр у доповнювальному коді, тобто у регістрах *DL* та *DH* після виконання перших двох команд буде записане число -28*H* у доповнювальному коді, тобто число

*D*8*H*. Для подання цього даного у прямому коді можна використати команду *NEG DL*, яка подасть це число без знака, а саме 28*H*, та команду *OR DL*,80*H*, яка установить 1 у старшому розряді. В результаті отримаємо прямий код числа -28*H*, який дорівнює *A*8*H* (фрагмент а)). У фрагменті б) число *D*8*H*, записане у регістр *DH* у доповнювальному коді, інвертується і до нього додається число 81*H*. Ця операція установлює 1 у старшому розряді результату і додає 1 до молодшого розряду; у регістрі *DL* буде записане також число *A*8*H*.

*Приклад 9.4.3* Охарактеризувати ознаками *ZF, SF* та *PF* дане, яке зберігається у регістрі *AL*. Команда

*OR AL,AL*

не змінює дане, яке зберігається в *AL*, і виставляє зазначені ознаки. У припущенні, що дане є число 56*H*, ознаки дорівнюватимуть:

*ZF*=0;
*SF*=0;
*PF*=1.

*Приклад 9.4.4* Знайти вміст акумулятора та ознак *CF, SF, AF* після виконання фрагментів програм:

| а) | б) | в) |
|---|---|---|
| MOV AL,EH | MOV AL,2AH | MOV AL,2AH |
| SUB AL,2AH | SUB AL,4EH | SUB AL,E4H |
| NOP | NOP | NOP |

При виконанні віднімання отримуємо вміст акумулятора:

| а) | б) | в) |
|---|---|---|
| _4*EH* | _2*AH* | _2*AH* |
| 2*AH* | 4*EH* | *E*4*H* |
| 24*H* | *DCH* | 46*H* |
| *CF*=0 | *CF*=1 | *CF*=1 |
| *AF*=0 | *AF*=1 | *AF*=0 |
| *SF*=0 | *SF*=1 | *SF*=0 |

Ознака *CF* установлюється в 1, якщо мала місце позика, тобто у фрагментах б) та в). При відніманні із меншого числа більшого ознака *CF* завжди установлюється в 1, а ознака знаку *SF*=1 тільки у разі отримання "коректного" результату, тобто за відсутності переповнення розрядної сітки. Для виявлення меншого з двох чисел слід користуватися ознакою *CF*, якщо аналізується результат виконання команд віднімання або порівняння.

*Приклад 9.4.5* При завантаженні комп'ютера тестуються наявні асинхронні адаптери й ініціалізуються перші два з них: *COM*1 та *COM*2, які мають базові адреси 0000:0400*H* та 0000:0300*H*. Наступний фрагмент програми дозволяє прочитати з регістра керування з адресою 3FBH порту *COM*1 байт стану режиму адаптера:

```
MOV DX,3FBH ; Завантаження адреси керувального регістра у DX
IN AL,DX    ; Введення поточного режиму адаптера
```

При правильній ініціалізації байт стану дорівнюватиме 000X0011*B* = 03*H* або 13*H*. Це відповідає такому режиму: немає контролю на парність (*D*3 = 0, *D*4 = *X*), один стоповий біт (*D*2 = 0), кількість бітів дорівнює 8 (*D*1 = 1, *D*2 = 1).

*Приклад 9.4.6* Два байти *D*1 та *D*2 є 3- та 5-м елементами масиву, що починається з ефективної адреси 0010H. Інвертувати перший байт, вирізати 0, 2, 7 розряди другого; отримані результати зберегти у стеку. Припустимо, що перший байт *D*1 дорівнює 25*H*, а другий *D*2 – 73*H*. Виконаємо вказані операції порозрядно та вкажемо ознаки результату (прапорці) на кожному кроці виконання. Сегменти стека та даних є суміщені.

$F1 = (\overline{D1}) = \overline{00100101} = 11011010 = DAH.$

$F2 = D2 \wedge mask;\ mask = 01111010 = 7AH.$

$$F2 = \wedge \begin{array}{r} 01110011 \\ 01111010 \\ \hline 01110010 \end{array} \qquad F2 = 72H \quad OF = 0,\ CF = 0,\ SF = 0,\ AF = 0,\ ZF = 0,\ PF = 1.$$

Задамо довільно 16 елементів масиву байтів, з яких третій дорівнює 25*H*, а п'ятий дорівнює 73*H*.

7000:0010 12 34 56 25 43 73 43 54 65 76 82 37 15 13 14 61 52.

Вказівник стека задамо таким, який дорівнює 2080*H*.
Структурна схема програми наведена на рис. 9.12.
Фрагмент програми, яка вирішує цю задачу, має вид

```
MOV AX,7000H      ; Завантаження
MOV DS,AX         ; сегментних
MOV SS,AX         ; регістрів
MOV SP,2080H      ; Завантаження вказівника стека
MOV BX,0010H      ; Завантаження базового регістра
MOV AL,[BX+03]    ; Пересилання першого байта до AL
NOT AL            ; Інвертування першого байта
```

```
MOV AH,[BX+05]      ; Пересилання другого байта до AH
AND AH,7AH          ; Змінення другого байта
PUSH AX             ; Запам’ятовування результатів у стеку
```

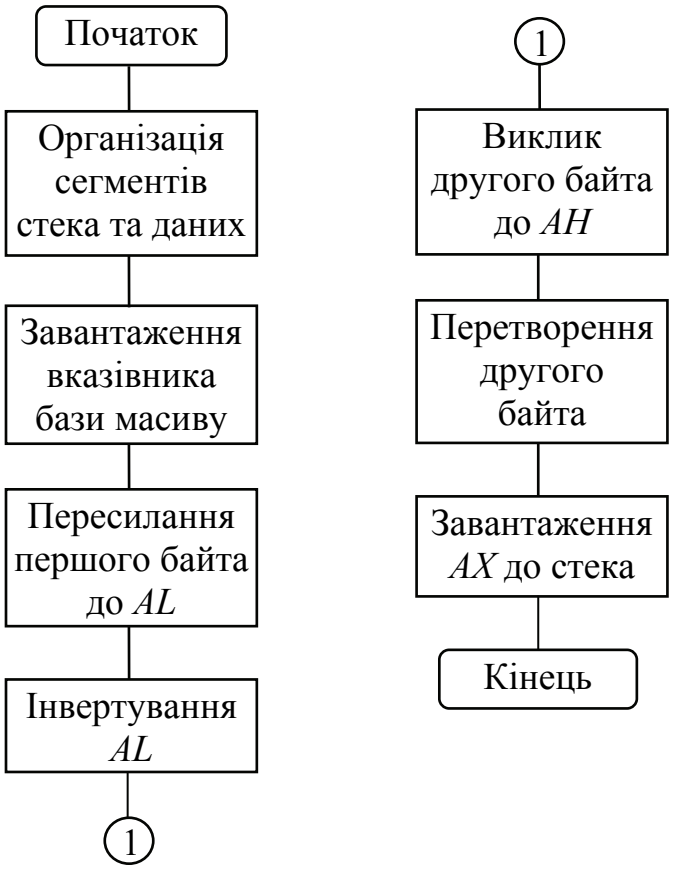



Рисунок 9.12 – Структурна схема програми

Програма виконується так.

Усі команди пересилань та команда *NOT* не змінюють прапорці; результатом виконання команди *NOT* є число *DAH*, що вміщується у регістрі *AL*, а результатом команди *AND* є число 72*H*, що вміщується в *AH*; ця команда установлює прапорці *OF* = 0, *CF* = 0, *SF* = 0, *AF* = 0, *ZF* = 0, *PF* = 1.

*Приклад 9.4.7* Виконати операцію виключного АБО над двома словами, *D*1 та *D*2, що розміщені у пам’яті. Помножити результат на беззнакове число 33H і розташувати добуток на місці другого слова. Перше знаходиться у масиві, що починається зі зміщення 10*H* та адресоване вмістом індексного регістра *SI*, а друге знаходиться у масиві, що починається зі зміщення 20*H* та адресується вмістом індексного регістра *DI* і зміщенням 2*H*. Початкова адреса сегмента даних є 7000*H*.

Створимо два масиви, починаючи зі зміщення 10*H* та 20*H* відповідно. Припустимо, що у регістрі *SI* міститься число 6*H*, а у регістрі *DI* – число 2*H*.

7000:0010 12 34 56 25 40 43 71 43 65 76 82 37 15 13 18 60
7000:0020 14 36 58 27 45 73 48 54 65 76 82 39 17 15 14 61

Тоді перше слово $D1$ становить $4371H$, а друге $D2$ дорівнює $7345H$. Перший проміжний результат $F1$ становить:

$F1=D1 \oplus D2=3034H$.

$$F1 = \oplus \begin{array}{l} 0100001101\ 110001 \\ \underline{0111001101\ 000101} \\ 0011000000\ 110100 \end{array}$$

Ознаки результату становлять $OF = 0$, $CF = 0$, $SF = 0$, $AF = 0$, $ZF = 0$, $PF = 0$.

Другий результат $F2$ становить:

$F2 = F1 \times 33H = 3034H \times 33H = 99A5CH$.

Ознаки результату дорівнюють: $OF = 1$, $CF = 1$, $SF = 1$, $AF = 1$, $ZF = 0$.

Програма має вигляд:

```
MOV AX,7000H        ; Організація
MOV DS,AX           ; сегмента даних
MOV BX,0010H        ; Завантаження до BX початкової адреси першого
                    ; масиву
MOV DI,2H           ; Завантаження
MOV SI,6H           ; індексних регістрів
XOR DX,DX           ; Обнулення DX
MOV AX,[BX+SI]      ; Завантаження першого слова до AX
MOV BX,0020H        ; Завантаження до BX початкової адреси другого
                    ; масиву
XOR AX,[BX+DI+2]    ; Складання за модулем 2 з другим словом
MOV CX,33H          ; Завантаження множника до CL
MUL CX              ; Множення на константу 33H
MOV [BX+DI+2],AX    ; Пересилання молодшого слова добутку на місце
                    ; другого слова
MOV [BX+DI+4],DX    ; Пересилання старшого слова добутку до другого
                    ; масиву
```

Після виконання програми в регістр $AX$ буде записане число $9A5CH$, а в регістр $DX$ – число $0009H$.

**Контрольні питання:**

1 Як подаються числа беззнакові та зі знаком у МП сім'ї *Intel*?

2 Подані в яких системах числення дані можуть оброблюватись у МП сім'ї *Intel*?

3 Четвертий елемент першого масиву дорівнює 32*H*. Поділити його на п'ятий елемент другого масиву, що дорівнює 4*H*, усіма відомими вам способами.

4 Інвертувати слово, що становить п'ятий елемент першого масиву; поділити результат на третій елемент другого масиву. Частку разом з регістром прапорців запам'ятати у стеку.

5 Знайти логічний добуток четвертого байта першого масиву та третього байта другого масиву. Результат помножити на 44*H*, добуток запам'ятати за ефективною адресою, що визначається вмістом регістра *DI*.

6 Заповнити масив з чотирьох слів, починаючи з адреси *DS:*50, використовуючи команду *STOSW*.

7 Порівняти нульовий елемент першого масиву з другим елементом другого масиву. Запам'ятати у стеку вміст регістра прапорців та суму вказаних елементів.

**Контрольні питання підвищеної складності:**

1 Завантажити одночасно регістр сегмента даних та акумулятор двобайтовими елементами першого масиву, починаючи з другого елемента за допомогою прямого адресування.

2 Сумістити сегмент даних та додатковий сегмент даних.

### *9.4.2 Розгалужені програми*

**Вхідний контроль:**

1 За якими умовами (прапорцями) можна реалізувати умовні переходи?

2 Напишіть команду безумовного переходу на адресу команди, яка знаходиться в іншому сегменті кодів.

У послідовних ділянках програми виконання команд відбувається згідно з їхнім розташуванням. Однак за умовами тієї чи іншої задачі може виникнути необхідність зміни порядку виконання команд. Це реалізується за допомогою команд, призначених для передачі керування з однієї точки програми до іншої. При виконанні будь-якої команди відбувається нарощування вмісту вказівника команд *IP*, що дозволяє одержати адресу наступної виконуваної команди у лінійних програмах. Якщо ж треба перейти до потрібної адреси у програмі, використовують безпосередню зміну значення вказівника команди *IP* та/або значення сегментного регістра коду *CS* за допомогою команд умовного та безумовного переходів. Розрізняють команди дальнього (*far*), близького (near) та короткого (*short*) переходів. При дальньому переході змінюється не тільки вміст вказівника команд *IP*, але й вміст сегментного регістра коду *CS*, тобто можливий перехід до будь-якої комірки пам'яті, на відстань, більшу ніж $2^{16}$

адрес комірок пам'яті, а не тільки у межах сегмента коду. Це дає можливість реалізувати міжсегментні переходи до будь-якого сегмента, наприклад, при виконанні підпрограм. Цей сегмент потім стає поточним сегментом коду. При близькому переході змінюється тільки вміст вказівника команд *IP* у межах [-$2^{16}$... +$2^{16}$-1] адрес, а вміст сегментного регістра *CS* не змінюється, тобто перехід можливий тільки у сегменті коду *CS*. При короткому переході межі змінення вказівника команд *IP* становлять [-128...+127] адрес.

**Безумовний перехід**

Команди безумовного переходу (БП) дозволяють перейти на задану адресу програми без запам'ятовування адреси повернення. Є можливими три форми мнемонічного подання команди безумовного переходу:

```
JMP 0110H          ; Прямий перехід на адресу 0110H
JMP [offset]       ; Перехід за зміщенням відносно поточного вмісту
                   ; вказівника команд IP
JMP [operand]      ; Непрямий перехід; адреса переходу вміщується до
                   ; регістрів загального призначення або до комірок
                   ; пам'яті
```

При прямому переході адреса переходу замінює вміст вказівника команд і, за необхідності, вміст сегментного регістра коду.

Зміщення являє собою 8-, 16- та 32-розрядне число (останнє тільки для МП *I*80386 та старших).

Зміщення вважається знаковим, тому є можливий перехід у бік початку або кінця програми. Змінюється тільки вказівник команд *IP*, а сегментний регістр *CS* залишається незмінним.

При непрямому переході адреса продовження програми визначається вмістом одного із 16- (*AX, BX, CX, DX, SI, DI, BP, SP*) або 32- (*EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP* – для старших моделей) розрядних регістрів загального призначення або ділянки пам'яті. У першому випадку реалізується короткий або близький перехід, а у другому і третьому випадках перехід може бути також і далеким. Вміст вказаного у команді регістра або ділянки пам'яті є новим вмістом вказівника команд *IP* та за необхідності сегментного регістра коду *CS*. У наступних фрагментах програми показані можливі варіанти указання адреси при безумовних переходах.

```
  CS : IP
7000 : 30 MOV DS,AX          ; Завантаження DS  з AX
7000 : 32 MOV AX,1234H       ; Завантаження AX даним 1234H
7000 : 35 JMP 0039H          ; Обхід команди MOV CX,AX
7000 : 37 MOV CX,AX          ; Завантаження CX з AX
7000 : 39 MOV DX,AX          ; Завантаження DX з AX
```

```
  CS : IP
7000 : 30 MOV DS,AX          ; Фрагмент
7000 : 32 MOV AX,0039H       ; програми
7000 : 35 JMP AX             ; аналогічний
7000 : 37 MOV CX,AX          ; попередньому
7000 : 39 MOV DX,AX

  CS : IP
7000 :  30 MOV AX,003CH      ; Фрагмент
7000 :  33 MOV [ 50],AX      ; програми
7000 :  36 JMP [50]          ; аналогічний
7000 :  3A MOV CX,AX         ; попередньому
7000 :  3C MOV DX,AX
```

Прапорці при виконанні команд безумовних переходів не змінюються.

**Умовні переходи**

Умовні переходи (УП) у програмах здійснюються при виконанні вказаних у них умов залежно від одержаного перед УП результату і не реалізуються, якщо зазначені умови не виконані, тоді виконується ділянка програми, що йде безпосередньо за командою умовного переходу. Команди умовних переходів наводяться у табл. 9.4. Для деяких умов існують кілька команд умовного переходу. Використання певної команди залежить тільки від програміста. Умови “вище” або “нижче” відносяться до величин без знака, а “більше” та “менше” до величин зі знаком. Зміщення у командах умовного переходу – це величини у діапазоні -128...+127, тобто вони займають 1 байт. Для МП *I*80386 та старших можна використовувати зміщення, що займає 2 або 4 байти. Вміст регістра *CS* не змінюється.

Якщо ж потрібна адреса знаходиться поза даним сегментом коду, використовують команду умовного переходу з протилежною умовою, а за нею вміщують вже команду безумовного переходу з необхідною адресою.

Достатньо розповсюдженим є випадок, коли розгалуження програм виконується залежно від результату порівняння двох операндів за допомогою команди CMP – порівняння, як знакових, так і беззнакових чисел.

*Приклад 9.4.8* Два однобайтових беззнакових даних *D*1 та *D*2 зберігаються у стеку. Перше дане міститься за молодшою адресою стека, друге – за старшою. Якщо різниця між першим та другим даними є негативна і кількість одиниць в ній непарна, то слід:

– збільшити перше дане на 5*H*;

– інвертувати друге дане;

– отримані результати запам’ятати відповідно за суміжними адресами у сегменті даних зі зміщеннями 10*H* та 11*H*.

У разі парної кількості одиниць у різниці даних необхідно:
– збільшити перше дане на 02*H*;
– збільшити друге слово на 01*H*;
– отримані результати записати за тими самими адресами.
Якщо різниця даних позитивна, їх слід зберегти у стеку.
Структурна схема алгоритму має вид (рис. 9.13).

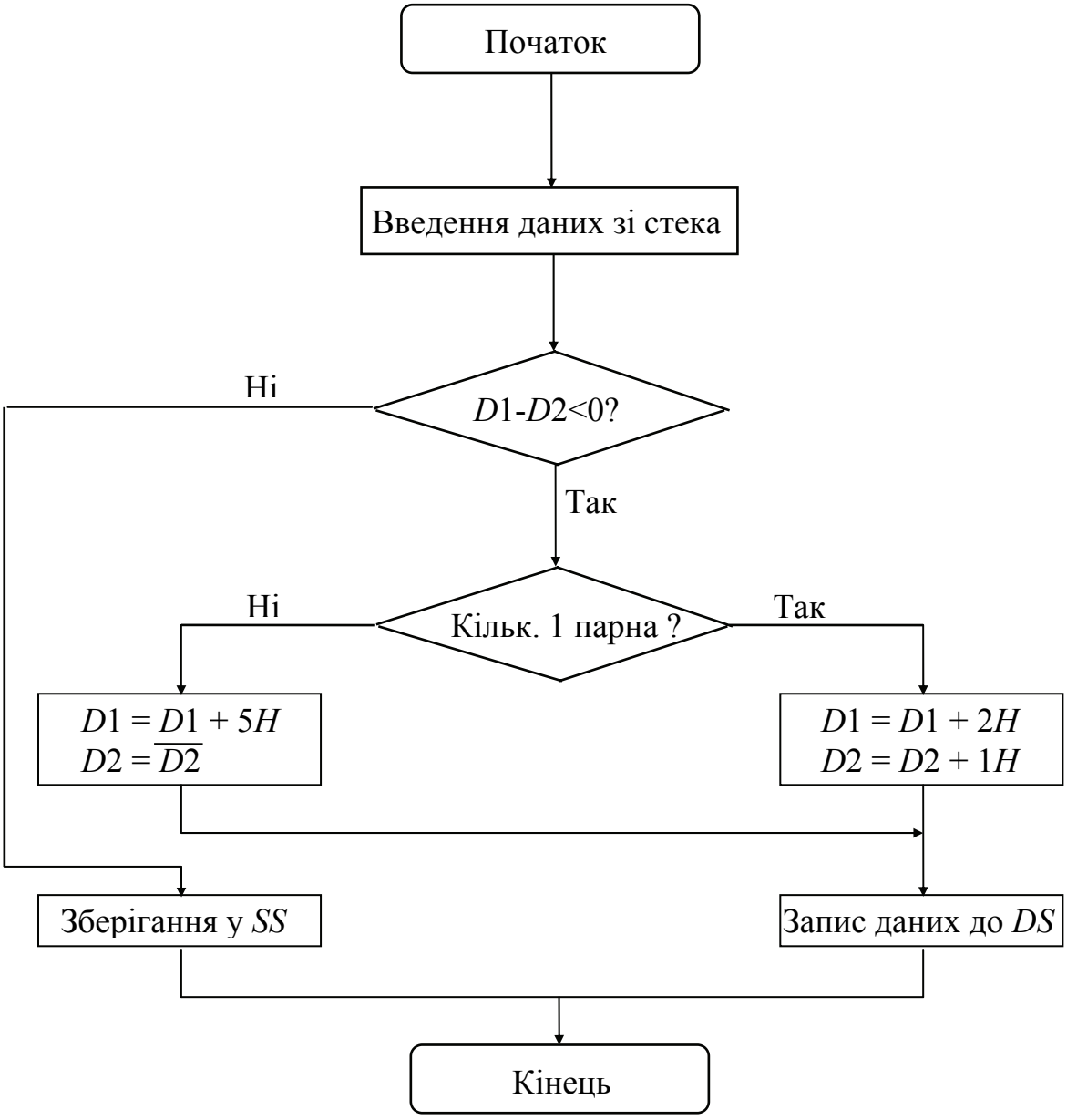



Рисунок 9.13 – Структурна схема алгоритму

Фрагмент програми розв’язання задачі:

```
MOV AX,0F0AH      ; Завантаження даних до AX
PUSH AX           ; Завантаження даних до стека
POP AX            ; Завантаження даних із стека до AX
MOV BX,AX         ; Збереження вмісту AX в BX
SUB BL,BH         ; Різниця даних
```

```
        JNS POSIT       ; позитивна ?
        JP EVEN         ; Ні, кількість одиниць парна?
        ADD AL,05H      ; Ні, зміна першого слова
        NOT AH          ; Зміна другого даного
        MOV [10H],AX    ; Запам’ятовування нових даних у
                        ; сегменті даних
        JMP KINEC       ; БП на кінець програми
EVEN:   ADD AL,02H      ; Так, зміна першого даного
        INC AH          ; Інкрементування другого даного
        MOV [10],AX     ; Запам’ятовування даних у сегменті даних
        JMP KINEC       ; БП на кінець програми
POSIT:  PUSH AX         ; Запам’ятовування вхідних даних у стеку
KINEC:  NOP             ; Кінець програми
```

Перед виконанням програми до стека, враховуючи значення вказівника стека *SP*, треба по черзі завантажувати дані *D*1 та *D*2 так, щоб кожного разу виконувалася одна гілка розгалуженої програми , а комірки пам’яті у сегменті даних зі зміщеннями 10*H* та 11*H* попередньо обнулити, щоб перевірити правильність виконання програми. Якщо перше дане дорівнює 0*AH*, а друге – 0*FH* , то при завантаженні регістра *AX* із стека після виконання команди *POP AX* у регістрі *AL* буде розміщене перше дане – 0*AH*, а у регістрі *AH* – друге дане, 0*FH*. Різниця першого та другого даних після виконання команди *SUB BL, BH* становить *FBH*. Установлюється ознака *SF* = 1, що вказує на одержання негативного результату. Через це буде виконуватися команда *JP*, чергова за списком, а умовний перехід за командою *JNS* не виконується.

Кількість одиниць у різниці непарна, тому установлюється ознака (прапорець) *PF* = 0, і виконується лінійна ділянка програми, що йде за командою УП *JP*, включаючи команду БП на кінець програми. Сам умовний перехід не виконується. У результаті виконання цього фрагмента у комірці пам’яті зі зміщенням 10*H* буде записане число 00*H*, а у комірці пам’яті зі зміщенням 11*H* – число F0*H*.

Якщо вхідні дані становили відповідно 0*FH* та 0*AH*, то їхня різниця буде позитивною – 05*H*, і після її обчислення здійснюється перехід до команди з ефективною адресою 012*F PUSH AX*, за якою вхідні дані будуть розміщені у стеку за попередніми адресами.

Якщо вхідні дані дорівнюють відповідно 0*AH* та 0*EH*, то в їх різниці *FCH* буде парна кількість одиниць, і за ознакою *PF* = 1 команда *JP* 0126 реалізує умовний перехід. У результаті виконання послідовної ділянки програми, включаючи команду БП на кінець програми, у комірці пам’яті зі зміщенням 10H буде завантажене число 0*CH*, а зі зміщенням 11*H* – число 0*FH*.

У табл. 9.5...9.7 подані протоколи виконання різних гілок розгалуженої програми залежно від різних значень вхідних даних *D*1 та *D*2. Для наочності у стовпчику *IP* показано адресу поточно виконуваної команди, а не наступної.

Слід зазначити, що приклад має виключно навчальне значення.

Таблиця 9.5 – Протокол виконання програми *D*1 = 0*AH*, *D*2 = 0*FH*

| № пп. | Мнемоніка команд | *IP* | *AX* | *BX* | *SP* | *DS* | *SS* | *CS* | *PF* | *SF* | *ZF* | *CF* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | *MOV AX*,0*F*0*A* | 0109 | 0*F*0*A* | 0000 | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 2 | *PUSH AX* | 010*C* | 0*F*0*A* | 0000 | *FFEC* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 3 | *POP AX* | 010*D* | 0*F*0*A* | 0000 | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 4 | *MOV BX,AX* | 010*E* | 0*F*0*A* | 0*F*0*A* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 5 | *SUB BL,BH* | 0110 | 0*F*0*A* | 0*FFB* | *FFEE* | 7000 | 7000 | 7000 | 0 | 1 | 0 | 1 |
| 6 | *JNS* 0128 | 0112 | 0*F*0*A* | 0*FFB* | *FFEE* | 7000 | 7000 | 7000 | 0 | 1 | 0 | 1 |
| 7 | *JP* 011*F* | 0114 | 0*F*0*A* | 0*FFB* | *FFEE* | 7000 | 7000 | 7000 | 0 | 1 | 0 | 1 |
| 8 | *ADD AL*,05 | 0116 | 0*F*0*F* | 0*FFB* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 9 | *NOT AH* | 0118 | *F*00*F* | 0*FFB* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 10 | *MOV*[10],*AX* | 011*A* | *F*00*F* | 0*FFB* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 11 | *JMP* 0129 | 0126 | *F*00*F* | 0*FFB* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 12 | *NOP* | 0129 | *F*00*F* | 0*FFB* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |

Таблиця 9.6 – Протокол виконання програми D1 = 0FH, D2 = 0AH

| № пп. | Мнемоніка команд | *IP* | *AX* | *BX* | *SP* | *DS* | *SS* | *CS* | *PF* | *SF* | *ZF* | *CF* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | *MOV AX*,0*A*0*F* | 0109 | 0*A*0*F* | 0000 | *FFEE* | 7000 | 7000 | 7000 | 0 | 0 | 0 | 0 |
| 2 | *PUSH AX* | 010*C* | 0*A*0*F* | 0000 | *FFEC* | 7000 | 7000 | 7000 | 0 | 0 | 0 | 0 |
| 3 | *POP AX* | 010*D* | 0*A*0*F* | 0000 | *FFEE* | 7000 | 7000 | 7000 | 0 | 0 | 0 | 0 |
| 4 | *MOV BX,AX* | 010*E* | 0*A*0*F* | 0*A*0*F* | *FFEE* | 7000 | 7000 | 7000 | 0 | 0 | 0 | 0 |
| 5 | *SUB BL,BH* | 0110 | 0*A*0*F* | 0*A*05 | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 6 | *JNS* 0129 | 0112 | 0*A*0*F* | 0*A*05 | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 7 | *PUSH AX* | 0128 | 0*A*0*F* | 0*A*05 | *FFEC* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 8 | *NOP* | 0129 | 0*A*0*F* | 0*A*05 | *FFEC* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |

Таблиця 9.7 – Протокол виконання програми *D*1 = 0*AH*, *D*2 = 0*EH*

| № пп. | Мнемоніка команд | *IP* | *AX* | *BX* | *SP* | *DS* | *SS* | *CS* | *PF* | *SF* | *ZF* | *CF* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | *MOV AX*,0*E*0*A* | 0109 | 0*E*0*A* | 0000 | *FFEE* | 7000 | 7000 | 7000 | 0 | 0 | 0 | 0 |
| 2 | *PUSH AX* | 010*C* | 0E0*A* | 0000 | *FFEC* | 7000 | 7000 | 7000 | 0 | 0 | 0 | 0 |
| 3 | *POP AX* | 010*D* | 0*E*0*A* | 0000 | *FFEE* | 7000 | 7000 | 7000 | 0 | 0 | 0 | 0 |
| 4 | *MOV BX,AX* | 010*E* | 0*E*0*A* | 0*E*0*A* | *FFEE* | 7000 | 7000 | 7000 | 0 | 0 | 0 | 0 |
| 5 | *SUB BL,BH* | 0110 | 0*E*0*A* | 0*EFC* | *FFEE* | 7000 | 7000 | 7000 | 1 | 1 | 0 | 1 |
| 6 | *JNS* 0128 | 0112 | 0*E*0*A* | 0*EFC* | *FFEE* | 7000 | 7000 | 7000 | 1 | 1 | 0 | 1 |
| 7 | *JP* 011*F* | 0114 | 0*E*0*A* | 0*EFC* | *FFEE* | 7000 | 7000 | 7000 | 1 | 1 | 0 | 1 |
| 8 | *ADD AL*,02 | 011*F* | 0*E*0*C* | 0*EFC* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 9 | *INC AH* | 0121 | 0F0*C* | 0*EFC* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 10 | *MOV*[0010],*AX* | 0123 | 0F0*C* | 0*EFC* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 11 | *JMP* 0129 | 0126 | 0*F*0*C* | 0*EFC* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |
| 12 | *NOP* | 0129 | 0*F*0*C* | 0*EFC* | *FFEE* | 7000 | 7000 | 7000 | 1 | 0 | 0 | 0 |

**Контрольні питання:**

1 З якою метою у програмах реалізуються БП?

2 Чи зберігається адреса повернення до основної програми, якщо реалізується безумовний перехід?

3 Знайти добуток двох однобайтових даних, що зберігаються у регістрі *AX*. Якщо одержаний результат від'ємний і вміщує парну кількість одиниць, то до нього треба додати одиницю і запам'ятати у стеку. Якщо кількість одиниць непарна, то добуток треба подати у прямому коді і записати у сегменті даних за зміщенням 0012*H*. Якщо добуток додатний, його треба запам'ятати у сегменті даних за адресою 7000:0014*H*. Значення вхідних даних задати самостійно.

**Контрольні питання підвищеної складності:**

1 Знайти частку від ділення двох однобайтових даних. Ділене знаходиться у регістрі *AH*, а дільник у регістрі *AL*. Якщо одержаний результат від'ємний і вміщує парну кількість одиниць, то до нього треба додати одиницю і запам'ятати у стеку. Якщо кількість одиниць непарна, то частку треба подати у прямому коді за зміщенням 0012*H*. Якщо частка додатна, її треба запам'ятати у сегменті даних за адресою 7000:0016*H*. Значення вхідних даних задати самостійно.

2 Перевірити, чи буде вірний фрагмент програми

```
7000:0120       CMP AL,BL
7000:0122       JZ 8000:0100
```

### *9.4.3 Циклічні програми*

**Вхідний контроль:**

1 Чи завжди у циклічній частині програми треба вказувати необхідну кількість повторень циклу?

2 Наведіть приклади організації циклів за допомогою команд мови Асемблер і покажіть, як вони реалізуються.

Програми, в яких багаторазово виконуються ті ж самі ділянки обчислень, як правило, з різними значеннями вхідних даних, називаються **циклічними**, а послідовність команд, що повторюються, **циклами**. Цикли з наперед заданою кількістю повторень називаються **арифметичними**. Цикли, що не мають заздалегідь заданої кількості повторень, називаються **ітераційними**. Вихід з них відбувається при виконанні заданої умови. Для реалізації арифметичних циклів треба організувати лічильник циклів. За умовчанням для організації лічильника циклів призначений регістр *CX (ECX* у МП *I*80386 та старших). Якщо лічильник спадний, то при виконанні кожного циклу із його вмісту віднімається одиниця. Цикли повторюються доти, доки вміст лічильника буде більший за нуль. Циклічна програма вважається завершеною, якщо лічильник дорівнює нулю. Рідше використовуються зростаючі лічильники. При роботі з масивами закінчувати цикли можна також при досягненні поточної адреси

елемента масиву адреси його останнього елемента. Як лічильник циклів можна використовувати будь-який регістр загального призначення, але у системі команд мови Асемблера існують спеціальні команди умовного переходу, в яких перевіряється вміст регістра *CX (ECX*).

*Приклад 9.4.9* Перед записом байта даних у регістр передавача послідовного асинхронного адаптера *RS*-232-*C* треба перевірити, чи вільний є регістр зберігання передавача, тобто чи завершено передавання попереднього символу. Ознакою "порожнього" регістра зберігання є установлений в 1 п'ятий біт регістра стану лінії з адресою 3*FDH*. Наступний фрагмент програми реалізує цю перевірку у циклі:

```
     MOV DX,3FDH ; Завантаження адреси регістра стану лінії
M1:  IN AL,DX    ; Введення байта стану лінії
     AND AL,20H  ; Регістр зберігання передавача
     JZ M1       ; порожній, D5 = 1?
     NOP
```

*Приклад 9.4.10* Перед прийомом даних з порту приймача послідовного асинхронного адаптера *RS*-232-*C* треба перевірити, чи прийняте з лінії дане перебуває у буферному регістрі приймача. Ознакою цього є наявність 1 у нульовому біті регістра стану лінії з адресою 3*FDH*. Наступний фрагмент програми реалізує цю перевірку в циклі:

```
     MOV DX,3FDH ; Завантаження адреси порту стану лінії
M1:  IN AL,DX    ; Введення байта стану лінії
     AND AL,01H  ; Перевірка наявності даного у буферному регістрі
                 ; приймача
     JZ M1       ; D0 = 1
```

*Приклад 9.4.11* Знайти у чотириелементному рядку байт, що дорівнює 43*H,* якщо рядок розміщений, починаючи з ефективної адреси 0200H, а вміст сегментного регістра даних *ES* дорівнює 7000*H*:

7000:0200 41 42 43 44.

Лічильник байтів у рядку організуємо у регістрі *CL*, ефективна адреса байтів вміщується в індексний регістр *DI*, а еталон шуканого байта завантажуємо до *AL*. Для перепускання неспівпадаючих елементів використовується префікс *REPNZ*.

```
MOV AX,7000H  ;
MOV ES,AX     ;
MOV CL,0004H  ; Організація лічильника циклів
MOV DI,0200H  ; Організація непрямого регістрового адресування
```

```
                    ; елементів рядка
MOV AX,0043H        ; Завантаження еталонного байта до AL
STD                 ; Установлення прапорця напряму
CLD                 ; Скидання прапорця напряму
REPNZ               ; Організація перепускання неспівпадаючих елементів
SCASB               ; Сканування байтів рядка
```

Програма виконується циклічно чотири рази, кожний цикл являє собою лінійну програму. Після сканування усіх байтів рядка в регістрі *CX* буде записане число 0001*H*, а в регістрі *DI* – число 0203*H*, на одиницю більше, ніж адреса шуканого байта. Прапорець *ZF* установлюється після виконання програми.

*Приклад 9.4.12* Одновимірний масив *M(N), N* = 10*H* однобайтових елементів зі знаками розміщений у сегменті даних, починаючи з ефективної адреси 20*H*; розсортувати елементи масиву на додатні *M(P)* та від'ємні *M(NEG*) і запам'ятати додатні елементи в області пам'яті *MP*, починаючи з ефективної адреси 30*H*, а від'ємні елементи в області пам'яті *MNEG*, починаючи з ефективної адреси 40*H*. Сегменти *DS* та *ES* суміщені.

Структурна схема алгоритму наведена на рис. 9.14.

Фрагмент програми розв'язання задачі.

```
        MOV CX,0010H    ; Завантаження лічильника циклів
        MOV SI,0020H    ; Завантаження адреси первісного масиву
        MOV DI,0030H    ; Завантаження початкової адреси масиву
                        ; додатних елементів
        MOV BX,0040H    ; Завантаження початкової адреси масиву
                        ; від'ємних елементів
        CLD             ; Виставлення автоінкрементування SI та DI
CYCLE:  LODSB           ; Виклик чергового елемента  масиву
        OR AL,AL        ; Виставлення ознак елемента масиву
        JS NEG          ; Елемент є додатний?
P:      STOSB           ; так, запис до масиву додатних чисел
        JMP COUNT       ; БП на команду перевірки лічильника
                        ; циклів
NEG:    MOV [BX],AL     ; ні, запис до масиву  чисел
        INC BX          ; Нарощування адреси масиву від'ємних
                        ; чисел
COUNT:  LOOP CYCLE      ; Перевірка лічильника циклів
        NOP
```

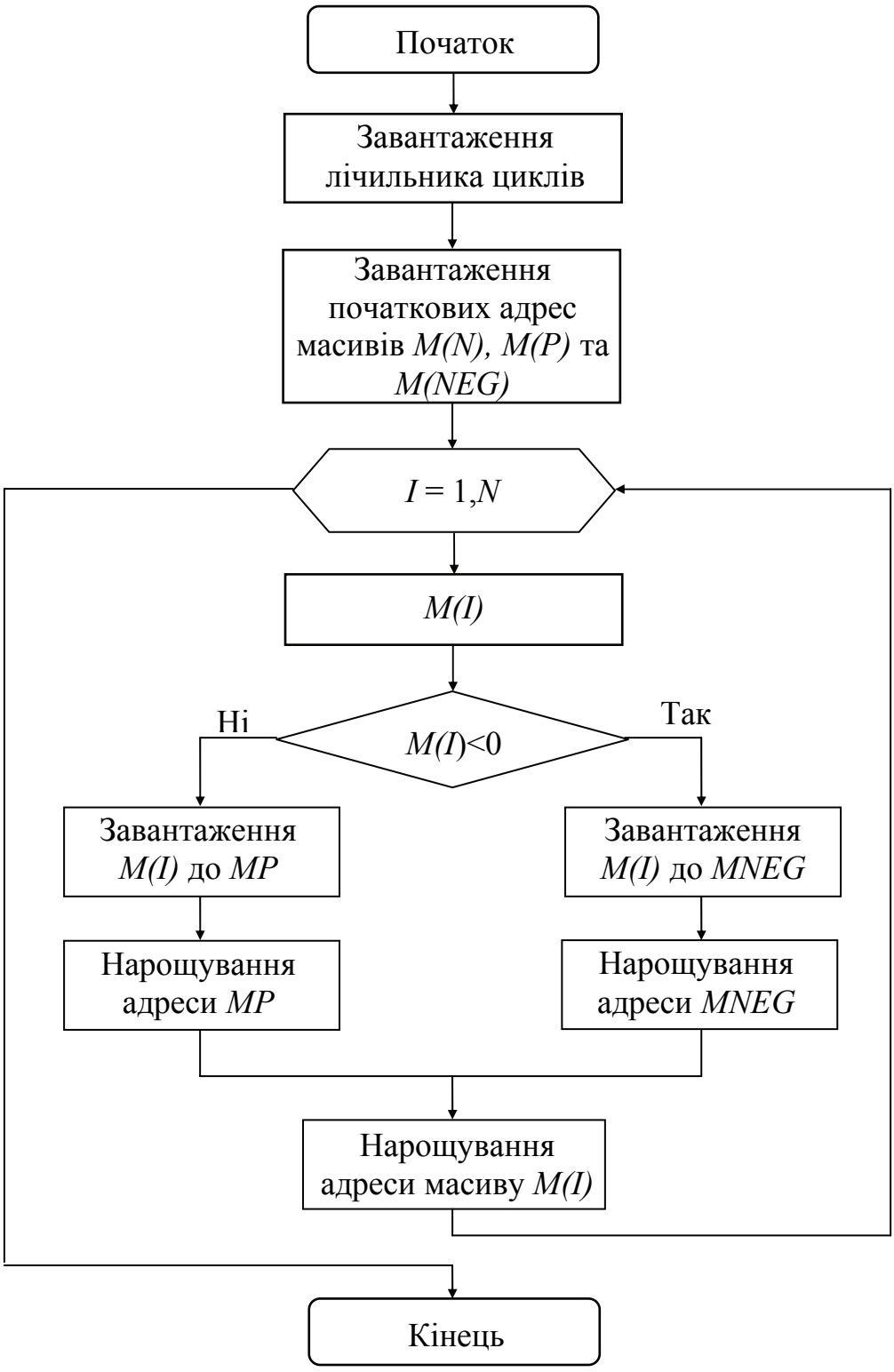



Рисунок 9.14 – Структурна схема алгоритму

*Приклад 9.4.13* Знайти середнє арифметичне *N* двобайтових елементів масиву *M(N), N* = 10*H*, що зберігаються у пам'яті в доповняльному коді, починаючи з адреси 7000:0020. Результат розмістити за ефективною адресою 30*H*. Структурна схема алгоритму подана на рис. 9.15.

Рисунок 9.15 – Структурна схема алгоритму

Програма розв'язання задачі.

```
MOV AX,7000H      ; Організація сегмента даних,
MOV DS,AX         ; починаючи з ефективної адреси 7000:0000
```

```
        MOV BX,0020H    ; Завантаження адреси початку масиву
        MOV AX,0000H    ; Обнулення акумулятора (S = 0)
M1:     ADD AX,[BX]     ; Додавання чергового елемента масиву
        ADD BX,2        ; Нарощування адреси чергового елемента
                        ; масиву
        CMP BX,0030H    ; Чи дорівнює адреса чергового елемента масиву
        JNZ M1          ; адресі останього+2? Якщо ні, повернення на
                        ; початок циклу
        MOV BX,0010H    ; Завантаження дільника до BX
        IDIV BX         ; Визначення середнього арифметичного
        MOV [30H],AX    ; Запам’ятовування середнього арифметичного
        NOP
```

Сума елементів масиву вміщується в акумуляторі *AX*, результат треба запам’ятовувати за ефективною адресою 30*H*.

*Приклад 9.4.14* Написати підпрограму часової затримки тривалістю 100 мкс за умови, що тривалість такту 0,1 мкс. Для організації підпрограми затримки, яка найчастіше має назву *DELAY*, треба використати лічильник для зберігання числа повторень циклів, від якого залежить час затримки. Значення *X* обчислюється за формулою

$$X = \left]\frac{t_{\text{з}} - t_{\text{о}}}{t_{\text{ц}}}\right[,$$

де дужки ] [ означають, що дрібна частина відкидається; $t_{\text{з}}$ – задане значення затримки; $t_{\text{о}}$ – час, потрібний для одноразово виконуваних команд; $t_{\text{ц}}$ – час, потрібний для виконання циклічно повторюваних команд. Вибір застосовуваних у підпрограмі команд та обчислення *X* являє собою ітераційний процес, тривалість якого залежить від необхідної точності затримки. За наявності високих вимог до точності забезпечуваної затримки можливі два шляхи розв’язання цієї задачі:

1) зменшують отримане значення *X* на кілька одиниць, а отримане зменшення компенсують командою *NOP*, яка виконується багаторазово;

2) змінюють значення $t_{\text{ц}}$ за рахунок включення у цикл інших команд.

Наступний фрагмент підпрограми *DELAY* дає уявлення про принцип програмної реалізації часової затримки:

```
        CALL DELAY      ; 19 тактів за умови внутрішньосегментного
                        ; переходу
DELAY:  MOV AL,X        ; 2 такти, X обчислюється за формулою та
                        ; задається у команді з безпосереднім
                        ; адресуванням
TIME:   DEC AL          ; 3 такти
```

```
        JNZ TIME        ; 8 тактів
        NOP             ; 2 такти
        RET             ; 8 тактів
```

На початку підпрограми *DELAY*, як і в кожній підпрограмі, треба запам'ятати вміст використовуваних регістрів та регістра прапорців:

```
        CALL DELAY      ; 19 тактів
DELAY:  PUSH BX         ; 11тактів
        PUSHF           ; 10 тактів
        MOV BL,X        ; 2 такти, X повинен завантажуватись у BL
                        ; як операнд з безпосереднім адресуванням
M1:     DEC BL          ; 3 такти
        JNZ M1          ; 16 тактів
        POP F           ; 8 тактів
        POP BX          ; 8 тактів
        RET             ; 8 тактів
```

$t_о = (19+11+10+2+8+8+8)\cdot 0{,}1 = 7{,}3$ мкс;
$t_ц = (3+16)\cdot 0{,}1 = 1{,}9$ мкс;
$t_з = 100$ мкс.

$$X = \left\lceil \frac{100 - 7{,}3}{1{,}9} \right\rceil = 49D = 31H .$$

При $X = 49D$ отримаємо $t_з = 6{,}6 + 49 * 1{,}9 = 6{,}6 + 93{,}1 = 99{,}7$ мкс.

Така точність формування затримки не є достатня і тому до одноразово виконуваних команд треба додати команду, яка б займала 3 такти і виконувалась 0,3 мкс. Це може бути команда *OR BL, BL*. Остаточно фрагмент підпрограми затримки має вигляд:

```
DELAY: PUSH BX
       PUSHF
       MOV BL,31H
M1:    DEC BL
       JNZ M1
       OR BL,BL
       POP F
       POP BX
       RET
```

Слід зазначити, що доцільно перед зверненням до підпрограми *DELAY* заборонити масковані переривання від зовнішніх пристроїв і наступною після команди *CALL DELAY* вжити команду дозволу переривань.

**Контрольні питання:**

1 Які програми називаються циклічними?

2 Які види циклічних програм ви знаєте?

3 Які команди умовних переходів використовуються для пошуку в рядку першого нульового або ненульового елемента?

4 Який спосіб адресування використовується при роботі з двовимірними масивами?

5 У масиві *N* однобайтових чисел, розміщених у пам'яті, починаючи з адреси 7000: 0010*H*, підрахувати суму парних чисел. Результат розмістити за зміщенням 0027*H*.

6 У масиві *M(N)* однобайтових чисел, розміщених у пам'яті, починаючи з адреси 7000: 0010*H*, підрахувати суму непарних чисел. Результат розмістити за зміщенням 0022*H*.

7 У масиві *M(N)* однобайтових чисел, розміщених у пам'яті, починаючи з адреси 7000:0010*H*, підрахувати кількість непарних чисел. Результат розмістити за зміщенням 0030*H*.

8 У масиві *M(N)* двобайтових чисел, розміщених у пам'яті, починаючи з адреси 7000:0010*H*, підрахувати кількість парних чисел. Результат розмістити за зміщенням 0026*H*.

**Контрольні питання підвищеної складності:**

1 Знайти суму елементів головної діагоналі квадратної матриці *N×N* і розмістити результат за зміщенням 0020*H*. Початкова адреса двовимірного масиву 7000:0010.

2 Скільки разів буде повторено цикл, якщо у лічильних циклів *CL* буде спочатку занесено число 2?

# 10 ПРОГРАМНА РЕАЛІЗАЦІЯ ВУЗЛІВ ТЕЛЕКОМУНІКАЦІЙНОГО ОБЛАДНАННЯ МОВОЮ АСЕМБЛЕР-86

## 10.1 Способи реалізації алгоритмів

**Вхідний контроль:**

1 Що таке алгоритм?

2 Які засоби задання алгоритмів Ви знаєте?

Алгоритм будь-якого процесу можна реалізувати трьома основними способами – **апаратним**, **матричним** та **програмним**.

**Апаратний** спосіб реалізації алгоритмів роботи цифрових пристроїв, на яких будуються вузли телекомунікації, був до недавнього часу домінуючим. Основними його рисами є детермінізм реалізації: навіть незначні зміни в алгоритмі призводять до необхідності перероблення хоча б частини схеми, а також схемотехнічна надмірність: послідовні процеси, які реалізують алгоритм, можуть виконуватись паралельно. Апаратний спосіб забезпечує найвищу швидкість роботи цифрових пристроїв.

**Матричний** спосіб по суті є також детермінованим, він реалізується за матричною технологією на кристалі мікросхеми. Логічні функції програмованої логічної матриці (ПЛМ, *PAL – Programming Array Logic*) описуються у вигляді функцій алгебри логіки – булевих рівнянь. Ці рівняння транслюються у карту плавких перемичок спеціалізованим компілятором на комп’ютері, результуюча карта виводиться у вигляді стандартизованого файла. Цей файл завантажується у програматор, і плавкі перемички перепалюються. До переваг ПЛМ відносять програмованість, ПЛМ відносно дешеві порівняно з апаратно реалізованими вузлами. До недоліків ПЛМ слід віднести відсутність у них пристроїв пам’яті та низьку швидкодію. ПЛМ використовуються для реалізації інтерфейсів, контролерів магістралей у мікропроцесорних системах, декодерів адрес тощо.

**Програмний** спосіб базується на поетапно-послідовній реалізації алгоритмів зі зберіганням проміжних результатів при програмному керуванні усіма виконуваними діями. Такий спосіб є найбільш універсальним і об’єднує можливості оптимального використання апаратних та програмних засобів. Програмний спосіб реалізації вузлів будь-якої апаратури базується на використанні програмованих ВІС-мікропроцесорів та мікроконтролерів і поєднує їх великі можливості за обробки інформації, досягану високу швидкодію, низьку вартість. Цей спосіб є найбільш використовуваним у сучасних реалізаціях вузлів телекомунікацій на апаратно-програмних комплексах.

Програмні засоби можуть замінити апаратні, якщо ця заміна задовольняє вимогам до швидкодії обчислювальної або мікропроцесорної системи і ця заміна економічно доцільна, а також якщо програмна модель реалізує повністю функції апаратури, а саме сприймання, зберігання, оброблення та видавання даних.

У загальному випадку програмна модель апаратних засобів вміщує програму роботи та значення вхідних змінних (сигналів), які програма переробляє у набори вихідних сигналів.

**Контрольні питання:**

1 Чи можна реалізувати матричний спосіб задання алгоритмів на ПЛІС?

2 В якому вигляді повинен бути заданий алгоритм для реалізації на ПЛІС?

3 Чи є можливість реалізувати невироджені цифрові автомати на ПЛІС?

4 Які способи підвищення швидкодії реалізації алгоритмів, заданих програмно, Ви знаєте?

**Контрольні питання підвищеної складності:**

1 Як можна апаратним способом реалізувати розгалуження?

2 Як за допомогою ПЛМ реалізувати цикли?

## 10.2 Розробка апаратно-програмних комплексів

**Вхідний контроль:**

1 Які вузли телекомунікацій Ви знаєте?

2 Яку роль у пристроях телекомунікацій відіграють системи синхронізації та генератори імпульсів?

Розробка програмного забезпечення є застосування принципів, навичок та творчого підходу до складання програмних комплексів. Середовищем цих програм є обчислювальна система, на якій вони використовуються. До характерних рис програмного забезпечення можна віднести:

– програми виконуються під єдиним керуванням, частіш за все під керуванням операційної системи, яка працює на центральному процесорі;

– взаємне керування окремими програмами комплексу є підлеглим по відношенню до цього єдиного керування, яке реалізується на більш високому рівні пріоритету та рівнях привілеїв;

– внутрішні рівні операційної системи є сховані від користувачів обчислювального комплексу і не можуть бути змінені користувачем.

При проектуванні програмного комплексу створюється його повний та точний опис, який є комплектом підописів програмних та апаратних компонентів та їх взаємодій. Повний опис програмного компонента – це програма на визначеній мові програмування, семантика якої повинна бути відома до початку програмування.

Розробники програмного забезпечення повинні досягти чотирьох цілей: заданого функціонування, правильності, продуктивності та надійності.

Задане функціонування – це отримання потрібного та правильного результату при введенні заданих вхідних даних. Вхід – це вся інформація, яка сприймається програмним комплексом. Вихід – це видавана інформація. Правильність визначає, наскільки точно програмний комплекс відповідає

поставленим цілям. Під продуктивністю розуміють швидкість та ефективність програмного комплексу при споживанні ним ресурсів у процесі досягнення мети. Надійність – це здатність програмного комплексу вірно виконувати свої функції, не зважаючи на відмови компонентів обчислювальної системи. Під відмовою розуміють тимчасову або постійну зміну характеристик компонентів, яке призводить до порушення їх функцій.

Програмне забезпечення розробляється за модульним принципом, використовує стандартизовані елементи та обмежені розміри заради гнучкості та універсальності, і компонується шляхом об'єднання модулів.

Мікропроцесорні системи часто вбудовують у вузли систем керування, вимірювальних систем, вузлів обладнання різного призначення, в тому числі телекомунікаційного. Розробники спеціалізованого програмного забезпечення мікропроцесорних систем повинні перш за все враховувати особливості технологічних пристроїв або процесів, якими керує програма. Розробка МПС здійснюється вузько спрямовано для реалізації конкретного алгоритму, і цей принцип розповсюджується, як на апаратну, так і на програмну складові. Апаратна частина складається з визначеної кількості модулів конкретного типу, необхідних для реалізації заданого алгоритму, а розроблений додаток фіксується найчастіше у ПЗП і часто не може з метою реалізації заходів безпеки зчитуватись, верифікуватись та змінюватись у процесі експлуатації. Таким чином, багатофункційність є тільки потенціальною властивістю МПС.

Телекомунікації являють собою апаратно-програмні комплекси реального часу, створення програмного забезпечення для яких вимагає чималого досвіду в багатьох галузях. Розвиток інтелектуальних та мультисервісних мереж відбувається високими темпами, розробляються все нові і нові принципи взаємодії різних вузлів та пристроїв, які входять у ці мережі, розробляються нові технології для узгодження й ефективного їх функціонування. В процесі створення програмного забезпечення необхідно врахувати ряд вимог, що є важливим фактором, який впливає на ефективність готового програмного продукту:

– відмовостійкість – можливість системи продовжувати нормальне функціонування в умовах наявності помилок. Висока вірогідність забезпечення завадостійкості веде до забезпечення необхідної якості обслуговування (*QoS*);

– супроводжуваність – сучасні системи розробляються з урахуванням їх використання протягом достатньо значного терміну. У цей період неодноразово виникає необхідність в її модифікації, що викликається різними причинами (розширення функціональних можливостей, заміна апаратних елементів на сучасніші, зміна стандартів і правил взаємодії із зовнішнім середовищем тощо);

– керованість процесом розробки – великі системи, як правило, відрізняються значним об'ємом і структурною складністю апаратних засобів, а так само значним набором виконуваних функцій. Все це вимагає розробки програмного забезпечення великого обсягу. Досягаючи певних меж, складність програмного забезпечення призводить до втрати контролю над його розробкою і вимушує розробників робити помилки, які згодом достатньо важко виправити;

– гнучкість – буває так, що є точно визначена структура апаратних засобів і параметри зовнішнього оточення, і більшість систем дозволяють налаштовуватись на ці характеристики. Для великих систем така структура реалізується за допомогою створення бази даних, що містить усі необхідні параметри. Задана структура може реалізуватись, як статично до запуску системи, так і динамічно в процесі її функціонування. Динамічне налаштування ускладнюється тим, що зміни у структурі необхідно робити “по живому”, тобто в даних, які можуть у цей момент використовуватися функціональними процесами, що вимагає узгодження між ними, і процесом налаштування;

– стійкість системи – означає її готовність реагувати на непередбачувану поведінку зовнішнього середовища;

– стабільність – вбудована система може перебувати в одному із двох станів: **активному** і **пасивному**. В **активному** стані здійснюється функціонування елементів (можливо, не всіх) програмного забезпечення системи, і система виконує всі або деякі зі своїх функцій; при цьому повний стан системи описується станом апаратури і станом даних в обчислювальному середовищі. Під час переходу системи в **пасивний** стан відбувається зупинка програмного забезпечення; при цьому не всі використовувані ним дані втрачають свою актуальність, частина їх повинна бути збережена до наступного запуску системи, що примушує зберігати ці дані в пам’яті; як засоби обробки таких даних є системи баз даних, що забезпечують організацію, зберігання і доступ до даних;

– безперервність – для більшості систем безперервне і надійне функціонування є критичним; безперервність функціонування означає неможливість зупинки системи, зв'язану, як правило, з важливістю виконуваних нею процесів;

– паралельність – є характерною властивістю більшості систем; наявність апаратних елементів, що працюють у реальному паралельному режимі, й одночасне виконання декількох функціональних процесів вимагають застосування паралельних обчислень; одна з проблем, пов’язана з розподілом ресурсів в програмно-апаратних системах, полягає в тому, що не завжди можна добитися їх монополізації; зокрема, це торкається даних, що відображають стан апаратних елементів системи; такі дані не можуть бути повністю монополізовані функціональним процесом, оскільки існує апаратний процес, який може зажадати їхньої асинхронної зміни;

– розподіленість – при проектуванні великих вбудованих систем часто використовується принцип модульності. Це дозволяє “збирати” системи різної конфігурації з типових елементів – модулів, збільшувати гнучкість системи. Ще одною перевагою модульної структури є можливість створення розподіленої архітектури вбудованої системи. При цьому кожен окремий модуль включає свій власний управляючий елемент – вбудований процесор. Усі процесори системи об'єднуються в обчислювальну мережу, що забезпечує обмін інформацією між ними. Зрозуміло, що багато з перерахованих властивостей характерні для всіх великих програмних систем (наприклад, супроводжуваність і керованість), частка характерна тільки для вбудованих систем. Важливо

відзначити, що для вбудованих систем докорінним чином змінюється система пріоритетів, наприклад, стійкість і безперервність потрібно забезпечувати навіть у збиток іншим характеристикам.

Розробка програмних моделей вузлів телекомунікацій здійснюється відповідно до рекомендацій *ITU* серії *Z* – *Z*.100, переважно табличним методом. Табличний метод є вільний від недоліків, пов'язаних з нерозумінням фахівців у процесі розробки програмного забезпечення завдяки високому ступеню абстрактності.

Нижче наведені приклади реалізації деяких простих вузлів телекомунікацій.

**Контрольні питання:**

1 Які переваги для реалізації вузлів телекомунікацій має програмний метод?

2 Як Ви розумієте модульність побудови програмного забезпечення?

3 Які вимоги пред'являються до програмного забезпечення, яке реалізує вузли телекомунікацій?

4 Які рекомендації міжнародних організацій регламентують створення моделей вузлів телекомунікацій?

**Контрольні питання підвищеної складності:**

1 Наведіть приклад, коли при переході системи у пасивний стан, дані повинні зберігатись до наступного запуску системи?

2 Де, на Ваш погляд, у ПК зберігаються дані про адреси векторів переривань *BIOS*?

3 Чи знищуються вони при вимкненні комп'ютера?

## 10.3 Приклади реалізації простих вузлів телекомунікацій

### *10.3.1 Ініціалізація послідовного асинхронного адаптера RS-232-C*

**Вхідний контроль:**

1 Які адреси мають порти послідовного асинхронного адаптера?

2 Чому для передавання даного 41*H* (п. 6.2) використовується керувальне слово 1*EH*?

3 З якою метою у цьому ж прикладі використовується один стоповий біт?

Перше, що повинна зробити програма, яка працює з асинхронним адаптером, – це установити протокол обміну та швидкість передавання даних. Після завантаження операційної системи установлюється швидкість 2400 Бод, не виконується перевірка на парність, використовується один стоповий біт та восьмибітова довжина передаваного символу.

Для протоколу обміну даними, який застосовується у п. 6.2, керувальне слово становить 1*EH*. Цей режим забезпечує довжину надсилання у бітах – 7 бітів, два стопових біти та контроль на парність.

Для завдання нового значення швидкості обміну даними потрібно установити старший біт керувального слова в 1 та вивести 80*H* за адресою 3*FBH*. Після цього двома послідовними командами виведення потрібно завантажити подільник частоти. Молодший байт подільника завантажується у порт 3*FBH*, а старший – у порт 3*F9H*. Виберемо швидкість обміну 1200 Бод, якій відповідає подільник 96*D* = 60*H*. Перед початком роботи ініціалізується регістр керування перериваннями (порт 3*F9H*), якщо у програмі навіть не використовуються переривання від самого адаптера. Якщо переривання не потрібні, у порт записується нуль.

Для дозволу переривань необхідно установити в 1 біти порту керування перериваннями 3*F9H*, відповідні тим перериванням, які потрібно опрацьовувати. Коли відбулося переривання, програма-опрацьовувач переривання повинна проаналізувати його причину за значенням вмісту порту ідентифікації переривання з адресою 3*FAH*.

Якщо водночас виникають декілька запитів на переривання, біт 0 регістра ідентифікації буде установлено в 1. У такому разі перед завершенням опрацювань переривань треба знов прочитати регістр ідентифікації переривань і опрацювати наступне переривання. Процес повторюється доти, доки біт 0 регістра ідентифікації переривань не дорівнюватиме 0. На цьому ініціалізація завершується.

Нижче наведено фрагмент програми ініціалізації асинхронного адаптера мовою Асемблер-86.

```
        MOV DX,3FBH   ; Завантаження адреси керувального регістра у DX
        MOV AL,80H    ; Порт керування настроюється на забезпечення
                      ; передавання подільника: D7 = 1, інші біти дорівнюють
                      ; 0, керувальне слово дорівнює 80H
        OUT  DX,AL    ; Виведення керувального слова
        MOV DX,3F8H   ; Завантаження адреси порту 3F8H
                      ; та
        MOV AL,60H    ; завантаження молодшого байта
        OUT  DX,AL    ; подільника частоти
        MOV DX,3F9H   ; Завантаження адреси порту 3F9H
                      ; та
        MOV AL,00H    ; завантаження старшого байта
        OUT  DX,AL    ; подільника частоти
M3:     MOV AL,1EH    ; Завантаження керувального слова в AL
        MOV DX,3FBH   ; Завантаження адреси керувального регістра
        OUT  DX,AL    ; Виведення керувального слова
        MOV AL,00H    ; Заборона переривань від RS-232-C
        MOV DX,3F9H   ;Завантаження адреси керувального регістра переривань
        OUT  DX,AL    ; Виведення керувального слова
```

### *10.3.2 Фрагмент програми передавання даних через асинхронний адаптер RS-232-C*

```
M1:  MOV DX,3FDH   ; Завантаження адреси регістра стану лінії
     IN  AL,DX     ; Введення слова стану лінії
     AND AL,20H    ; Регістр зберігання даних є
     JZ  M1        ; порожній, D5 = 1?
     MOV AL,41H    ; Так, завантаження даного в AL
     MOV DX,3F8H   ; Завантаження адреси порту передавання даних
     OUT DX,AL     ; Виведення даних
     NOP
```

### *10.3.3 Фрагмент програми приймання даних через асинхронний адаптер RS-232-C*

```
     MOV DX,3FDH   ; Завантаження адреси порту стану лінії
M2:  IN  AL,DX     ; Введення слова стану лінії
     AND AL,01H    ; Перевірка наявності даних в лінії (D0 = 1?)
     JZ  M2        ; у циклі
     MOV DX,3F8H   ; Завантаження адреси порту приймання даних 3F8H
                   ; у DX
M5:  IN  AL,DX     ; Введення даних
     NOP
```

### *10.3.4 Приклад програми ініціалізації RS-232-C та введення-виведення даних, написаної у програмному середовищі TURBO ASSEMBLER (TASM)*

```
; Програма ініціалізації послідовного адаптера та введення-виведення
; даних TITLE RS232.ASM

MODEL SMALL
STACK 256
DATA SEGMENT
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA,ES:DATA

START:  MOV   AX,DATA  ; Суміщення
        MOV   DS,AX    ; сегментів
        MOV   ES,AX    ; даних
        MOV   DX,3FBH  ; Завдання
        MOV   AL,80H   ;
        OUT   DX,AL    ; подільника
```

```
            MOV     DX,3F8H   ;
            MOV     AL,60H    ;
            OUT     DX,AL     ; частоти
            MOV     DX,3F9H   ;
            MOV     AL,00H    ; генератора
            OUT     DX,AL     ;
                              ; Завдання нового режиму адаптера
CONTR:      MOV     AL,1EH    ; Завантаження
            MOV     DX,3FBH   ; керувального
            OUT     DX,AL     ; слова (1EH)
INTER:      MOV     AL,00H    ; Заборона
            MOV     DX,3F9H   ; переривань
            OUT     DX,AL     ; від RS-232-C
                              ; Передавання даних
STATUS:     MOV     DX,3FDH   ; Перевірка
            IN      AL,DX     ; стану
            AND     AL,20H    ; буфера передавання
            JZ      STATUS    ; у циклі
SEND:       MOV     AL,41H    ; Передавання
            MOV     DX,3F8H   ; даного
            OUT     DX,AL     ; 41H
                              ; Приймання даних
            MOV     DX,3FDH   ; Перевірка
READY:      IN      AL,DX     ; наявності
            AND     AL,01H    ; даних у лінії
            JZ      READY     ; у циклі
            MOV     DX,3F8H   ; Приймання
            IN      AL,DX     ; даного
            JMP     STATUS    ; Циклування програми
            NOP
EXIT:       MOV     AH,4CH    ; Вихід
            INT     21H       ; з програми
      CODE ENDS
      END
```

Циклування програми командою *JMP STATUS* здійснюється з метою забезпечення перегляду часових діаграм за допомогою осцилографа на перемкнутих контактах 2 та 3 з'єднувача *DB*9*P COM*1. Часові діаграми для наведеного прикладу будуть співпадати з наведеними на рис. 6.2.

**Контрольні питання:**

1 З якою метою програмно ініціалізується асинхронний адаптер *RS*-232-*C*?

2 З якою метою адреси портів послідовного асинхронного адаптера завантажуються у регістр *DX*?

3 З якою метою у фрагменті програми у п. 10.3.2 виконується команда *AND AL*,20*H*?

4 За якої умови у фрагменті програми у п. 10.3.3 після команди *JZ M*2 буде виконуватись команда *MOV DX*,3*F*8*H*?

5 Як розподіляється пам'ять по сегментах після виконання директиви *MODEL SMALL*?

**Контрольні питання підвищеної складності:**

1 Як у програмі, написаній у середовищі *TASM*, завантажуються регістри *CS, DS, SS, ES*?

2 З якою метою у цій програмі використовується фрагмент:

```
MOV     AH,4CH
INT     21H
```

3 Яку адресу має у таблиці переривань комірка, в якій зберігається адреса підпрограми оброблення *INT* 21*H*?

### *10.3.5 Програмна реалізація генератора імпульсних послідовностей*

**Вхідний контроль:**

1 Генератор, який синхронізує роботу МПС, видає імпульси на МП з частотою 10 МГц; яка тривалість тактового імпульсу МП?

2 Як можна реалізувати програмно часову затримку на 10 мкс?

В основу методу створення програмних моделей генераторів імпульсних послідовностей покладено три прийоми:

– використання різних підпрограм часових затримок;

– використання таблиць в ОЗП, які вміщують інформацію про імпульсні послідовності;

– використання алгоритмів обчислення часових логічних функцій.

Вираз для часової логічної функції генератора імпульсної послідовності $Y = f(t)$, де $t$ – дискретний час, показує залежність вихідного сигналу від дискретного реального часу. Обчислення часової логічної функції повинне відбуватися за рівні інтервали часу, інакше синхронізація формованих послідовностей буде порушуватись. На рис. 10.1 показано змодельовану імпульсну послідовність кінцевої довжини, яка виводиться через молодший розряд паралельного порту *PORT*.

Імпульсна послідовність розбивається на інтервали $\Delta t$ та кодується, закодований опис завантажується у пам'ять.

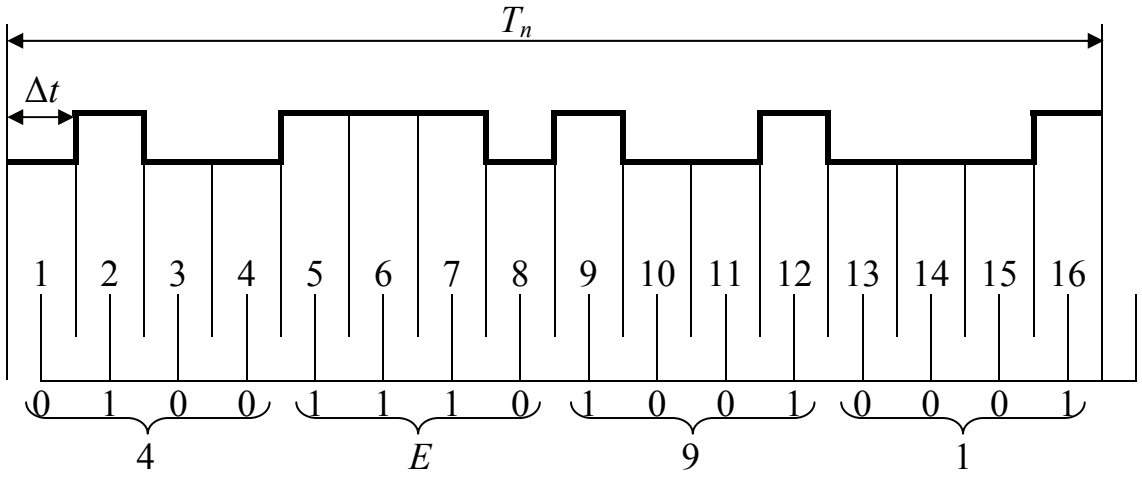



Рисунок 10.1 – Кодування імпульсної послідовності

Наведена нижче підпрограма *IM* реалізації імпульсної послідовності працює таким чином: через рівні інтервали часу Δ*t* наступний байт опису послідовності виводиться у *PORT*. Інтервал Δ*t* визначається кількістю тактів синхронізації МП, які потрібні для виводу байта у порт (10 тактів = 1 мкс) та кодом часу затримки у підпрограмі *DELAY*. Таким чином $\Delta t = 1_{\text{МКС}} + t_{DELAY}$. Підпрограма моделює тільки період імпульсної послідовності, для моделювання довгої серії імпульсів треба організувати цикл звернення до підпрограми *IM*:

```
IM: MOV  BX,A1ABH  ; Занесення до BX адреси таблиці опису
M2: MOV  CL,02H    ; Занесення до CL кількості байтів опису
    MOV  DH,8H     ; Занесення до DH кількості бітів у байті
    MOV  AL,[BX]   ; Занесення до AL чергового байта опису
    MOV  DL,AL     ; Запам'ятовування цього байта у DL
M1: AND  AL,01H    ; Виділення молодшого біта
    OUT  PORT      ; Вивід його у PORT
    SHR  DL,1      ; Логічний зсув праворуч на 1 розряд
    MOV  AL,DL     ; Запам'ятовування нового значення байта
    CALL DELAY     ; Затримка сигналу на виході порту
    DEC  DH        ; Декрементування лічильника бітів
    JNZ  M1        ; Повернення на початок циклу
    INC  BX        ; Нарощування адреси елемента таблиці
    LOOP M2        ; Звернення до наступного елемента таблиці
    RET            ; Повернення з підпрограми
    NOP
```

### *10.3.6 Програмне вимірювання періоду імпульсної послідовності DET*

Період імпульсної послідовності $T_{\text{пер}}$ вимірюється підрахуванням кількості відомих малих інтервалів часу Δ*t*, які накопичуються за один період імпульсної послідовності. Імпульсна послідовність показана на рис. 10.1.

Частота імпульсів повинна знаходитись у межах 5 Гц ... 167 кГц. Підпрограма *DET* кожні $\Delta t = 3$ мкс опитує порт, на молодший розряд якого надходить послідовність імпульсів. Фіксується фронт імпульсу, після чого у лічильнику *CX* накопичується кількість дискретних інтервалів часу *T*І, протягом яких у порт надходить високий рівень напруги, а у лічильнику *BX* накопичується кількість інтервалів часу *T*П, протягом яких зберігається низький рівень напруги. Числа *T*І та *T*П запам'ятовуються відповідно за адресами 20*H* та 22*H* у сегменті даних, а кількість інтервалів часу за період $T_{\text{пер}} = T\text{І} + T\text{П}$ запам'ятовується у комірці пам'яті за адресою 24*H* у сегменті даних.

В основній програмі після повернення з підпрограми *DET* можна підрахувати тривалість імпульсу $t_{\text{і}}$, тривалість паузи $t_{\text{п}}$ та період $T_{\text{пер}}$ у секундах

$t_{\text{і}} = \Delta t(T\text{І}) = 3$ мкс ($T$І);
$t_{\text{п}} = \Delta t(T\text{П}) = 3$ мкс ($T$П);
$T_{\text{пер}} = \Delta t(T\text{І} + T\text{П}) = 3$ мкс ($T$І + $T$П).

Фрагмент програми *DET* наведено нижче:

```
      MOV DX,PORT   ; Завантаження до DX адреси порту
DET:  IN AL,DX      ; Фіксація
      AND AL,01H    ; фронту
      JNZ DET       ; імпульсу
M1:   IN AL,DX
      AND AL,01H
      JNZ M1
      MOV CX,00H    ; Скидання лічильника часу (ТІ) імпульса
M2:   INC CX        ; Вимірювання ТІ
      IN AL,DX
      AND AL,01H
      JNZ M2
      MOV [20H],CX  ; Занесення ТІ у пам'ять
      MOV BX,00H    ; Скидання лічильника часу (ТП) паузи
M3:   INC BX        ; Вимірювання ТП
      IN AL,DX
      AND AL,01H
      JNZ M3
      MOV [22H],BX  ; Занесення ТП у пам'ять
      ADD BX,CX     ; Визначення періоду послідовності
                    ; імпульсів (Tпер)
      MOV [24H],BX  ; Занесення Tпер у пам'ять
      RET           ; Повернення з підпрограми
```

**Контрольні питання:**

1 Яка підпрограма використовується для формування будь-яких часових послідовностей?

2 Як за допомогою табличного метода програмно реалізувати генератор імпульсних послідовностей?

3 Як, використовуючи комп'ютер або МПС, здійснити вимірювання періоду імпульсної послідовності?

**Контрольні питання підвищеної складності:**

1 Напишіть програму формування імпульсної послідовності з коефіцієнтом заповнення 0,5.

2 Виведіть цю імпульсну послідовність через порт *RS*-232-*C*.

### *10.3.7 Програмна реалізація мультиплексора*

**Вхідний контроль:**

1 Напишіть у вигляді таблиці алгоритм роботи мультиплексора з вісім'ю інформаційними входами.

2 Який активний рівень має сигнал дозволу роботи мультиплексора?

Реалізувати програмно мультиплексор з вісім'ю інформаційними входами; залежно від коду на адресних входах один із входів підключається до виходу; сигнал дозволу має нульове активне значення. Нижче наведено фрагмент підпрограми, яка моделює такий мультиплексор.

```
        MOV AH,04H    ; Завантаження керувального слова у регістр AH
        MOV BH,63H    ; Завантаження байта даних
        CALL PMS      ; Звернення до підпрограми, яка реалізує
                      ; мультиплексор
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
PMS:  MOV AL,AH       ; Запам'ятовування керувального слова в AL
      TEST AL,10H     ; Дозвіл на підключення біта даних до виходу є?
      JNZ PP2         ; Ні, перехід на установлення дозволу
      OR AL,AL        ; Так, адреса є нульова?
      JNZ M1          ; Ні, перехід на оброблення адреси
      RCR BH,1H       ; Так, запам'ятовування молодшого біта даних
                      ; у CF
      RCL AL,1H       ; Перенесення молодшого біта даних у нульовий
                      ; розряд AL
      JMP EXIT        ; Перехід на повернення з підпрограми
M1:   RCR AL,1H       ; Оброблення вказаної
      AND AL,07H      ; адреси
      MOV BL,AL       ; Запам'ятовування адреси у лічильнику адрес
      RCR BH,1H       ; Виключення з розряду нульового біта даних
PP1:  RCR BH,1H       ; Заміщення CF бітами даних, починаючи
```

```
                    ; з першого у циклі
        DEC BL      ; Зменшення адреси до нуля
        JNZ PP1     ; у циклі
        RCL AL,1H   ; Внесення адресованого біта даних
                    ; у молодший розряд AL
M2:     MOV AH,AL   ; Повернення керувального слова з адресованим
                    ; бітом даних у AH
        JMP EXIT    ; Безумовний перехід на повернення
                    ; з підпрограми
PP2:    AND AH,EFH  ; Установлення дозволу на підключення біта
                    ; даних до виходу при збереженні
                    ; керувального слова
EXIT:   RET
```

Вісім інформаційних входів мультиплексора моделюються регістром *BH*. У нульовому розряді регістра *AL* будемо отримувати результат – прямий вихід мультиплексора. У розряди 1, 2, 3 регістра *AL* заноситься код адреси біта даних, а розряд 4 моделює вхід дозволу. Код адреси зменшується у циклі при одночасній фіксації в ознаці CF значення біта даних в обраному розряді.

**Контрольні питання:**

1 Чому у програмі, яка зреалізовує мультиплексор, окремо розглядається випадок, коли код адреси дорівнює нулю?

2 Яким способом, апаратним або програмним, доцільніше реалізувати мультиплексор на 32 інформаційних входи і чому?

**Контрольні питання підвищеної складності:**

1 Як треба змінити фрагмент програми реалізації мультиплексора, щоб перевіряти наявність дозволу на роботу мультиплексора поза підпрограмою *PMS*?

2 Як у програмі здійснюється синхронізація адреси й значення біта даних в обраному розряді?

## СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ДО Частини II 1-го МОДУЛЯ


1 Брэй Б. Микропроцессоры *Intel*: 8086/8088, 80186/80188, 80286, 80386, 80486, *Pentium, Pentium Pro Processor, Pentium* II, *Pentium* III, *Pentium* 4. Архитектура, программирование и интерфейсы. – [6-е изд.]; пер. с англ. – С.Пб.: БХВ – Петербург, 2005. – 1328 с.: ил.

2 Юров В. Assembler: учебный курс / В. Юров, С. Дорошенко – С.Пб.: Изд. «Питер», 1999. – 672 с.: ил.

3 Митрофанов Ю. М. Програмування на мові Асемблер: [підручник для самостійної роботи з курсу «Цифрова техніка та мікропроцесори»] / Митрофанов Ю. М., Ошаровська О. В., Хіхловська І. В. – Одеса: УДАЗ, 1997. – 25 с.: іл.

4 Брамм П. Микропроцессор 80386 и его программирование / П. Брамм, Д. Брамм; пер. с англ. – М.: Мир, 1990. – 448 с.: ил.

5 Майоров В. Г. Практический курс программирования микропроцессорных систем / В. Г. Майоров, А. И. Гаврилов – М.: Машиностроени, 1989. – 272 с.: ил.

6 Абель П. Язык Ассемблер для *IBM PC* и программирования / Абель П. – М.: Высшая школа, 1992. – 447 с.: ил.

7 Лю Чжен–Ю. Микропроцессоры семейства 8080/8088 / Лю Чжен–Ю, Г. Гибсон – М.: Радио и связь, 1987. – 512 с.: ил.

8 Микропроцессорный комплект К1810: Структура, программирование, применение: справочная книга / [Ю. М. Казаринов, В. Н. Номоконов, Г. С. Подклетнов, Ф. В. Филиппов]; под ред. Ю. М. Казаринова. – М.: Высшая школа, 1990. – 269 с.: ил.

9 *INTERNATIONAL TELECOMMUNICATION UNION. TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Addendum* 1 (10/96). *SERIES Z: PROGRAMMING LANGUAGES. Specification and Description Language (SDL).*

10 Вивчення архітектури та програмних моделей мікропроцесорів фірми Intel. Укладачі: І. В. Хіхловська, О. С. Антонов. Одеса 2000.

11 Сегментування пам’яті та способи адресування операндів у МП фірми *INTEL*. Укладачі: І. В. Хіхловська, О. С. Антонов. Одеса 2000.

12 Програмування мікропроцесорів фірми *INTEL* на мові Асемблер. Укладачі: І. В. Хіхловська, О. С. Антонов. Одеса 2000.

13 Порт послідовного передавання даних *RS*-232-*C*. Укладачі: І. В. Хіхловська, О. С. Антонов. Одеса 2000.

14 Системне та прикладне програмне забезпечення в телекомунікаціях. Конспект лекцій з дисципліни “Системне та прикладне програмне забезпечення в телекомунікаціях” для спеціальності 7.092402. Укладач: І. В. Хіхловська. Одеса 2004.

**15** Системне програмне забезпечення. Конспеки лекцій з дисципліни “Системне програмне забезпечення” для спеціальностей 7.092401, 7.092402, 7.092407. Укладач О. С. Антонов. Одеса 2004.

# МОДУЛЬ 2.
# Частина І МІКРОПРОЦЕСОРНІ СИСТЕМИ НА МІКРОПРОЦЕСОРАХ ФІРМИ MOTOROLA ТА ЇХНЄ ПРОГРАМУВАННЯ

## 11 МІКРОПРОЦЕСОРНІ СИСТЕМИ НА УНІВЕРСАЛЬНИХ МП ФІРМИ MOTOROLA

**Вхідний контроль:**

1 Які *32-розрядні універсальні МП фірми Intel* Вам відомі?

2 У яких пристроях обчислювальної техніки використовуються 32-розрядні універсальні МП фірми *Intel*?

3 У яких пристроях телекомунікацій використовуються 32-розрядні універсальні МП фірми *Intel*?

4 Під якими операційними системами працюють *Intel*-сумісні мікропроцесори?

5 У який спосіб зорганізовується захищений режим МП фірми Intel?

МП сімейства *MC*680*X*0 фірми *Motorola* належать до універсальних МП з класичною *CISC* архітектурою і застосовуються у персональних комп'ютерах, серверах, цифрових системах комутації, маршрутизаторах, мобільному зв'язку, контрольно-вимірювальній апаратурі, літакобудівництві, військово-промисловому комплексі, системах промислової автоматики тощо. Фірма *Motorola* випускає також широкий спектр напівпровідникових пристроїв від транзисторів до надвеликих інтегральних мікросхем. Вироби фірми *Motorola* відрізнюються високими технічними характеристиками, якістю та надійністю, а вихід бездефектної продукції становить 99,9997 %. Все це зумовило широке розповсюдження виробів фірми *Motorola* на світовому ринку.

Базовою моделлю МП *MC*680*X*0 є МП *MC*68000.

### 11.1 16-розрядні мікропроцесори фірми *Motorola*

**Вхідний контроль:**

1 Які пристрої входять до програмної моделі мікропроцесора *I*8086?

2 Відмінності фоннеймановської і гарвардської архітектури МП.

3 До якої з архітектур належать МП фірми *Intel*?

4 Призначення регістра прапорців (ознак результату).

5 У який спосіб змінюється вміст регістра прапорців?

6 Яке призначення має регістр-лічильник команд?

7 У який спосіб змінюється вміст регістра-лічильника при виконуванні різних команд?

Сімейство 16- та 32-розрядних мікропроцесорів *MC*680*x*0 фірми *Motorola* містить низку процесорів, контролерів та співпроцесорів, які мають однакову базову архітектуру. Ця архітектура складається з наборів, однакових для всіх представників сімейства регістрів, внаслідок чого всі пристрої мають ідентичні

базові способи адресування, базову систему команд, ідентичні принципи взаємодії з пам'яттю і взаємодії із зовнішніми пристроями. Задля всіх пристроїв сімейства виконується принцип програмної сумісності «знизу-догори». При цьому системи команд наступних пристроїв доповнюється новими командами і способами адресування, зберігаючи усі базові.

Мікропроцесори сімейства *MC*680*x*0 фірми *Motorola* використовуються в персональних комп'ютерах *Macintosh*, а також в розподілених системах керування складними об'єктами, обмін даними з якими виконується стандартною шиною *VME*. Для використовування в системах керування розроблено низку модифікацій МП *MC*68000, які сполучено під назвою *MC*68*EC*0*x*0 (*EC – Embedded Controller*). До їхнього складу входять інтегровані процесори, комунікаційні контролери тощо.

Базова модель – мікропроцесор *MC*68000 – має 16-розрядну зовнішню шину даних та 24-розрядну шину адреси, що дозволяє адресувати простір пам'яті 16 Мбайт. Максимальна тактова частота для цього процесора – 16,7 МГц. МП *MC*68000, який продукується за *n*-МОП-технологією, споживає близько 1,6 Вт, а мікропроцесор *MC*68*HC*000, який створено за КМОП-технологією, – 260 мВт. Тактову частоту процесора *MC*68*EC*000 доведено до 20 МГц за споживання потужності 380 мВт.

Залежно від класу розв'язуваних завдань, що вирішуються, програмні й апаратні ресурси системи, в складі якої працюють МП сімейства *MC*680*x*0, поділяються в такий спосіб, щоби забезпечити передусім керування роботою власне мікропроцесорної системи за допомогою системного програмного забезпечення, а також задля розв'язування прикладних завдань користувача. Задля розв'язування згаданих завдань зреалізовано два режими МП:

- режим супервізора;
- режим користувача.

В режимі супервізора дозволено виконування усіх команд і забезпечено доступ до всіх регістрів МП. В режимі користувача заборонено виконування деяких команд і обмежено доступ до певних регістрів задля запобігання внесення таких змін у режимі МП, щоби розв'язання низки завдань стало неможливим.

Поточний режим функціонування визначається значенням біта *S* регістра стану. При вмиканні МПС мікропроцесор автоматично встановлює режим супервізора $S = 1$, і працює під керуванням операційної системи. Перехід до режиму користувача здійснюється встановлянням біта *S* до стану логічного 0. У режимі користувача змінювання режиму роботи МП не дозволяється; перехід може статися в разі виникання виняткових ситуацій (переривань) чи внаслідок нового запуску МП (*RESET*).

До складу регістрової моделі МП *MC*68000, наведеної на рис. 11.1, входять два набори 32-розрядних регістрів, кожний з котрих вміщує по вісім однакових регістрів, програмний лічильник та регістр стану.

Вісім регістрів даних *D*7...*D*0 призначено задля зберігання даних.

Вісім регістрів адреси *A*7...*A*0 призначено для зберігання адрес комірок пам'яті, визначених програмістом. Регістр *A*7 (*A*′7) є дубльований і має

заздалегідь визначене функціональне призначення. Він використовується як вказівник стека. Залежно від режиму функціонування МП, він слугує за вказівник стека користувача *USP* чи то за вказівник стека супервізора – *SSP*. Це дозволяє розділити стеки при розв'язуванні завдань користувача і супервізора.

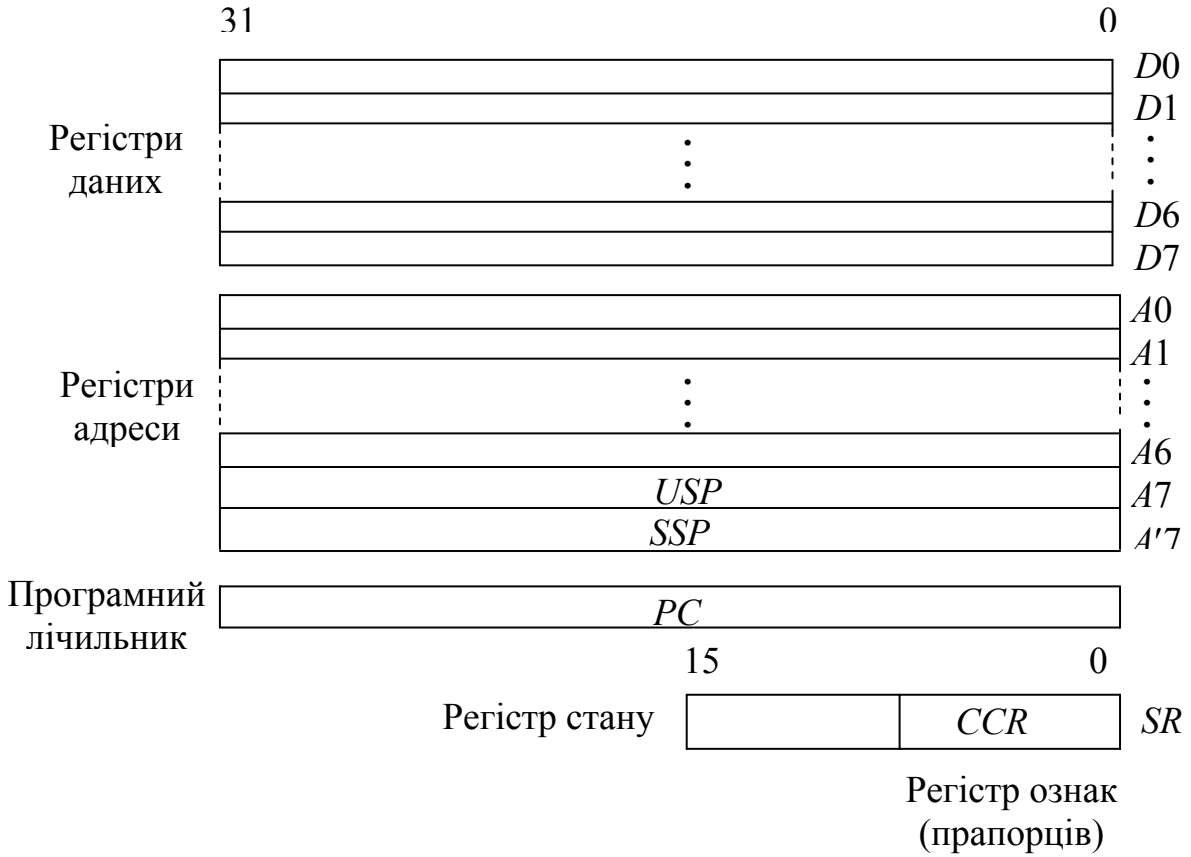



Рисунок 11.1 – Регістрова модель мікропроцесора *MC*68000

Програмний лічильник *PC* призначено для зберігання адрес команд. Зважаючи на те, що МП має 24-розрядну шину адрес, у програмному лічильнику використовуються лише 24 розряди.

Регістр стану *SR* складається з системного байта та байта користувача. Повністю цей регістр є доступний лише в режимі супервізора. В режимі користувача доступ є лише до байта користувача – регістра ознак *CCR*, який використовується окремо. Формат регістра стану подано на рис. 11.2.

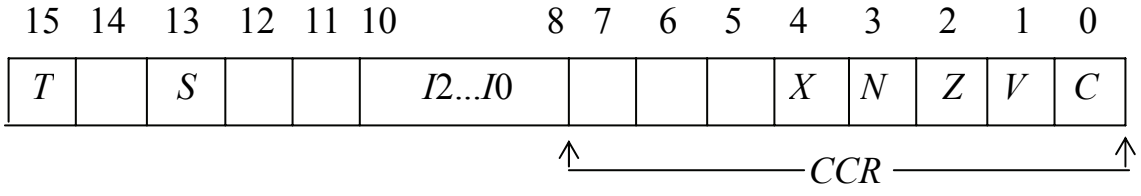



Рисунок 11.2 – Формат регістра стану *SR*

Регістр ознак (прапорців) *CCR* складається з восьми тригерів, п'ятеро з яких призначено для зберігання певних ознак, якими можна схарактеризувати результат виконання операції АЛП.

Окремі біти регістра ознак мають таке призначення:

*C* (ознака перенесення) – перенесення зі старшого розряду при виконуванні арифметичних операцій та зберігання вмісту висуваного розряду при виконуванні операцій зсувів. Набирає значення $C = 1$ за виникнення перенесення зі старшого розряду результату виконуваної операції;

*V* (ознака переповнення) – набуває значення $V = 1$ в разі переповнення розрядної сітки при обробці операндів зі знаком;

*Z* (ознака нульового результату) – за здобуття результату, який дорівнює 0, ознака набуває значення 1;

*N* (ознака знаку) – зберігає копію знакового розряду результату операції. Значення ознаки $N = 1$ відповідає від'ємному результатові;

*X* (ознака розширення) – при виконуванні переважної більшості операцій копіює значення біта *C*. В певних випадках значення цього біта формується залежно від виконуваної операції.

Біти системного регістра *SR* визначають режим функціонування МП і мають призначення:

*T* (ознака трасування) – за допомогою цієї ознаки здійснюється перехід до покрокового виконування програми за $T = 1$;

*S* (ознака супервізора) – визначає режим роботи МП. Значення ознаки $S = 1$ відповідає функціонуванню МП в режимі супервізора, а $S = 0$ – в режимі користувача;

*I*2...*I*0 – поле маски переривань, визначає мінімальний рівень обслуговуваних запитів переривань.

Інші біти регістра *SR* не використовуються або їх зарезервовано задля використовування в наступних моделях.

Мікропроцесор *MC*68000 обробляє дані – операнди, які можуть бути бітами, байтами, словами та довгими словами, а також дані, які подано у вигляді двійково-десяткових чисел. Операнди можуть розміщуватись у регістрах мікропроцесора (даних чи адреси), а також у комірках пам'яті. При цьому, позаяк шина даних МП має місткість 16 біт, а пам'ять має байтову організацію, то для роботи зі словами чи довгими словами слід задавати адресу старшого байта операнда, яка має бути парною чи кратною до 4. Це дозволяє звертатися спочатку до старшого байта слова, а потім до молодшого, який розміщується у комірці пам'яті, адреса котрої є більша на 1 за адресу старшого байта. Таке послідовне розміщення операндів у пам'яті відповідає звичайному розміщенню розрядів числа при написанні чисел. Такий порядок адресування байтів називається *big-endian*, на відміну від порядку адресування *little-endian*, який використовується в МП фірми *Intel*. Розміщення слів (а) і довгих слів (б) у пам'яті та їхнє адресування подано на рис. 11.3. Адреса *N* завжди має бути парною.

Підмикання ВІС мікропроцесора до шин МПС здійснюється за допомогою виводів, функціональне призначення котрих та їхні умовні назви подано на рис. 11.4 (виводи живлення на схемі не зазначено). Напрям проходження сигналів зазначено стрілками. Підключення МП до шини адреси МПС здійснюється за допомогою 24-розрядної шини адреси (*A*23...*A*0), на яку

виставляється адреса пристрою, до котрого здійснюється звернення. Команди та дані надходять на 16-розрядну шину даних (*D*15...*D*0), якою здійснюється обмін словами. Передавання довгих слів здійснюється за два цикли шини (спочатку старша частина довгого слова, потім молодша).

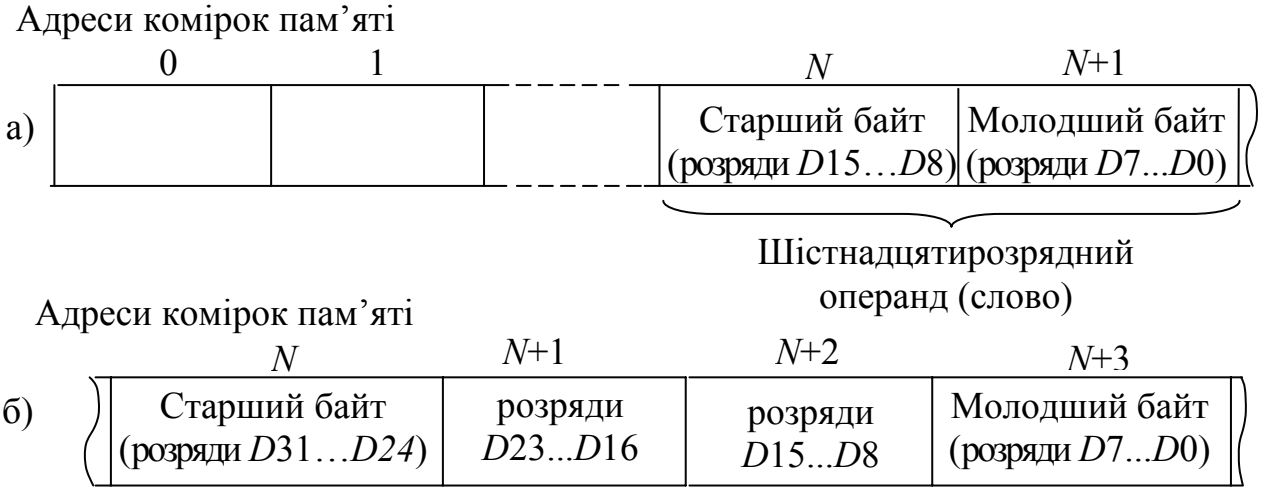



Рисунок 11.3 – Адресування байтів в слові (а) і довгому слові (б)

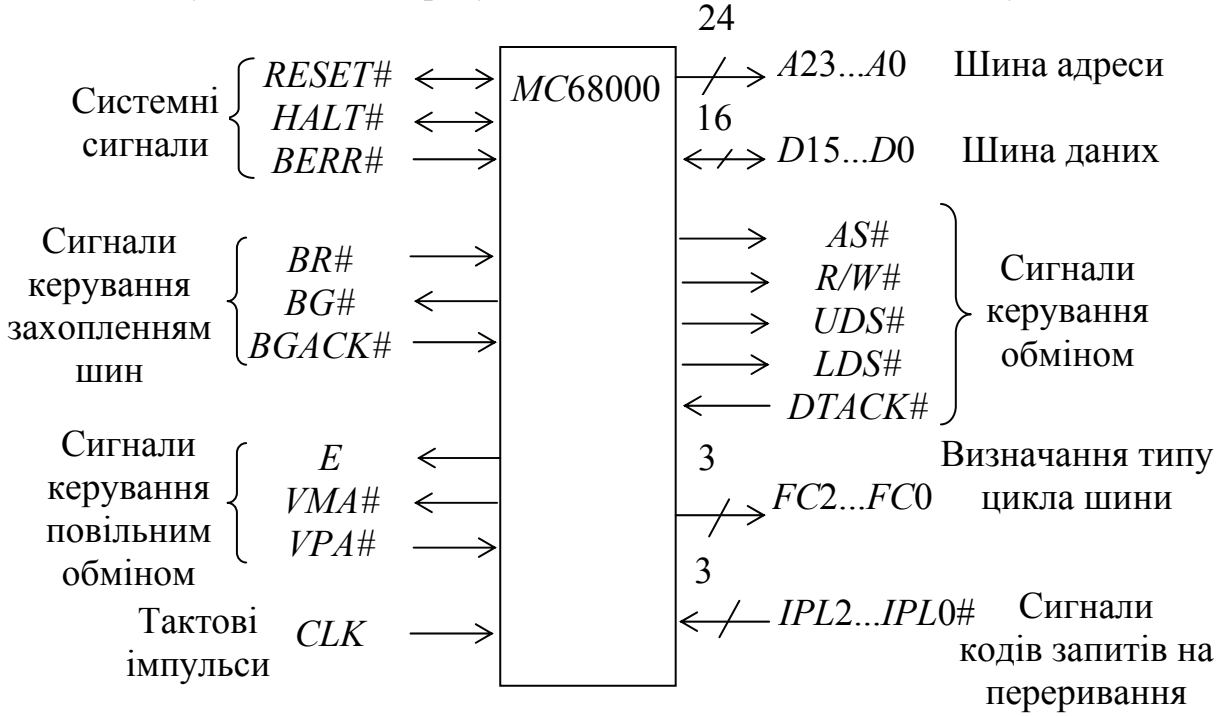



Рисунок 11.4 – Призначення виводів ВІС *MC*68000

Сигнали керування поділено на групи відповідно до функціонального призначення, як подано на рис. 11.4. Знак # засвідчує, що активним рівнем сигналу є значення логічного 0 (низький рівень).

*CLK* – сигнал тактової синхронізації. Частота цього сигналу зумовлює тривалість такту машинного часу.

Групу системних сигналів становлять три сигнали:

– *RESET#* – сигнал скидання. Вихідний сигнал *RESET#* формується при виконуванні команди *RESET* і встановлює всі пристрої МПС до початкового

стану. Вхідний сигнал *RESET#* призводить до апаратного виключення, яке встановлює початковий стан МП. При цьому вміст регістрів адреси обнулюється, значення біта S встановлюється в 1, що відповідає режимові супервізора. Далі з комірки пам'яті з адресою $000000 завантажується значення вказівника стека супервізора *SSP*, а з комірки з адресою $000004 до програмного лічильника *PC* – адреса підпрограми ініціалізації системи. Ця підпрограма завантажує потрібні початкові значення до регістра *SR*, регістрів адреси та даних, а також виконує ініціалізацію пристроїв, які входять до складу МПС. Програмне скидання виконується в режимі супервізора командою *RESET*, яка також встановлює початковий стан МП.

– *HALT#* – сигнал зупину. Вхідний сигнал *HALT#* припиняє виконання поточної програми, переводить шину адреси та шину даних до відключеного стану, а виходи керування – до неактивного стану. Вихідний сигнал *HALT#* формується за подвійної помилки звернення до шини. Вихід зі стану зупину відбувається подаванням вхідного сигналу *RESET#.*

– *BERR#* – вхідний сигнал помилки звернення до шини адреси. Цей сигнал формується схемою контролера шини, якщо на шину адреси подано адресу пристрою, що не існує, або адресу, яка не входить до адресного простору МПС. Цей сигнал також формується за тривалої відсутності сигналу готовності *DTACK#* і за інших порушень обміну.

Групу сигналів керування обміном подавано п'ятьма сигналами:

– *AS#* – адресний строб є сигнал, формування якого зумовлює момент завершення формування адреси на шині; активний стан цього сигналу зберігається до завершення циклу обміну.

– *R/W#* – читання/запис. Значення сигналу зумовлює напрям передавання даних. Сигнал *R/W#,* котрий дорівнює 1, зумовлює читання (введення) даних до МП; значення сигналу, що він дорівнює 0, – навпаки, виведення (запис) даних з МП.

– *UDS#* – передавання старшого байта даних.

– *LDS#* – передавання молодшого байта.

Сигнали *UDS#, LDS#* зумовлюють розмір і порядок передавання даних шиною. Значення *LDS#* = 0 водночас з *UDS#* = 1 відповідає передаванню молодшого байта, *LDS#* = 0 та *UDS#* = 0 – передаванню слова.

– *DTACK#* – вхідний сигнал підтвердження готовності до обміну, який надходить від пам'яті чи зовнішніх пристроїв.

Групу сигналів керування захопленням шин становлять сигнали, які визначають порядок спільного використання шини даних кількома пристроями системи. Ці сигнали використовуються задля забезпечення режиму безпосереднього доступу до пам'яті, а також для організації спільної роботи кількох МП у багатопроцесорних системах. До цієї групи входять сигнали:

– *BR#* – вхідний сигнал будь-якого зовнішнього пристрою на захоплення шини. Отримавши такий сигнал, якщо переривання дозволено, МП завершує виконання поточного циклу обміну, припиняє виконування команди, переводить власні виводи *A*23...*A*0 та *D*15...*D*0 до відключеного стану, а виходи керування – до неактивного стану.

– *BG#* – вихідний сигнал дозволу захоплення шин. Формується після відключення МП від шини.

– *BGACK#* – вхідний сигнал підтвердження захоплення шини зовнішнім пристроєм. Формується зовнішнім пристроєм після отримання сигналу *BG#* і переходу до режиму керування шиною. Формування сигналу *BR#* після цього припиняється.

Після завершення обміну зовнішній пристрій встановлює значення сигналу *BGACK* у 1 і МП продовжує виконування своєї роботи, яку він перервав.

Сигнали керування повільним обміном дозволяють підключати до МП зовнішні пристрої, які мають відносно невелике значення тактової частоти. Задля організації роботи з ними використовуються відповідні сигнали керування. До цих сигналів належать:

– *E* – тактовий сигнал для зовнішнього пристрою. На цьому виході формується сигнал, який має частоту, вдесятеро меншу за тактову частоту МП (*CLK*). Цей сигнал може використовуватися зовнішніми пристроями в якості сигналу тактової частоти. При цьому сигнали адреси й сигнали *AS#* та *R/W#* формуються як за звичайного обміну даними.

– *VPA#* – сигнал готовності до повільного обміну. Цей сигнал формує зовнішній пристрій, адресу якого встановлено на шині адреси і він є готовий до роботи.

– *VMA#* – сигнал підтвердження повільного обміну. Цей сигнал формує МП у відповідь на отримання ним сигналу *VPA#.* Після його встановлення розпочинається обмін даними у повільному режимі.

У кожному циклі обміну МП визначає тип циклу, який виконується. Задля цього МП формує код, який виставляється на шину визначення типа циклу *FC*2...*FC*0. Відповідність типа циклу і коду наведено у табл. 11.1.

Таблиця 11.1 – Типи виконуваних циклів

| *FC*2 | *FC*1 | *FC*0 | Тип циклу |
|---|---|---|---|
| 0 | 0 | 0 | резервовано |
| 0 | 0 | 1 | вибирання даних користувача |
| 0 | 1 | 0 | вибирання команд користувача |
| 0 | 1 | 1 | резервовано |
| 1 | 0 | 0 | резервовано |
| 1 | 0 | 1 | вибирання даних супервізора |
| 1 | 1 | 0 | вибирання команд супервізора |
| 1 | 1 | 1 | підтвердження переривання |

Дешифрування коду виконуваного циклу надає можливість розділити пам’ять для режимів супервізора та користувача, сформувати сигнал підтвердження переривання *INTA#* або ідентифікувати поточний стан МП.

Переривання і виключення – це спеціальні процедури, виконувані МП, якщо при роботі МПС виникають ситуації, котрі потребують тимчасового

припинення головної програми й обслуговування інших програм (обробки переривань) задля відповідної реакції системи на обставини, які призвели до цього.

МП може обслуговувати до семи запитів на переривання від зовнішніх пристроїв, які формують сигнали *IRQ*1...*IRQ*7, відповідно до свого пріоритету. Ці сигнали оброблюються за допомогою пріоритетного шифратора і надходять на входи *IPL#.* Відповідність пріоритетів запиту і сигналів *IPL*2...*IPL*0# подано у табл. 11.2.

Таблиця 11.2 – Кодування запитів на переривання

| Пріоритет | Запити на переривання відповідно до пріоритету | Код запиту *IPL*2...*IPL*0 |
|---|---|---|
| 0 (мінімальний) | відсутність запиту | 111 |
| 1 | *IRQ*1 | 110 |
| 2 | *IRQ*2 | 101 |
| 3 | *IRQ*3 | 100 |
| 4 | *IRQ*4 | 011 |
| 5 | *IRQ*5 | 010 |
| 6 | *IRQ*6 | 001 |
| 7 (максимальний) | *IRQ*7 | 000 |

**Виключення.** Виключення (*Exception*) – особливі ситуації при роботі МПС, які є реакцією системи на непередбачувані, переважно аварійні, ситуації, обслуговування яких передбачає переривання головної програми і перехід до підпрограми обробки цього виключення. Виключення бувають апаратними або програмними. Апаратні виключення зумовлено будь-якими порушеннями функціонування пристроїв системи, як периферійних, так і внутрішніх, або аварійними (вимкнення живлення, відмикання з'єднувальних ліній тощо). Програмні виключення виникають за неправильного функціонування програми: формування неіснуючої адреси, формування помилкового коду команди, ділення на 0, покрокового виконання програми тощо. При обслуговуванні виключень МП автоматично переходить до режиму супервізора, що надає підпрограмі обслуговування виключення доступ до всіх ресурсів системи.

Апаратні виключення зумовлено такими причинами:

– надходження зовнішнього сигналу *RESET#* ;

– формування зовнішнього сигналу помилки звернення до шини *BERR#;*

– надходження від периферійних пристроїв запитів на переривання;

– відсутність сигналів підтвердження готовності до обміну *DTACK#* або сигналу готовності до повільного обміну *VPA#;*

– надходження запиту на переривання від периферійного пристрою, який попередньо не ініційовано;

– відключення живлення та інші аварійні ситуації.

Програмні виключення можуть виникати з таких причин:

– надходження команд, які спричинюють виключення *RESET, TRAP, TRAPV, CHK;*

– надходження привілейованих команд у режимі користувача;

– ділення на 0 (команди *DIVS, DIVU*);

– надходження команди, яка має помилковий код операції;

– встановлення покрокового режиму роботи (встановлення біта $T = 1$);

При цьому виключення відбувається на кожному кроці виконування програми і МП виводить на екран монітора результати виконання поточної команди й дозволяє змінювати ці результати в перебігу налагодження;

– надходження команди, яка має у полі коду операції значення 1010. Цей код забезпечує перехід до підпрограми-емулятора команд, які не входять до системи команд МП;

– формування непарної адреси.

Обслуговування всіх виключень відбувається однаково: МП переходить до режиму супервізора, встановлюючи значення біта $S = 1$, потім МП зберігає у стеку вміст програмного лічильника *PC* і регістра стану *SR* задля можливості повернення до перерваної програми. Потім до програмного лічильника завантажується вектор виключення ($V_e$), котрий є адресою початку відповідної програми обслуговування. Якщо виключення зумовлено помилкою шини чи формуванням помилкової адреси, то до стека також завантажується значення адреси, яка спричинилася до виключення, код виконуваної команди і код циклу (*FC*2...*FC*0), який при цьому виконувався. Ця інформація використовується підпрограмою обслуговування переривання задля з'ясування причини, яка зумовила виключення.

Вектори виключень розміщено у пам'яті МПС і поєднано у таблицю виключень. Цю таблицю розміщено в області пам'яті з адресами \$000...\$3*FF*. Кожний вектор адресовано двома словами, тому в таблиці є входи для 256 підпрограм виключень. Кожний вектор має власний номер $N_e$ від 0 до 255 і кожному номеру відповідає власна адреса $A_e$, в якій зберігається вектор $V_e$ відповідної підпрограми. Адреса визначається в результаті множення номера вектора на 4 – ($A_e = 4\ N_e$). Множення відбувається у результаті логічного зсуву на два розряди ліворуч. Різновиди виключень та їхніх векторів подано у табл. 11.3.

Повернення з підпрограми обслуговування виключення здійснюється за допомогою команд *RTE* чи *RTS*. Командою *RTE* завершується підпрограма обслуговування виключення в режимі супервізора, а командою *RTS* – якщо при виконуванні підпрограми обслуговування МП перейшов до режиму користувача.

Виключення з номером $N_e = 0$ може бути програмним, за надходження команди *RESET*, або апаратним, за формування сигналу зовнішнього скидання *RESET#.* В обох випадках з комірки пам'яті з адресою \$000 завантажується значення вказівника стека супервізора – *SSP*, а з комірки пам'яті з адресою \$004 – початкова адреса підпрограми ініціалізації всієї системи. Отже, це виключення має два вектори виключення $V_e$.

Таблиця 11.3 – Різновиди виключень та їхні адреси

| Номер виключення $N_e$ | Адреса виключення $A_e$ | Вид виключення |
|---|---|---|
| 0 | \$000 | Встановлення початкового стану (завантаження *SSP*) |
| – | \$004 | Встановлення початкового стану (завантаження *PC*) |
| 2 | \$008 | Помилка звернення до шини |
| 3 | \$00*C* | Формування помилкової адреси (непарної) |
| 4 | \$010 | Помилковий код операції |
| 5 | \$014 | Ділення на 0 |
| 6 | \$018 | Команда *CHK* |
| 7 | \$01*C* | Команда *TRAPV* |
| 8 | \$020 | Порушення привілеїв |
| 9 | \$024 | Покроковий режим ($T = 1$) |
| 10 | \$028 | Код емуляції 1010 |
| 11 | \$02*C* | Код емуляції 1111 |
| 12...14 | \$030...\$038 | Резервовано |
| 15 | \$03C | Неініційоване переривання |
| 16...23 | \$040...\$05*E* | Резервовано |
| 24 | \$060 | Помилкове переривання |
| 25...31 | \$064...\$07*C* | Автовекторні переривання |
| 32...47 | \$080...\$0*BF* | Команди *TRAP* з номерами $N_t = 0...15$ |
| 48...63 | \$0C0...\$0*FF* | Резервовано |
| 64...255 | \$100...\$3*FF* | Векторні переривання користувача |

Виключення з номером $N_e = 2$ (помилка звернення до шини) здійснюється за сигналом спеціального пристрою, якщо на шині адреси формується адреса неіснуючого (непідімкненого) пристрою. Також це виключення може відбуватися за сигналом вартового таймера, якщо певного часового інтервалу не формується сигнал підтвердження готовності до обміну *DTACK#* від пам'яті або зовнішніх пристроїв. Підпрограма обслуговування цього виключення визначає тип помилки, адресу і команду, яка зумовила виключення. Якщо при обслуговуванні виникає повторний сигнал помилки звернення, то МП переходить до режиму зупину і формує відповідний вихідний сигнал *HALT#*.

Виключення кодів емуляції 1010 та 1111 дозволяють виконувати команди, які не входять до системи команд МП *MC*6800. Приміром, підпрограма обслуговування виключення, зустрівши команду, яка має код 1010, аналізує інші розряди слова команди і залежно від результатів аналізу виконує певну послідовність команд, яка відповідає виконуванню макрокоманд, що розширює можливості МП і сприяє реалізації макроасемблерів. Код емуляції 1111 використовується задля реалізації команд арифметичного співпроцесора. Отже, існує можливість за допомогою МП *MC*68000 налаштовувати і

виконувати програми, які використовують команди арифметичного співпроцесора.

Виключення з номерами 15 (неініційоване переривання) і 24 (помилкове переривання), а також векторні й автовекторні переривання обслуговуються за надходження запитів від зовнішніх пристроїв.

Номери виключень команди *TRAP* формуються відповідно до виразу $N_e = 32 + D_t$. Це виключення використовується задля завдавання контрольних точок зупину при налагоджуванні програми, а також задля включення підпрограм обслуговування пристроїв, які використовуються спільно з МП.

**Переривання** (*Interruption*) – особливі ситуації, які виникають при функціонуванні МПС задля обслуговування певних пристроїв, робота яких відбувається за участі центрального процесора і не передбачена програмою, яка виконується. Як було розглянуто у розділі 7.2.2, переривання виникають за надходження відповідних команд (програмні переривання) або сигналів від зовнішніх пристроїв (апаратні переривання).

Задля обслуговування переривань МП припиняє виконування головної програми, зберігає у стеку адресу останньої команди, яку він виконав (адреса повернення), завантажує початкову адресу підпрограми обслуговування переривання і розпочинає її виконувати. Після завершення підпрограми МП поновлює у програмному лічильнику адресу повернення і продовжує виконування головної програми. Обслуговування переривання розпочинається за запитом на переривання, який формується пристроєм, що він потребує обслуговування.

Зовнішні пристрої, які можуть спричинити переривання, поділено відповідно до важливості їхнього функціонування, і кожному з них призначено власний пріоритет обслуговування. МП може обслуговувати до семи запитів на переривання від різних пристроїв. Кодування запитів наведено у табл. 11.2.

Обслуговування зовнішніх пристроїв відбувається відповідно вмісту поля маски переривання (*I*2...*I*0) у регістрі *SR*. Це поле визначає рівень пріоритету пристрою (*IRQ*) і дозволяє чи не дозволяє його обслуговування. Відповідність коду запитів і можливості його обслуговування наведено у табл. 11.4.

Таблиця 11.4 – Кодування можливості обслуговування переривань

| Запит на переривання | Маска *I*2...*I*0 | Запити, що обслуговуються |
|---|---|---|
| відсутність запиту | 000 | *IRQ*1...*IRQ*7 |
| *IRQ*1 ($L_i = 1$) | 001 | *IRQ*2...*IRQ*7 |
| *IRQ*2 ($L_i = 2$) | 010 | *IRQ*3...*IRQ*7 |
| *IRQ*3 ($L_i = 3$) | 011 | *IRQ*4...*IRQ*7 |
| *IRQ*4 ($L_i = 4$) | 100 | *IRQ*5...*IRQ*7 |
| *IRQ*5 ($L_i = 5$) | 101 | *IRQ*6...*IRQ*7 |
| *IRQ*6 ($L_i = 6$) | 110 | *IRQ*7 |
| *IRQ*7 ($L_i = 7$) | 111 | *IRQ*7 |

Адресування підпрограм обслуговування запитів переривань може виконуватися у два способи. При автовекторному перериванні кожному запитові з рівнем $L_i$ відповідає номер виключення згідно з табл. 11.3 з адресами $A_e$ = \$064...\$07*C*. За векторного переривання зовнішній пристрій формує і передає до МП будь-яке значення номера $N_e$. Пристрої, які можуть спричинити векторні переривання, мають спеціальний векторний регістр *IVR*, в якому зберігається відповідне значення $N_e$, яке записується до нього за ініціалізації. Якщо ініціалізацію не проведено, то пристрій формуватиме код з номером $N_e$ = 15 (неініціалізоване переривання).

Кожне переривання розпочинається з того, що пристрій, який потребує обслуговування, формує запит *IRQ#,* котрий надходить на пріоритетний шифратор, який формує сигнали коду запиту *IPL*0...*IPL*2, відповідно до призначеного пріоритету пристрою. Якщо код запиту є не менший за значення *I*2...*I*0 у регістрі *SR*, то МП завершує виконування поточної команди і виконує цикл підтвердження переривання. В цьому циклі МП формує на виводах *FC*2...*FC*0 код 111 – підтвердження переривання (табл. 11.1), який перетворюється на сигнал *IACK#,* котрий, відповідно до адреси, надходить на пристрій, який потребує обслуговування. Приклад схеми формування сигналів наведено на рис. 11.5.

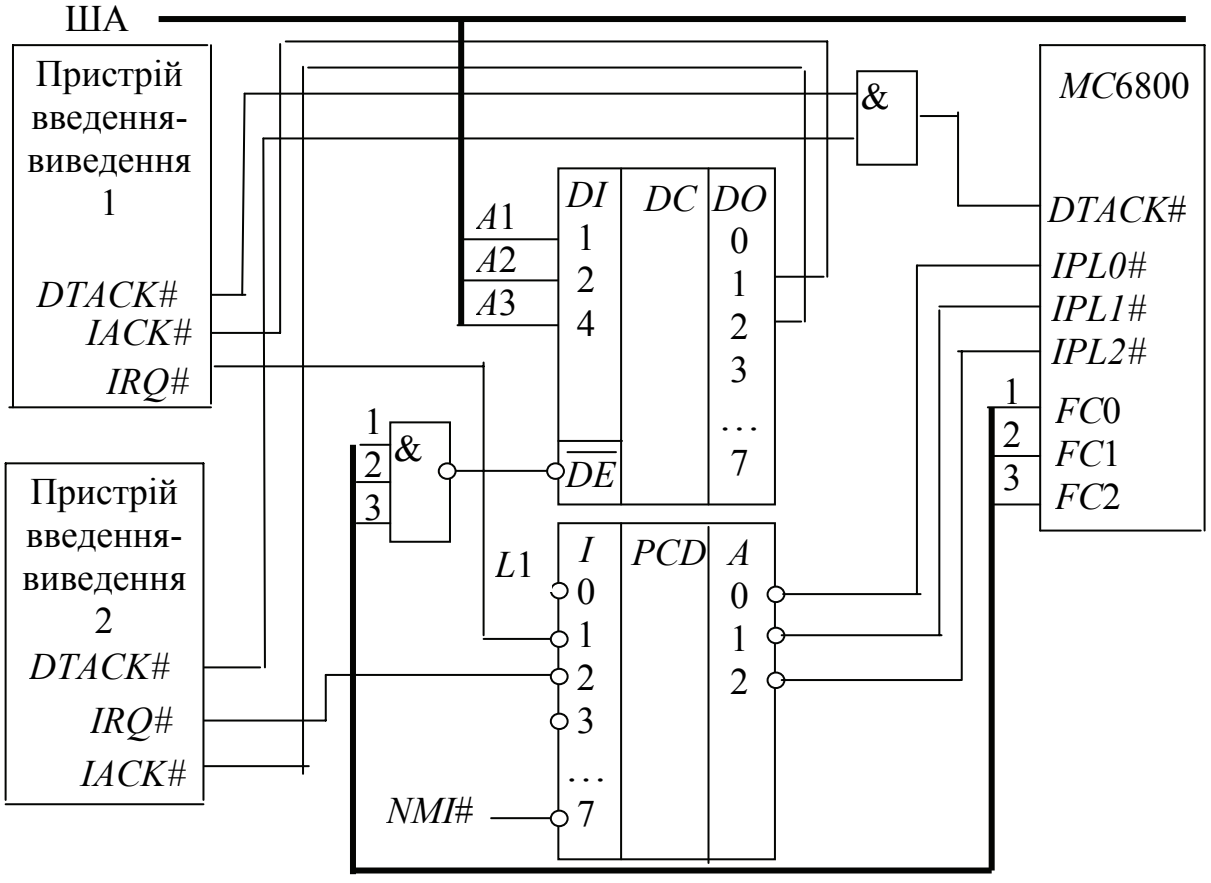



Рисунок 11.5 – Схема формування сигналів переривання

При виконуванні автовекторного запиту на переривання зовнішній пристрій, який отримав сигнал *IACK#,* підтверджує свій запит, формуючи сигнал низького рівня на вході *VPA#* мікропроцесора. Після цього МП формує

адресу вектора виключення $A_V$ = \$064...\$07*C* відповідно до рівня пріоритету пристрою, котрий потребує обслуговування.

При опрацьовуванні векторного переривання зовнішній пристрій, який отримав сигнал *IACK#,* встановлює на молодші розряди шини даних *D*7...*D*0 власний номер виключення $N_e$ і підтверджує передання номера, формуючи сигнал *DTACK#.* МП приймає значення номера і, відповідно до нього, формує адресу вектора виключення $A_V$ = \$100...\$3*FF*.

Якщо при підтвердженні переривання не буде сформовано сигнали *VPA#* або *DTACK#,* то МП виконуватиме виключення з номером 24 – «помилкове переривання».

Наступна модель *MC*68010 передбачає підмикання мікросхеми диспетчера пам'яті *MC*68851 (*Memory Management Unit – MMU*).

Блок *MMU* зорганізовує сторінкове адресування пам'яті з розміром сторінки від 256 байт до 32 кбайт. Адреса комірки пам'яті, яку формує МП, сприймається як ефективна або виконавча. На виходи *MMU* ця адреса транслюється як фізична. Якщо логічна адреса відповідає сторінці, яка є відсутня в оперативній пам'яті, то формується сигнал помилки звернення до пам'яті *BERR#.* Цей сигнал спричинює переривання, – і операційна система виконує перезапис вмісту відсутньої сторінки з жорсткого диску до адресованої оперативної пам'яті замість сторінки, до якої, довш за все, не було звернення. Вміст старої сторінки переписується на жорсткий диск, а МП обирає команду або дані зі сторінки, перенесеної до оперативної пам'яті. Зреалізований в такий спосіб механізм віртуалізації пам'яті дозволяє здійснювати обмін сторінками із зовнішньою пам'яттю великого обсягу, використовуючи оперативну пам'ять меншого обсягу. Сигнал *BERR#* формується також у разі помилки при задаванні адреси пам'яті або пристроїв введення-виведення.

**Контрольні питання:**

1 Чим відрізнюється програмна модель МП *MC*68000 від програмної моделі фірми Intel?

2 В яких двох режимах може працювати МП *MC*68000?

3 Чим відрізняються режими, в яких може працювати МП *MC*68000?

4 Які регістри входять до регістрової моделі користувача МП *MC*68000?

5 В який спосіб можна перейти до режиму супервізора при роботі з МП *MC*68000?

6 З даними якої розмірності працює МП *MC*68000 і в який спосіб відбувається звернення до шини даних при роботі з кожним з них?

7 У який спосіб виконується адресування байтів даних при обробці слів та довгих слів?

8 Які сигнали визначають розмірність даних, котрі оброблюються командою?

9 В чому полягає призначення сигналу BERR# і за яких умов відбувається його формування?

**Контрольні питання підвищеної складності:**

1 Якими сигналами обмінюються МП та зовнішній пристрій за взаємодії за різних способів обміну?

2 Які типи циклів роботи може зреалізовувати МП *MC*68000?

3 Чим відрізнюються тип циклу вибору даних супервізора від вибору даних користувача?

## 11.2 Побудова МПС на 16-розрядних мікропроцесорах фірми Motorola

### *11.2.1 Підсистема центрального процесорного елемента MC68000*

**Вхідний контроль:**

1 Яку розрядність мають ШД та ША МП *MC*68000?
2 Які системні сигнали ВІС МП68000 Вам відомі?
3 Чи є згадані в п. 2 сигнали односпрямовані або двоспрямовані й чому?
4 За яким алгоритмом працює пріоритетний шифратор?
5 На якій частоті працює МП *MC*68000?

До підсистеми центрального процесорного елемента (ЦПЕ) МПС входять пристрої (ВІС), які забезпечують його роботу, власне мікропроцесор *MC*68000 і пристрої, до яких належать:

– генератор тактових імпульсів, який формує послідовність імпульсів тактової частоти для всієї МПС;

– формувач сигналів керування МПС, який формує всі сигнали, необхідні для вибору вузлів МПС, керування вибором розрядності операндів, контролю за формуванням адреси пристроїв, перериваннями тощо;

– буфер шини даних – пристрій, який забезпечує необхідний рівень навантажувальної здатності виходів шини даних ВІС *MC*68000. Він являє собою двоспрямований приймач-передавач, який підключається до виходів центрального процесора (ЦП).

Умовне графічне позначення ВІС *MC*68000 наведено на рис. 11.6. На цьому рисунку також подано рекомендоване фірмою підмикання виводів ВІС до джерела живлення. Призначення виводів відповідає рис. 11.4.

Формування сигналів синхронізації та скидання (*RST*) виконується за допомогою схеми генератора тактових імпульсів, поданої на рис. 11.7. В схемі використовується мікросхема *MC*88916 фірми *Motorola*. Задля стабілізації частоти використовується кварцовий резонатор *Z*1. Підключення виводів генератора до ЦП проводиться відповідно до назв виводів; з’єднуються лінії, що мають однакові назви. Сигнал тактової частоти подається на ЦП й інші пристрої схеми.

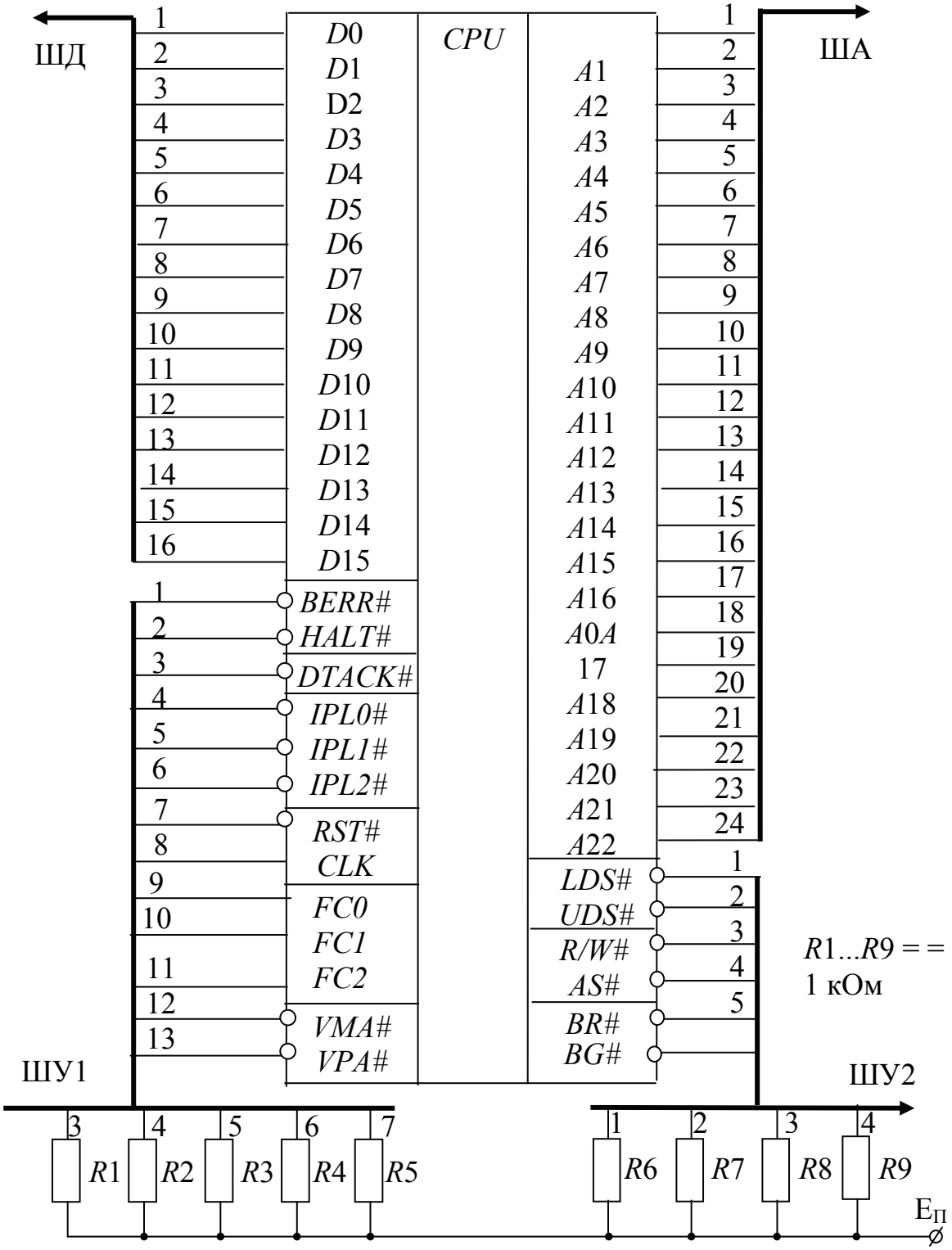



Рисунок 11.6 – Умовне графічне позначення і принципова схема підключення ВІС *MC*68000

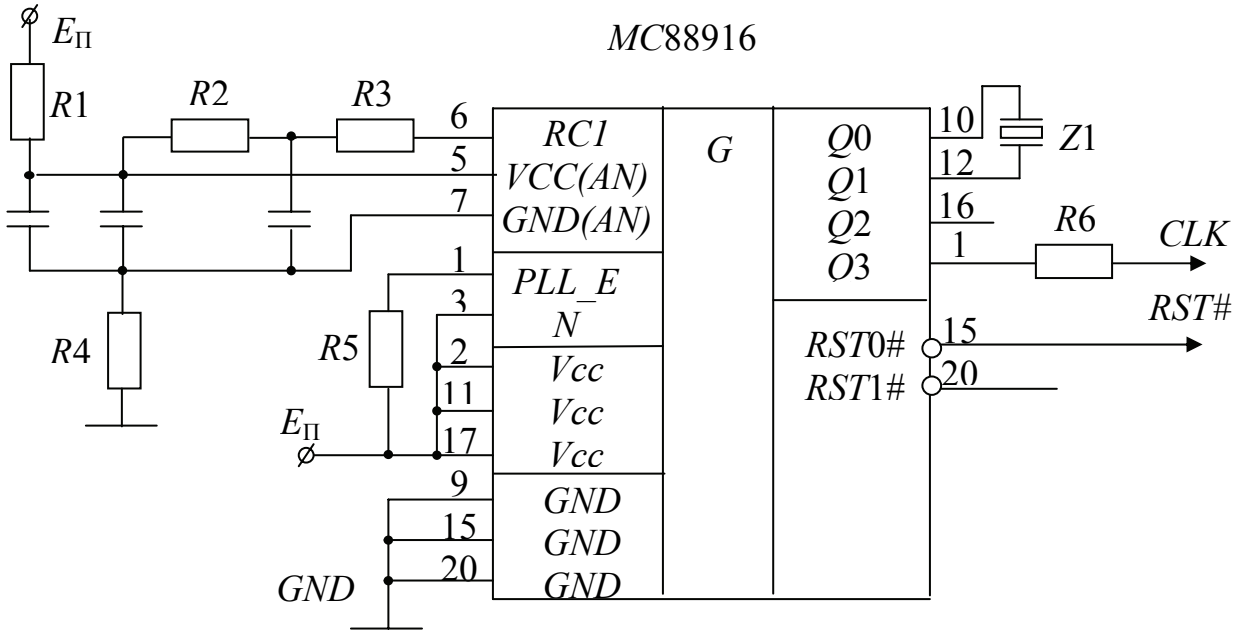



Рисунок 11.7 – Схема генератора тактових імпульсів

Виходи шини даних ВІС *MC*68000 не мають вбудованих підсилювачів потужності вихідних сигналів, тому задля використовування у МПС ця шина потребує використовування спеціальних схем – буферів. В якості таких схем рекомендовано застосовувати приймачі-передавачі 74*F*245, які є 8-розрядними буферними схемами. Кількість мікросхем визначається розрядністю шини даних ВІС *MC*68000 – 16, тому в МПС слід використовувати дві мікросхеми 74*F*245, одна з котрих обслуговує молодший байт шини, а друга – старший. Обробка довгих слів здійснюється за два такти, тому молодша і старша частини будуть обслуговуватися окремо. Для керування роботою схеми слід дешифрувати сигнали *UDS, LDS*, а також сигнали формування типу циклу. Сигнали керування буфером шини даних на схемі позначено *PDEN0*, PDEN1. Принципову схему буфера шини даних наведено на рис. 11.8.

Сигнали керування МПС формуються при дешифруванні сигналів ЦП та сигналів адреси. Отже, ця схема являє собою низку різних дешифраторів, підключених до відповідних кіл схеми. Задля уніфікації схеми та зручності використовування рекомендовано застосовувати програмовану логічну інтегральну схему *FPGA*, запрограмовану відповідно до алгоритмів роботи всіх необхідних дешифраторів та інших схем, потрібних для керування МПС. Умовне графічне позначення програмованої логічної інтегральної схеми *FPGA* подано на рис. 11.9. На цьому рисунку подано також вхідні й вихідні сигнали на виводах схеми *FPGA*.

Приміром, сигнали *BYTE0* (1,3) формуються при дешифруванні сигналів *UDS, LDS*, відповідно до табл. 11.5. Обробка довгих слів здійснюється за два такти, під час виконування котрих зберігається значення коду. Значення сигналу *BYTE0* зберігається при обробці байтів, слів та довгих слів, а *BYTE*1 – при обробці слів та довгих слів, що дозволяє спрощувати організацію багатошарової пам’яті.

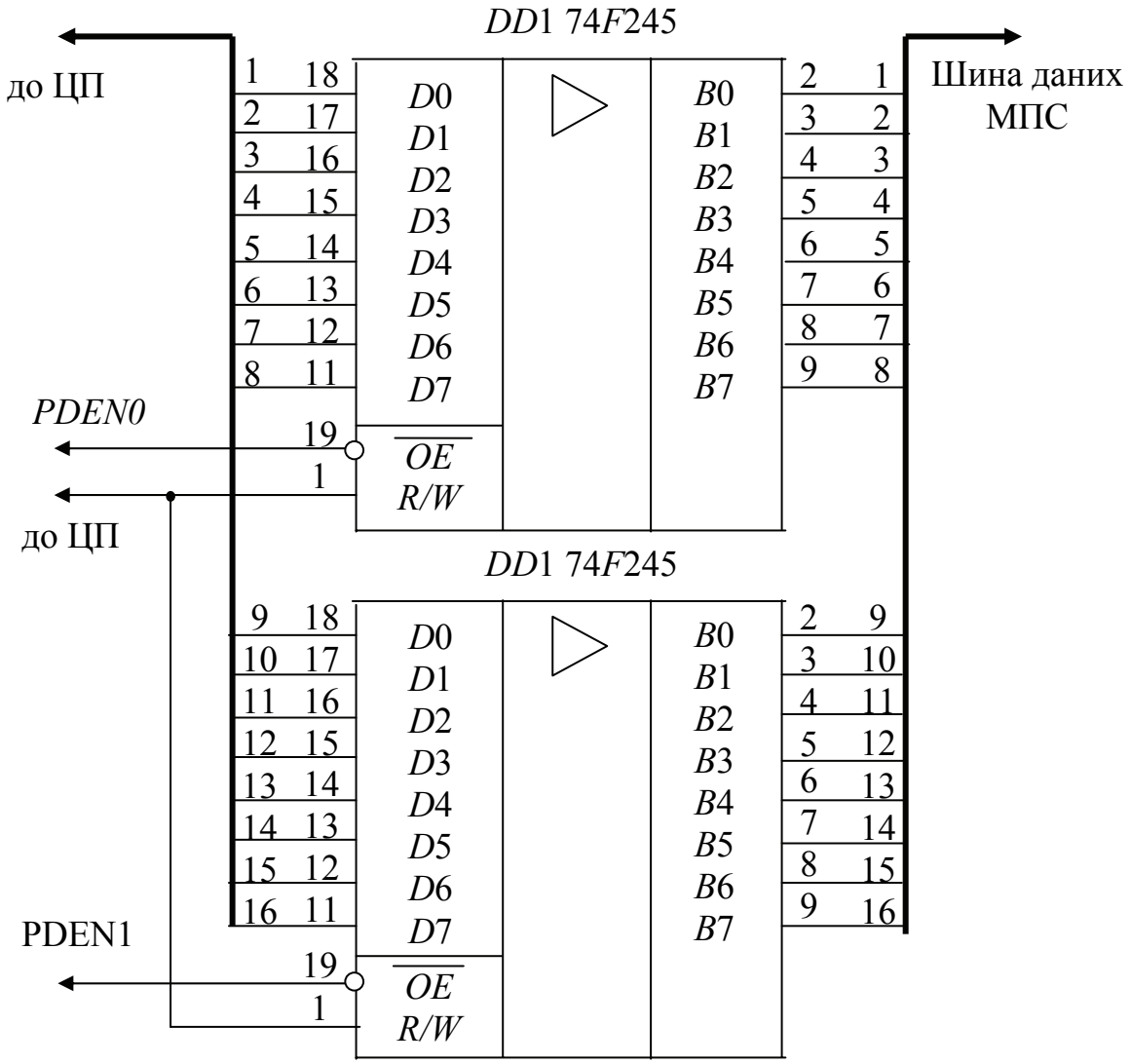



Рисунок 11.8 – Принципова схема буфера шини даних

Таблиця 11.5 – Визначення довжини операнда

| *LDS* | *UDS* | Обробка |
|---|---|---|
| 0 | 0 | слова |
| 0 | 1 | байта |

Сигнали *BACK*1 (2, 3, 4, 5) використовуються для керування пристроями, які входять до складу МПС, формуючи сигнали, котрі переводять певний пристрій до активного стану. Задля формування цих сигналів використовуються сигнали адреси і виконуваного циклу. Визначено, що сигнал *BACK*1 відповідає переведенню до активного стану ПЗП, *BACK*2 – ОЗП, *BACK*3 – асинхронний послідовний приймач-передавач, *BACK*4 – паралельний периферійний інтерфейс, *BACK*5 – таймер.

Умовне графічне позначення схеми *FPGA*, сигнали і виводи, на яких вони формуються, а також адреси підключення показано на рис. 11.9.

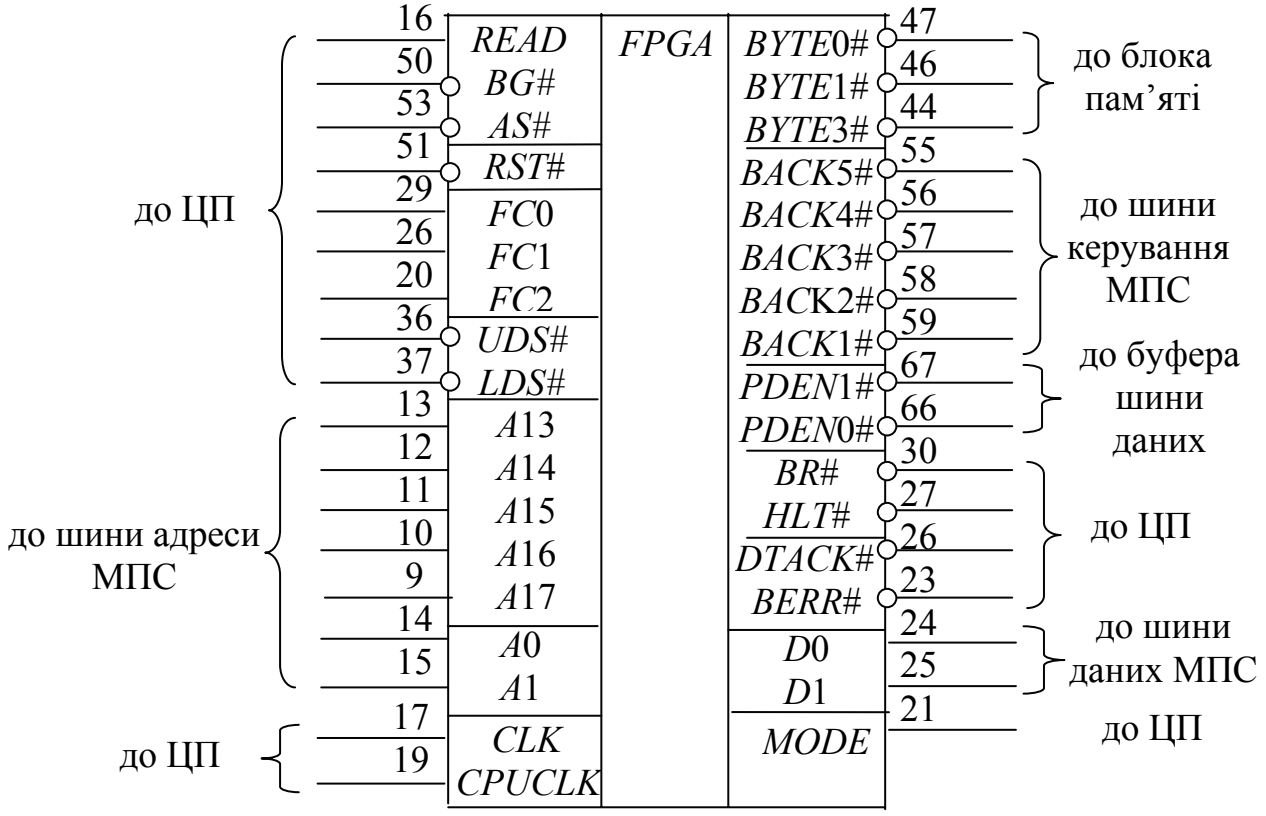



Рисунок 11.9 – Умовне графічне позначення ВІС *FPGA*

**Контрольні питання:**

1 З якою метою до підсистеми ЦП *MC*68000 включено буфер шини даних?

2 Чим відрізнюються сигнали на входах і виходах буфера шини даних?

3 Чим зумовлюється стабільність частоти тактового генератора *MC*88916?

**Контрольні питання підвищеної складності:**

1 Для чого використовуються сигнали *BACK*5…*BACK*1?

2 Для чого на схему *FPGA* надходять сигнали адреси *A*17…*A*13?

3 В чому полягає призначення сигналу *BERR*, який формується схемою *FPGA*?

4 Для формування яких сигналів використовуються сигнали *UDS, LDS*?

5 Чи використовуються сигнали *FC*2...*FC*0 для реалізації переривань?

### *11.2.2 Розподіл адресного простору МПС*

**Вхідний контроль:**

1 З якою метою адресний простір розподіляється поміж різних типів підсистем МПС?

2 Які вимоги визначають розподіл адресного простору поміж підсистемами МПС?

3 Які підсистеми МПС входять до адресного простору?

4 Чим визначається максимальний обсяг адресного простору МПС?

Кожна з підсистем МПС має власну область адрес у адресному просторі МПС. Розподіл адресного простору визначається мапою адрес, котра задає межі адресного простору для кожної з підсистем. Розподіл адресного простору МПС на базі *MC*68000 здійснено відповідно до мапи адрес материнської плати інтегрованої платформи *M*68*EC*0*X*0*IDP*. Мапу адрес для МПС наведено на рис. 11.10. Адресний простір, відповідно до мапи, розподіляється поміж модулями: постійного запам'ятовувального пристрою (ПЗП) – *ROM*, оперативного запам'ятовувального пристрою (ОЗП) – *RAM*, послідовного порту (інтерфейсу) – *DUART*, паралельного порту (інтерфейсу) та таймера – *PI/T*.

Розподіл адресного простору зреалізовано на ВІС *FPGA*, яка була розглянута вище, і сигнали *BACK*1 (2, 3, 4, 5) формуються відповідно до мапи. Звернення до ПЗП при вмиканні і перезавантаженні системи виконується апаратно – сигнал *BACK*1 у ці моменти формується примусово.

Рисунок 11.10 – Мапа розподілу адресного простору МПС *MC*68000

Системний ПЗП призначено для зберігання системних програм: самотестування, початкового завантаження системи тощо.

Область системного таймера призначено задля обслуговування вбудованого таймера, який визначає часові інтервали при роботі системи. Він є енергонезалежним і продовжує функціонувати за вимикання живлення.

**Контрольні питання:**

1 Визначте інформаційну ємність кожної з областей мапи розподілу адресного простору МПС МП *MC*68000?

2 Які підсистеми МПС позначено в адресному просторі?

3 Яка з підсистем МПС займає найбільший адресний простір і чому?

**Контрольні питання підвищеної складності:**

1 Призначення програмованої логічної інтегральної схеми *FPGA* у складі підсистеми центрального процесорного елемента *MC*68000.

2 Для керування якими пристроями використовуються сигнали *BACK*1 (2, 3, 4, 5), які формуються на виходах програмованої логічної інтегральної схеми *FPGA*?

3 За допомого яких сигналів здійснюється визначання довжини даних, які оброблюються?

### *11.2.3 Організація підсистеми пам'яті*

**Вхідний контроль:**

1 Якими параметрами схарактеризовуються пристрої пам'яті?

2 Визначте кількість мікросхем *RAM*, необхідних задля побудування блока пам'яті з організацією 64*K*×8, з мікросхем, які мають організацію 32*K*×4?

3 Скільки входів адреси повинна мати мікросхема *ROM* з організацією 32*K*x8?

4 Які сигнали необхідні задля запису інформації до мікросхем пам'яті *RAM*?

5 Які сигнали необхідні задля зчитування інформації з мікросхеми пам'яті *RAM*?

6 Яких станів можуть набувати вихідні сигнали тристабільних мікросхем пам'яті?

7 У який спосіб взаємодіють сигнали $\overline{OE}, \overline{CS}, \overline{CE}$ поміж собою задля виконування запису до мікросхеми пам'яті *RAM*?

Побудова підсистеми пам'яті здійснюється відповідно до положень розділу 5, котрі доповнюються тим, що ПЗП і ОЗП МПС *MC*68000 повинні працювати з даними, які можуть бути байтами, словами та довгими словами. Відповідно до цього, водночас можливе звернення до однієї, двох чи чотирьох комірок пам'яті. Шина даних в МПС *MC*68000 16-розрядна, і робота з довгими словами виконується за два цикли шини, тому при роботі з байтами і словами відбувається звернення до комірок пам'яті з однією адресою, а молодше і старше слова довгих слів розміщуються у двох сусідніх парах комірок. Задля реалізації такого принципу побудови пам'яті необхідно будувати пам'ять з чотирьох блоків, кожен з яких призначено для роботи з байтами даних, поєднуючи їх відповідно до довжини операндів. Організацію такої пам'яті подано на рис. 11.11. Блоки ПЗП та ОЗП будуються в однаковий спосіб.

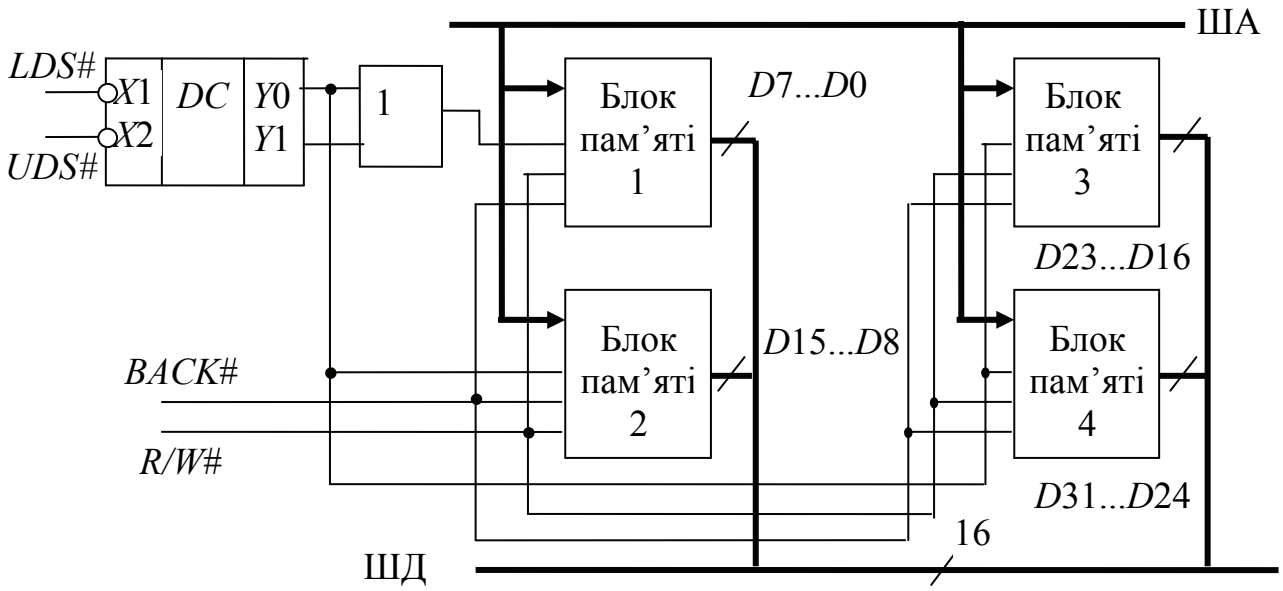



Рисунок 11.11 – Організація чотириблокової пам'яті

В чотириблоковій пам'яті всі блоки пам'яті до шини адреси під'єднуються паралельно (до одних і тих самих розрядів), що забезпечує звернення до комірок з однаковими номерами. Сигнали вибору блока (*BACK*) і читання/запис (*R/W*) також надходять одночасно. Вибір відповідного блока здійснюється за допомогою дешифратора і схеми АБО. Якщо на входи дешифратора надходить код 00 (робота зі словами), то активний сигнал формується лише на виході *Y*0 і надходить на відповідні входи всіх блоків (на вхід блока 1 він проходить через логічну схему АБО). Отже, водночас всі 16 виводів двох блоків пам'яті будуть з'єднані з шиною даних. Якщо на входи дешифратора надходить код 01 (робота з байтом), то сигнал *Y*1 дозволить роботу лише блокові 1; інші блоки в цей момент будуть перебувати в режимі зберігання інформації (будуть неактивними). Задля нарощування інформаційної ємності блока пам'яті збільшується кількість однотипних мікросхем пам'яті у кожному з блоків, відповідно до положень розділу 5.5. Сукупність мікросхем пам'яті в усіх чотирьох блоках, які обслуговують однакові адреси, можна назвати шаром пам'яті. Отже, якщо кожен з блоків пам'яті складається з кількох мікросхем пам'яті, то можна говорити про використовування багатошарової пам'яті.

Побудова кожного з блоків пам'яті здійснюється відповідно до положень розділу 5.5. Припустімо, що треба побудувати ОЗП з організацією 115*K*×8 з мікросхем *AM*21*C*512, які було розглянуто в розділі 5.5. ОЗП має працювати з байтами, словами та подвійними словами. Кількість мікросхем, потрібних для побудови, визначатиметься за виразом

$$N = 4\frac{115K \times 8}{64K \times 8} = 7{,}1872 \approx 8,$$

де 4 – кількість блоків пам’яті; 115*К*×8 – організація ОЗП кожного з блоків; 64*К*×8 – організація мікросхеми *AM*21*C*512.

Якщо у результаті здобуто дробове число, то його слід заокруглювати **обов’язково** до більшого цілого числа.

Отже, блок ОЗП вміщуватиме чотири блоки пам’яті, кожен з яких складатиметься з двох мікросхем. Схему цієї пам’яті наведено на рис. 11.12.

Керування схемою здійснюється сигналами, які формуються ВІС *FPGA*. Сумарний обсяг пам’яті кожного блока 128*К*. Блок 1 може працювати з байтами, сумісно з блоком 2 – зі словами і всі чотири блоки – з довгими словами. Молодша частина довгого слова оброблюється блоками 1 та 2, старша – блоками 3 та 4. Керування шарами пам’яті в кожному з блоків здійснюється за сигналом адреси *A*16 у такий спосіб, що молодші комірки пам’яті з адресами \$00000...\$0*FFFF* оброблюються верхньою (за схемою) мікросхемою, а старші з адресами \$10000...\$1*FFFF* – нижньою. Задля керування використовується вхід $\overline{OE}$, на котрий подається або сам сигнал *A*16, або його інверсія. Отже, якщо на вхід верхньої мікросхеми надходить сигнал безпосередньо з шини адреси, то в діапазоні адрес \$10000...\$1*FFFF* роботу цього блока буде заборонено.

**Контрольні питання:**

1 Чому при побудові МПС на МП *MC*68000 рекомендовано використовувати пам’ять, яка складається з чотирьох блоків?

2 У який спосіб зорганізовується робота блоків при роботі з даними різної розрядності?

3 Який пристрій керує роботою блоків пам’яті МПС на МП *MC*68000?

4 Для чого використовується багатошарова пам’ять?

5 Чи є обов’язковим використовування багатошарової пам’яті?

6 За допомогою яких сигналів та пристроїв здійснюється вибір шарів блоків пам’яті?

**Контрольні питання підвищеної складності:**

1 Розробити схему ПЗП з організацією 128×8 задля зберігання даних у вигляді байта, слова і довгого слова, використовуючи мікросхеми пам’яті, які наведено в розділі 5.5.

2 Які сигнали керують вибором шару для роботи блоків пам’яті і з якого пристрою вони надходять?

3 У який спосіб взаємодіють поміж собою блоки пам’яті при роботі з операндами різної розрядності?

4 За скільки циклів 32-розрядні дані може бути записано до пам’яті *RAM*?

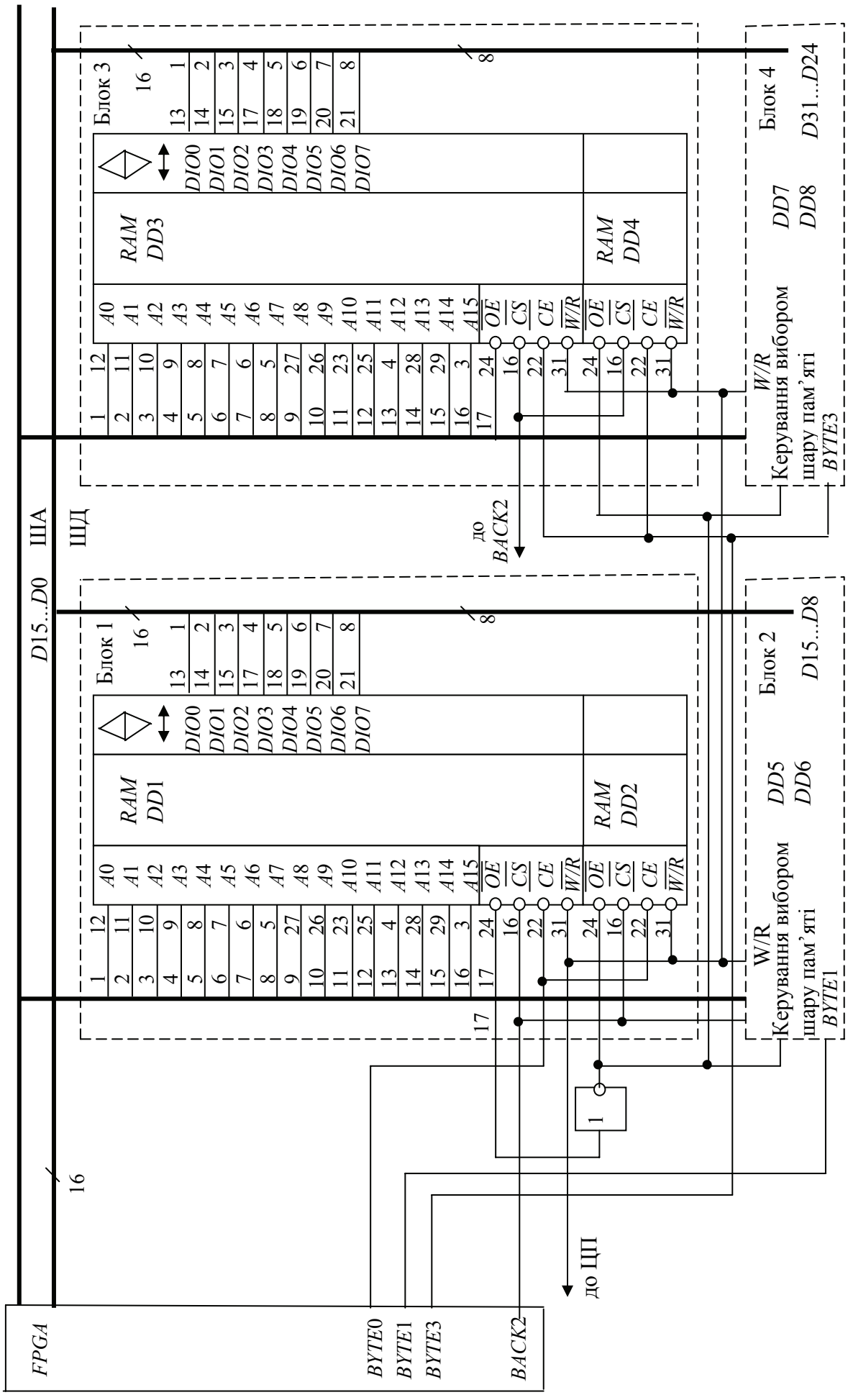



Рисунок 11.12 – Принципова схема блока пам'яті

### *11.2.4 Організація підсистеми введення-виведення*

**Вхідний контроль:**

1 Чим відрізнюються паралельний і послідовний способи обміну даними у МПС?

2 Даними якої розмірності може обмінюватися МПС через послідовний порт *RS*-232-*C*?

3 Які службові символи можуть входити до складу інформації, яка передається через послідовний порт *RS*-232-*C*?

4 В чому полягає різниця поміж синхронним і асинхронним способами обміну інформацією?

5 В який спосіб слід зорганізовувати послідовний обмін інформацією поміж МПС, якщо в кожній з них інформацію подано у паралельному коді?

6 У яких випадках застосовується паралельний обмін даними поміж МПС та периферійними пристроями й чому?

7 У яких випадках дані поміж МПС та периферійними пристроями пересилаються у послідовному коді й чому?

8 Які послідовні адаптери та приймачі-передавачі Вам є відомі?

9 З яких бітів складається формат даних у послідовному адаптері *RS*-232-*C*?

*Асинхронний послідовний інтерфейс DUART – MC68681 фірми Motorola*

ВІС фірми *Motorola MC*68681 (*DUART – DUAL ASYNCHRONOUS RESEIVER/TRANSMITTER*) – подвійний асинхронний приймач-передавач, призначений для організації двох незалежних послідовних асинхронних дуплексних каналів обміну (*A* та *B*) із зовнішніми пристроями; також за його допомогою існує можливість організації обміну службовою інформацією через 6-розрядний паралельний порт приймання і 8-розрядний паралельний порт введення-виведення інформації.

*DUART* забезпечує такі можливості як:

– організація двох незалежних асинхронних дуплексних послідовних каналів введення-виведення;

– робота кожного з каналів на 18-ти фіксованих швидкостях обміну, які можуть програмуватися незалежно для кожного з каналів;

– робота з даними, довжина котрих може становити від 5 до 8 бітів даних;

– у повідомлення встановлюються стартовий і один-два стопових біти, додатково може використовуватися біт паритету;

– можуть вибиратися такі режими роботи каналів:

нормальний (повний дуплекс);
автоматичного ехо-сигналу;
місцева кільцева перевірка;
віддалена кільцева перевірка;

– робота в режимі селекторного виклику;

– багатофункціональний 16-розрядний лічильник/таймер може працювати у двох режимах:

формування часових інтервалів;

генерування імпульсів;

– 6-розрядний паралельний порт введення може слугувати за багатофункціональний пристрій введення інформації або використовувати чотири входи для вимірювання часових характеристик сигналів;

– багатофункціональний 8-розрядний паралельний порт введення-виведення дозволяє індивідуально виконувати встановлення інформації на кожному з розрядів і задавати режим роботи;

– порт забезпечує багатовекторну систему переривань (до восьми пристроїв);

– визначення помилок роботи (зупин лінії, фальшстарт, надходження переривання тощо);

– робота від внутрішнього генератора з кварцовим стабілізуванням частоти або від зовнішнього генератора імпульсів;

– ВІС є сумісна з ТТЛ-логікою;

– живлення від джерела +5 В.

Структурну схему *DUART* наведено на рис. 11.13. До структурної схеми входять:

– системний інтерфейс – схема, яка забезпечує керування роботою вузлів *DUART*, а також керування обміном інформацією з ЦП. За допомогою цієї схеми проводиться обмін сигналами поміж ЦП і вузлами *DUART*. На схему надходять сигнали:

*R/W#* – читання /запис;

*CS#* – вибір мікросхеми, при надходженні цього сигналу ВІС переводиться до активного стану;

*RESET#* – сигнал скидання;

*RS*4-*RS*1 (*Register Select*) – відповідні розряди шини адреси, які визначають адресу внутрішнього вузла регістра *DUART*;

*DTACK#* – сигнал підтвердження готовності ВІС до обміну, підтверджує наявність інформації на шині даних;

– буфер шини даних – схема, котра забезпечує правильність обміну поміж шиною даних МПС і внутрішньою шиною *DUART*;

– схема керування перериваннями – забезпечує роботу за перериваннями. На її входи надходить сигнал *IACK#* – сигнал підтвердження переривання – і формується сигнал *IRQ#* – номер запиту на переривання. До схеми входять такі вузли:

*IMR (Interrupt Mask Register*) – регістр маски переривання;

*ISR (Interrupt Status Register*) – регістр стану переривання;

*ACR (Auxiliary Control Register*) – допоміжний регістр керування;

*IVR (Interrupt Vector Register*) – регістр вектора переривань;

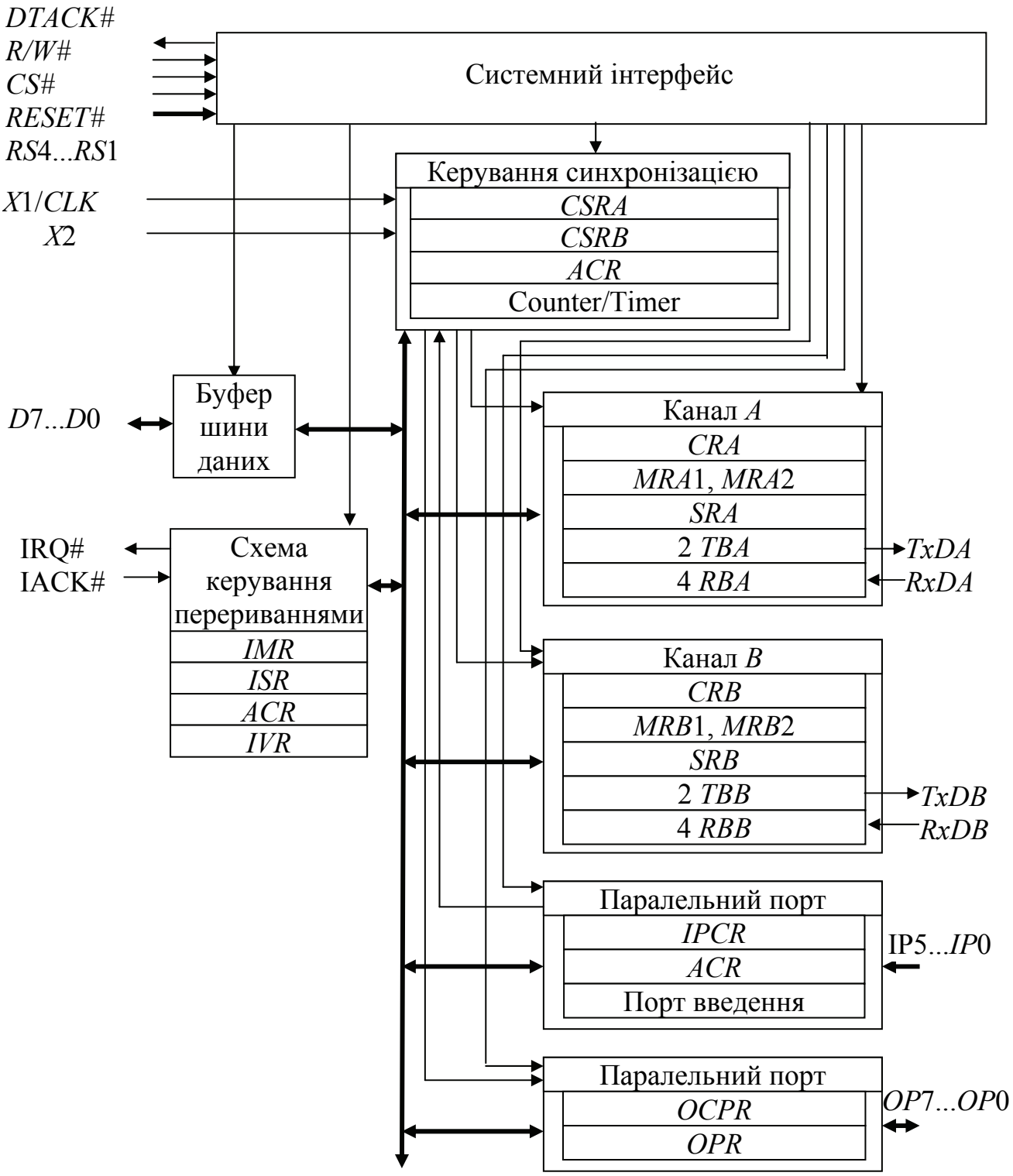



Рисунок 11.13 – Структурна схема *DUART*

– схема керування синхронізацією. Схема визначає швидкість обміну інформацією у послідовних каналах. До її складу входять:

*CSRA* (*Clock Select Register channel A*) – регістр визначення швидкості обміну інформацією в каналі А;

*CSRB* (*Clock Select Register channel B*) – регістр визначення швидкості обміну інформацією в каналі *B*;

*ACR* (*Auxiliary Control Register*) – допоміжний регістр контролю;

*Counter/Timer* – 16-розрядний лічильник/таймер, складається з двох 8-розрядних регістрів, кожен з яких програмується окремо;

– два аналогічних асинхронних послідовних канали – *A* і *B*, які побудовано з однакових вузлів:

*CRA (B) (Command Register channel A (B))* – командний регістр, який керує виконанням команд;

*MRA*1, *MRA*2 (*Mode Register* 1, 2) – регістри, вміст котрих визначає режим роботи каналів введення-виведення даних;

*SRA (B) (Status Register Channel A (B))* – регістри стану, зберігають інформацію про результати обміну;

*TBA (B) (Transmit Buffer channel A (B))* – буферні регістри передавача відповідного каналу, до їхнього складу входять два регістри: *THRA (B)* – регістр тимчасового зберігання передавача і *TSRA (B)* – регістр зсуву передавача;

*RBA (B) (Receive Buffer channel A (B))* – буферні регістри приймача відповідного каналу. Кожний складається з чотирьох регістрів: трьох регістрів *RHRA (B)* – регістр тимчасового зберігання приймача, і *RSRA (B)* – регістр зсуву приймача;

– 6-розрядний паралельний порт введення. За необхідністю виводи каналу може бути запрограмовано на виконування певних допоміжних функцій обслуговування каналів зв'язку. До його складу входять:

*IPCR (Input Port Change Register)* – регістр стану порту введення. За відсутності сигналів перебуває у скинутому стані;

*ACR (Auxiliary Control Register)* – допоміжний регістр контролю;

порт (6-розрядний регістр) введення;

– 8-розрядний паралельний порт введення-виведення. Багатоцільовий універсальний порт. Усі розряди порту може бути індивідуально скинуто або встановлено. До його складу входять:

*OPCR (Output Port Configuration Register)* – регістр стану. Програмується задля визначення конфігурації паралельного порту (*OP*7...*OP*0) для поточного режиму роботи.

*OPR (Output Port Register)* – 8-розрядний паралельний порт.

Розподіл адресного простору *DUART* відбувається відповідно до табл. 11.6.

Кожен з двох каналів забезпечує роботу на 18-ти фіксованих швидкостях від внутрішнього генератора, частота котрого визначається кварцовим стабілізатором, який має частоту 3,6864 МГц, або від зовнішнього генератора, сигнал з котрого надходить на вивід *X*1/*CLK*. Взаємозв'язок швидкості й частот зовнішнього генератора наведено в табл. 11.7.

Таблиця 11.6 – Розподіл адресного простору *DUART*

| *RS*4 | *RS*3 | *RS*2 | *RS*1 | Читання (*R/W* = 1) | Запис (*R/W* = 0) | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | *Mode Register A* (*MRA*1, 2) | *Mode Register A* (*MRA*1, 2) | |
| 0 | 0 | 0 | 1 | *Status Register A (SRA*) | *Clock Select Register (CSRA)* | |
| 0 | 0 | 1 | 0 | Не використовується | *Command Register A (CRA)* | |
| 0 | 0 | 1 | 1 | *Receive Buffer (RBA)* | *Transmit Buffer A (TBA)* | |
| 0 | 1 | 0 | 0 | *Input Port Change Register (IPCR)* | *Auxiliary Control Register (ACR)* | |
| 0 | 1 | 0 | 1 | *Interrupt Status Register (ISR)* | *Interrupt Mask Register (IMR)* | |
| 0 | 1 | 1 | 0 | Старший байт лічильника/таймера | Старший байт лічильника/таймера | |
| 0 | 1 | 1 | 1 | Молодший байт лічильника/таймера | Молодший байт лічильника/таймера | |
| 1 | 0 | 0 | 0 | *Mode Register B* (*MRB*1,2) | *Mode Register B* (*MRB*1,2) | |
| 1 | 0 | 0 | 1 | *Status Register B (SRB)* | *Clock Select Register (CSRB)* | |
| 1 | 0 | 1 | 0 | Не використовується | *(Command Register B (CRB)* | |
| 1 | 0 | 1 | 1 | *Receive Buffer B (RBB)* | *Transmit Buffer B (TBA)* | |
| 1 | 1 | 0 | 0 | *Interrupt Vector Register (IVR)* | *Interrupt Vector Register (IVR)* | |
| 1 | 1 | 0 | 1 | Порт введення | *Output Port Configuration Register (OPCR)* | |
| 1 | 1 | 1 | 0 | Старт лічби | *Output Port Register* | Встановлення біта |
| 1 | 1 | 1 | 1 | Стоп лічби | | Скидання біта |

Таблиця 11.7 – Відповідність швидкості й частоти зовнішнього генератора

| Швидкість, Бод | Частота зовнішнього генератора, кГц | Швидкість, Бод | Частота зовнішнього генератора, кГц |
|---|---|---|---|
| 50 | 0,8 | 1200 | 19,2 |
| 75 | 1,2 | 1800 | 28,8 |
| 100 | 1,759 | 2000 | 32,056 |
| 134,5 | 2,153 | 2400 | 38,4 |
| 150 | 2,4 | 4800 | 76,8 |
| 200 | 3,2 | 7200 | 115,2 |
| 300 | 4,8 | 9600 | 153,6 |
| 600 | 9,6 | 19,2 кБод | 307,2 |
| 1050 | 16,756 | 38,4 кБод | 614,4 |

Програмування ВІС *DUART* здійснюється за допомогою запису сигналів керування (байтів) до відповідних регістрів.

Керування роботою каналів *A* і *B* здійснюється за допомогою вмісту регістрів *MRA*1, *MRA*2 (*MRB*1, *MRB*2). Призначення бітів регістра *MRA*1 (*MRB*1) подано в табл. 11.8.

Програмування регістрів *MRA*2 та *MRB*2 задля визначання режиму роботи каналу здійснюється відповідно до табл. 11.9.

Вибір необхідної швидкості роботи передавача та приймача здійснюється завантаженням до регістрів *CSRA* та *CSRB* байта, вміст котрого формується відповідно до табл. 11.10. За коду 1111 для синхронізації використовуються виводи *IP*3 або *IP*4 паралельного порту введення.

Вміст регістрів команд *CRA* (*B*) керує виконуванням програми функціонування ВІС *DUART*. Його програмування наведено у табл. 11.11.

Передавачі каналів *A* і *B* програмуються відповідно до наведених таблиць перед початком передавання інформації. Про готовність до передавання повідомляється встановленням біта 2 у регістрі статусу (*SRA (B)*). Цей біт також може використовуватися задля генерації запиту на переривання для каналу *A* на виводі *OP*6 паралельного порту або *OP*7 – для каналу *B*. Передавання дозволяється встановленням біта 2 і скиданням біта 3 у командному регістрі *CRA (B)*.

Передавання інформації розпочинається з її завантаження до буферного регістра передавача *TBA (B),* звідки вона передається до регістра зсуву *TSRA* (*B*). В цьому регістрі здійснюється її перетворювання на послідовний код, формування стартового біта і передавання інформації до каналу. Завершується передавання формуванням необов'язкового біта парності та обов'язкових одного чи двох стопових бітів. Передавання розпочинається за зрізом сигналу у каналі *TxD*, що встановлює значення сигналу *TxRDY* у 1. Скидання цього сигналу буде відбуватися після завантаження до буфера передавання нової інформації, що призводить до формування імпульсу сигналу $\overline{DTAC}$, котрий дозволяє передавання інформації. Після проходження стопового біта лінія залишається у стані логічної 1, якщо немає наступного байта. Скидання бітів 2 і 3 командного регістра *CRA (B)* є забороною передавання; при цьому сигнали *TxRDY* та $\overline{DTAC}$ не формуються і інформація *D*4 не передається.

Передавання інформації подано за допомогою часових діаграм, які наведено на рис. 11.14.

Про готовність до приймання інформації повідомляється встановлЯнням біта 0 в регістрі статусу (*SRA (B)).* Приймачі відповідних каналів розпочинають працювати при надходженні на вхід зрізу вхідного сигналу і визначають тривалість одного біта. Тривалість біта визначається кількістю імпульсів тактової частоти від зрізу до першого фронту вхідного сигналу. Зазвичай тривалість біта становить 7,5 періодів тактової частоти; якщо таке не виконується, подальша інформація ігнорується і приймач продовжує

Таблиця 11.8 – Програмування режиму роботи каналів регістрів *MRA*1 (*MRB*1)

| Контроль *RxRTS* | Вибір *RxIRQ#* | Помилка | Перевірка парності | | Тип перевірки | Довжина повідомлення |
|---|---|---|---|---|---|---|
| біт 7 | біт 6 | біт 5 | біт 4 | біт 3 | біт 2 | біт 1 біт 0 |
| 0 – не дозволено<br>1 – дозволено | 0 – *RxRDY*<br>1 – повний канал *FIFO* | 0 – обнулення<br>1 – блокування | 0 0 | Перевірка парності | З перевіркою<br>0 – дорівнює<br>1 – не дорівнює<br>–<br>Адреси/дані<br>0 – дані<br>1– адреси | 0 0 – 5 біт<br>0 1 – 6 біт<br>1 0 – 7 біт<br>1 1 – 8 біт |
| | | | 0 1 | – | | |
| | | | 1 0 | Немає перевірки | | |
| | | | 1 1 | Біти використовуються як адреси/дані | | |

Таблиця 11.9 – Програмування режиму роботи каналів регістрів *MRA*2 (*MRB*2)

| Режим роботи каналу | | Контроль *TxRTS* (вивід *OP*0) | *CTS* дозвіл передавання | Кількість стопових бітів | |
|---|---|---|---|---|---|
| біт 7 | біт 6 | біт 5 | біт 4 | біт 3 біт 2 біт 1 біт 0 | |
| | | 0 – не дозволено<br>1 – дозволено | 0 – не дозволено<br>1 – дозволено | 6-8 бітів даних | 5 бітів даних |
| 0 0 | Нормальний | | | 0000 – 0,563 | 1,063 |
| 0 1 | Автоматичного exo-сигналу | | | 0001 – 0,625 | 1,125 |
| 1 0 | Місцева кільцева перевірка | | | 0010 – 0,688 | 1,188 |
| 1 1 | Віддалена кільцева перевірка | | | 0011 – 0,750 | 1,250 |
| | | | | 0100 – 0,813 | 1,313 |
| | | | | 0101 – 0,875 | 1,375 |
| | | | | 0110 – 0,938 | 1,438 |
| | | | | 0111 – 1,000 | 1,500 |
| | | | | 1000 – 1,563 | 1,563 |
| | | | | 1001 – 1,625 | 1,625 |
| | | | | 1010 – 1,688 | 1,688 |
| | | | | 1011 – 1,750 | 1,750 |
| | | | | 1100 – 1,813 | 1,813 |
| | | | | 1101 – 1,875 | 1, 875 |
| | | | | 1110 – 1,938 | 1,938 |
| | | | | 1111 – 2,000 | 2,000 |

Таблиця 11.10 – Програмування швидкості обміну *DUART*

| Вибір швидкості приймача | | | | | | Вибір швидкості передавача | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Біт 7 | Біт 6 | Біт 5 | Біт 4 | Швидкість, Бод залежно від *ACR* | | Біт 3 | Біт 2 | Біт 1 | Біт 0 | Швидкість, Бод залежно від *ACR* | |
| | | | | Біт 7 = 0 | Біт 7 = 1 | | | | | Біт 7 = 0 | Біт 7 = 1 |
| 0 | 0 | 0 | 0 | 50 | 75 | 0 | 0 | 0 | 0 | 50 | 75 |
| 0 | 0 | 0 | 1 | 110 | 110 | 0 | 0 | 0 | 1 | 110 | 110 |
| 0 | 0 | 1 | 0 | 134,5 | 134,5 | 0 | 0 | 1 | 0 | 134,5 | 134,5 |
| 0 | 0 | 1 | 1 | 150 | 150 | 0 | 0 | 1 | 1 | 150 | 150 |
| 0 | 1 | 0 | 0 | 200 | 200 | 0 | 1 | 0 | 0 | 200 | 200 |
| 0 | 1 | 0 | 1 | 300 | 300 | 0 | 1 | 0 | 1 | 300 | 300 |
| 0 | 1 | 1 | 0 | 600 | 600 | 0 | 1 | 1 | 0 | 600 | 600 |
| 0 | 1 | 1 | 1 | 1050 | 1050 | 0 | 1 | 1 | 1 | 1050 | 1050 |
| 1 | 0 | 0 | 0 | 2400 | 2400 | 1 | 0 | 0 | 0 | 2400 | 2400 |
| 1 | 0 | 0 | 1 | 4800 | 4800 | 1 | 0 | 0 | 1 | 4800 | 4800 |
| 1 | 0 | 1 | 0 | 7200 | 1800 | 1 | 0 | 1 | 0 | 7200 | 1800 |
| 1 | 0 | 1 | 1 | 19,2 кБод | 9600 | 1 | 0 | 1 | 1 | 19,2 кБод | 9600 |
| 1 | 1 | 0 | 0 | 38,4 кБод | 19,2 кБод | 1 | 1 | 0 | 0 | 38,4 кБод | 19,2 кБод |
| 1 | 1 | 0 | 1 | Таймер | Таймер | 1 | 1 | 0 | 1 | Таймер | Таймер |
| 1 | 1 | 1 | 0 | Зовн. сигн. | Зовн. сигн. | 1 | 1 | 1 | 0 | Зовн. сигн. | Зовн. сигн. |
| 1 | 1 | 1 | 1 | Зовн. сигн. | Зовн. сигн. | 1 | 1 | 1 | 1 | Зовн. сигн. | Зовн. сигн. |

Таблиця 11.11 – Програмування командного регістра *CRA(B)*

| | Різні команди | Команди передавача | Команди приймача |
|---|---|---|---|
| Біт 7 | Біт 6 Біт 5 Біт 4 | Біт 3 Біт 2 | Біт 1 Біт 0 |
| | 000 – Команди немає<br>001 – Скидання режиму<br>010 – Скидання приймача<br>011 – Скидання передавача<br>100 – Скидання статусу помилок<br>101 – Скидання канальних переривань<br>110 – Зупин<br>111 – Скидання зупину | 00 – Не виконуються активні дії. Зберігається поточний стан режимів | 00 – Не виконуються активні дії. Зберігається поточний стан режимів |
| | | 01 – Передавання дозволено | 01 – Приймання дозволено |
| | | 10 – Передавання заборонено | 10 – Приймання заборонено |
| | | 11 – Не використовується | 11 – Не використовується |

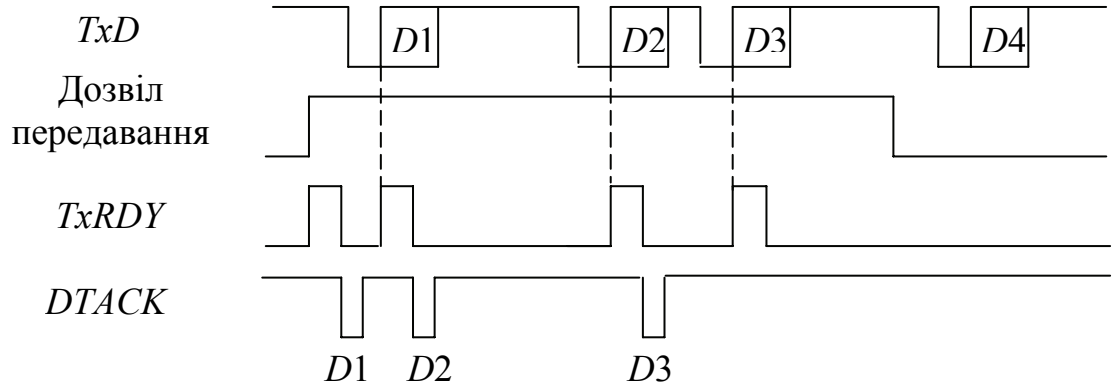



Рисунок 11.14 – Передавання інформації послідовним каналом

контролювати надходження імпульсів на його вхід. Якщо часові співвідношення виконуються, то приймач записує дані до буфера приймача *RHR* і скидає біт 0 у регістрі статусу (*SRA (B)*). Якщо віднайдено помилку кадрової синхронізації (відсутність стопового біта), то у лінії зберігається нульовий (*L*0) стан упродовж половини періоду тривалості біта, після чого робота триває так само, як і після отримання стартового біта.

Поява інших помилок (парності, зміна швидкості надходження інформації тощо) змінює значення відповідних прапорців (ознак) у регістрах статусу (*SRA (B)*).

Після завершення приймання байта на лінії повинен встановлюватися високій рівень сигналу (*L*1) тривалістю не менш за половину періоду тривалості біта, після чого порт продовжує пошук наступного стартового біта.

Функціонування *DUART* у режимах контролю забезпечується встановленням бітів 7, 6 у регістрі режиму роботи каналів *MRA*2 (*MRB*2) відповідно до табл. 11.8.

В режимі автоматичного ехо-сигналу (код бітів 7, 6, ..., 1) в каналі автоматично ретранслюються дані. Місцевий зв’язок поміж центральним процесором і приймачем продовжує нормально функціонувати, а зв’язок поміж центральним процесором і передавачем заблоковано. В такий спосіб можна контролювати стан лінії і роботу робочої станції.

В режимі місцевої кільцевої перевірки (код – 10) вихід передавача у ВІС з’єднується з входом приймача. Передавач і приймач в цьому режимі працюють відповідно до встановлених режимів у штатному розписі. Передаючи дані і контролюючи їхнє приймання, можна перевірити правильність роботи місцевого каналу приймання-передавання *DUART*.

В режимі віддаленої кільцевої перевірки каналом автоматично ретранслюються дані і канал передавача робочої станції сполучено з іншими станціями мережі (дозволено до 256 інших станцій). При цьому місцевий зв’язок поміж центральним процесором і приймачем заблоковано. До складу даних, що передаються, входять стартовий біт, блок даних, біт адреси/даних

(*A/D*) і стопові біти. Цей блок призначено одній зі станцій. Станції мережі невпинно здійснюють моніторинг каналу й ідентифікують біт адреси/даних задля визначення виду інформації. Якщо біт *A/D* встановлено, то передається адреса, якщо скинуто – дані. Станція визначає власну адресу, сповіщає про це головну станцію, яка вмикає власний приймач, і генерує переривання, після чого отримується повідомлення, і знову заблоковує власний канал.

Багатофункціональний 16-розрядний лічильник/таймер може працювати в режимі формування часових інтервалів і в режимі генерування імпульсів. В кожному з режимів лічильник/таймер можна запрограмувати на отримання сигналу тактової частоти від кількох джерел і виведення результату через вивід *OP*3 паралельного порту. До лічильника можна завантажити число від $0002 до $*FFFF,* котре можна змінити будь-якого моменту. Програмування здійснюється завантаженням до допоміжного регістра керування (*ACR*) байта, що відповідає потрібному режимові роботи. Формат цього регістра наведено у табл. 11.12. Біт 7 цього регістра визначає швидкість обміну інформацією відповідно до табл. 11.6. В режимі формування часових інтервалів лічильник/таймер запускається і зупиняється ЦП, тому цей режим можна використовувати як системний годинник реального часу задля генерування переривань і як вартовий пристрій. В режимі генерування імпульсів лічильник/таймер працює невпинно незалежно від ЦП, тому такий режим можна використовувати задля програмної реалізації сигналів тактової частоти для каналів *A* і *B*, періодичного звернення до певних пристроїв і в якості автогенератора періодичних імпульсних сигналів.

В режимі формування часових інтервалів лічильник/таймер працює як лічильник, який декрементує попередньо завантажений вміст лічильника. За сигнал тактової частоти може слугувати сигнал від зовнішнього генератора, який надходить на вхід *X*1/*CLK*, поділений на 16, або сигнал зі входу *IP*2. Після програмування і запуску лічильник/таймер розпочинає працювати від значень попереднього завантаження до досягнення значення $0000, після чого встановлює прапорець у біті 3 регістра *ISR* і продовжує лічбу від значення $*FFFF*. Задля перевантаження лічильник/таймер потрібно заново зініціалізувати.

В режимі генерування імпульсів таймер формує періодичну послідовність прямокутних імпульсів, частота слідування котрих визначається значеннями попередніх завантажень і сигналом, який надходить від джерел: внутрішній кварцовий генератор; зовнішній сигнал на виводі *X*1/*CLK* або сигнал на цьому виводі, частота котрого поділена на 16; зовнішній сигнал, що надходить на вивід *IP*З2. Таймер виконує лічбу до досягнення значення $0000, після чого перевантажує значення попередніх завантажень і продовжує роботу.

Таблиця 11.12 – Програмування режиму роботи таймера

| Вибір швид-кості | Програмування режиму і джерела сигналу | | Виводи запитів переривання | | | |
|---|---|---|---|---|---|---|
| | | | *IP*3 *IRQ* | *IP*2 *IRQ* | *IP*1 *IRQ* | *IP*0 *IRQ* |
| Біт 7 | Біт 6 \| Біт 5 \| Біт 4 | | Біт 3 | Біт 2 | Біт 1 | Біт 0 |
| | Режим | Джерело часових сигналів | 0 – дозвіл, 1 – заборона | 0 – дозвіл, 1 – заборона | 0 – дозвіл, 1 – заборона | 0 – дозвіл, 1 – заборона |
| 0 або1 (табл. 11.5 | 000 – лічильник | зовнішній IP2 | | | | |
| | 001 – лічильник | *TxCA* – 1*X* визначення часових інтервалів каналу передавача *A* | | | | |
| | 010 – лічильник | *TxCB* – 1*X* визначення часових інтервалів каналу передавача *B* | | | | |
| | 011 – лічильник | кварцовий резонатор або зовнішній сигнал (*X*1/*CLK*), поділений на 16 | | | | |
| | 100 – таймер | зовнішній *IP*2 | | | | |
| | 101 – таймер | зовнішній *IP*2 | | | | |
| | 110 – таймер | кварцовий резонатор або зовнішній сигнал (*X*1/*CLK*) | | | | |
| | 111 – таймер | кварцовий резонатор або зовнішній сигнал (*X*1/*CLK*), поділений на 16 | | | | |

Нижче наведено фрагмент програми роботи послідовного порту *DUART* задля передавання байта зі швидкістю 300 Бод каналом А:

```
MOVEA.L #$B00000,A0     ; Завантаження базової адреси DUART у A0
MOVE.B #$07,(A0)        ; Завантаження до регістра режиму порту A
                        ; MR2A даного $07 (00000111) для
                        ; встановляння нормального режиму
                        ; передавання з одним стоповим бітом
MOVE.B #$44,($01,A0)    ; Завантаження до регістра визначання
                        ; швидкості обміну інформацією у каналі A
                        ; даного $44 (01000100) для  забезпечення
                        ; швидкості 300 Бод
MOVE.B #$34,($02,A0)    ; Завантаження до регістра команд каналу A
                        ; даного $34 (01010101), яке перевантажує
                        ; передавач
```

```
MOVE.B #$A5,($03,A0)      ; Завантаження до буферного регістра
                          ; передавача каналу А даного $A5 (10100101),
                          ; яке є інформацією, котру буде передано
                          ; каналом А
MOVE.B ($03,A0),($0E,A0)  ; Передавання інформації каналом А
```

Підмикання ВІС *DUART* до шин МПС здійснюється відповідно до функціонального призначення виводів мікросхеми та сигналів, що формуються іншими пристроями. Умовні графічні позначення та схему підключення асинхронного послідовного порту *DUART* подано на рис. 11.15. Збільшення кількості послідовних портів здійснюється при дешифруванні сигналів адреси. Кожна з ВІС *DUART* займає адресний простір 32 байти. Отже, для адресування наступних ВІС можна використовувати розряди *A*5 і наступні.

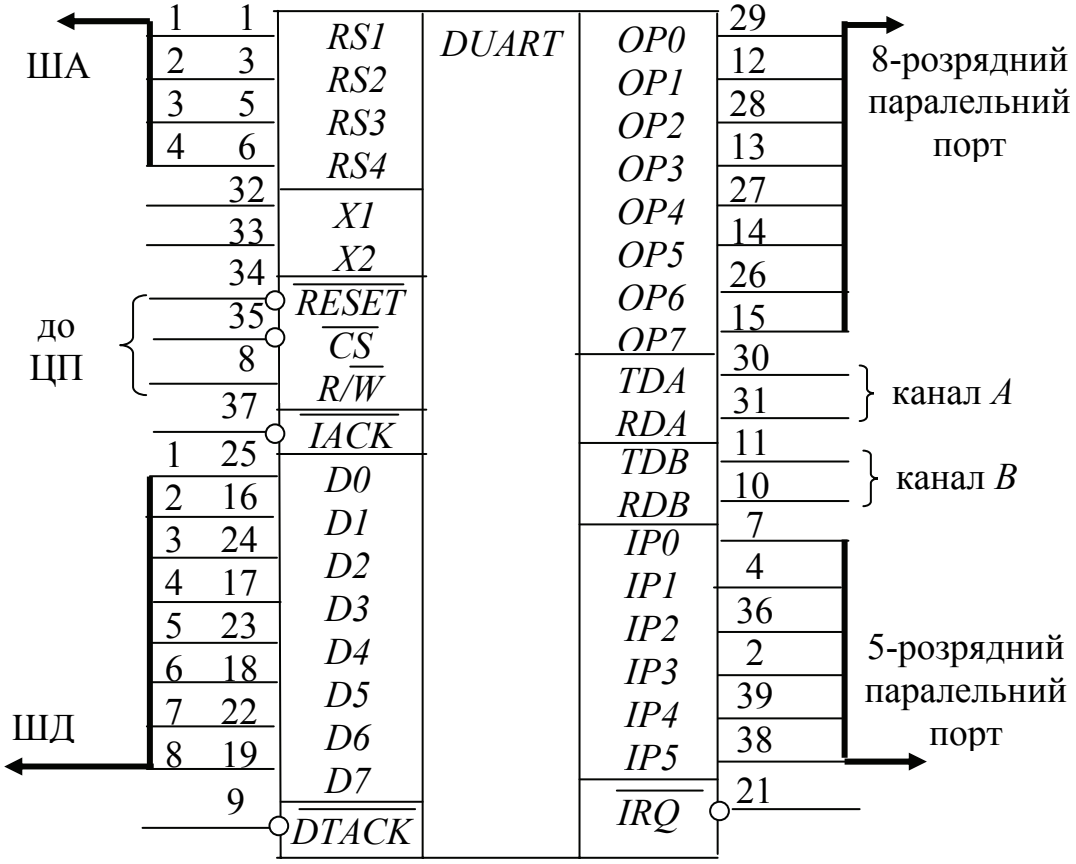



Рисунок 11.15 – Схема підключення ВІС *DUART*

**Контрольні питання:**

1 Які функціональні можливості має ВІС *DUART*?

2 Які пристрої входять до складу ВІС *DUART*?

3 Який обсяг пам’яті відведено для адресування блоків ВІС *DUART*?

4 У яких режимах може працювати блок приймача-передавача ВІС *DUART*?

5 Скільки дуплексних каналів можливо зорганізувати за допомогою однієї ВІС *DUART*?

6 Для чого можуть використовуватися виводи порту *IP*0...*IP*5?

**Контрольні питання підвищеної складності:**

1 Для чого призначено режим автоматичного ехо-сигналу?
2 У яких режимах може працювати 16-розрядний лічильник/таймер?
3 У який спосіб програмується швидкість обміну інформацією ВІС *DUART*?
4 Чи існує можливість перевірки працездатності тракту приймача-передавача без наявності абонента?
5 Скільки окремих швидкостей обміну можна зреалізовувати за допомогою ВІС *DUART*?

*Периферійний інтерфейс/таймер (PIT) MC68230 фірми Motorola*

ВІС *PI/T* є паралельний інтерфейс/таймер, призначений для обміну 8-розрядними даними поміж мікропроцесором та зовнішніми пристроями (датчиками, пристроями керування тощо). Структурну схему *PI/T* наведено на рис. 11.16.

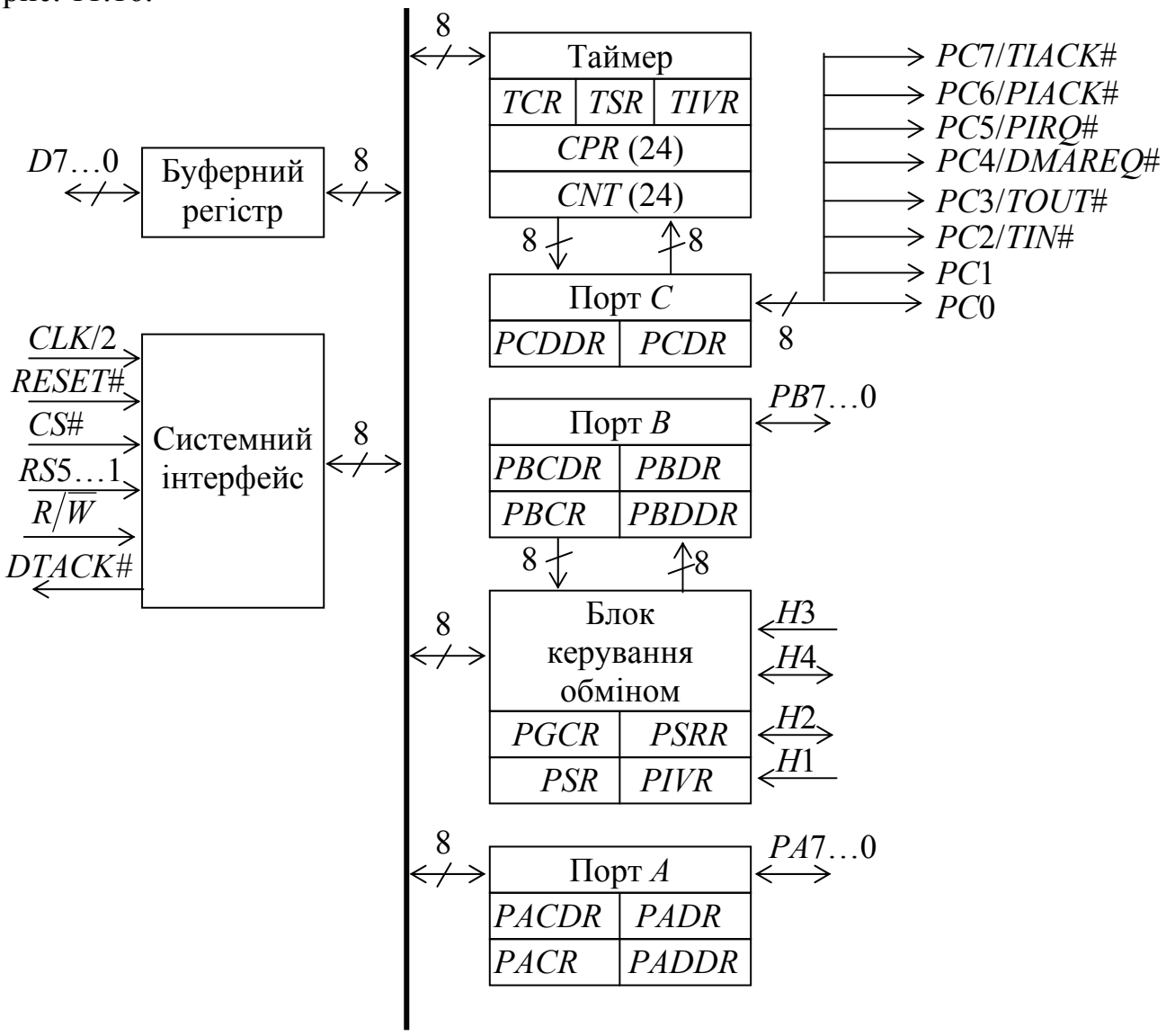



Рисунок 11.16 – Структурна схема *PI/T*

*PI/T* складається з блоків, які забезпечують зв'язок з мікропроцесором (буферний регістр, системний інтерфейс), і блоків, які обслуговують зовнішні пристрої (8-розрядні порти *A, B, C*, 24-розрядний таймер, блок керування обміном, який може використовуватись pfдля реалізації переривань, паралельного введення-виведення даних або формування сигналів квитування при пересиланні даних через порти *A, B*). Виводи *PC*7...*PC*2 можуть програмуватись для передавання сигналів таймера *TIN*#, *TOUT*#, переривання *PIRQ*#, *PIACK*#, *TIACK*, запиту прямого доступу *DMAREQ*#. Зв'язок *PI/T* з МП зреалізовується шляхом обміну даними по виводах *D*7…*D*0 у циклі читання або запису. МП видає на вхід системного інтерфейсу сигнал $R/\overline{W}$, а *PI/T* видає сигнал підтвердження готовності *DTACK*#.

Дані зчитуються або записуються до одного з регістрів таймера, порту *A, B, C* або блока керування обміном. Вибір регістра визначається кодом адреси, який надходить на входи адреси *RS*5…*RS*1. В адресному просторі *PI/T* займає 32-байтові комірки, з яких 23 зайнято регістрами, а решта не використовуються. Всі регістри, окрім лічильника *CNT* та регістра попереднього встановляння лічильника CPR, які мають 24 розряди, є 8-розрядні і адресуються як байт. *CNT* та *CPR* адресуються як три окремих 8-розрядні регістри. У табл. 11.13 показано адреси регістрів *PI/T* та зазначено зміщення відносно базової адреси $800001, які використовуються у команді *MOVEP* для формування ефективної адреси регістрів *PI/T*.

При зверненні до *PI/T* на входи *RS*5...*RS*1 надходить адреса регістра, що її формує МП, а на вхід вибирання *CS*# подається сигнал *CS*# від адресного дешифратора, який визначає саме цей *PI/T* серед усіх, які входять до МПС. На вхід *CLK* надходять синхросигнали *CLK*/2, а на вхід *RESET*# – загальний для всієї системи сигнал скидання.

*PI/T* є програмована ВІС, до керувальних регістрів якої задля реалізації різних режимів роботи треба записати керувальні коди, тобто зініціалізувати її.

Порти *A* та *B* забезпечують паралельний обмін даними поміж МП та зовнішніми пристроями. Задля виведення даних МП записує їх до регістра даних відповідного порту: *PADR* або *PBDR*, а введення здійснюється шляхом їхнього зчитування з регістрів даних. Кожний вивід портів *A* та *B* може програмуватись окремо на введення або виведення встановлянням у 1 (виведення) або у 0 (введення) бітів відповідного регістра – *PADDR* або *PBDDR*. Порти мають також додаткові регістри даних – *PACDR* та *PBCDR*, які дозволяють зберігати дані, якщо необхідно ввести від зовнішнього пристрою наступний біт до того, як попередній було зчитано мікропроцесором, або вивести наступний біт з мікропроцесора до того, як попередній біт було прийнято зовнішнім пристроєм. Отже, *PACDR* та *PBCDR* слугують за буферні регістри.

Блок керування обміном здійснює загальне керування портами *A* та *B*, а також керування функціями виводів *H*3...*H*1. При ініціалізації портів *A* та *B* до регістрів *PGCR*, *PSRR* блока керування, регістрів *PACR*, *PADDR* та *PBCR*, *PBDDR* портів МП записує керувальні коди, які визначають їхнє функціонування.

Таблиця 11.13 – Адреси регістрів *PI/T*

| Назва регістра | Призначення регістра | Адреса | Зміщення |
|---|---|---|---|
| *PGCR* | Головний регістр керування | $800001 | $00 |
| *PSRR* | Регістр обслуговування переривань | $800003 | $02 |
| *PADDR* | Регістр напрямку обміну даними порту *A* | $800005 | $04 |
| *PBDDR* | Регістр напрямку обміну даними порту *B* | $800007 | $06 |
| *PCDDR* | Регістр напрямку обміну даними порту *C* | $800009 | $08 |
| *PIVR* | Регістр вектора переривань | $80000*B* | $0*A* |
| *PACR* | Регістр керування порту *A* | $80000*D* | $0*C* |
| *PBCR* | Регістр керування порту *B* | $80000*F* | $0*E* |
| *PADR* | Регістр даних порту *A* | $8000011 | $10 |
| *PBDR* | Регістр даних порту *B* | $8000013 | $12 |
| *PAAR* | Додатковий регістр даних порту *A* | $8000015 | $14 |
| *PBAR* | Додатковий регістр даних порту *B* | $8000017 | $16 |
| *PCDR* | Регістр даних порту *C* | $8000019 | $18 |
| *PSR* | Регістр стану *PI/T* | $800001*B* | $1*A* |
| *TCR* | Регістр керування таймером | $8000021 | $20 |
| *TIVR* | Регістр вектора переривань таймера | $8000023 | $22 |
| *CPRH* | Регістр попереднього встановляння старшого байта лічильника | $8000027 | $26 |
| *CPRM* | Регістр попереднього встановляння середнього байта лічильника | $8000029 | $28 |
| *CPRL* | Регістр попереднього встановляння молодшого байта лічильника | $800002*B* | $2*A* |
| *CNTRH* | Регістр старшого байта лічильника | $800002*F* | $2*E* |
| *CNTRM* | Регістр середнього байта лічильника | $8000031 | $30 |
| *CNTRL* | Регістр молодшого байта лічильника | $8000033 | $32 |
| *TSR* | Регістр стану таймера | $8000035 | $34 |

Фрагмент програми демонструє керування виведенням даних *PI/T:*

```
MOVEA.L #$800001,A0     ; Завантаження базової адреси PI/T до A0
MOVE.B #0,($0C,A0)      ; Завантаження до регістра керування порту A PACR
                        ; 0 задля встановляння нульового режиму
MOVE.B #$FF,(4,A0)      ; Завантаження до регістра напрямку обміну даними
                        ; порту A даного $FF задля забезпечення виведення
                        ; по всіх розрядах
MOVE.B #$55,($0,A0)     ; Завантаження до регістра даних порту A даного
                        ; $55 (01010101)
```

```
JSR.B *+12              ; Звернення до підпрограми затримки
BRA.B *-16              ; Циклування програми
MOVE.L #1000000,D0      ; Завантаження лічильника підпрограми затримки
SUBQ.L #1,D0            ; Декрементування лічильника
BNE.B *-2               ; Організація підпрограми затримки на 1 с
RTS                     ; Повернення з підпрограми затримки
```

Наведений фрагмент програми є драйвер виведення *PI/T*.

Фрагмент драйвера введення наведено нижче:

```
MOVE.B $800015.L,D0     ; Введення молодшого байта даного з додаткового
                        ; регістра порту A PADR до регістра D0
```

За бажання ввести до регістра *D*0 слова з портів *A* та *B* можна застосовувати команду *MOVEP*, яка виконується тільки з непрямою регістровою адресацією:

```
MOVEA.L $800001,A5
.
.
.
MOVEP ($14,A5),D0
```

За командою *MOVEP* біти *D*15...*D*8 з додаткового регістра порту *A* пересилаються до першого байта регістра *D*0, а біти *D*7...*D*0 пересилаються з додаткового регістра порту B до нульового байта регістра *D*0.

Виводи *H*3...*H*1 може бути запрограмовано на виконування функцій входів на запит переривання, входів-виходів даних або передавання сигналів квитування при обміні даними через порти *A, B*. Якщо виводи *H*3 та *H*1 використовуються для введення даних, то поточні сигнали на цих виводах дублюються у чотирьох розрядах регістра стану *PSR*. Для виведення даних використовуються виводи *H*2 та *H*4, значення даних на них встановлюються відповідно до вмісту певних розрядів регістра *PSR*. Через читання або запис вмісту *PSR* мікропроцесор керує введенням або виведенням даних.

Порти *A, B* можуть програмуватись для реалізації чотирьох різних режимів обміну.

Режим 0 – односпрямований обмін 8-розрядними даними через окремі порти A і B. Напрямок обміну для кожного виводу задається вмістом регістрів *PADDR* та *PBDDR*. Виводи *H*1 і *H*2 можуть використовуватись для квитування передавання через порт *A*, а виводи *H*3 і *H*4 – для квитування передавання через порт *B*. При введенні даних до відповідного порту сигнали *H*1 і *H*3 вказують на надходження даних від зовнішнього пристрою, при виведенні даних – на їхнє приймання зовнішнім пристроєм. Сигнали *H*2 і *H*4 при введенні підтверджують

отримання даних від зовнішніх пристроїв, а при виведенні слугують за запити на приймання даних зовнішніми пристроями.

Режим 1 – односпрямований обмін 16-розрядними даними через подвійний порт *A-B*. Для квитування обміну використовуються сигнали на виводах *H*3, *H*4, функціональне призначення яких при введенні та виведенні даних є таке ж саме, як і в режимі 0. Виводи *H*1 та *H*2 використовуються для введення-виведення даних або для подавання запитів на переривання.

Режим 2 – двоспрямований обмін 8-розрядними даними через порт B. Виводи *H*1 та *H*2 слугують для квитування виведення даних до зовнішнього пристрою, виводи *H*3 та *H*4 – для квитування введення даних до порту *B*. Порт *A* у цьому режимі використовується для односпрямованого обміну даними без квитування.

Режим 3 – двоспрямований обмін 16-розрядними даними, які зчитуються мікропроцесором через подвійний порт *A-B*. Виводи *H*1 та *H*2 слугують для квитування виведення даних до зовнішнього пристрою, а виводи *H*3 та *H*4 – для квитування введення даних до порту.

Порт *C* може використовуватись для обміну 8-розрядними даними, які записуються або зчитуються мікропроцесором через регістр *PCDR*. Обмін використовується без квитування. Напрямок передавання даних для кожного виводу *PCi* задається відповідним розрядом вмісту регістра *PCDDR*, який вводиться до нього у перебігу ініціалізації порту C. Виводи *PC*2, *PC*3, *PC*7 може бути запрограмовано для обслуговування таймера, виводи *PC*5, *PC*6 – для запиту на переривання та приймання підтвердження переривання від мікропроцесора. Вивід *PC*4 може використовуватись для формування запиту на прямий доступ до пам’яті.

Якщо виводи *H*3...*H*1 або частина з них програмується для роботи на приймання запитів на переривання за ініціалізації блока керування обміном, то надходження на них запиту зумовлює формування сигналу *PIRQ#* = 0 на виводі *PC*5 порту *C*. Цей сигнал подається на вхід пріоритетного шифратора *PRCD*, який формує відповідний код запиту на переривання *IPL*2...*IPL*0, який подається на мікропроцесор. Коли мікропроцесор переходить до обслуговування цього запиту, він видає сигнал підтвердження, який надходить на вхід *PIACK#* – вивід *PC*6. Мікропроцесор формує адресу регістра векторів переривань PIVR і зчитує вектор при зверненні до таблиці переривань для адресування підпрограми обслуговування цього переривання. Кожному запитові Ні відповідає свій вектор переривань. Старші шість розрядів цього вектора записуються до регістра *PIVR PI/T* у перебігу ініціалізації, а два молодших розряди відповідають номерові входу Ні, на який надійшов запит. Пріоритети запитів від зовнішніх пристроїв і порядок їхнього обслуговування визначаються вмістом регістра *PSSR PI/T,* який завантажується у перебігу ініціалізації.

Таймер, який входить до складу *PI/T,* зреалізовано на базі 24-розрядного лічильника *CNT*, який працює на віднімання. Початковий стан лічильника встановлюється за ініціалізації через запис 3-х байтів до регістра попереднього встановляння *CPR*. Запуск таймера здійснюється через запис до регістра

керування таймером *TCR* керувального коду, який визначає також режим його функціонування. Сигнали на *CNT* можуть надходити як від генератора синхроімпульсів *CLK*, так і від зовнішніх пристроїв на вхід *TIN*# (вивід *PC*2).

У режимі лічби імпульсів *CLK* таймер через інтервали часу, які визначаються вмістом *CPR*, формують сигнали на виході *TOUT*# (вивід *PC*3). У режимі лічби подій (лічба сигналів на вході *TIN*#) поточний вміст *CNT* вказує на кількість імпульсів, що надійшли. Також можна запрограмувати ділення частоти вхідних імпульсів на 32.

У режимі лічби синхроімпульсів сигнал на вході *TIN*# слугує для запуску (*TIN*# = 1) або зупину (*TIN*# = 0) лічильника. Сигнал на виході таймера *TOUT*# (вивід *PC*3) має початкове значення *TOUT*# = 1 і змінює своє значення, коли вміст лічильника дорівнює 0. Після цього лічильник може відновити початкове значення і повторити лічбу або завантажити нове значення з регістра *CPR*. Коли стан лічильника стає нульовим, у регістрі *TSR* встановлюється значення біта *ZDS* = 1 (прапорець нуля). Поточний стан таймера контролюється мікропроцесором через зчитування та аналіз вмісту регістра *TSR*.

При роботі таймера на виході *TOUT*# формуються прямокутні імпульси, які можуть використовуватись для керування зовнішніми пристроями (періодичне включення-виключення, синхронізація тощо). Сигнал *TOUT* може подаватися на вхід приорітетного шифратора як сигнал запиту для переривання для мікропроцесора. Сигнал підтвердження переривання надходить на вхід *TIACK* (вивід *PC*7). Для зчитування вектора переривання мікропроцесор звертається до регістра *TIVR*, до якого цей вектор записується за ініціалізації таймера.

Таймери використовуються у вартових пристроях, для затримки сигналів, для запуску підпрограм у задані моменти часу тощо.

На рис. 11.17 подано часові діаграми передавання слова через 8-розрядний інтерфейс *PI/T*. Сигнали *SIZ*1, *SIZ*0 мають відповідно значення 10...2 байти в перебігу першого циклу і 1 байт (значення 01) – в перебігу другого циклу. Сигнали *DSACK*1# та *DSACK*0#, які вказують на готовність порту до роботи в кожному циклі, мають значення 11 в перебігу очікування і 10 (1 байт) – у кожному з циклів передавання даних до *PI/T*.

**Контрольні питання:**

1 Які функції можна зреалізовувати на ВІС *PI/T*?

2 Які пристрої входять до складу ВІС *PI/T*?

3 Скільки регістрів може бути адресовано до ВІС *PI/T*?

4 У скількох режимах роботи може працювати ВІС *PI/T* і чим вони відрізнюються один від одного?

5 Поясніть термін – обмін інформацією з квитуванням.

6 Для чого використовуються виводи *H*1...*H*4 в різних режимах роботи?

7 У яких режимах може працювати таймер ВІС *PI/T*?

8 Які виводи ВІС *PI/T* використовує таймер?

**Контрольні питання підвищеної складності:**

1 Чи можна використовувати ВІС *PI/T* для обміну словами?

2 Проілюструйте за допомогою часових діаграм процес передавання слова.

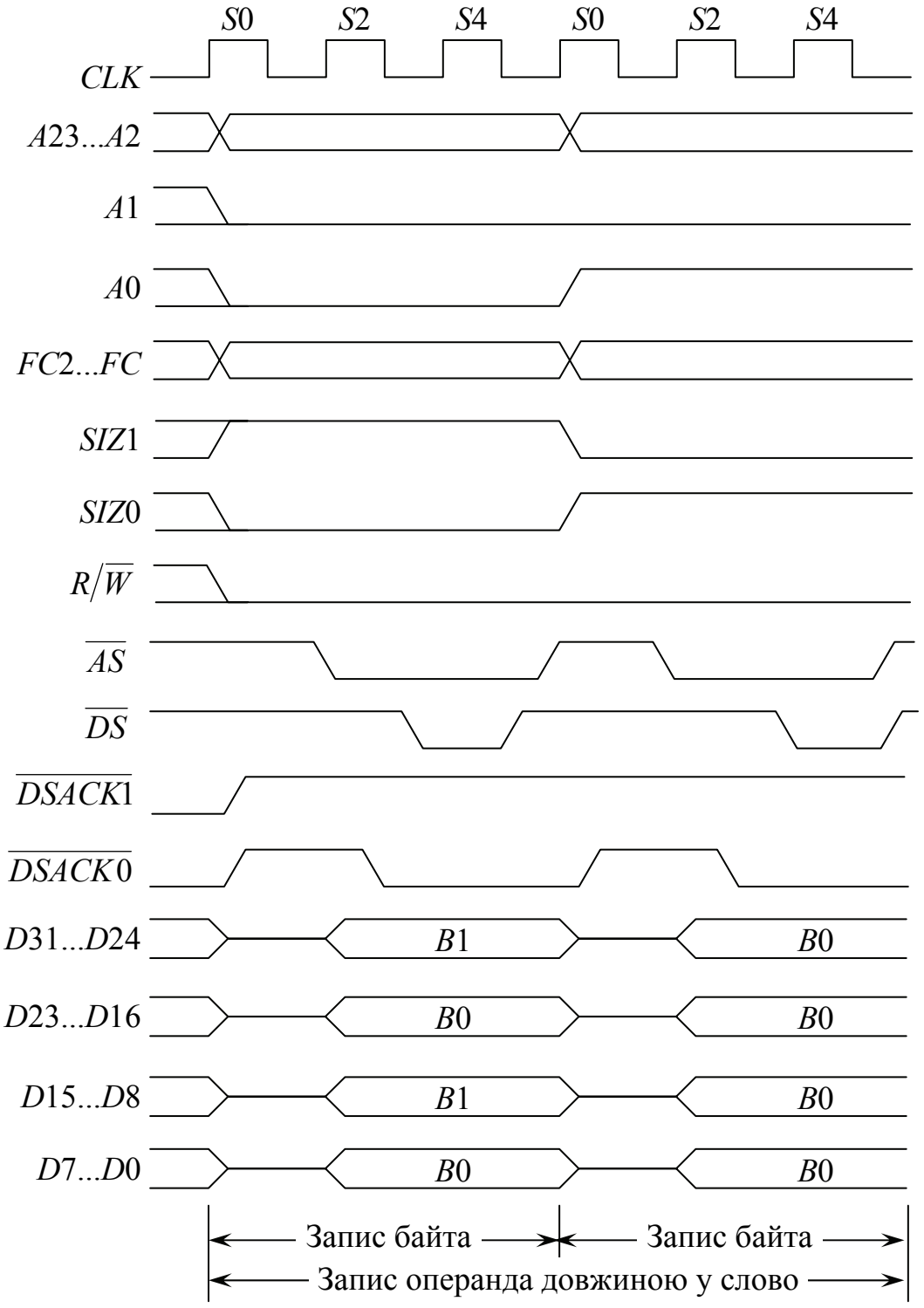



Рисунок 11.17 – Часові діаграми передавання слова через 8-розрядний інтерфейс *PI/T*

## 11.3 32-розрядні мікропроцесори сімейства *M*680*XX* фірми *Motorola*

**Вхідний контроль:**

1 Яку назву мають регістри загального призначення МП фірми *Intel*?

2 Чи закріплено згадані у п. 1 регістри за умовчанням за певними функціями?

3 Що таке сегментування пам'яті і в яких МП воно використовується?

4 З якою метою пам'ять поділяється на сегменти?

5 Чи є сумісні різні моделі універсальних МП фірми *Intel* і, якщо так, то як цього досягають?

6 Розпочинаючи з якої моделі МП фірми *Intel* використовують конвеєр команд?

7 Розпочинаючи з якої моделі МП фірми *Intel* використовують кеш-пам'ять?

32-розрядні універсальні мікропроцесори фірми *Motorola* *MC*68020, *MC*68030, *MC*68040, *MC*68060 побудовано на підставі базової моделі *MC*68000. Їхньою характерною особливістю є наявність засобів, які забезпечують їхню спільну роботу з співпроцесорами задля обробки чисел з плаваючою точкою. Розпочинаючи з моделей *MC*68040, 32-розрядні процесори побудовано за гарвардською архітектурою з розділенням потоків даних та команд; кеш-пам'ять є багаторівневою; є апаратна підтримка реалізації мультипроцесорних систем; введено внутрішні засоби самотестування та налагодження. МП *MC*68060 має суперскалярну структуру з паралельною роботою двох виконавчих пристроїв обробки цілочисельних операндів і одного пристрою обробки операндів з плаваючою точкою. На базі моделі MC68020 розроблено сімейство спеціалізованих комунікаційних контролерів *M*683*XX*.

32-розрядні МП *MC*680*X*0 сумісні програмно "знизу догори" і мають ту ж саму програмну модель користувача, що й базова модель МП *MC*68000.

МП *MC*68020 зреалізовує розширений набір команд та способів адресування, які стали базовими для всіх наступних моделей сімейства, має внутрішню кеш-пам'ять команд на 256 байт і передбачає підключення зовнішніх співпроцесорів. МП *MC*68030 і старші моделі вміщують вже додатково кеш-пам'ять даних та блок керування пам'яттю, який виконує сторінкове адресування.

Програмну модель користувача МП сімейства *M*680*X*0 подано на рис. 11.18.

Програмна модель складається з восьми 32-розрядних регістрів даних – *D*7…*D*0, восьми 32-розрядних регістрів адрес – *A*7…*A*0, з яких регістр *A*7 є вказівник стека користувача *USP*, 32-розрядного регістра-лічильника команд *PC* та 16-розрядного регістра стану *SR***,** молодші 8 розрядів якого є регістр коду умов (прапорців) *CCR* користувача, а 8 старших розрядів вміщують біти стану процесора в режимі супервізора. Регістрову модель супервізора подано на рис. 11.19.

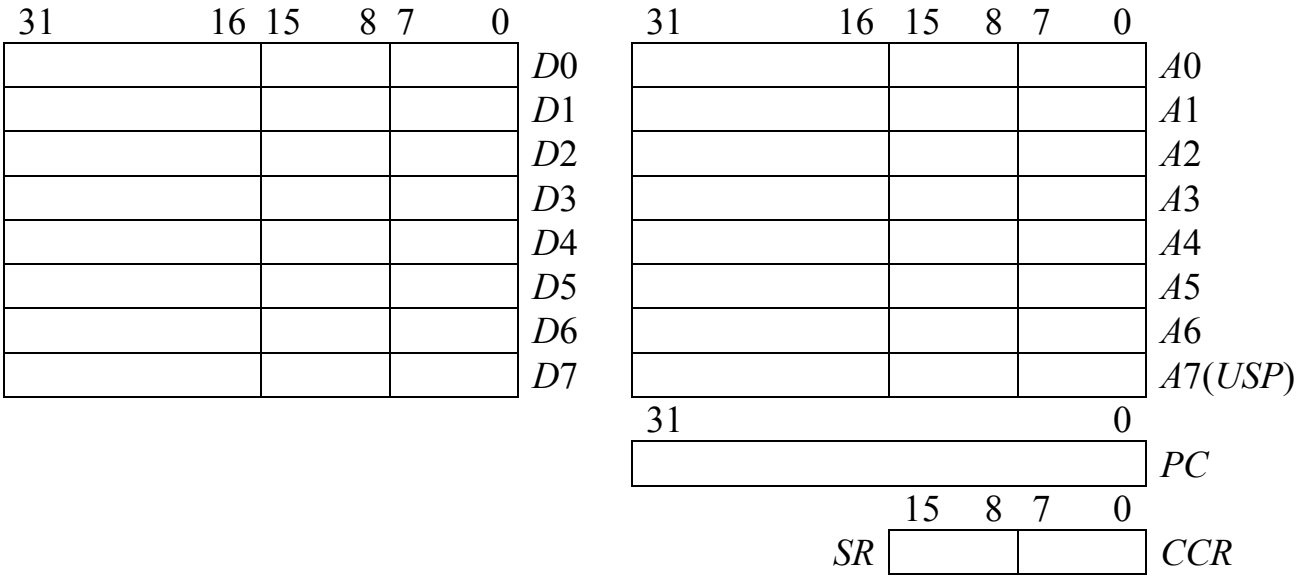



Рисунок 11.18 – Програмна модель користувача

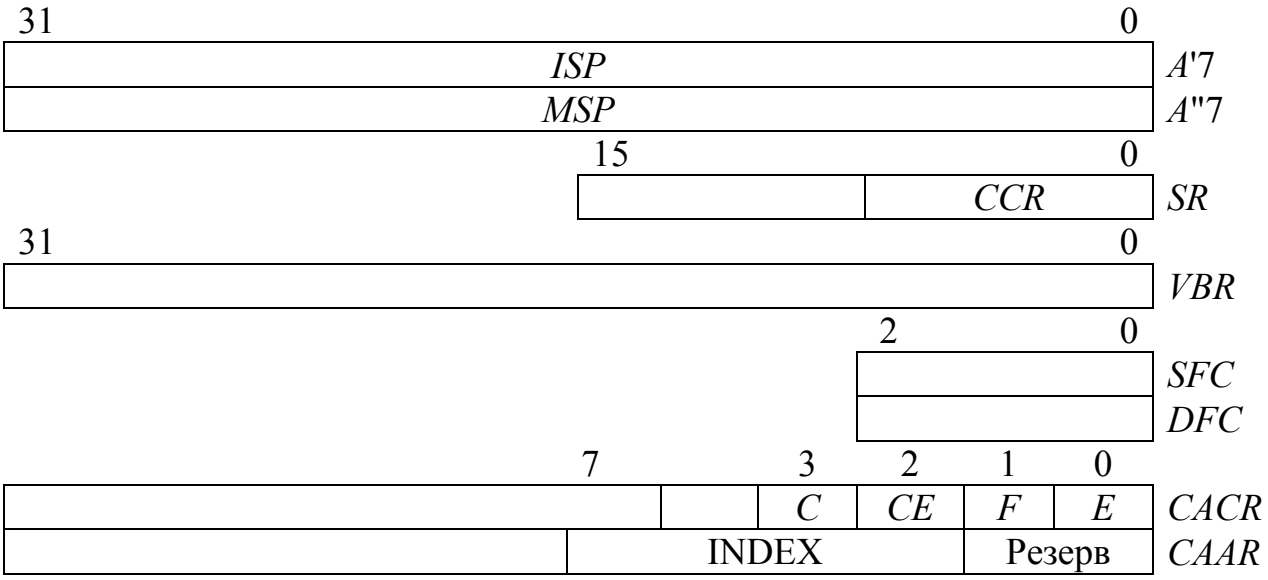



Рисунок 11.19 – Регістрова модель супервізора *MC*68020

Регістрова модель вміщує додатково два 32-розрядні регістри *ISP* (*A*7') та *MSP* (*A*7"). Регістр *ISP* виконує функції вказівника стека супервізора при обробці переривань та виключень, а регістр *MSP* використовується операційною системою при роботі у багатозадачному режимі. Він вказує на головний стек, до якого при перемиканні задач заноситься адреса повернення до попередньої задачі. Регістр *VBR* вміщує базову адресу таблиці векторів переривань, яка може переміщуватись у адресному просторі.

До трьох молодших розрядів регістрів *SFC* та *DFC* заноситься код адресного простору, який надходить на зовнішні виводи *FC*2…*FC*0 при пересиланні даних поміж регістрами процесора *D*7…*D*0 або *A*7…*A*0 та зовнішньою пам’яттю у привілейованих командах.

Під час запису даних з регістра *Dn* або *An* у пам’ять функціональний код *FC*2…*FC*0 відповідно до таблиці:

| *FC*2 | *FC*1 | *FC*0 | Простір пам’яті |
|---|---|---|---|
| 0 | 0 | 1 | Пам’ять даних користувача |
| 0 | 1 | 0 | Пам’ять команд користувача |
| 1 | 0 | 1 | Пам’ять даних супервізора |
| 1 | 1 | 0 | Пам’ять команд супервізора |

надходить на входи МП з регістра DFC, а при зчитуванні з пам’яті – до регістра Dn або *An* – з регістра *SFC*. Одночасно функціональний код надходить до блока керування пам’яттю *MMU*, який звертається до відповідного адресного простору в пам’яті.

Регістри *CACR* та *CAAR* керують кеш-пам’яттю; окремі біти регістра *CACR* мають таке призначення:

*E* = 1 дозволяє роботу кеш-пам’яті, *E* = 0 забороняє звернення до кеша, але вміст кеша зберігається;

*CE* = 1 анулює вміст рядка кеша, номер якого задається вмістом поля *INDEX* у регістрі *CAAR*;

*F* = 1 забороняє заповнення рядка кеша у разі кеш-промаху, фіксуючи поточний стан кеша;

*C* = 1 анулює вміст всіх рядків кеш-пам’яті.

На рис. 11.20 подано біти умов регістра стану.

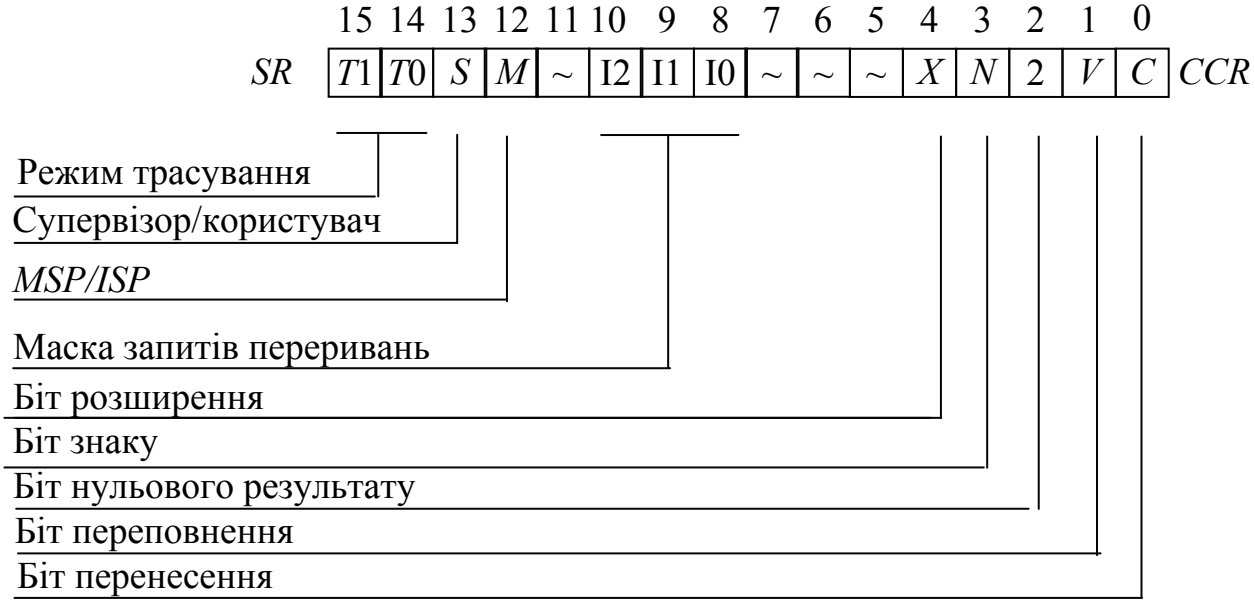



Рисунок 11.20 – Регістр стану *SR*

Біти *T*1…*T*0 задають режими трасування:

| *T*1 | *T*0 | Режим |
|---|---|---|
| 0 | 0 | Трасування відсутнє |
| 0 | 1 | Покроковий режим |
| 1 | 0 | Зупин виконування програми після команд *JMP, JSR, BRA* тощо |

Біт $S = 1$ визначає режим супервізора, $S = 0$ – режим користувача.

Біт $M = 1$ при значенні біта $S = 1$ зумовлює вибір головного вказівника стека MSP, а $M = 0$ – вибір регістра *ISP* – вказівника стека переривань.

**Контрольні питання:**

1 Які нові регістри порівняно з МП *MC*68000 вміщує програмна модель користувача МП *MC*68020?

2 Які основні регістри має регістрова модель супервізора MC68020?

3 Чи має МП *MC*68020 конвеєр команд?

4 В чому полягає різниця щодо розміщення у пам’яті таблиць векторів переривань у мікропроцесорів фірми *Intel* та *Motorola*?

5 Які нові прапорці вміщують старші 8 розрядів регістра *SR* і на що вони вказують?

6 Що таке маска переривань і з якою метою вона використовується?

**Контрольні питання підвищеної складності:**

1 Що таке *MMU* і які функції він виконує?

2 Як працює блок *MMU* за взаємодії з жорстким диском?

3 Як зорганізовується у МП *MC*68010 і старших його моделей процедура перезапису відсутньої у ОЗП сторінки з жорсткого диску: апаратно чи програмно?

## 11.4 Побудова МПС на 32-розрядних мікропроцесорах фірми *Motorola*

### *11.4.1 Підсистема центрального процесорного елемента*

**Вхідний контроль:**

1 Яку розрядність мають ШД та ША МП *MC*68000?

2 Які системні сигнали ВІС МП *MC*68000 Вам відомі?

3 Чи є згадані в п. 2 сигнали односпрямовані або двоспрямовані й чому?

4 За яким алгоритмом працює пріоритетний шифратор?

5 На якій частоті працює МП *MC*68000?

6 Який сигнал треба подати на вхід *AVEC*# ВІС МП *MC*68000?

За основу МПС було взято контролер – модуль розвитку фірми *Flight Electroniks – FLIGHT-68EC020 EVM.*

У якості центрального процесора в МПС використовується МП *MC*68*EC*020, який має 32-розрядну шину даних та 24-розрядну шину адреси, що є достатнім для більшості застосувань МПС у складі пристроїв телекомунікацій. В іншому параметри *MC*68*EC*020 збігаються з параметрами *MC*68020, збігаються також програмні моделі супервізора і користувача. Однаковими, за незначними винятками, є і системи команд. Зовнішні виводи МП *MC*68*EC*020 подано на рис. 11.21.

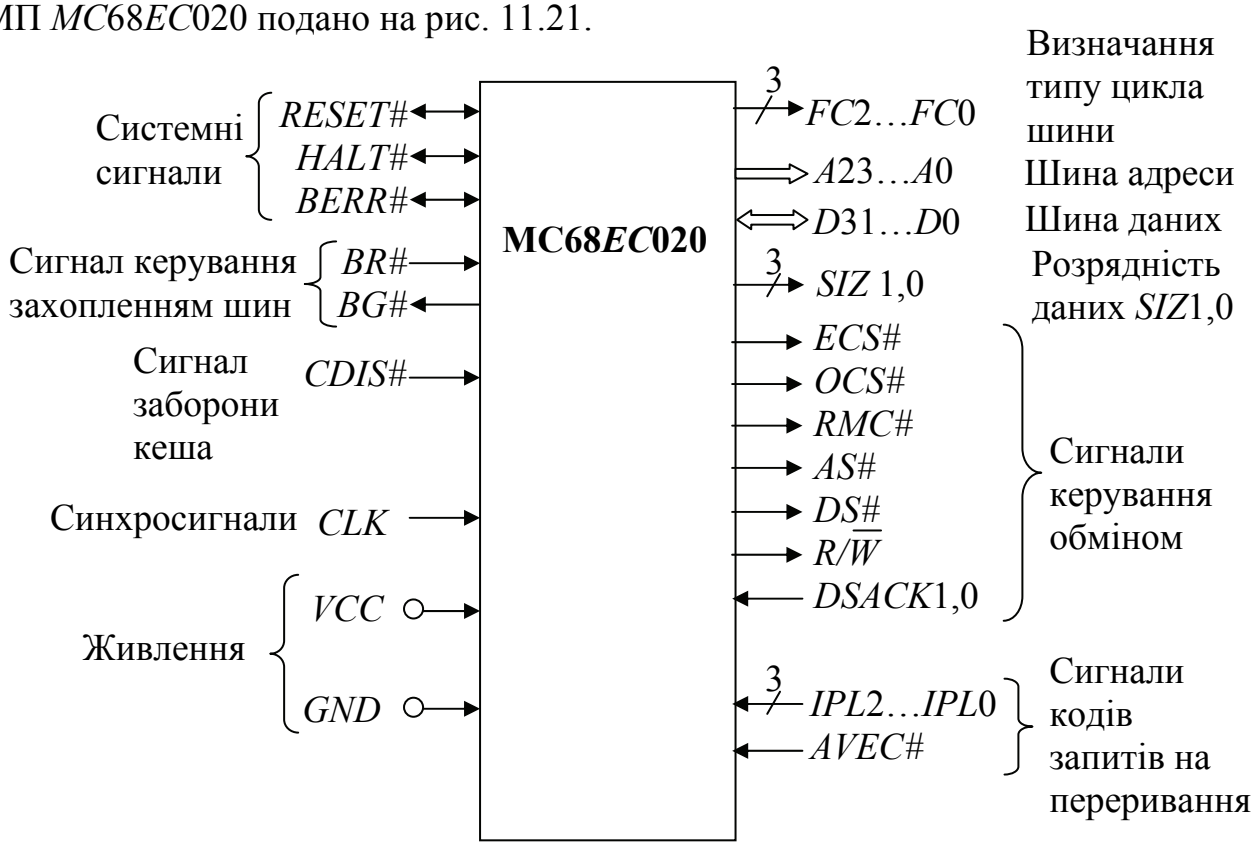



Рисунок 11.21 – Зовнішні виводи МП *MC*68*EC*020

На виводи МП *A*23...*A*0, *D*31...*D*0 надходять 24-розрядні адреси та 32-розрядні дані. Сигнали на виходах *FC*2...*FC*0 зазначають тип виконуваного циклу відповідно до табл. 11.14.

За *FC*2 = *FC*1 = *FC*0 = 1 може працювати підключений співпроцесор.

Таблиця 11.14 – Типи виконуваних циклів

| *FC*2 | *FC*1 | *FC*0 | Тип виконуваного циклу |
|---|---|---|---|
| 0 | 0 | 0 | резервовано для подальшого розвитку |
| 0 | 0 | 1 | вибирання даних користувача |
| 0 | 1 | 0 | вибирання команд користувача |
| 0 | 1 | 1 | резервовано |
| 1 | 0 | 0 | резервовано |
| 1 | 0 | 1 | вибирання даних супервізора |
| 1 | 1 | 0 | вибирання команд супервізора |
| 1 | 1 | 1 | цикл центрального процесора |

Особливістю 32-розрядних мікропроцесорів є динамічне визначення використовуваної розрядності шини даних, яка може буди 8-, 16-, 32-розрядною. Значення вихідних сигналів *SIZ*1,0 вказують на розрядність даних, які передаються у циклі (табл. 11.15).

Таблиця 11.15 – Вихідні сигнали *SIZ*1,0, які визначають розрядність передаваного операнда

| *SIZ*1 | *SIZ*0 | Розрядність операнда |
|---|---|---|
| 0 | 0 | 4 байти |
| 0 | 1 | 1 байт |
| 1 | 0 | 2 байти |
| 1 | 1 | 3 байти |

Передавання трьох байтів (*SIZ*1,0 = 11) здійснюється при виборі довгого слова за непарною адресою, за два послідовних цикли при 32-разрядній шині даних.

Для керування обміном використовуються такі сигнали:

*AS*# – адресний стробімпульс, набирає активного значення при надходженні адреси на шину *A*23...*A*0 і зберігає це значення до завершення циклу обміну;

$R/\overline{W}$ – сигнал, який визначає напрямок передавання даних шиною *D*31…*D*0, введення (читання) відбувається за високого рівня сигналу $R/\overline{W} = 1$, а виведення (запис) – за $R/\overline{W} = 0$;

*DSACK*1,0 – вхідні сигнали підтвердження готовності зовнішніх пристроїв до обміну, які вказують на розрядність шини даних згідно з табл. 11.16; вийти з циклу очікування можна за надходження сигналу *BERR*# = 0 від зовнішньої схеми або при зміні сигналів *DSACK*1# = *DSACK*0 = 1 на інше сполучення. Сигнали *DSACK*1# та *DSACK*0# вказують на кількість байтів, які залишилось передати у даному циклі, наприклад, при передаванні довгого слова через 8-розрядний пристрій виведення;

Таблиця 11.16 – Вхідні сигнали *DSACK*1,0, які вказують на розрядність шини даних

| *DSACK*1# | *DSACK*0# | Розрядність шини даних |
|---|---|---|
| 0 | 0 | 4 байти |
| 0 | 1 | 2 байти |
| 1 | 0 | 1 байт |
| 1 | 1 | Цикл очікування |

*DS*# – стробімпульс даних, який у циклі читання дорівнює 0 водночас зі стробом адреси; у циклі запису набирає значення *DS*# = 0 через один такт після адресного стробімпульсу; він сигналізує, що МП видав дані на шину даних;

*ECS*# – вихідний сигнал початку циклу обміну, який набирає значення *ECS* ≠ 0 упродовж першого такту кожного нового циклу;

*OCS*# – вихідний сигнал початку циклу обміну даними, який має значення *OCS*# = 0 упродовж першого такту початкового циклу передавання даних;

*RMC*# – вихідний сигнал циклу "читання-модифікація-запис", який зреалізовується при виконуванні команди *TAS*, значення *RMC*# = 0 встановлюється на початку першого циклу цієї команди і зберігається в перебігу її виконування.

Сигнали переривань *IPL*2, *IPL*1, *IPL*0, системні сигнали початкового встановлення *RESET*#, зупину *HALT*#, помилки звернення до шини *BERR*#, а також сигнали керування захопленням шини *BR*#, *BG*# виконують ті ж самі функції, що й у мікропроцесорі *MC*68000.

Сигнал заборони роботи кеша *CDIS*# 0 зазвичай використовується у режимі налагодження МПС.

Сигнал *AVEC*#, який дорівнює логічному нулю, дозволяє автовекторні переривання.

На вхід *CLK* подаються зовнішні синхросигнали з частотою 16 МГц. Напруга живлення процесора становить 5 В, споживана потужність не перевищує 2 Вт.

МП *MC*68*EC*020 підтримує оброблення до семи запитів на переривання від пристроїв введення-виведення, які повинні подати відповідні сигнали *IRQ*7#...*IRQ*1# (*INT*7#...*INT*1#) на входи схеми пріоритетного шифратора *PCD* (рис. 11.22). Для кожного запиту встановлено пріоритет обслуговування: найвищий *Li* = 7 – для запиту на вході *IRQ*7, найнижчий *Li* = 1 – для запиту на вході *IRQ*1. Запит переривання ініціюється при надходженні логічного нуля на відповідний вхід *IRQi*#; на виходах *PCD* формується трирозрядний код *IPL*2#, *IPL*1#, *IPL*0#, який відповідає номерові запиту, який має максимальний пріоритет. Вхід *IRQ*0 завжди підмикається до низького потенціалу і, за відсутності запитів на входах *PCD,* формує код *IPL*0#, *IPL*1#, *IPL*2# = 111, поданий у інверсній формі, який не спричинює переривань.

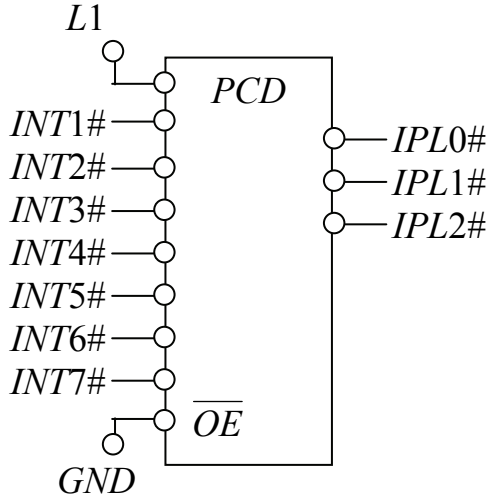



Рисунок 11.22 – Схема формування кодів пріоритетів запитів переривань

**Контрольні питання:**

1 З якою метою МП *M*68*EC*020 формує сигнали керування обміном *ECS*# та *OCS*#?

2 Що визначають вихідні сигнали *SIZ*1 та *SIZ*0?

3 Що визначають вхідні сигнали *DSACK*1# та *DSACK*0#?

4 Чому не можна переривати цикл “читання–модифікація–запис”?

5 На який вхід пріоритетного шифратора *PCD* треба підімкнути пристрій введення-виведення, щоби надати йому 3-й пріоритет у інверсному коді?

6 Який код має виставити МП на виходах *FC*2, *FC*1, *FC*0 у режимі циклу центрального процесора (циклу підтвердження переривань)?

7 З якою метою подається рівень логічного нуля на вхід $\overline{OE}$ пріоритетного шифратора?

**Контрольні питання підвищеної складності:**

1 На входи пріоритетного шифратора надходять одночасно запити на переривання *INT*1#, *INT*5#, *INT*6#. Який код – *IPL*0#, *IPL*1#, *IPL*2# – буде сформовано на виходах пріоритетного шифратора?

2 Маска переривань має значення $I2 = 1$, $I1 = 0$, $I0 = 0$. Зазначте номери пріоритетів обслуговуваних запитів на переривання.

### *11.4.2 Розподіл адресного простору МПС*

**Вхідний контроль:**

1 З якою метою адресний простір розподіляється поміж підсистемами МПС різних типів?

2 Чи можуть займати адресний простір периферійні пристрої?

3 Чи займає адресний простір центральний процесор?

4 Чи займає адресний простір співпроцесор?

За основу підходу до розділу адресного простору, який формує МП *MC*68*EC*020, було взято мапу пам’яті контролера *FLIGHT*-68*EC*020*EVM*. Розподіл адресного простору МПС подано на рис. 11.23.

Адресний простір розподіляється поміж ПЗУ – адреси \$000000...\$3*FFFFF*, ОЗП – адреси \$400000...\$7*FFFFF*, паралельними периферійними інтерфейсами-таймерами *PI/T* – адреси \$800000...\$9*FFFFF*, та асинхронними послідовними адаптерами *DUART* – адреси *A*00000...*FFFFFF*.

Розподіл адресного простору поміж типами пристроїв можна зреалізовувати на програмованій логічний матриці – ПЛМ, наприклад 16*L*8*D* (*PAL*2) фірми *Motorola* (рис. 11.24).

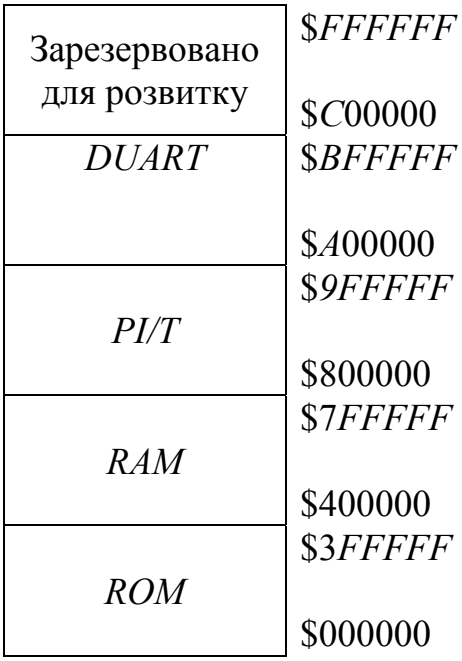



Рисунок 11.23 – Розподіл адресного простору МПС

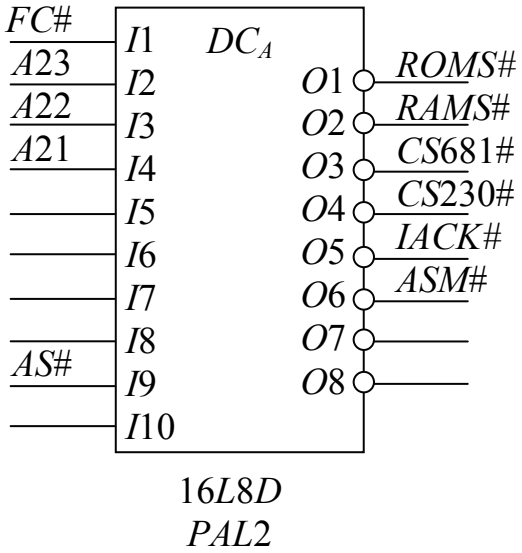



Рисунок 11.24 – Схема формування сигналів розподілу адресного простору

Вихідні логічні функції – це сигнали: *ROMS#* – дозвіл на роботу декодерів адреси ВІС постійної пам'яті, *RAMS#* – дозвіл на роботу декодерів адреси ВІС оперативної пам'яті, *CS230#* – дозвіл на роботу декодерів адреси ВІС *PI/T, CS*681# – дозвіл на роботу декодерів адреси ВІС *DUART*. Отже, розподіл адресного простору зреалізовується за допомогою ієрархічної структури: ПЛМ розподіляє адресний простір на підпростори для пристроїв різних типів, а всередині кожного підпростору адреси розподіляються поміж окремими ВІС одного призначення за допомогою декодерів адреси.

Перед тим як застосовувати декодер адресного простору *PAL*2, треба за допомогою логічної схеми сформувати сигнал $\mathrm{FC\#} = \overline{\mathrm{FC2} \wedge \mathrm{FC1} \wedge \mathrm{FC0}}$. На рис. 11.25 подано схему формування сигналу *FC*# – циклу центрального процесора в інверсній формі.

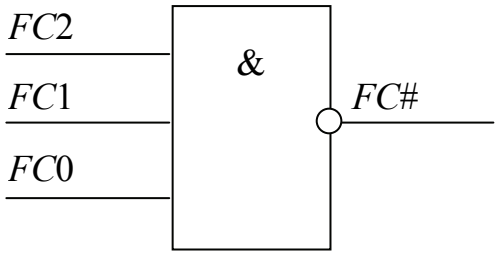



Рисунок 11.25 – Формування сигналу *FC*#

Вихідні сигнали *PAL2* може бути описано в аналітичний формі як функції вхідних сигналів:

$$IACK\# = \overline{FC} \wedge \overline{AS}$$

$$ASM\# = FC \wedge \overline{AS}$$

$$ROMS\# = \overline{ASM} \wedge IACK \wedge \overline{A23} \wedge \overline{A22} \wedge \overline{A21}$$

$$RAMS\# = \overline{ASM} \wedge IACK \wedge \overline{A23} \wedge A22 \wedge \overline{A21}$$

$$CS230\# = \overline{ASM} \wedge IACK \wedge A23 \wedge \overline{A22} \wedge \overline{A21}$$

$$CS681\# = \overline{ASM} \wedge IACK \wedge A23 \wedge \overline{A22} \wedge A21$$

З цих виразів видно, що сигнал *ROMS*# матиме низький рівень, коли розряди адреси *A*23, *A*22, *A*21 дорівнюватимуть 0, тобто розпочинаючи з адреси $000000.

Сигнал *RAMS*# дорівнюватиме 0, коли адресні розряди *A*23, *A*22, *A*21 матимуть відповідно значення 010, тобто розпочинаючи з адреси $400000.

Сигнал *CS*230# дорівнюватиме 0, коли адресні розряди *A*23, *A*22, *A*21 матимуть відповідно значення 100, тобто розпочинаючи з адреси $800000.

Сигнал *CS*681# дорівнюватиме 0, коли адресні розряди *A*23, *A*22, *A*21 матимуть відповідно значення 101, тобто розпочинаючи з адреси $*A*00000.

Нульові значення сигналів *ROMS*#, *RAMS*#, *CS*230#, *CS*681# слугують розподілу адресного простору.

Формування сигналу читання/запису *RWAS*#, який подається на входи *RAM*, здійснюється на логічних елементах (рис. 11.26).

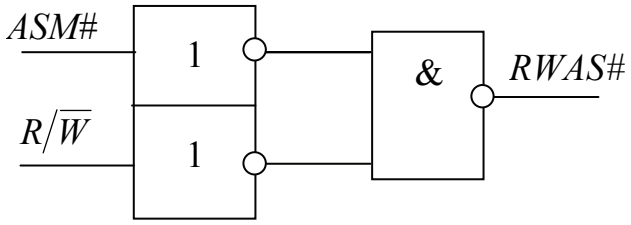



Рисунок 11.26 – Схема формування сигналу RWAS#

**Контрольні питання:**

1 Який обсяг пам'яті у Мбайтах займає кожний підпростір пам'яті у мапі пам'яті *FLIGHT*-68*EC*020*EVM*?

2 Які початкові адреси мають адресні підпростори, призначені для адресування кожного типу пристроїв, які входять до підсистем МПС?

3 Чим визначаються значення початкових адрес адресних підпросторів?

4 З якою метою формується сигнал читання/запису *RWAS*#?

5 У який спосіб можна сформувати сигнал *RWAS*#?

6 З якою метою формується сигнал *FC*#?

**Контрольні питання підвищеної складності:**

1 Як програмуються ПЛМ у якості дешифратора адреси?

2 Проілюструйте, як можна на ПЛМ 16*L*8*D* сформувати сигнал дозволу роботи співпроцесора?

### *11.4.3 Організація підсистеми пам'яті МПС*

**Вхідний контроль:**

1 Скільки мікросхем *RAM* з організацією 16x4 треба задіяти і як їх поєднати, якщо треба побудувати підсистему пам'яті з організацією 16x8?

2 Скільки мікросхем *ROM* з організацією 16x8 треба задіяти і як їх поєднати, якщо треба побудувати підсистему пам'яті з організацією 64x8?

3 За допомогою якого пристрою формуються сигнали вибирання мікросхем пам'яті $\overline{CS}$?

4 Які вхідні сигнали треба подати на мікросхеми пам'яті *RAM*?

5 Які вхідні сигнали треба подати на мікросхеми пам'яті *ROM*?

6 Які вихідні сигнали знімаються з мікросхем пам'яті і скільки їх може бути?

Шина даних *D*31…*D*0 МП *MC*68*EC*020 є тристабільна, двоспрямована, немультиплексована паралельна шина, яка слугує для обміну даними поміж мікропроцесором, пам'яттю та пристроями введення-виведення. За один цикл шини можуть пересилатися 8-, 16-, 24- або 32-розрядні дані.

Пам'ять заданого типу – постійна або оперативна – великого розміру, яка сягає сотен кбайтів або десятків Мбайтів, може бути побудована банками,

кожний з яких складається з чотирьох однакових ВІС. Кожна ВІС призначена для зберігання байтів з однаковими індексами: всіх нульових (В0), перших (*B*1), других (*B*2) та третіх (*B*3). Така конфігурація пам'яті дозволяє легко адресувати будь-яку комірку пам'яті у будь-якому з чотирьох шарів банка. Кількість банків залежить від заданого обсягу пам'яті і ємності ВІС, які утворюють ці банки. Припустимо, що треба забезпечити чотиришарову організацію пам'яті *M* x 8 кбайт, а в наявності є ВІС оперативної пам'яті з організацією *n* x 8. Враховуючи, що один банк складається з чотирьох ВІС, ємність одного банка дорівнює 4 x *n* x 8 кбайт, а кількість банків, з яких буде побудовано пам'ять, можна обчислити за формулою

$$P = \frac{M \times 8}{4 \times n \times 8},$$

де *M* – заданий обсяг пам'яті, а *n* – обсяг пам'яті однієї ВІС. Припустимо, що заданий обсяг пам'яті дорівнює 750 кбайт, а ємність однієї ВІС – 64 кбайт. Ємність одного банка становить 4 x 64 = 256 кбайт, а кількість банків дорівнюватиме трьом.

Зрозуміло, що адресування будь-якої комірки пам'яті такої конфігурації повинно мати ієрархічну структуру. У адресному просторі пам'яті заданого типу спочатку адресується банк пам'яті, потім шар в обраному банку, а в шарі вже адресується комірка пам'яті.

Адресування банків оперативної чи постійної пам'яті можна зреалізовувати за допомогою звичайного декодера адреси, дозвіл на роботу якого дають сигнали *RAMS*# або *ROMS*#, сформовані *PAL*2. На рис. 11.27 наведено формування сигналів дозволу на роботу банків *RAM*.

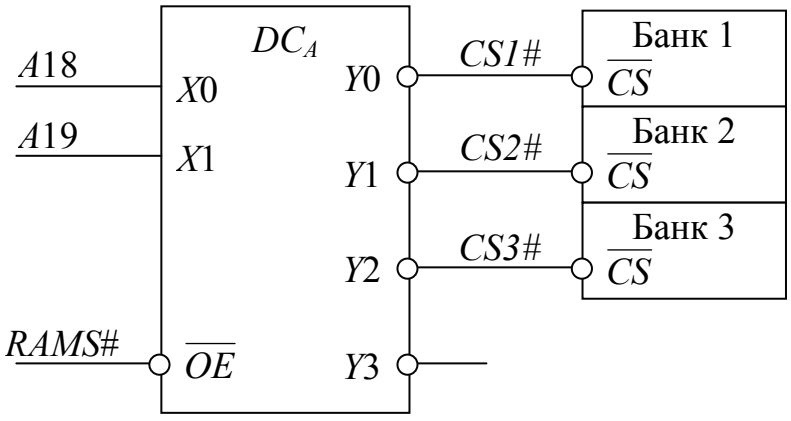



Рисунок 11.27 – Дешифратор адреси банків *RAM*

ВІС оперативної пам'яті повинні мати два керувальних входи – $\overline{CS}$, $\overline{OE}$ та вхід $R/\overline{W}$, на який подаються сигнали читання-запису *RWAS*#. В одному банку всі входи $\overline{CS}$ може бути сполучено, через те що банк обирається у

цілому. Вибір шару здійснюється сигналами *RAMU#* (*B*3), *RAMMU#* (*B*2), *RAMLU#* (*B*1), *RAML#* (*B*0), які можуть формуватися ПЛМ *PAL* 16*L*8*D* фірми *Motorola* (*PAL*1). Формування вихідних сигналів здійснюється відповідно до аналітичних виразів:

$$
\begin{aligned}
RAMU\# &= \overline{RDY} \wedge R/\overline{W} \wedge \overline{DS} \vee \overline{RDY} \wedge \overline{A1} \wedge \overline{A0} \wedge \overline{DS} \\
RAMMU\# &= \overline{RDY} \wedge R/\overline{W} \wedge \overline{DS} \vee \overline{RDY} \wedge \overline{A1} \wedge A0 \wedge \overline{DS} \vee \\
&\quad \overline{RDY} \wedge \overline{A1} \wedge SIZ1 \wedge \overline{DS} \vee \overline{RDY} \wedge \overline{A1} \wedge \overline{SIZ0} \wedge \overline{DS} \\
RAMLU\# &= \overline{RDY} \wedge R/\overline{W} \wedge \overline{DS} \vee \overline{RDY} \wedge A1 \wedge \overline{A0} \vee \\
&\quad \overline{RDY} \wedge \overline{A1} \wedge \overline{SIZ0} \wedge \overline{SIZ1} \wedge \overline{DS} \vee \\
&\quad \overline{RDY} \wedge \overline{A1} \wedge A0 \wedge \overline{SIZ0} \wedge \overline{DS} \\
RAML\# &= \overline{RDY} \wedge R/\overline{W} \wedge \overline{DS} \vee \overline{RDY} \wedge A0 \wedge A1 \wedge \overline{DS} \vee \\
&\quad \overline{RDY} \wedge A0 \wedge SIZ0 \wedge SIZ1 \wedge \overline{DS} \vee \\
&\quad \overline{RDY} \wedge \overline{SIZ0} \wedge \overline{SIZ1} \wedge \overline{DS} \vee \\
&\quad \overline{RDY} \wedge A1 \wedge SIZ1 \wedge \overline{DS}
\end{aligned}
$$

На рис. 11.28 подано схему декодера адреси шарів пам’яті у банку.

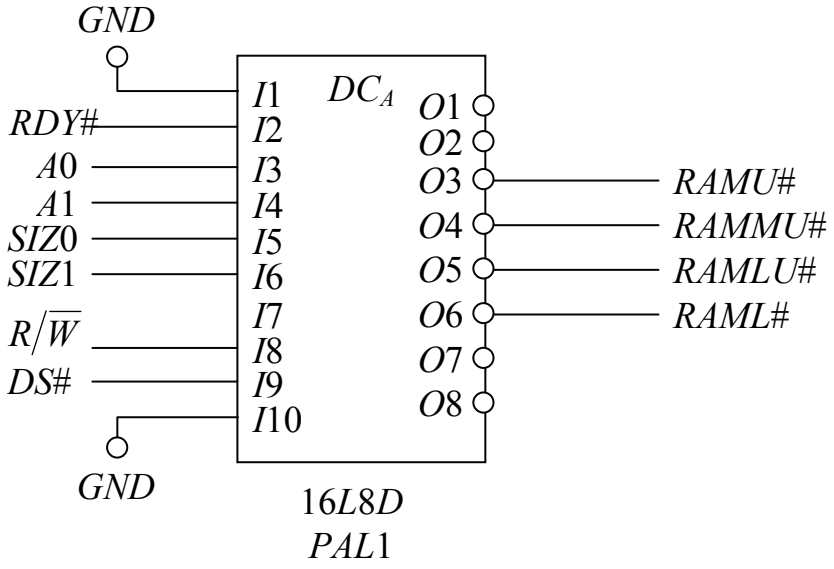



Рисунок 11.28 – Декодер адреси шарів пам’яті у банку

Сигнал *RDY* є сигнал дозволу роботи *RAM* або *ROM*; він може бути сформований на логічних елементах як логічна сума сигналів *ROMS#* та *RAMS#*:

$$RDY\# = ROMS\# \vee RAMS\#.$$

Оскільки розряди адреси *A*1...*A*0 використовуються для формування сигналів адреси шарів, на адресні входи ВІС оперативної пам'яті на 64 К подаються адресні розряди *A*17...*A*2, тому на вхід дешифратора банків можна подавати розряди *A*18, *A*19 і, в разі необхідності, – *A*20 (адресування 8-ми банків).

На рис. 11.29 подано схему підсистеми пам'яті, яка вміщує оперативну пам'ять. Підсистема пам'яті на *ROM* будується аналогічно, але сигнал $R/\overline{W}$ не подається (рис. 11.30).

**Контрольні питання:**

1 З якою організацією доцільно використовувати мікросхеми *RAM* при побудові банка пам'яті обсягом 64 *K*, яка є чотиришарова?

2 Скільки мікросхем *ROM* з організацією 64 *K* треба задіяти для побудови пам'яті з організацією 612 *K* і скільки банків, які вміщують чотири шари, треба зорганізувати?

3 Який пристрій чи програма забезпечують організацію окремих банків пам'яті для режимів супервізора та користувача МП *MC*680*X*0 фірми *Motorola*?

4 За допомогою якого пристрою розподіляється підпростір адрес, призначених для адресування банків підсистеми пам'яті?

5 За допомогою якого пристрою формуються сигнали дозволу роботи мікросхем різних шарів?

6 Як називаються сигнали, які дозволяють роботу шарів банків пам'яті? Як розшифровуються назви цих сигналів?

7 З якою метою з шини адреси МП *M*680*X*0 на адресування мікросхем пам'яті надходять адресні розряди, розпочинаючи з *A*2?

8 Чому на входи *X*0 та *X*1 $DC_A$ (див. рис. 11.27) подаються розряди адреси саме *A*18, *A*19, а не інші?

**Контрольні питання підвищеної складності:**

1 Чи потребують виходи адресних розрядів *A*31...*A*0 шинних формувачів і чому?

2 Чи можуть бути доступними для МП усі 32 розряди даних з чотиришарового банку пам'яті за один цикл шини?

### *11.4.4 Організація підсистеми введення-виведення*

**Вхідний контроль:**

1 За скільки циклів мікропроцесора можна передати слово через периферійний інтерфейс/таймер *MC*68230?

2 За скільки циклів мікропроцесора можна прийняти довге слово з периферійного інтерфейса/таймера *MC*68230?

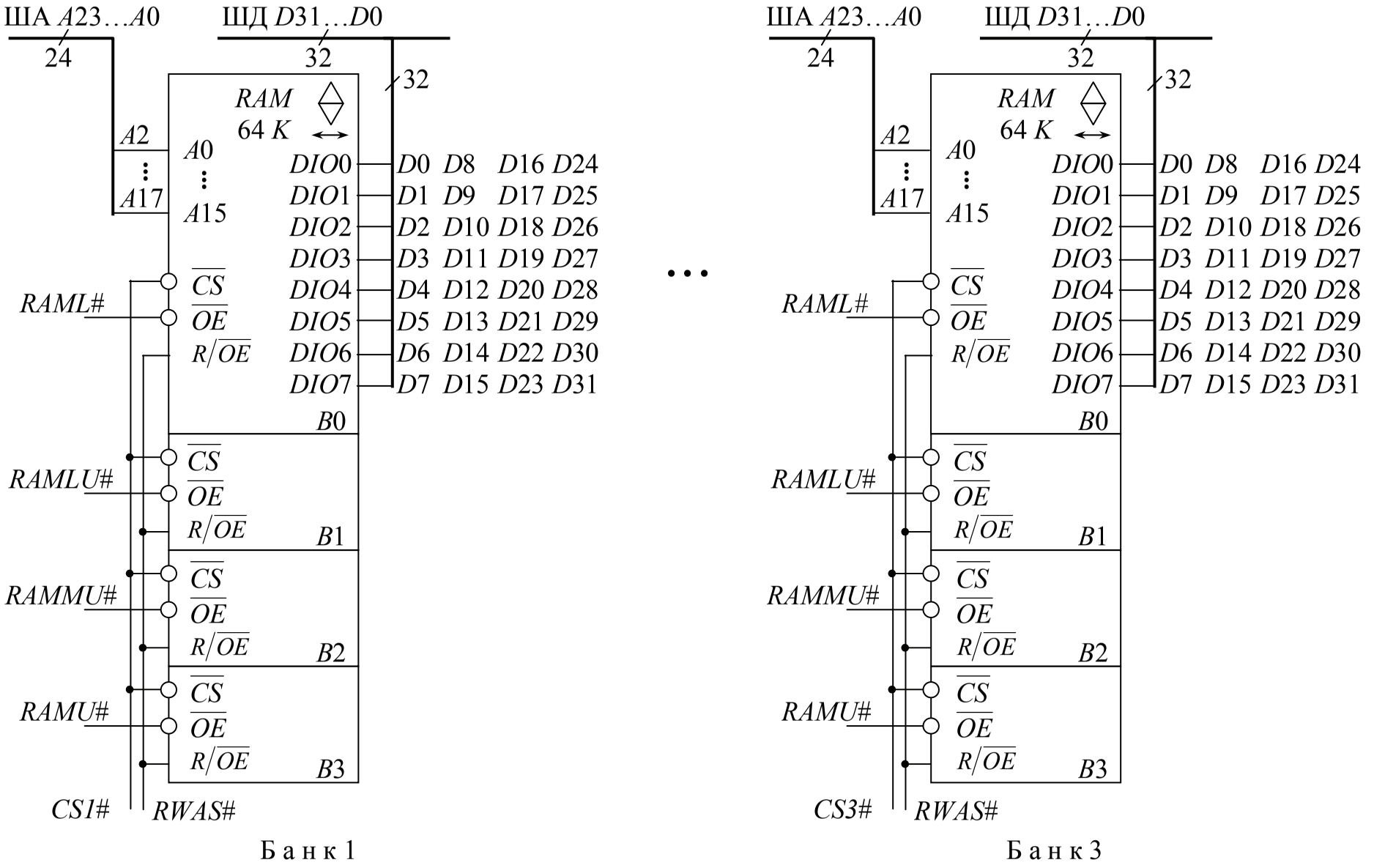



Рисунок 11.29 – Схема підсистеми пам’яті *RAM*

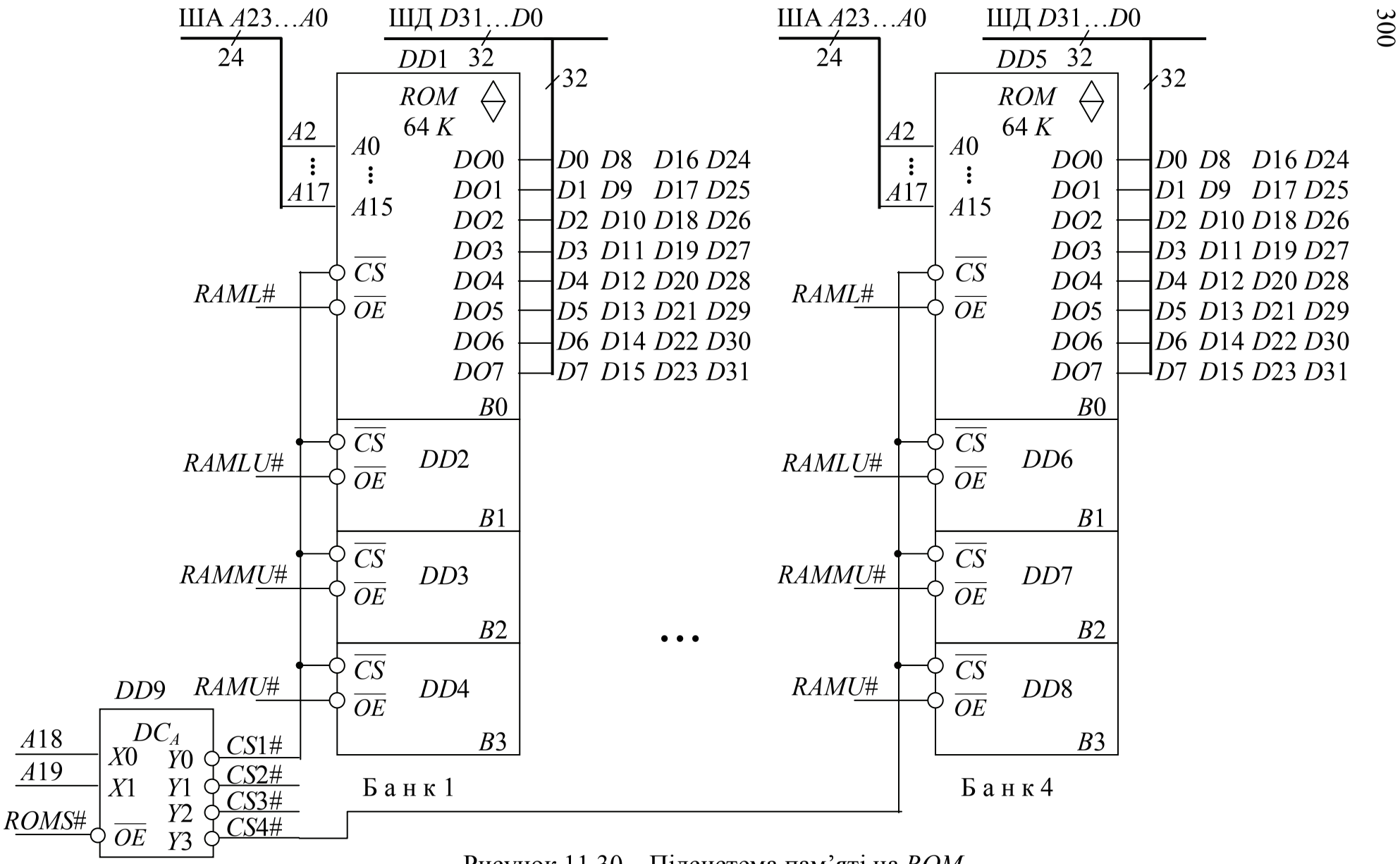



Рисунок 11.30 – Підсистема пам’яті на *ROM*

300

3 До яких розрядів шини даних МПС підмикаються входи *PI/T*?

4 Чи може таймер у складі *PI/T* викликати переривання роботи МП?

5 Чи можна через *PI/T* обмінюватись даними поміж периферійними пристроями та пам’яттю у режимі ПДП?

6 Якщо у МПС треба зорганізувати два послідовних канали на приймання та три на передавання, скільки ВІС *DUART MC*68681 має бути задіяно?

Підключення кількох периферійних інтерфейсів-таймерів та їхнє адресування здійснюється аналогічно до підключення банків пам’яті за допомогою декодерів адреси. У якості сигналу дозволу роботи дешифратора адреси використовується сигнал *CS*230#. На рис. 11.31 подано підключення трьох *PI/T*.

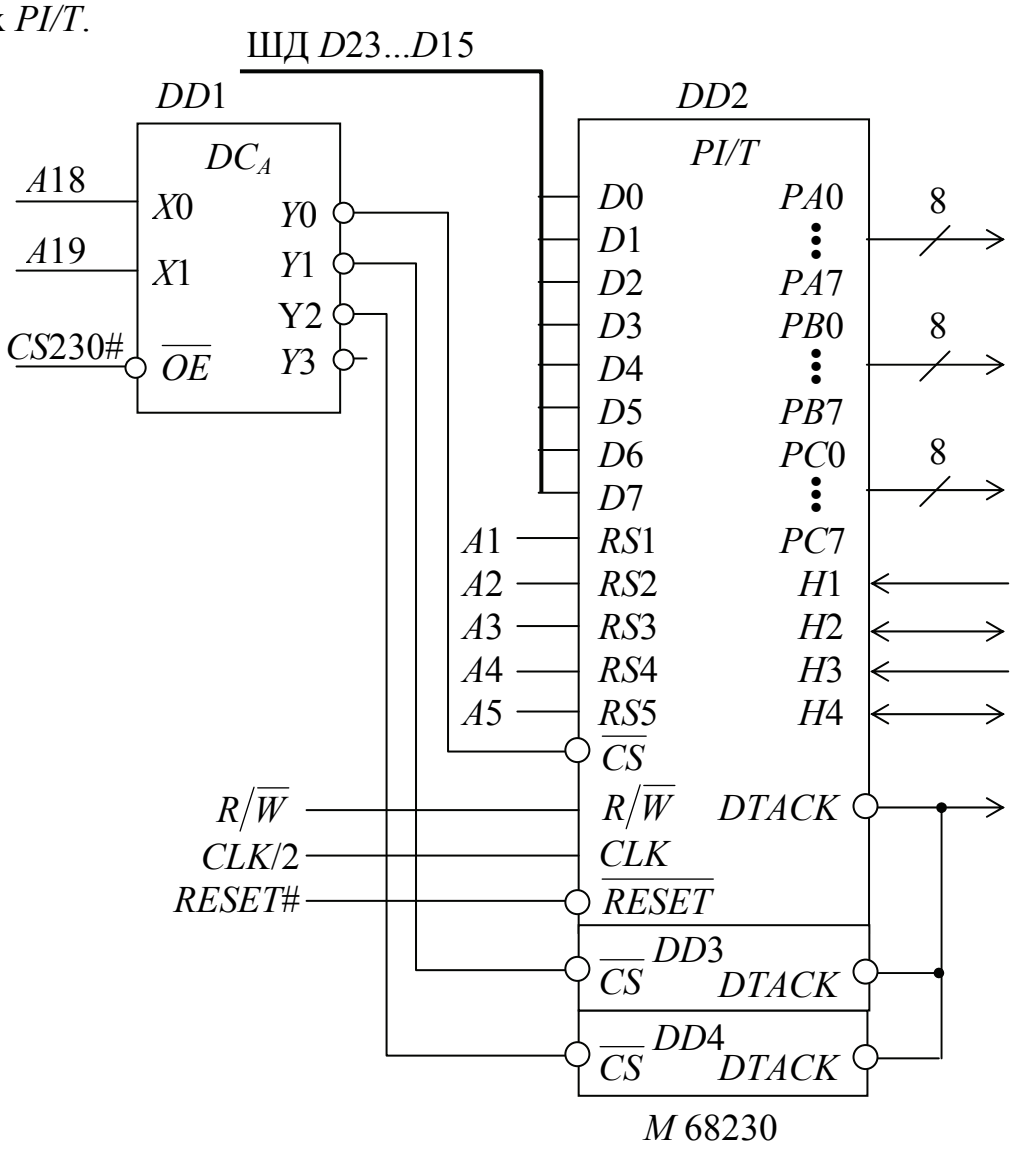



Рисунок 11.31 – Підсистема введення-виведення на трьох *PI/T*

Завдяки наявності тристабільних виводів *PI/T* при побудуванні підсистеми введення-виведення з кількох ВІС відповідні розряди *D*7…*D*0, виводи *DTACK* можна підмикати, так само як і входи $R/\overline{W}$, *CLK* та $\overline{RESET}$.

Сигнали вибору чіпа подаються з виходів декодера окремо до входів $\overline{CS}$ кожного *PI/T*. Виходи портів *PA*7...*PA*0, *PB*7...*PB*0, *PC*7...*PC*0, а також виводи *H*1, *H*2, *H*3, *H*4 підмикаються до зовнішніх пристроїв через розмикачі, які і на рис. 11.31 зазначено лише для верхньої ВІС.

Побудова підсистеми введення-виведення на ВІС *DUART* здійснюється аналогічно, а адресний простір для побудови підсистеми можна робити спільним для різних видів інтерфейсів – паралельних та послідовних, якщо їхня кількість є невелика.

На рис. 11.32 наведено частину підсистеми введення-виведення, яка є побудована на *DUART*.

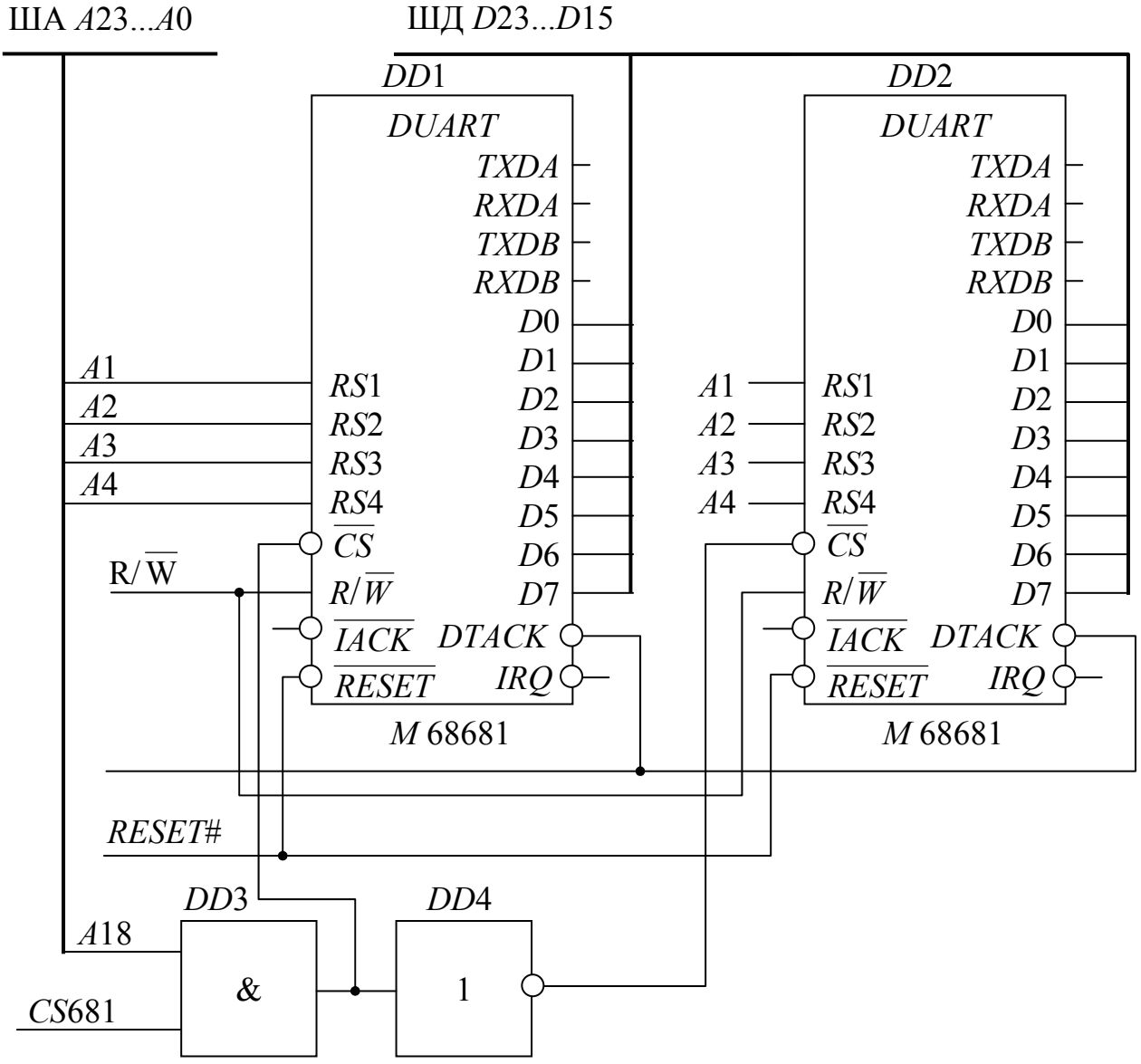



Рисунок 11.32 – Підсистема введення-виведення на двох *DUART*

Сигнали *D*7…*D*0 *DUART* можна подавати до шини даних, поєднуючи однакові розряди, так само можна поєднувати сигнали $R/\overline{W}$, *RESET*# та *DTACK*. На входи $\overline{CS}$ різних ВІС подаються у загальному випадку сигнали з декодера адреси; у простому випадку, коли ВІС *DUART* є лише дві, розряд *A*18, у

протилежних фазах на входи $\overline{CS}$ подається поєднаний з сигналом *CS*681 на логічному елементі ТА.

Виходи двох приймачів *RXDA* та передавачів *TXDA* через підсилювачі-формувачі підмикаються до лінії зв'язку, яка підмикає їх до зовнішніх пристроїв (датчиків, вимірювальних приладів тощо).

На рис. 11.33 наведено схему дешифратора, який формує сигнали підтвердження переривань *IACK*, які надходять на сім пристроїв введення-виведення (*PI/T* та *DUART*).

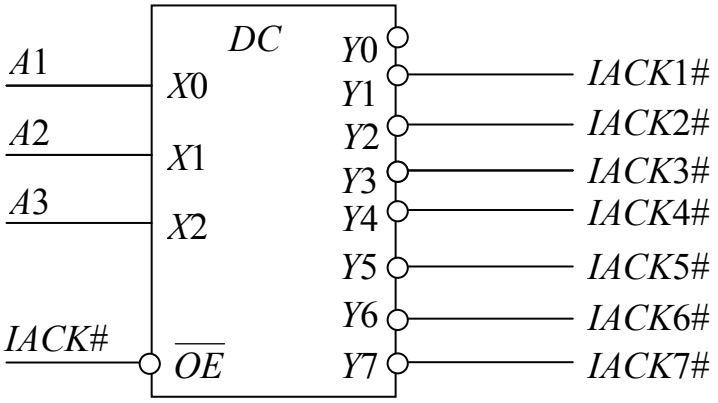
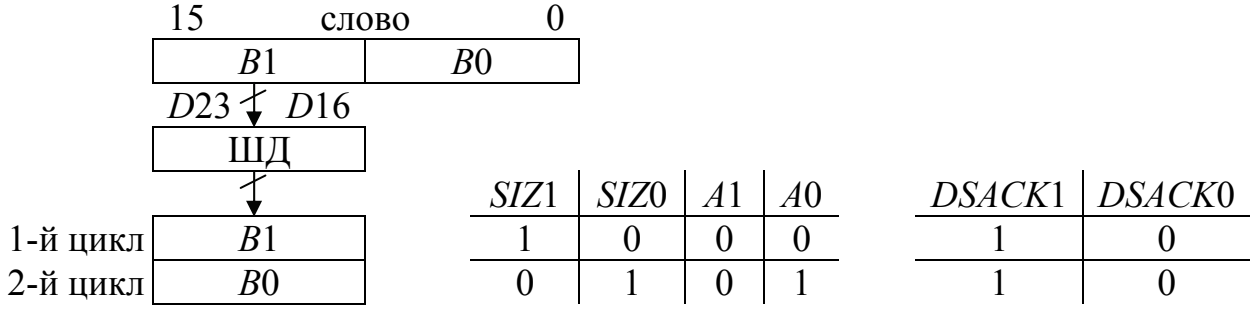



Рисунок 11.33 – Схема формування сигналів *IACK*

На рис. 11.34 наведено передавання слова через *PI/T* за два цикли шини.




| | *SIZ*1 | *SIZ*0 | *A*1 | *A*0 |
|---|---|---|---|---|
| 1-й цикл | 1 | 0 | 0 | 0 |
| 2-й цикл | 0 | 1 | 0 | 1 |

| *DSACK*1 | *DSACK*0 |
|---|---|
| 1 | 0 |
| 1 | 0 |

Рисунок 11.34 – Передавання слова через 8-розрядний інтерфейс

Сигнали *DTACK*# у разі обміну байтами можуть поєднуватись від усіх пристроїв введення-виведення через логічний елемент АБО-НІ і подаватись на входи *DSACK*1 та *AVEC* мікропроцесора.

На вхід *DSACK*0 можна подати рівень логічного нуля для визначення розміру передаваного операнда як байта. Сигнал *BERR*# формувати не треба, через те що усі розряди шини адрес є задіяні. Режим ПДП не є передбачений, тому на вхід *BR* мікропроцесора треба подати рівень *L*1.

**Контрольні питання:**

1 Який адресний простір займають три ВІС *MC*68230?

2 За допомогою якого комбінаційного вузла можна підімкнути до МПС три ВІС М68230?

3 Який адресний простір для кожної ВІС *M*68230 виокремлюється у схемі, яку наведено на рис. 11.31?

4 Який адресний простір виокремлюється для кожної ВІС *MC*68681 у схемі, яку наведено на рис. 11.32?

**Контрольні питання підвищеної складності:**

1 Куди надходять сигнали *DTACK* ВІС *MC*68230 та ВІС *MC*68681?

2 У який спосіб формуються сигнали *DSACK*1 та *DSACK*0, які подаються на входи МП?

### *11.4.5 Підключення співпроцесора*

**Вхідний контроль:**

1 Які функції виконує співпроцесор у складі двопроцесорної системи: центральний процесор – співпроцесор?

2 Які типи співпроцесорів Ви знаєте?

3 Який пристрій – центральний процесор або співпроцесор – є ініціатором їхньої спільної роботи?

Робота співпроцесора здійснюється в циклі, коли *FC*2 = *FC*1 = *FC*0 = 1, або, інакше, у просторі *CPU*.

Мікропроцесор *MC*68*EC*020 припускає підключення співпроцесора обробки чисел з плаваючою точкою типу MC68881; тип процесора визначається кодом, який вміщує поле формату команди *CPID* співпроцесора (рис. 11.35). Мнемоніка команди співпроцесора розпочинається з символу F (1111).

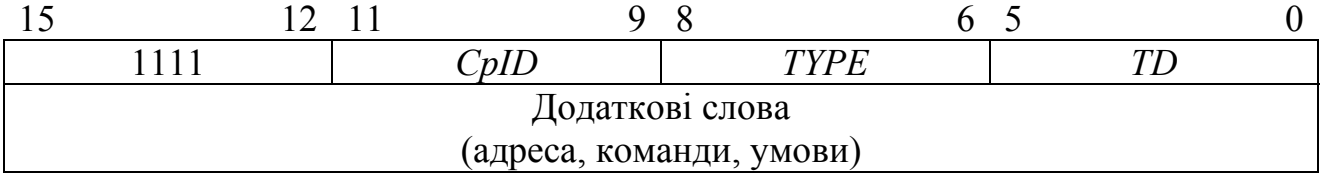



Рисунок 11.35 – Формат команд співпроцесора

Поле *TYPE* визначає тип команди, яку виконує співпроцесор, – код операції, розгалуження за умовою, реалізація циклу тощо. Поле *CD* й решта слів команди вміщують додаткову інформацію: адресу операнда, вибір операції, умови тощо.

Після отримання команди співпроцесора мікропроцесор виконує цикл звернення до співпроцесора: *FC*2, *FC*1 та *FC*0 дорівнюють 1, а на шину адреси

(рис. 11.36) надходить інформація, яка описує стан шини адреси при зверненні до співпроцесора. Вміст розрядів *A*19...*A*13 використовується для адресування операнда зазначеного в команді співпроцесора, а розряди *A*4...*A*0 обирають регістр у складі інтерфейсу співпроцесора, який слугує за джерело або приймач інформації.

| 31 ... 20 | 19 ... 16 | 15 ... 13 | 12 ... 5 | 4 ... 0 |
|---|---|---|---|---|
| 0 ... 0 | 0010 | *CpID* | 0 ... 0 | Регістр |

Рисунок 11.36 – Призначення розрядів шини адреси при зверненні до співпроцесора

При зверненні до співпроцесора мікропроцесор видає сигнали *AS*#, *DS*#, $R/\overline{W}$ , *RESET*# і приймає від нього сигнали підтвердження *DSACK*1#, *DSACK*0#.

При виконуванні команди співпроцесора програмний лічильник мікропроцесора *PC* вміщує адресу коду операції (першого слова команди). На рис. 11.37 подано ВІС математичного співпроцесора *MC*68881. Сигнал вибору співпроцесора *CPCS* формується на ПЛМ або на декодері адреси.

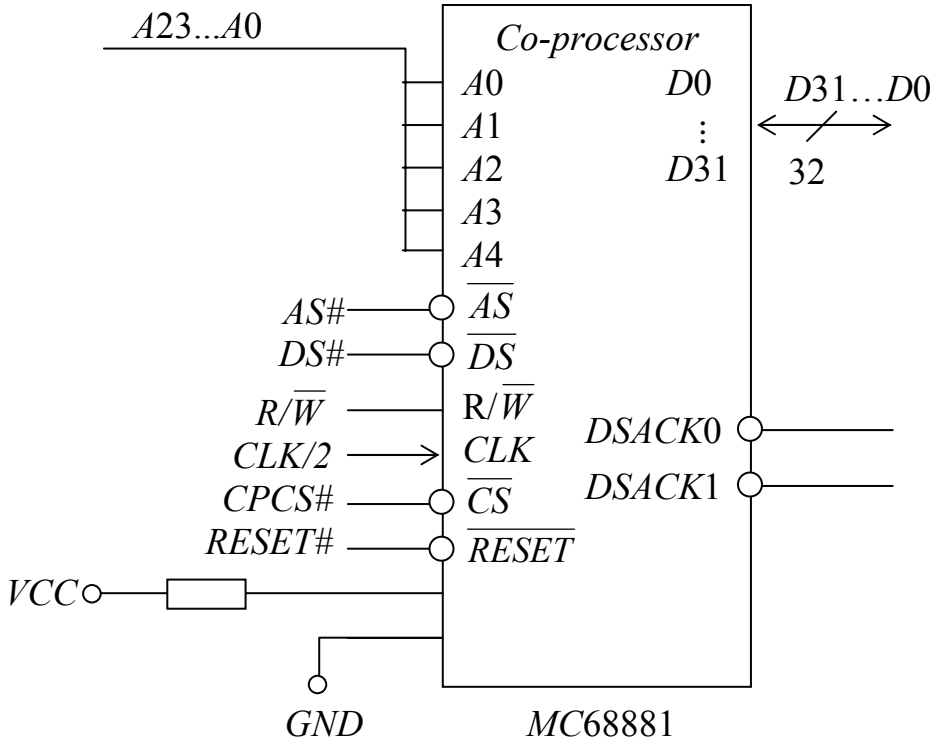



Рисунок 11.37 – ВІС співпроцесора та її підключення

**Контрольні питання:**

1 Які поля входять до складу формату команди співпроцесора й що вони визначають?

2 З якого символу розпочинається кожна команда співпроцесора й чому?

3 Які пристрої можуть формувати сигнал *CS#* вибору мікросхеми співпроцесора?

4 Яку розрядність має шина даних співпроцесора й чому?

**Контрольні питання підвищеної складності:**

1 Яку функцію виконують у МПС вихідні сигнали співпроцесора *DSACK*0# та *DSACK*1#?

2 Які типи співпроцесорів може кодувати поле *CpID*:

а) 000;

б) 001?

# 12 ПРОГРАМУВАННЯ УНІВЕРСАЛЬНИХ МП ФІРМИ *MOTOROLA*

## 12.1 Мова програмування Асемблер МП фірми *Motorola*

**Вхідний контроль:**

1 Який формат мають типові команди мови Асемблер-86?

2 Де у форматі команди пересилань мови Асемблер-86 є джерело, а де приймач?

3 У який спосіб у командах мови Асемблер-86 зазначається розрядність операндів?

4 У яких системах числення може бути подано операнди у командах мови Асемблер-86?

5 Які способи адресування підтримують МП фірми *Intel*?

Мова Асемблер МП сімейства *M*680*X*0 є спільна для МП *MC*68000 як базової моделі та подальших моделей і використовує формат двоадресної типової команди.

Формат типової двоадресної команди подано на рис. 12.1.

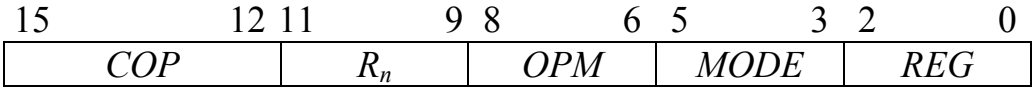



Рисунок 12.1 – Формат типової двоадресної команди

Поле *COP* визначає код виконуваної операції, поле $R_n$ вміщує номер регістра – *n*, в якому зберігається операнд “приймач”, *OPM* – розрядність і розміщення результату; поля *MODE* та *REG* – спосіб адресування і розміщення операнда “джерело”.

Якщо виконується адресування з індексуванням, код команди вміщує друге слово, формат котрого наведено на рис. 12.2.

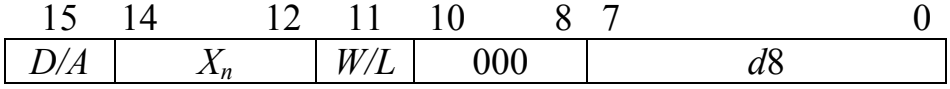



Рисунок 12.2 – Формат другого слова команди при адресуванні з індексуванням

Поле *D/A* визначає тип регістра, який використовується в якості індексного. За *D/A* = 0 в якості індексного використовується один з регістрів даних, а за D/A = 1 – регістр адреси.

Поле $X_n$ вміщує номер регістра, який використовується в якості індексного.

Поле *W/L* визначає розрядність індексу. За *W/L* = 0 в якості індексу використовуються 16 молодших розрядів індексного регістра з поширенням

знаку до 32-х розрядів. За *W/L* = 1 за індекс слугує 32-розрядний вміст індексного регістра.

Поле *d*8 вміщує 8-розрядне зміщення, подане у доповнювальному коді.

Деякі способи адресування потребують доповнення кодів команди, які розглянуто. Приміром, за безпосереднього адресування код команди доповнюється одним чи двома словами, які вміщують безпосередній операнд Im8, Im16 або *Im*32. Якщо при адресуванні операнда “джерело” чи “приймач” використовується 16-розрядне зміщення або абсолютна адреса, яка має розмір “слово” (*Abs.W*) або “подвійне слово” (*Abs.L*), то код команди доповнюється відповідними словами.

У загальному вигляді типова двоадресна команда мови Асемблер МП *MC*68000 має вигляд

*COP.x <src>,<dst>,*

де COP – мнемокод відповідної команди, замість x ставиться символ, який визначає розрядність операндів: *B* – байт, *W* – слово (16 розрядів), *L* – довге слово (32 розряди); якщо символ розрядності є відсутній, то за умовчанням операнд є слово. Операнди умовно позначаються як *<src>* – джерело, *<dst>* – приймач, який слугує за власне приймач результату операції. При записуванні конкретних команд замість *<src>* та *<dst>* зазначаються символічні адреси операндів мовою Асемблера відповідно до способу їхнього адресування. За безпосереднього адресування замість *<src>* зазначається число, перед яким ставиться префікс #.

Операнди, адреси та зміщення у командах можуть подаватись у системах числення, які зазначаються символами:

& – десяткова система,
% – двійкова система,
@ – вісімкова система,
$ – шістнадцяткова система,

символи розміщують перед даними.

За відсутності символу, який зазначає систему числення, дане сприймається як десяткове число.

Мікропроцесор *MC*68000 виконує читання з пам’яті 16-розрядних операндів, так само як їхній запис до пам’яті, навіть коли у команді зазначено в якості операнда байт; у циклі звернення до пам’яті обирається слово, молодший байт якого використовується як операнд. Довге слово обирається за два послідовних цикли шини, причому першими обираються старші 16 розрядів. Отже, адреси команд чи даних, що їх формує МП, мають бути парними.

Мікропроцесори *MC*68000 та старші моделі використовують такі способи адресування операндів:

- регістрове (операнд у регістрі даних чи адреси), наприклад
  MOVE.B D1,D0
  MOVEA.L A2,A3;

− непряме регістрове (операнд у комірці пам’яті, яка є адресована вмістом регістра адреси), наприклад

```
MOVE (A0),D1
```

− непряме регістрове з постінкрементом (операнд у комірці пам’яті, адреса якої розміщено в адресному регістрі і після виконання команди нарощується на 1, 2 чи 4 залежно від довжини зазначеного операнда), наприклад

```
MOVE D3,(A1)+          ; Вміст A1 після виконання команди
                       ; нарощується на 2
```

− непряме регістрове з предекрементом (операнд у комірці пам’яті, адреса якої розміщено в адресному регістрі і перед виконанням команди зменшується на 1, 2 чи 4 залежно від довжини зазначеного операнда), наприклад

```
MOVE.L -(A1),D2        ; Вміст A1 перед виконанням команди
                       ; зменшується на 4
```

− непряме регістрове зі зміщенням (операнд у комірці пам’яті, адреса якої є алгебраична сума вмісту регістра адреси та 16-розрядного зміщення, яке задається у команді), наприклад

```
MOVE.B -$A(A0),D3    ; Для МП MC68000
MOVE.B ($44,A3),D4   ; Для старших моделей
```

− непряме регістрове з індексуванням (операнд у комірці пам’яті, адреса якої є сума вмісту регістра адреси, індексного регістра та зміщення, заданого у команді), наприклад

```
CLR.B $40(A0,D3.W)   ; Для МП MC68000
CLR ($1234,A2,D3.L)  ; Для старших моделей
```

− пряме (операнд у комірці пам’яті, адреса якої задається числом зі знаком, безпосередньо зазначеним у команді), наприклад

```
JMP $1234              ; Для МП MC68000
MOVE.B D0,-$07FF.W     ; Для старших моделей ефективна адреса
                       ; становить $FFF801
```

− відносне (операнд у комірці пам’яті, адреса якої є сума поточного вмісту програмного лічильника *PC* та заданого у команді зміщення); поточний вміст PC, який використовується для обчислення відносної адреси, дорівнює адресі першого слова виконуваної команди плюс 2, наприклад

```
JMP *+$10              ; Для МП MC68000
JMP (*+$10,PC)         ; Для старших моделей
```

− відносне з індексуванням (операнд у комірці пам’яті, адреса якої є сума вмісту *PC*, індексного регістра та зміщення, заданого у команді), наприклад

```
CLR $10(PC,A2.W)       ; Для старших моделей
CLR ($1000,PC,A2.W)    ; Для старших моделей
```

− безпосереднє (значення операнда задано безпосередньо у команді), наприклад

```
MOVEQ #$40,D3          ; Швидке завантаження
MOVEQ #64,D3           ; безпосередньо заданого операнда у D3
```

МП сімейства *M*680*X*0 підтримують усі способи адресування, що і МП *MC*68000, й, окрім того, додаткові типи непрямого регістрового адресування з індексуванням.

**Непряме регістрове адресування з постіндексуванням**

Синтаксис Асемблера має вигляд

*([bd,An],Ri.s*SCALE,od),*

де *bd* – базове зміщення; *Ri.s*SCALE* – значення індексного регістра *An* або *Dn*, яке множиться на масштабний множник *SCALE* – 1, 2 ,4 або 8; *od* – вихідне зміщення; s – символ розрядності індексного регістра, може дорівнювати *W* або *L*. Вміст індексного регістра *Ri.s* трактується як число зі знаком і у разі *s = W* знак поширюється на 32 розряди.

Наприклад, запис *A*3.*W**2 означає, що за індекс у команді слугує вміст 16-ти молодших розрядів регістра адреси *A*3, зсунутий на один розряд ліворуч та розширений знаком до 32-х розрядів.

Команда

*CLR* ([$1234,*A*2],*D*3.*L*,$5678)

очищуватиме комірку пам’яті з ефективною адресою *EA*, яка підраховується в такий спосіб. Припустимо, що вміст регістра *A*2 становить $400600, а регістра *D*3 – відповідно $1000. Частина ефективної адреси *EA*, подана у квадратних дужках, має вигляд

$$\begin{array}{rl} & \$400600 \\ + & \underline{\$001234} \\ & \$401834 \end{array}$$

Припустимо, що за цією адресою у пам’яті зберігається число $600600. Тоді повна ефективна адреса, зазначена у команді, матиме вигляд

*EA* = $600600 + $1000 + $5678 = $606C78.

Внаслідок виконання команди буде обнулено 16 молодших розрядів даного, яке зберігається у пам’яті, розпочинаючи з адреси $606C78.

**Непряме регістрове адресування з преіндексуванням**

Синтаксис Асемблера має вигляд

*([bd,An,Ri.s*SCALE],od)*

Ефективна адреса *EA* підраховується аналогічно до попереднього способу.

Команда

*CLR* ([$1234,*A*2,*D*3.*L*]$5678)

очищуватиме комірку пам'яті з ефективною адресою *EA*, частина якої, подана у квадратних дужках, матиме вигляд

$400600 + $1234 + $1000 = $402834.

Припустимо, що, розпочинаючи з цієї адреси, у пам'яті зберігається довге слово $600600, тоді

*EA* = $600600 + $5678 = $605*C*78.

Внаслідок виконання команди буде обнулено 16 молодших розрядів даного, яке зберігається у пам'яті, розпочинаючи з адреси $605*C*78.

**Непряме відносне адресування з індексуванням**

Синтаксис Асемблера має вигляд

*(*+d,PC,Ri.s*SCALE)*

або

*(xxx,PC,Ri.s*SCALE)*.

За вміст *PC* вважається адреса наступної команди.

За непрямого відносного адресування з постіндексуванням та преіндексуванням ефективна адреса формується як за непрямого регістрового адресування, але замість вмісту регістра An використовується вміст програмного лічильника *PC*:

*EA = [PC + bd] + (Xn) * SCALE + od* – у разі постіндексування
*EA = [PC + (Xn) * SCALE + bd] + od* – у разі преіндексування

Якщо замість *PC* вказати *ZPC*, то можна задати в команді значення програмного лічильника, дорівнюване 0.

Нижче наводяться приклади використовування команди *JMP* з різними способами адресування:

*JMP* ($400610,*PC,A2.W*)
*JMP* (*+$10,*PC,A*2.*W*)
*JMP* ($4,*PC,A*2.*W*)
*JMP* (*+$1000,*PC,A*2.*W*)

*JMP* (\$400610,*ZPC*,*A*2.*W*)

Команди Асемблера *M*П68000 можуть займати від одного до 5 байт, а старших моделей – до 6 байт.

**Контрольні питання:**

1 Наведіть формат типової команди МП *M*680*X*0.

2 Зазначте, де у форматі типової команди є джерело, а де – приймач.

3 У який спосіб у форматі команди зазначається розрядність операндів?

4 У яких системах числення можна подавати операнди, адреси та зміщення у командах мови Асемблера МП фірми *Motorola*?

5 Які способи адресування операндів підтримує МП *MC*68000?

6 Які способи адресування операндів підтримує додатково МП *MC*68020 і старших моделей і з чим це пов'язано?

**Контрольні питання підвищеної складності:**

1 З якою метою, на Ваш погляд, використовується при роботі з масивами непряме регістрове адресування з постіндексуванням?

2 З якою метою, на Ваш погляд, використовується при роботі з масивами непряме регістрове адресування з преіндексуванням?

3 У яких випадках доцільно використовувати кожний із зазначених в пп. 1 та 2 видів адресування?

## 12.2 Система команд МП *MC*680X0 (Для самостійного вивчення)

**Вхідний контроль:**

1 Зазначте, де у команді мови Асемблер-86 *SUB AX,BX* є джерело, а де – приймач.

2 Зазначте, де у команді Асемблера МП *M*680*X*0 *SUB D*0,*D*1 є джерело, а де – приймач.

3 Яку розрядність має операнд у команді мови Асемблер-86 MOV AX,70H?

4 Яку розрядність має операнд у команді мови Асемблера МП *M*680*X*0 *MOVE D*0,*D*1?

5 Чи виставляють команди пересилань мови Асемблер-86 прапорець знаку?

6 Чи виставиться прапорець *ZF* при виконуванні команди *MOV BX*,0000*H*?

### *12.2.1 Команди пересилань*

**Команда *MOVE*** пересилає вміст джерела до приймача, наприклад:

```
MOVE.L #$12345678,$400700.L  ; Запис безпосереднього даного до
                             ; комірки пам'яті з адресою $ 400700
MOVEA A3,A4                  ; Пересилання молодших 16-ти розрядів
```

```
                        ; адресного регістра A3 у A4, старші
                        ; 16 розрядів A4 розширюють знаком
MOVEA.L D0,A0           ; Пересилання вмісту регістра D0 до A0
PEA (A0)                ; Запис вмісту регістра D0 до стека
MOVEA.L #$800015,A0     ; Завантаження адреси додаткового
                        ; регістра PAAR порту A PI/T до A0
MOVEP (0, A0),D0        ; Введення слова з додаткових регістрів
                        ; PAAR та PBAR за адресами $800015 та
                        ; $800017 до D0
MOVEP D0,(-4,A0)        ; Зберігання слова з D0 у регістрах
                        ; даних PADR та PBDR, які мають
                        ; відповідно адреси $800011 та $800013
```

**Команда *MOVEP*** забезпечує пересилання 16- або 32-розрядних операндів через 8-розрядні периферійні пристрої за два або чотири цикли обміну.

```
EXG D1,D3               ; Обмін вмістом регістрів D1 та D3
EXG A4,D2               ; Обмін вмістом регістрів A4 та D2
```

Слід відзначити, що команда пересилання *MOVE* виставляє прапорець $N = 1$ у разі негативного операнда, який пересилається, і скидає всі інші прапорці. Якщо пересилається операнд, який дорівнює 0, встановлюється прапорець $Z = 1$, а інші скидаються. Прапорець $X$ є індиферентний відносно значення операнда, який пересилається.

### *12.2.2 Команди арифметичних операцій*

```
ADD.B (A3),D2           ; Додавання молодшого байта регістра D2
                        ; до вмісту комірки пам’яті, яка є
                        ; адресована A3
ADDA D3,A0              ; Додавання слова з A0 до слова у D3,
                        ; результат записується до A0
ADDI #$1234,(A3)+       ; Додавання слова з комірок пам’яті, які
                        ; мають адреси (A3) та (A3) + 1,
                        ; до числа $1234, після чого до вмісту A3
                        ; додається число 2
ADDQ.L #8,(22,A3)       ; Довге слово з комірок пам’яті, які є
                        ; адресовані вмістом (A3) + 22, (A3) + 23,
                        ; (A3) + 24 та (A3) +25, додається до даного
                        ; 8, результат записується за цими ж
                        ; адресами, швидке додавання
ADDX -(A3),-(A4)        ; Перед додаванням адреси комірок пам’яті
                        ; A3 та A4 зменшуються на 2, після чого
                        ; до вмісту комірки пам’яті, адресованої A4,
```

```
                        ; додається вміст комірки пам'яті,
                        ; адресованої A3, та значення прапорця X
SUB.L #1,D0             ; Віднімання від вмісту 32-розрядного
                        ; регістра D0 одиниці
SUBA.L #$400,A6         ; Віднімання від довгого слова, яке
                        ; вміщується у адресному регістрі A6,
                        ; безпосереднього даного $400
SUBI #250,(A6)+         ; Віднімання від слова, розташованого у
                        ; комірках пам'яті з адресами A6 та
                        ; A6 + 1, десяткового даного 250; після
                        ; операції вміст A6 збільшиться на 2
SUBQ.L #2,A2            ; Швидке віднімання безпосередньо
                        ; заданого операнда, який не може
                        ; перевищувати 3-х розрядів
NEG D0                  ; Віднаходження доповнення до 2 – D0: =
                        ; ( D0 ) + 1, знак даного змінюється
CMP (A4),D7             ; Порівняння вмісту регістра D7 із вмістом
                        ; комірки пам'яті, яка адресується (A4), за
                        ; рахунок внутрішнього віднімання,
                        ; результат нікуди не записується, а
                        ; виставлені прапорці використовуються
                        ; для реалізації розгалужень
CMPM.B (A0)+,(A1)+      ; Порівняння відповідних однобайтових
                        ; елементів масивів, адресованих вмістом
                        ; A0 та A1; при виконуванні команди від
                        ; елементів масиву (A1)+ віднімаються
                        ; елементи масиву (A0)+
```

**Команди додавання та віднімання** встановлюють усі прапорці залежно від здобутого результату, окрім прапорця *X*.

```
MULS (A3),D2            ; Множення зі знаком слова з D2 на слово, яке є
                        ; розташоване у комірках пам'яті, адресованих
                        ; вмістом A3 та A3 + 1; 32-розрядний результат
                        ; розміщується у регістрі D2
MULU (A3),D0            ; Множення без знаку слова з регістра D0 на слово,
                        ; розташоване у пам'яті, адресу якого зазначено у
                        ; регістрі A3; 32-розрядний результат розміщується
                        ; в регістрі D0
DIVS D1,D3              ; Ділення зі знаком довгого слова з D3 на довге
                        ; слово, розміщене у D1; біти результату
                        ; розміщуються у бітах D0...D15 регістра D3, а
                        ; стача – у бітах D16...D31 регістра D3
DIVSL D1,D3             ; Ділення зі знаком довгого слова з D3 на довге
                        ; слово, розміщене у D1; біти результату
```

```
                    ; розміщуються у бітах D0...D31 регістра D3, а
                    ; стача губиться
```

Розширення набору команд 32-розрядних процесорів пов'язано із введенням нових груп команд, які забезпечують операції з бітовими полями, поширюють групу арифметичних та інших операцій, а також команд, які забезпечують роботу зі співпроцесором.

**Команди пересилань** з регістрів *CCR* та *SR* та їхнє завантаження мають вигляд

```
MOVE CCR,D3     ; Пересилання вмісту регістра CCR у D3
MOVE #0,CCR     ; Обнулення регістра CCR
MOVE SR,D0      ; Пересилання вмісту регістра SR у D0
MOVE #$700,SR   ; Встановлення маски пріоритетів переривань, яка
                ; дорівнює 7
```

**Команди обміну** поміж вказівником стека користувача *USP* та адресним регістром мають вигляд

```
MOVE USP,A0     ; Пересилання вказівника стека USP до A0
MOVE A7,USP     ; Задання вказівника стека
```

**Команди множення** 32-розрядних операндів мають вигляд

```
MULS.L (A3),D2      ; Множення зі знаком 32-розрядного даного з D2 на
                    ; 32-розрядне дане, адресоване A3; результат
                    ; записується до D2; якщо він перевищує
                    ; 32 розряди, встановлюється прапорець V = 1
                    ; до CCR
MULS.L (A3),D2:D3   ; Множення зі знаком 32-розрядного даного з D3 на
                    ; 32-розрядне дане, адресоване A3; 64-розрядний
                    ; результат записується до регістрової пари D2:D3,
                    ; старші 32 біти – до D2, молодші – до D3
```

**Команди *MULU* та *MULS***, відповідно беззнакове множення та множення зі знаком, встановлюють такі прапорці:

*N* = 1, якщо результат є від'ємний;
*Z* = 1, якщо результат дорівнює нулю;
*V* = 1, якщо переповнення є;
*C* = 0, завжди;
*X* – не встановлюється.

**Команди ділення** 64-розрядного операнда на 32-розрядний дільник:

```
DIVSL D1,D2:D3     ; Ділення зі знаком 64-розрядного операнда з
                   ; регістрової пари D2:D3 на 32-розрядний операнд
                   ; з D1; 64-розрядний результат записується до
                   ; регістрової пари D2:D3, старші 32 розряди – до D2,
                   ; молодші – до D3
DIVSL.L D1,D2:D3   ; Ділення зі знаком 32-розрядного операнда з D3 на
                   ; 32-розрядний операнд з D1; 32-розрядний
                   ; результат – біти 0...31 – записуються до D3, а
                   ; стача – біти 0...31 – до регістра D2
```

**Команди *DIVU* та *DIVS***, відповідно беззнакове ділення та ділення зі знаком, встановлюють такі прапорці:

*N* = 1, якщо частка є від'ємна, та N є невизначений, якщо є переповнення або ділення на нуль;

*Z* = 1, якщо частка дорівнює нулю;

*V* = 1, у разі переповнення;

*C* = 0, завжди;

*X* – не змінюється.

**Команди бітових операцій** наводяться у табл. 12.1. Якщо операнд розміщено у регістрі даних, він трактується як довге слово, якщо у комірці пам'яті, – як байт. Номер тестованого біта задається вмістом регістра Dn або безпосереднім операндом *Nb*. Значення *Z* = 1 встановлюється, якщо тестований біт $b_n = 0$, і *Z* = 0, якщо $b_n = 1$.

**Команда *BTST*** зберігає значення тестованого біта незмінним, **команда *BSET*** після тестування встановлює значення $b_n = 1$, а **команда *BCLR*** – значення $b_n = 0$. **Команда *BCHG*** інвертує значення біта $b_n$ після тестування.

Зазначені команди можуть виконуватися на МП *MC*68000 та *MC*68020.

У МП68020 введено також нову групу команд, яка виконує операції з бітовими полями, довжиною до 32-х бітів. Для обрання бітового поля <*bf*> у команді зазначається ефективна адреса *EA* операнда, зміщення *Of*, яке визначає номер молодшого (першого) біта в полі <*bf*>, а також розмір поля *Wf* (кількість бітів у полі). При виконуванні кожної з цих команд у регістрі *CCR* встановлюються ознаки *N, Z*, які схарактеризовують вміст зазначеного поля <*bf*>: ознака *N* набирає значення старшого біта поля, ознака *Z* = 1 встановлюється, якщо усі біти поля мають значення 0. Ознаки *C* та *V* обнулюються, *X* не змінюється. Після встановлення ознак **команда *BFCHG*** інвертує значення бітів у полі <*bf*>, **команда *BFCLR*** заповнює поле нулями, **команда *BFSET*** – одиницями. **Команда *BFTST*** тільки встановлює ознаки і не змінює значення бітів. **Команди *BFEXTS*** та ***BFEXTU*** завантажують обране бітове поле до регістра Dn з розширенням його до 32-х розрядів знаком або нулями. **Команда *BFINS*** виконує обернену процедуру і пересилає *Wf*

молодших розрядів з регістра *Dn* до зазначеного бітового поля. **Команда *BFFFO*** визначає *Nb* – номер першого біта у полі <*bf*>, який має значення 1, і заносить цей номер до регістра *Dn*.

Таблиця 12.1 – Команди бітових операцій

| Синтаксис Асемблера | Розрядність | Операції | Адресування |
|---|---|---|---|
| *BTST Dn,<EA>* | *B, L* | $\overline{b}_n \to Z$ | *Dn* – регістрове, операнда – усі види |
| *BTST #Nb, <EA>* | *B, L* | $\overline{b}_n \to Z$ | *Nb* – безпосереднє, операнда – усі види |
| *BSET Dn, <EA>* | *B, L* | $\overline{b}_n \to Z,\ 1 \to b_n$ | *Dn* – регістрове, операнда – усі види |
| *BSET #Nb, <EA>* | *B, L* | $\overline{b}_n \to Z,\ 1 \to b_n$ | *Nb* – безпосереднє, операнда – регістрове, непряме регістрове, відносне, відносне з індексуванням |
| *BCLR Dn, <EA>* | *B, L* | $\overline{b}_n \to Z,\ 0 \to b_n$ | *Dn* – регістрове, операнда – регістрове, непряме регістрове, відносне, відносне з індексуванням |
| *BCLR #Nb, <EA>* | *B, L* | $\overline{b}_n \to Z,\ 0 \to b_n$ | *Nb* – безпосереднє, операнда – регістрове, непряме регістрове, відносне, відносне з індексуванням |
| *BCHG Dn, <EA>* | *B, L* | $\overline{b}_n \to Z,\ \overline{b}_n \to b_n$ | *Dn* – регістрове, операнда – регістрове, непряме регістрове, відносне, відносне з індексуванням |
| *BCHG #Nb, <EA>* | *B, L* | $\overline{b}_n \to Z,$ $\overline{b}_n \to b_n$ | *Nb* – безпосереднє, операнда – регістрове, непряме регістрове, відносне з індексуванням |

Значення зміщення й розміру поля {*Of:Wf*} можуть задаватися у команді безпосередньо числами або зазначатися вмістом регістрів даних як *Of* та *Wf*, наприклад: {*D*1:*D*2} або {*10*:*D*3}. Значення Wf повинні перебувати у діапазоні 0…31, а *Wf* = 0 визначає розмір поля у 32 біти. Значення *Of*, яке задається безпосередньо, також повинне бути у діапазоні 0…31. Якщо для віднайдення Of зазначається регістр даних, то його вміст трактується як число зі знаком.

Приклади команд бітових операцій:

*BFCHG* D1{#6: #3}

Команда інвертує 3 біти числа, яке вміщено до регістра *D*1, розпочинаючи з 6-го біта ліворуч.

```
MOVEA.L     #400700,A4   ; Завантаження адреси до регістра A4
   CLR.L    (A4)         ; Обнулення комірки пам‘яті за EA = (A4)
   MOVEQ    #10,D0       ; Завантаження номера першого
                         ; перетворюваного біта до D0
   MOVEQ    #15,D1       ; Завантаження кількості перетворюваних
                         ; бітів до D1 (F)
   BFCHG    (A4) {D0: D1}; Інвертування бітів
```

До виконання команди BFCHG комірку пам‘яті було обнулено. Після виконання цієї команди у комірці буде число

00 0000 0011 1111 1111 1111 1000 0000 *B*=$003*FFF*00

*BFCLR* ([$400700]){#2: #10}

Команда обнулить поле у 16 бітів довжиною, розпочинаючи зі зміщення 2 біти у довгому слові з адресою $400700.

*BFEXTS* ($400600,*A*0){*D*0: *D*1},*D*2

З комірки пам‘яті за адресою ($400600,*A*0) бітове поле, розпочинаючи зі зміщення, зазначеного у *D*0, і довжиною у кількість бітів, зазначену у *D*1, завантажується до регістра *D*2 і поширює знак до 32-х бітів.

*BFEXTU D*1{#0: #1},*D*2

Команда пересилає з регістра *D*1 один біт, розпочинаючи зі зміщення 0 до регістра D2, і заповнює нулями інші розряди.

*BFINS D*2,*D*3{#2:#$10}

Команда пересилає $10 молодших розрядів з регістра *D*2 у задане бітове поле регістра *D*3, розпочинаючи з другого розряду ліворуч. Якщо регістр *D*2 вміщує число $*FFFFFFFF*, а регістр *D*3 – усі нулі, то після виконання команди регістр *D*2 вміщуватиме число $*FFFFFFFF*, а регістр *D*3 – число $3*FFFC*000.

*BFSET D*1{*D*2:*D*3}

Команда встановлює одиниці у бітах числа, яке вміщується у *D*1, розпочинаючи зі зміщення (*D*2) у кількості (*D*3).

*BFTST D*0{$10:#4}

Команда тестує біти з *b*4 по *b*7 операнда довжиною у слово в регістрі *D*0. Якщо всі біти обнулено, у регістрі *CCR* встановлюється ознака *Z* = 1, якщо *b*7 дорівнює 1, встановлюється ознака *N* у регістрі *CCR*.

### *12.2.3 Команди логічних операцій*

```
AND.B D1,D3            ; Операція логічного ТА над молодшими
                       ; байтами операндів , розташованих у D1
                       ; та D3; результат записується до D3
ANDI #$FBFB,SR         ; Задання маски переривань відповідно до
                       ; рівня 4
ORI.B #4,CCR           ; Встановлення біта Z = 1 у регістрі
                       ; прапорців CCR
EOR.L DO,D2            ; Виконання операції виключного АБО над
                       ; довгими словами, які зберігаються у D2 та
                       ; D0
EORI.B #$55,$400700.L  ; Інвертування парних бітів у байті, який
                       ; зберігається у комірці пам'яті з адресою
                       ; $400700
```

Прапорець *N* у регістрі *CCR* при виконуванні логічних операцій встановлюється залежно від знаку результату. Прапорець *Z* = 1, якщо результат є нульовий . Прапорці *V* та *C* завжди встановлюються такими, що дорівнюють нулю. Прапорець *X* не змінюється.

### *12.2.4 Команди зсувів*

Команди арифметичних та логічних зсувів зреалізовуються за схемами рис. 12.3...12.5:

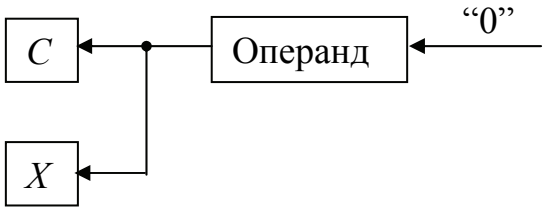



Рисунок 12.3 – Арифметичний та логічний зсуви ліворуч (**команди *ASL, LSL***)

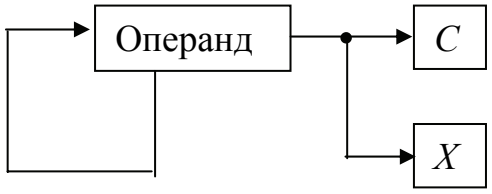



Рисунок 12.4 – Арифметичний зсув праворуч (**команда *ASR***)

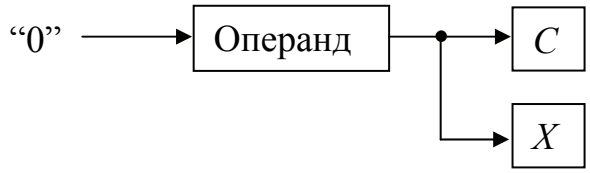



Рисунок 12.5 – Логічний зсув праворуч (**команда *LSR***)

Команди циклічних зсувів зреалізовуються за схемами рис. 12.6...12.9:

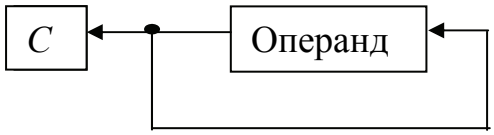



Рисунок 12.6 – Циклічний зсув ліворуч (**команда *ROL***)

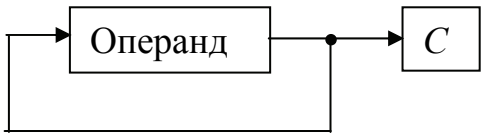



Рисунок 12.7 – Циклічний зсув праворуч (**команда *ROR***)

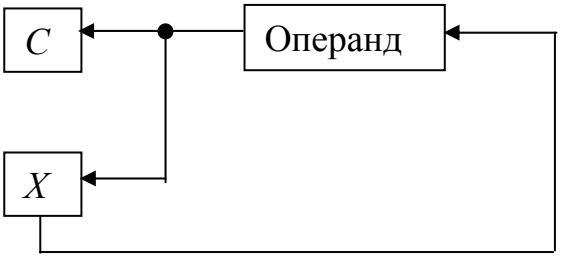



Рисунок 12.8 – Циклічний зсув ліворуч через прапорець *X*
(**команда *ROXL***)

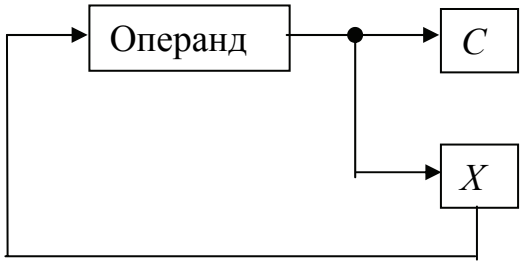



Рисунок 12.9 – Циклічний зсув праворуч через прапорець *X*
(**команда *ROXR***)

Кількість розрядів, на яку зреалізовується зсув, записується або безпосередньо у команді (1...8) або у регістрі даних, лічильнику зсувів. Операнд, який записано у пам’яті, можна зсувати тільки на 1 біт і його розмір має бути не більше за слово.

```
ASR #3,D4          ; Арифметичний зсув молодшого слова у регістрі
                   ; D4 праворуч на 3 розряди
ASL.L D0,D6        ; Зсув ліворуч довгого слова у регістрі D6 на
```

```
                    ; кількість розрядів, зазначену у D0
ASR (-2,A3,D2,L)    ; Зсув праворуч на один розряд слова, яке
                    ; зберігається у комірках пам'яті, ефективну
                    ; адресу першої зазначено за типом "непряме
                    ; регістрове адресування зі зміщенням та
                    ; індексуванням" та наступною
```

За арифметичного та логічного зсувів ліворуч за допомогою команд *ASL* та *LSL* молодші розряди операнда, які звільнюються, заповнюються нулями. За арифметичного зсуву праворуч за допомогою команди *ASR* старші розряди операнда, які звільнюються, заповнюються значенням його старшого (знакового) розряду. Це дозволяє зберегти значення знаку та форми подання операнда (двійковий код або доповняльний код). За логічного зсуву праворуч за командою *LSR* старші розряди операнда заповнюються нулями. При виконанні команди *ASL* значення $V = 1$ встановлюється, якщо відбулося хоча б одне змінення знакового розряду за кількох зсувів. За виконання команди *ASR* старший розряд не змінюється. За виконання усіх видів циклічних зсувів останній розряд операнда, що він висувається, зберігається як прапорець *C*.

### *12.2.5 Команди безумовних переходів*

Команди безумовних переходів (БП) дозволяють перейти у програмі на команду із заданою адресою без запам'ятовування адреси повернення. Команди безумовних переходів зреалізовуються у МП *MC*68000 з коротким та довгим зміщеннями та різними типами адресування. Короткі зміщення становлять $2^8$ адрес, а довгі – $2^{16}$ адрес. **Команда безумовного переходу *JMP*** завантажує до програмного лічильника *PC* ефективну адресу *EA*, яка подається за типами адресування: прямим, непрямим регістровим зі зміщенням, з індексуванням, відносним та відносним з індексуванням. Ця ефективна адреса і є адресою команди, якій передається керування програмою.

**Команда безумовного переходу *BRA*** відрізняється від *JMP* способом формування нового вмісту програмного лічильника. Він формується як сума вмісту програмного лічильника *PC* (адреса першого байта наступної команди програми) та зміщенням *Ds*, яке задається у команді. Величина *Ds*, яка вміщує до 16-ти розрядів, задається числом зі знаком, тому нове значення *PC* може бути більше чи менше за його поточне значення. Якщо вміст *Ds* становить не більше за 8 розрядів (коротке зміщення), то він розміщується у молодшому байті першого слова команди. Якщо ж він становить 16 розрядів (довге зміщення), то задається окремим словом команди. Тому команди з коротким зміщенням займають менший обсяг пам'яті й виконуються швидше, ніж такі ж самі команди з довгим зміщенням.

Нижче наводяться приклади деяких команд безумовних переходів з різним типом адресування для МП *MC*68000.

Команди безумовного переходу

```
$107008          JMP $1234
```

та

$107008 *BRA.L* $00107030

подано відповідно з коротким та довгим адресуванням.

Команди

$107030 *JMP* *+$10

та

$107030 *BRA.S* *+$10

виконуються аналогічно, адреса переходу зазначається відносно першого байта команди, наступної за командою безумовного переходу.

Мова Асемблера МП *MC*68020 дозволяє використовувати команди з коротким зміщенням до $2^{16}$ адрес та довгим – до $2^{32}$ адрес.

Команди

$400620 *JMP* $400634.*L*

та

$400620 *JMP* $500020.*L*

здійснюють відповідно короткий та довгий безумовний переходи.

Аналогічно будуть виконуватись команди

$400606 *BRA.B* *+$10,

$400606 *BRA* *+$10,

$400606 *BRA.L* *+$10

та

$ 400606 *BRA.L* $600000

Команди

*JMP* (*+ $10,*PC*)

та

*JMP* (*+ $100,*PC*)

використовують відносне адресування, а команди

*JMP* ($400610,*PC*,*A*2.*W*)

*JMP* (*+$10,*PC*,*A*2.*W*)

*JMP* ($8,*PC*,*A*2.*W*)

*JMP* (*+$1000,*PC*,*A*2.*W*)

*JMP* ($400610,*ZPC*,*A*2.*W*)

*JMP* ($400610,*A*2.*W*)

використовують непряме регістрове адресування з індексуванням.

### *12.2.6 Команди умовних переходів*

**Команди умовних переходів (розгалужень)** $B_{cc}$ мають 14 варіантів, які відрізнюються умовами. Якщо зазначена в команді умова виконується, то програма переходить на команду, адреса якої формується стосовно вмісту програмного лічильника *PC*, як у **команді *BRA*;** якщо – ні, то виконується наступна за командою умовного переходу команда. Як умови розгалуження використовуються 14 різних значень ознак *N, Z, C, V* та їхніх комбінацій. У багатьох випадках розгалуження програм виконується залежно від результату

порівняння двох операндів за допомогою команди порівняння знакових та беззнакових чисел: ***CMP, CMPA, CMPI, CMPM, TST*** та ***TAS***.

Порівняння операндів відбувається за їхнього віднімання згідно з табл. 12.2, внаслідок чого встановлюються ознаки *N, Z, V, C*. Сам результат не зберігається і значення операндів не змінюються.

Окремі команди дозволяють порівнювати операнд, що адресується *EA*, із вмістом регістра даних (**команда *CMP***), регістра адреси (**команда *CMPA***), безпосереднім операндом (**команда *CMPI***). **Команда *CMPM*** використовується задля порівняння розташованих у пам'яті елементів двох масивів операндів, **команди тестування *TST* та *TAS*** є одноопераднними варіантами команд порівняння.

Таблиця 12.2 – Команди порівняння та тестування

| Синтаксис Асемблера | Розрядність | Операції | Адресування |
|---|---|---|---|
| *CMP <EA>,Dn* | *B, W, L* | *Dn – <scr>* | *<scr>* – усі, *Dn* – регістрове |
| *CMPA <EA>,An* | *W, L* | *An – <scr>* | *<scr>* – усі, *An* – регістрове |
| *CMPI #Im,<EA>* | *B, W, L* | *<dst> – Im* | Im – безпосереднє, *<EA>* – регістрове, непряме регістрове з усіма модифікаціями або пряме, коротке чи довге |
| *CMPM (Ay)+,(Ax)+* | *B, W, L* | *<dst> – <scr>* | Обидва операнди подаються з постінкрементуванням |
| *TST <EA>* | *B, W, L* | *<dst> – 0* | Регістрове або непряме регістрове з усіма модифікаціями |
| *TAS <EA>* | 8 | *<dst>–0;1→b7* | Регістрове або непряме регістрове з усіма модифікаціями |

При виконуванні цих команд встановлюються ознаки *N, Z*, відповідно зі знаком і значенням (дорівнює чи не дорівнює 0) операнда, що адресується *EA*. **Команда *TAS*** після тестування встановлює у 1 старший біт операнда *b*7. Виконання цієї команди не можна перервати запитанням прямого доступу до пам'яті. **Команда *TAS*** використовується у мультипроцесорних системах задля встановлення семафора – спеціального біта, який дозволяє чи забороняє різним МП доступ до окремих блоків спільної пам'яті. **Команда *CMP*2** є властива тільки МП *MC*68020, вона перевіряє перебування операнда у регістрі даних або адреси в регістрі адреси у зазначених в команді межах. Нижня границя *LB* обирається з комірки з адресою *EA*, верхня *UB* – з наступної. Якщо операнд дорівнює *LB* або *UB*, то $Z = 1$, якщо не дорівнює, то $Z = 0$. Якщо операнд перебуває у заданих межах, $C = 0$, якщо виходить, то $C = 1$.

Рівність операндів визначається залежно від значення ознаки *Z*: умови *EQ* та *NE*. Якщо порівнюються числа без знаку, то їхні відносні значення: вище (>),

вище або дорівнює (>=), нижче (<), нижче або дорівнює (<=) – визначаються відповідно з умовами *HI, HS, LO, LS*. Як мнемокоди умов "вище або дорівнює" (HS) та нижче (LO) можна використовувати відповідно *CC* та *CS*.

Якщо порівнюються числа зі знаком, їхні відносні значення визначаються умовами *GT, GE, LT, LE*. Решта умов визначається знаком результату (*PL, MI*) та наявністю або відсутністю переповнення (*VS, VC*). Слід зауважити, що процесор *MC*680x0 не встановлює прапорець парності чи непарності кількості одиниць у результаті, який достатньо широко використовується, наприклад, при перевірці результату в обчислювальній техніці. Тому нижче наводиться підпрограма штучного формування цієї ознаки.

У запис команд умовного переходу мовою Асемблер мнемокод відповідної умови вводиться замість символу cc. Наприклад, *B + EQ = BEQ* – мнемокод команди розгалуження, якщо операнди дорівнюють один одному; *B + MI = BMI* – мнемокод команди розгалуження за від'ємного результату. Види умов, які використовуються у командах розгалужень, наводяться в табл. 12.3.

Таблиця 12.3 – Види умов

| Символи мови | Умова, що перевіряється | Значення ознак |
|---|---|---|
| *NE* | Не дорівнює (ненульовий результат) | $Z = 0$ |
| *EQ* | Дорівнює (нульовий результат) | $Z = 1$ |
| *HI* | Вище | $C + N = 0$ |
| *LS* | Нижче або дорівнює | $C + N = 1$ |
| *HS* (або *CC*) | Вище або дорівнює (перенесення немає) | $C = 0$ |
| *LO* (або *CS*) | Нижче (перенесення є) | $C = 1$ |
| *GE* | Більше або дорівнює | $N \oplus V = 0$ |
| *LT* | Менше | $N \oplus V = 1$ |
| *PL* | Результат додатний | $N = 0$ |
| *MI* | Результат від'ємний | $N = 1$ |
| *GT* | Більше | $Z + (N \oplus V) = 0$ |
| *LE* | Менше або дорівнює | $Z + (N \oplus V) = 1$ |
| *VC* | Переповнення немає | $V = 0$ |
| *VS* | Переповнення є | $V = 1$ |
| *T* | Розгалуження є | 1 |
| *F* | Розгалуження немає | 0 |

Умови *T* та *F* використовуються у складі **команд *DBF*** – безумовне виконання заданої кількості циклів та ***DBT*** – безумовний вихід з циклу.

### *12.2.7 Команди організації програмних циклів*

Для організації циклів використовується **команда $DB_{cc}$**. Зазначений у команді регістр *Dn* є лічильником циклів у цій циклічній програмі. При виконуванні команди $DB_{cc}$ спочатку перевіряється виконання умови, заданої у команді. Якщо умова виконується, то МП обирає наступну команду програми (умовний вихід з циклу). Якщо ж умова не виконується, то вміст регістра *Dn* декрементується. Якщо при цьому вміст регістра *Dn* становить –1, то також обирається наступна команда (цикл завершується). Якщо ж вміст Dn не дорівнює –1, то виконується перехід до команди з адресою (*PC* + *Ds*), яка є початком циклу. **Команда $DB_{cc}$** припускає використовування будь-яких умов, зазначених у табл. 12.3. **Команда організації циклу *DBF*** зреалізовує безумовне виконання заданої кількості циклів, а **команда *DBT*** – безумовний вихід з циклу.

### *12.2.8 Команди звернення до підпрограм*

**Команда виклику підпрограм *JSR*** завантажує до *PC* з комірки пам'яті, адресу якої зазначено в команді, адресу першої команди підпрограми. Перед цим поточний вміст *PC* (адреса наступної команди) запам'ятовується у стеку. Виклик підпрограми виконується також **командою *BSR***, яка використовує відносне адресування, аналогічно до **команди *BRA***. Повернення до програми, яка викликає, після завершення підпрограми здійснюється **командою *RTS***, яка повертає зі стека поточне значення *PC*, чим забезпечує перехід до виконування наступної команди програми. Як правило, перша команда підпрограми завантажує до стека поточне значення регістра SR з метою збереження ознак останнього результату. У цих випадках слід при поверненні з підпрограми використовувати **команду RTR**, яка відновлює поточне значення байта *CCR* (ознаки *X, N, Z, V, C*) та вміст лічильника команд *PC*.

Команди передавання керування наведено у табл. 12.4.

До команд керування також належать команди організації виключень:

***TRAP (TRAP)*** – команда звернення до підпрограми обслуговування виключень. Команда завантажує до стека супервізора поточний вміст регістрів *SR* і *PC*, а потім завантажує до *PC* початкову адресу (вектор) підпрограми обслуговування відповідного виключення, яке відповідає числу $\#D_t = 0...15$, що входить до команди;

***TRAPV (TRAP ON OVERFLOW)*** – команда виконується аналогічно до команди *TRAP* за умови встановлення ознаки переповнювання $V = 1$ і викликає виключення переповнювання;

***ILLEGAL (TAKE ILLEGAL INSTRUCTION TRAP)*** – команда містить відповідне виключення при надходженні помилкового коду команди;

***RTE (RETURN FROM EXCEPTION)*** – повернення з підпрограми обслуговування виключень. Команда є привілейованою і може виконуватися лише в режимі супервізора. При її виконуванні відбувається поновлення вмісту програмного лічильника і регістра *SR* зі стека

Таблиця 12.4 – Команди передавання керування

| Синтаксис Асемблера | Операція | Адресування |
|---|---|---|
| *JMP <EA>* | *<dst> → PC* | Непряме регістрове (усі види), пряме (коротке та довге), відносне, відносне з індексуванням |
| *BRA ds* | *PC + ds → PC* | Відносне |
| *JSR <EA>* | *SP – 4 → SP, PC → (SP), <dst> → PC* | Непряме регістрове (усі види), пряме (коротке та довге), відносне, відносне з індексуванням |
| *BSR ds* | *SP – 4 → SP, PC → (SP), PC + ds → PC* | |
| *RTS* | *(SP) → SP, SP + 4 → SP* | |
| *RTR* | *(SP) → CCR, SP + Z → SP, (SP) → PC, SP + 4 → SP* | |
| *Bcc ds* | Якщо (cc) виконується, то *PC + ds → PC* | |
| *DBcc ds* | Якщо (cc) не виконується, то *Dn*–1→*Dn*; якщо *Dn* ≠ -1, то *PC + ds → PC* | |

і спеціальні команди:

***NOP (NOP OPERATION)*** – команда здійснює перехід до наступної команди без виконання будь-яких операцій;

***STOP (LOAD STATUS REGISTER AND STOP)*** – є привілейованою командою і виконується в режимі супервізора. Завантажує до регістра *SR* слово, зазначене в команді, після чого процесор припиняє роботу;

***RESET (RESET EXTERNAL DEVICES)*** – команда формує сигнал RESET на відповідному виводі МП; використовується для початкового встановлення підсистем МПС;

***CHK (CHECK REGISTER AGAINST BOUNDS)*** – спеціальна команда, яка виконує порівнювання вмісту відповідного регістра з межею, завданою вмістом ефективної адреси; якщо значення виходить за межу, то виконується відповідне переривання;

***LINK (LINK AND ALLOCATE)*** – зменшує значення вказівника стека на 4, завантажує вміст регістра адреси, який зазначено в команді, до стека, завантажує значення вказівника стека до регістра адреси, збільшує значення вказівника стека на 4. Отже, команда завантажує нове значення вказівника стека, зберігаючи у стеку і регістрі адреси дані задля повернення до вихідних значень;

***UNLK (UNLINK)*** – команда скасовує зміни, які відбулися внаслідок виконування команди *LINK*.

Синтаксис команд організації переривань і спеціальних команд подано у табл. 12.5.

Таблиця 12.5 – Перелік команд організації переривань і спеціальних команд

| Синтаксис Асемблера | Операція | Адресування |
|---|---|---|
| *TRAP #$D_t$* | (*SSP*) – 2 → (*SSP*), (*SR*) → *SSP*<br>(*SSP*) – 4 → (*SSP*), (*PC*) → *SSP*;<br>$V_e$(#$D_t$) → *PC* | |
| *TRAPV* | Якщо $V_e$ = 1, то виконується аналогічно до *TRAP*; якщо $V_e$ = 0, то команда не виконується | |
| *ILLEGAL* | Якщо код команди є помилковий, то виконується аналогічно до *TRAP* | |
| *RTE* | (*SP*) → *SR*; *SR* + 2 → (*SP*); *SP* → *PC*;<br>(*SP*) + 4 → *SP* | |
| *NOP* | (*PC*) + 2 → *PC*; | |
| *STOP #$W_S$* | $W_S$ → *SR* і зупин програми | |
| *RESET* | Встановлення початкового стану пристроїв МПС | |
| *CHK <EA>,$D_n$* | Якщо $D_n$ < 0 або $D_n$ > <*EA*>, то виконується аналогічно до *TRAP* | Непряме регістрове (усі види), пряме (коротке та довге), відносне, відносне з індексуванням |
| *LINK $A_n$,$d_S$* | (SP) – 4 → (*SP*); $A_n$ → *SP*; (*SP*) → $A_n$;<br>(*SP*) + $d_S$ → (*SP*) | |
| *UNLK $A_n$* | $A_n$ → *SP*; (*SP*) → $A_n$; (*SP*) + 4 → (*SP*) | |

**Контрольні питання:**

1 Які прапорці виставляє МП фірми *Motorola* при виконуванні команд пересилань?

2 Які прапорці виставляються при виконуванні арифметичних операцій?

3 Які прапорці виставляються при виконуванні логічних операцій?

4 З якою метою використовується команда *CMP <src>,<dst>* і в який спосіб вона виконується?

5 Наведіть приклади виконування команд *ASL* та *LSL* з операндом, який дорівнює $82 на два розряди і поясність результати.

6 З якою метою виконуються команди циклічних зсувів?

7 Поясніть, в який спосіб виконується команда *CMPM* (*A*3)+,(*A*4)+.

8 Де вміщується адреса переходу при виконуванні команди *JMP <EA>*?

9 У який спосіб формується адреса нового вмісту програмного лічильника PC при виконуванні команди *BRA ds*?

10 Які умови можна зазначати у команді умовного переходу $B_{cc}$?

11 У який спосіб перевіряють команди $DB_{cc}$ та *DBF* умову завершення циклу, задану в команді, яка виконує вихід з циклу?

**Контрольні питання підвищеної складності:**

1 Які дії МП *MC*68*xxx* спричинює виконування команди виклику ПП *JSR <EA>*?

2 Які дії МП *MC*68*xxx* спричинює виконування команди повернення з ПП *RTS*?

3 Яка команда перевіряє потрапляння заданого операнда до зазначеного в ній діапазону значень і в який спосіб вона працює?

4 Які команди виконують операції з бітовими полями і в який спосіб вони виконуються?

## 12.3 Побудова програм з різною структурою мовою Асемблер МП фірми *Motorola*

### *12.3.1 Лінійні програми*

**Вхідний контроль:**

1 В який спосіб будуть розміщуватись у пам’яті байти команди мовою Асемблер-86 *MOV AX*,7000*H*, якщо команду розташовано розпочинаючи з адреси 7000:0100?

2 Команда з якою адресою виконуватиметься наступною у лінійній програмі?

3 Яка адреса вміщується у вказівнику команд *IP* МП фірми *Intel* на лінійних ділянках програми?

За приклад побудови лінійної програми розглянемо ділення 16-розрядного числа $5679 зі знаком на 16-розрядне число $0004.

```
MOVE.L #$00005679,D2
MOVE.L #$00000004,D1
DIVS D1,D2
NOP
```

Результатом виконання фрагмента буде частка від ділення (розряди 0...15), яка дорівнює $159*E*, вона буде розміщена у *D*2 (розряди 0...15), і стача $0001 (розряди 16...31), яку буде розміщено також у регістрі *D*2.

Зробимо перевірку правильності здобутого результату за допомогою фрагмента програми

```
EOR.L D3,D3
MOVE.L D2,D3
MULS D1,D2
SWAP D3
ADD D3,D2
```

Результатом виконання фрагмента буде наявність у регістрі *D*2 числа $5679, вміст регістра *D*1 не зміниться.

**Контрольні питання:**

1 Яку частку програм, на Ваш погляд, займають лінійні ділянки?

2 В який спосіб будуть розміщуватись у пам'яті байти команди мовою Асемблер МП *M*680*X*0 *MOVE* #$1234,*D*0, якщо команда розташована розпочинаючи з адреси $400600?

3 Команда з якою адресою буде виконуватись наступною у лінійній програмі?

4 Яка адреса вміщується у лічильнику команд *PC* МП фірми *Motorola* на лінійних ділянках програми?

**Контрольні питання підвищеної складності:**

1 Віднайти добуток даних $1234 та $2 усіма відомими Вам способами: написати фрагменти програми, які їх зреалізовують.

2 Віднайти частку від ділення здобутого у попередньому завданні результату на $2 усіма відомими Вам способами: написати фрагменти програми, які їх зреалізовують.

### *12.3.2 Розгалужені та циклічні програми. Підпрограми*

**Вхідний контроль:**

1 Наведіть приклад застосування арифметичного циклу.

2 Наведіть приклад застосування ітераційного циклу.

3 За яким принципом зреалізовуються підпрограми часової затримки?

*Приклад 12.3.1* Написати фрагмент програми, який здійснював би часову затримку на термін, визначуваний найбільшим числом, котре можна трактувати як байт.

```
400600 MOVE.B #$FF,D6   ; Затримка здійснюється за рахунок
400602 SUB #1,D6        ; повторювання у циклі команди віднімання 1 з
400604 BNE *-2          ; лічильника D6; цикли повторюються, допоки
400608 NOP              ; його вміст не дорівнюватиме нулю
```

Фрагмент програми, який зреалізовує затримку, може бути оформлено у вигляді підпрограми.

*Приклад 12.3.2* Написати фрагмент програми, який виводить слово $1234 до додаткового регістра *PAAR PI/T*, а через час, визначуваний вмістом *D*2 у підпрограмі *TIME*, слово $5678 – до того самого регістра *PAAR PI/T*.

```
        MOVE,B #$1234,D0     ; Завантаження даного $1234 до регістра D0
        MOVE,L #$800015,A0   ; Завантаження адреси регістра
                             ; PAAR PI/T у A0
        MOVEP.B D0,(A0)      ; Запис даного $1234 до регістра PAAR
        JSR TIME             ; Звернення до підпрограми TIME
        MOVE.B #$5678,D1     ; Завантаження даного $5678 до регістра D1
        MOVEP.B D0,(A0)      ; Запис даного до регістра PAAR PI/T
TIME: MOVE #$AB,D2           ; Підпрограма
  M2 : SUB #1,D2             ; TIME
        BNE M2               ;
        RTS                  ;
```

*Приклад 12.3.3* Написати підпрограму визначення парності чи непарності кількості одиниць у байті, який міститься в регістрі.

Задача розв’язується шляхом логічного зсуву байта у циклі, наприклад, праворуч, та підрахування кількості разів, коли встановлювався прапорець перенесення C. Структурну схему алгоритму підраховування кількості одиниць у байті зображено на рис. 12.10.

Програма розв’язання задачі на Асемблері МП *MC*68020:

```
EVEN:  MOVE    SR,D5      ; Завантаження регістра стану до D5
       CLR.L   D2         ; Обнулення D2, лічильника суми
       MOVE.L  #$7,D3     ; Організація лічильника циклів у D3
       MOVE.L  #$09,D0    ; Завантаження байта $9 до регістра D0
M1:    LSR.B   #$1,D0     ; Зсув праворуч регістра D0 на один розряд
       BCS.B   M2         ; Перехід до лічильника суми
       BRA.B   M3         ; Обхід підсумовування
M2:    ADDI    #$1,D2     ; Додавання 1, якщо C = 1
M3:    DBF     D3,M1      ; Організація повторення циклів
       BTST    #$0,D2     ; Перевірка парності кількості
                          ; одиниць суми у D2
       BNE.B   M4         ; Число непарне?
       CLR.L   D2         ; Ні, обнулення D2
       BRA.B   M5         ; Обхід запису числа FDH до D2
M4:    MOVE.L  #$FD,D2    ; Так, запис до D2 числа $FD
M5:    MOVE    D5,SR      ; Відновлення регістра стану з D5
       RTS                ; Повернення з підпрограми
```

Якщо кількість одиниць є непарна, то до регістра *D*2 записується довільна позначка $*FD*, а якщо ж парна, – то регістр *D*2 обнулюється.

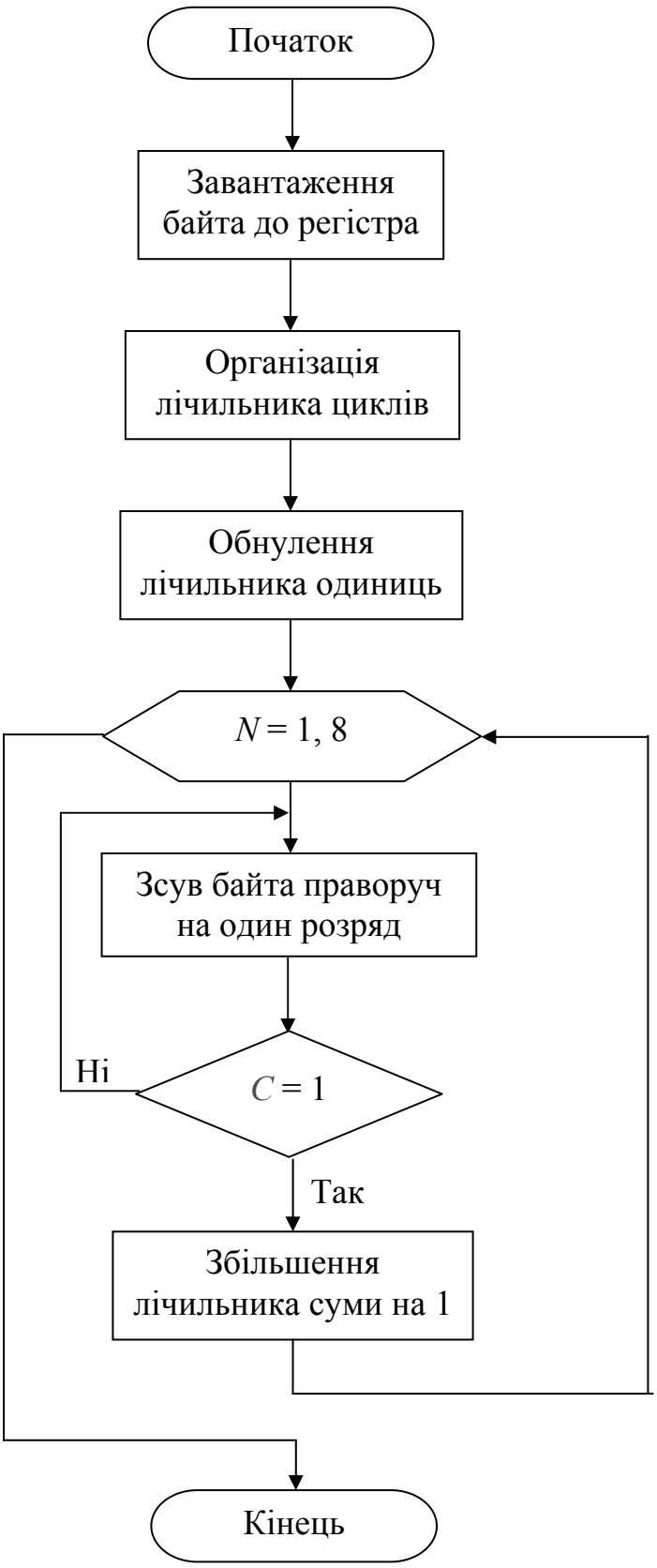



Рисунок 12.10 – Структурна схема алгоритму підраховування кількості одиниць у байті

*Приклад 12.3.4* Порівняти значення байтів у масивах (*A*0) та (*A*1), які складаються з чисел зі знаками у межах шести пар. За умовою *DBGT* зорганізувати вихід з циклу, зазначити, скільки циклів буде виконано.

| | (A0) | (A1) |
|---|---|---|
| 400600 MOVEA.L #$400700,A0 | 5F | 40 |
| 400606 MOVEA.L #$400800,A1 | 3F | F4 (-12) |
| 40060C MOVE #$5,D0 | (-1)FF | F0 (-16) |
| 400610 CMPM.B (A0)+,(A1)+ | 2F | F0 (-16) |
| 400612 DBGT D0,$400610 | (-32)E0 | A3 (-93) |
| 400616 NOP | (-16)F0 | AF (-81) |

Оскільки елементи масивів трактуються як числа зі знаками, то від'ємні числа подано у доповняльному коді (поряд ці самі числа подано зі знаком у десятковій системі числення). Враховуючи особливості виконання команди *CMPM* (*A*0)+, (*A*1)+ (від елемента, який адресується *A*1, віднімається елемент, адресований *A*0), можна свідчити, що вихід з циклу відбудеться за наявності в лічильнику циклів числа -1, за окресленою умовою *DBGT* цикл не завершується в межах порівняння шести пар елементів. Кількість циклів становить 6.

**Контрольні питання:**

1 Напишіть підпрограму визначення парності або непарності кількості одиниць у байті, який міститься у регістрі (приклад 12.3.3) при зсуві вміста цього регістра ліворуч; дайте пояснення здобутим результатам.

2 Поясніть, які команди у підпрограмі *TIME* визначають постійну складову затримки, а які – варіаційну.

**Контрольні питання підвищеної складності:**

1 Визначити час затримки, який зреалізовується фрагментом програми у прикладі 12.3.1.

2 Визначити час затримки, який зреалізовує підпрограма *TIME* у прикладі 12.3.2.

3 Написати підпрограму часової затримки на 2 мкс задля реалізації на МП *MC*68000 і на МП *MC*68020.

## 12.4 Створення програмного забезпечення МПС на МП фірми *Motorola*

**Вхідний контроль:**

1 Які пристрої входять до периферії МПС?

2 У який спосіб периферійні пристрої можуть підмикатись до МПС?

3 У який спосіб задаються режими роботи пристроїв введення-виведення?

4 У який спосіб програмно зреалізовується часова затримка?

МПС, яка розв’язує складні завдання керування або збирання та обробки даних, має розвинену периферію і повинна працювати під операційною системою. МП фірми *Motorola* мають всі необхідні структурні риси – режими супервізора та користувача, механізм підтримки сторінкового обміну даними поміж жорстким диском та оперативною пам’яттю задля побудови таких МПС, у тому числі комп’ютерів. Реалізація переривань, багатозадачного режиму, оброблення виключень виконується в режимі супервізора під керуванням ОС. Коли в якості обчислювального пристрою використовується комп’ютер або багатопроцесорна система, наприклад у цифрових системах комутації, вони водночас можуть розв’язувати завдання статистики, білінгу, підготовки документів тощо. При виборі обчислювального пристрою, в тому числі мікропроцесора, треба мати на увазі, що він визначається типом розв’язуваних завдань, вимогами щодо продуктивності, характеристиками периферійних пристроїв. МПС може бути просторово розподіленою – датчики, вимірювальні прилади, виконувальні та керувальні пристрої, МП (обчислювальний пристрій). Мікропроцесори намагаються розташовувати якомога ближче до периферійних пристроїв, якщо це є технічно можливим, і зв’язок поміж ними здійснюється за допомогою паралельних пристроїв введення-виведення задля досягнення найбільшої швидкості обміну даними. Водночас до МПС можуть входити послідовні адаптери та приймачі-передавачі задля зв’язку з віддаленими периферійними пристроями.

При створюванні програмного забезпечення МПС завжди слід розробляти підпрограми ініціалізації кожного з пристроїв введення-виведення на заданий режим. Головна робоча програма, яка керує МПС, незважаючи на різноманіття розв’язуваних задач, вміщує такі фрагменти:

- циклічний опит датчиків та вимірювальних приладів, перевірка готовності їх до роботи;
- введення даних з датчиків;
- оброблення даних за заданим алгоритмом;
- циклічне виведення результатів на виконувальні або керувальні пристрої;
- відображення проміжних або кінцевих результатів на моніторі чи інших пристроях.

З метою узгодження швидкостей роботи МП та периферії часто виникає потреба затримувати команди або сигнали за допомогою підпрограм затримки.

З метою економії пристроїв введення-виведення периферійні пристрої можуть підмикатись до одного чи кількох портів за допомогою комутаторів. У такому разі треба виводити у циклі головної програми номери або адреси кожного з периферійних пристроїв.

В головній програмі, як правило, є фрагменти порівняння введених параметрів зі стандартними, які зберігаються у пам’яті, результати порівняння визначають подальше виконування програми.

Нижче наведено приклади програм, які зреалізовують спрощений алгоритм роботи МПС різного призначення.

*Приклад 12.4.1* Приклад демонструє можливість реалізації пристрою телекомунікацій на комп'ютері або МПС з МП *M*680*X*0.

Скласти програмні моделі скремблера та дескремблера, які дозволяли б за допомогою генератора випадкових чисел кодувати 255 відліків цифрового сигналу, а потім їх декодувати. Надходження цифрових сигналів з лінії зв'язку у паралельному коді імітується зчитуванням однобайтових елементів масиву з пам'яті, розпочинаючи з адреси $400700. Кодовані скремблером значення відліків записуються до пам'яті, розпочинаючи з адреси $400800, потім декодуються у моделі дескремблера і первинні значення відліків для порівняння запам'ятовуються, розпочинаючи з адреси $400900. Для генерації псевдовипадкової послідовності (ПВП), як правило, використовуються кільцеві регістри зсуву з суматорами за модулем 2 у введених колах зворотного зв'язку. Не кожна конфігурація структури зворотного зв'язку забезпечує максимальний період псевдопослідовності, що дорівнює $2^N - 1$, де $N$ – кількість розрядів регістра. На рис. 12.11 подано одну з максимальних структур з трьома колами зворотного зв'язку, які пов'язують 0, 2, 4, та 7-й розряди 8-розрядного регістра зсуву. Ця структура забезпечує максимальний період генерації числової псевдопослідовності, що дорівнює 255.

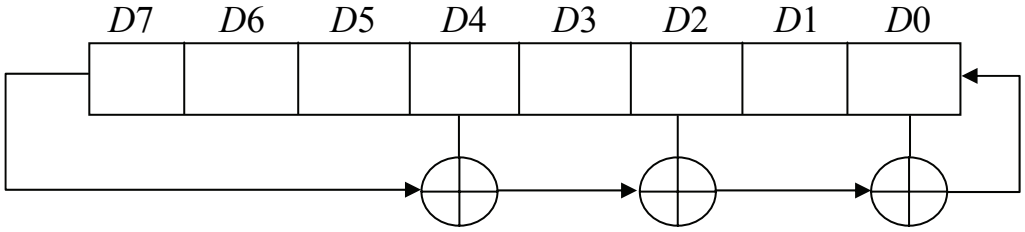



Рисунок 12.11 – Генератор псевдовипадкової послідовності

Задана структура зворотного зв'язку регістра зсуву здійснюється за допомогою маски $95. Для реалізації трьох послідовних операцій складання за модулем 2 прийнятої структури зручно використовувати ознаки парності чи непарності кількості одиниць у байті на кожному кроці зсуву, який зреалізовується програмно за допомогою команди *ROXL.B* – циклічного зсуву ліворуч з розширенням.

Скремблювання та дескремблювання відліків сигналів здійснюється шляхом операції додавання їх за модулем 2 до чергового псевдовипадкового числа. Вхідні та вихідні відліки сигналів при скремблюванні та дескремблюванні записуються до регістра зсуву та зчитуються з нього у паралельному коді.

Структурну схему алгоритму зображено на рис. 12.12.

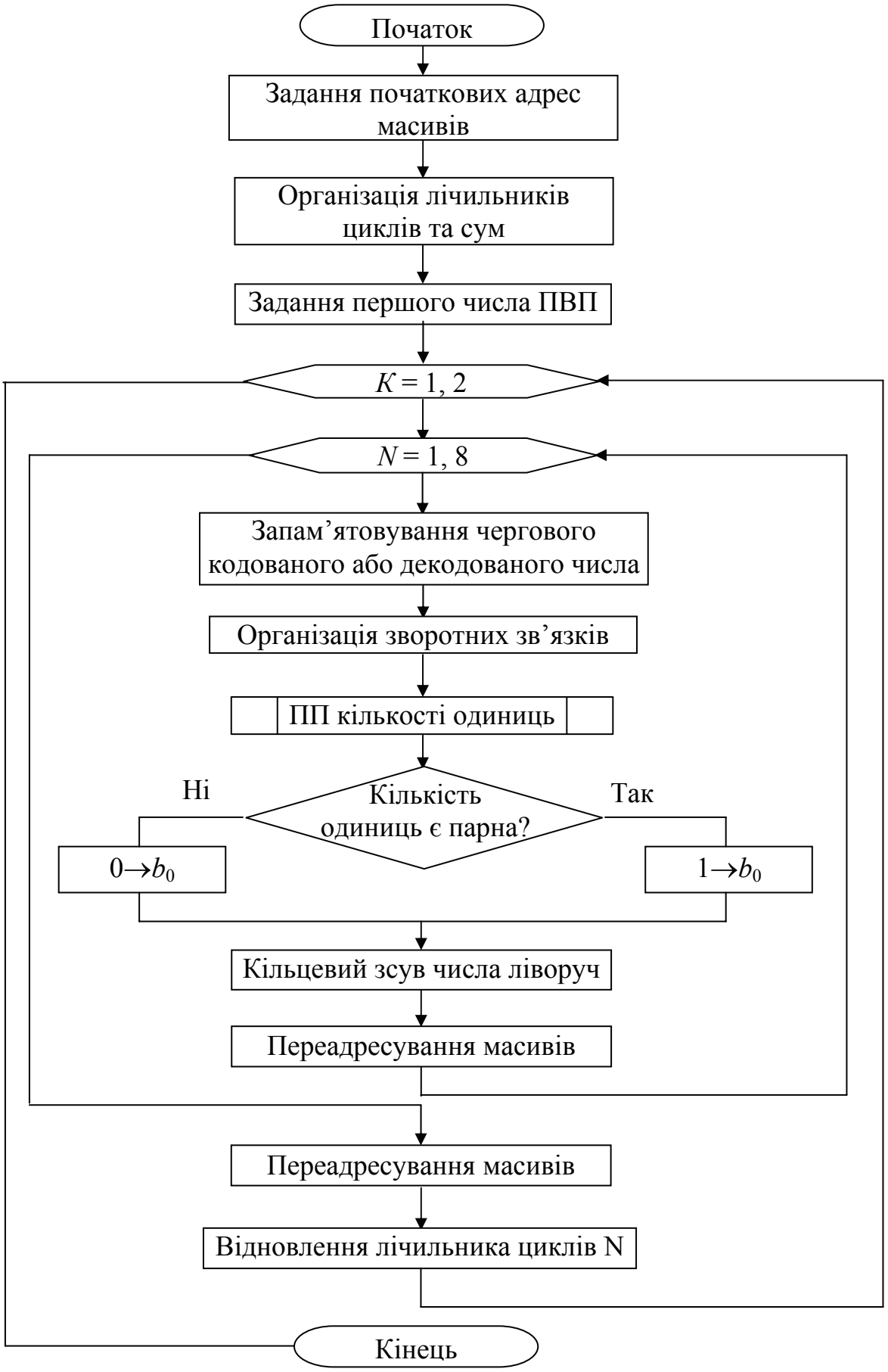



Рисунок 12.12 – Структурна схема алгоритму скремблера-дескремблера

Програма розв’язання завдання:

```
        MOVEA.L  #$400700,A0  ; Завантаження адреси масиву вхідних
                              ; сигналів до A0
        MOVEA.L  #$400800,A1  ; Завантаження адреси масиву кодованих
                              ; сигналів до A1
        MOVE.L   #$FE,D3      ; Завантаження лічильника циклів ПВП
        MOVE.L   #$1,D4       ; Завантаження лічильника циклів
                              ; скремблювання-дескремблювання
M4:     MOVE.L   #$9,D0       ; Завантаження першого числа ПВП
M1:     MOVE.L   D0,D1        ; Запам’ятовування чергового числа ПВП
                              ; у D1
        MOVE.B   (A0)+,D2     ; Скремблювання чергового елемента
        EOR.B    D0,D2        ; масиву вхідних сигналів
        MOVE.B   D2,(A1)+     ; та запам’ятовування його за адресою
                              ; (A1)+
        ANDI.B   #$95,D0      ; Організація зворотних зв’язків
        BSR      EVEN         ; Звернення до ПП визначення парності-
                              ; непарності кількості одиниць у байті
        MOVE     SR,D5        ; Ні, пересилання SR до D5
        BTST     #$0,D2       ; Кількість одиниць є
        BEQ.B    M2           ; парна?
        ANDI.B   #$FD,D5      ; Встановлення X до “0”
        BRA.L    M3           ; Обхід встановлення X до “1”
M2:     ORI.B    #$10,D5      ; Так, встановлення X до “1”
M3:     MOVE     D5,SR        ; Відновлення регістра стану
        MOVE.L   D1,D0        ; Відновлення чергового числа ПВП
        ROXL.B   #1,D0        ; Циклічний зсув D0 ліворуч через X
        DBF      D3,M1        ; Повернення до початку внутрішнього
                              ; циклу
        MOVEA.L  #$400800,A0  ; Переадресування вхідного
        MOVEA.L  #$400900,A1  ; та дескрембльованого масивів сигналів
        MOVE.L   #$FE,D3      ; Перевантаження лічильника циклів
        DBF      D4,M4        ; Повернення до початку зовнішнього
                              ; циклу
        NOP                   ; Завершення програми
```

Слід зауважити, що при зверненні до ПП визначення кількості одиниць у байті значення байта, що міститься у D0, треба передавати з головної програми, а команду MOVE.L #$9,*D*0 вилучити з ПП.

Результатом виконання програми є три масиви: перший, що складається з відліків вхідного сигналу, розпочинаючи з адреси $400700; другий розпочинається з адреси $400800 і складається з кодованих сигналів, а третій,

розташований за адресою $400900, складається з розкодованих відліків вхідного сигналу. Кожний з масивів складається з 255-ти елементів.

*Приклад 12.4.2* Вимірювальна МПС оцінює якість *N* каналів зв'язку за *P* параметрами і розпочинає свою роботу після надходження із зовнішнього пристрою ПВВ1 байта дозволу *B*1. Через пристрій виведення ПВИВ МПС видає номер поточного каналу, а через інтервал часу *T*1 – номер поточного вимірюваного параметра. До шини даних підмикається відповідний вимірювальний прилад, який через пристрій введення ПВВ2 вводить до МПС значення вимірюваного параметра у вигляді байта. Якщо це значення перебуває у заданих межах, МПС переходить до вимірювання наступного параметра, якщо ж ні, – то записує до пам'яті, розпочинаючи з адреси *ADDR*1 номер хибного каналу, номер хибного параметра та його значення, а потім також переходить до вимірювання наступного параметра. Задані межі відхилення кожного параметра зберігаються у пам'яті, розпочинаючи з адреси *ADDR*2. Скласти програму, яка забезпечує роботу МПС за заданим алгоритмом.

Розподіл регістрів:
*D*0 – лічильник кількості каналів
*D*1 – лічильник кількості параметрів
*D*2 – лічильник кількості циклів у підпрограмі затримки *DELAY*
*A*0 – адресний регістр, до якого завантажується *ADDR*1
*A*1 – адресний регістр, до якого завантажується *ADDR*2
*A*2 – адресний регістр, до якого завантажується *ADDR*3 (адреса ПВВ1)
*A*3 – адресний регістр, до якого завантажується *ADDR*4 (адреса ПВИВ)
*A*4 – адресний регістр, до якого завантажується *ADDR*5 (адреса ПВВ2)
*B*1 – байт дозволу роботи МПС
*Bi* – поточний байт, який вводиться з ПВВ2

Перед виконанням програми до пам'яті, розпочинаючи з адреси *ADDR*2, треба послідовно записати стандартні значення кожного з вимірюваних параметрів.

```
        MOVEA.L #$ADDR1,A0      ; Завантаження ADDR1 до A0
        MOVEA.L #$ADDR3,A2      ; Завантаження адреси ПВВ1
        MOVEA.L #$ADDR4,A3      ; Завантаження адреси ПВИВ
        MOVEA.L #$ADDR5,A4      ; Завантаження адреси ПВВ2
MN:     MOVE #$N-1,D0           ; Завантаження кількості каналів N-1
MP:     MOVE #$P-1,D1           ; Завантаження кількості параметрів P-1
M5:     MOVEA.L #$ADDR2-1,A1    ; Завантаження ADDR2-1 до A1
M1:     MOVE.B (A2),D3          ; Введення чергового байта Bi з ПВВ1
        CMP #$B1,D3             ; Порівняння введеного байта з B1
        BNE M1                  ; Повернення на початок циклу
        MOVE.B D0,(A3)          ; Виведення номера каналу до ПВИВ
        JSR DELAY
        MOVE.B D1,(A3)          ; Виведення номера параметра до
```

```
                                ; ПВИВ через час T
        JMP M3
DELAY: MOVE #$K,D2              ; Завантаження до D2 числа K, яке
                                ; визначає час затримки
M2:     SUBQ #1,D2              ; Декрементування регістра D2
        BNE M2                  ; Повернення на дві адреси назад, якщо
                                ; у регістрі D2 не 0
        RTS                     ; Повернення з підпрограми DELAY
M3:     MOVE.B (A4),D3          ; Введення значення вимірюваного
                                ; параметра
        ADDA,L #1,A1            ; Отримання адреси ADDR2+2
        CMP2.B (A1)+,D3         ; Перевірка перебування введеного
                                ; параметра у межах (A1) ≤ D3 ≤ (A1+1)
        BEQ M4
        MOVE.B D0,(A0)+         ; Завантаження до пам'яті номера
                                ; хибного каналу
        MOVE.B D1,(A0)+         ; Завантаження до пам'яті номера
                                ; хибного параметра
        MOVE.B D3,(A0)+         ; Завантаження до пам'яті значення
                                ; хибного параметра
M4:     DBF D1,M3               ; Повернення на початок внутрішнього
                                ; циклу по P, якщо D1 не -1
        MOVE #$P-1,D1           ; Відновлення лічильника циклів D1
        DBF D0,M5               ; Повернення на початок зовнішнього
                                ; циклу по N, якщо D0 не -1
        JMP MN                  ; Циклування програми
```

При налагодженні програми у разі неможливості підключати пристрої введення-виведення треба імітувати надходження байтів параметрів зчитуванням їх з пам'яті як елементів масивів.

*Приклад 12.4.3* Керувальна МПС послідовно перевіряє готовність до роботи *N* пристроїв з адресами *ADDR*1…*ADDRN*. МП видає через пристрій виведення ПВИВ адресу пристрою, який перевіряється на готовність, на комутатор. Комутатор підмикає пристрій до пристрою введення ПВВ, і МП може зчитати байт з виходу пристрою; якщо він дорівнює еталонному байтові $AA, то пристрій є готовий до роботи. Адреси готових до роботи пристроїв запам'ятовуються у послідовних комірках пам'яті, розпочинаючи з адреси *ADDRR* = *ADDRN*+1. Якщо в перебігу першого опитування виявляється, що жоден пристрій не є готовий до роботи, через час T опитування повторюється і, в разі неготовності жодного пристрою, МПС завершує роботу.

```
        MOVE.L #$ADDR1,A0       ; Завантаження адреси 1-го пристрою до A0
        MOVE #$N-1,D0           ; Організація лічильника циклів за
                                ; кількістю пристроїв
```

```
        MOVE #$1,D1             ; Організація лічильника циклів за
                                ; кількістю перевірок
        EOR D3,D3               ; Обнулення регістра D3 – лічильника
                                ; кількості готових пристроїв
        MOVEA.L #$ADDRO,A1      ; Завантаження адреси пристрою
                                ; виведення ПВИВ до A1
        MOVEA. L #$ADDRI,A2     ; Завантаження адреси пристрою ПВВ
                                ; до A2
        MOVEA. L #$ADDRR,A3     ; Завантаження початкової адреси
                                ; масиву адрес готових до роботи
                                ; пристроїв
M1:     MOVEA.L A0,(A1)         ; Виведення адреси і-го пристрою через
                                ; порт виведення ПВИВ
        ADDQ #1,A0              ; Нарощування адреси пристроїв
        MOVE.B (A2),D2          ; Введення байта даних з пристрою
                                ; ПВВ
        CMP $AA,D2              ; Порівняння (D2) з байтом дозволу
                                ; $AA
        BNE M2                  ; Ні, (D2) не дорівнює $AA
        ADDI #1,D3              ; Так, (D2) дорівнює $AA,
                                ; інкрекрементування лічильника D3
        MOVEA. L A0,(A3)+       ; Завантаження адреси готового до
                                ; роботи пристрою
M2:     DBF D0,M1               ; Організація циклу за кількістю
                                ; пристроїв
        CMPI 0,D3               ; Перевірка лічильника D3 на рівність 0
        BEQ M3                  ; Всі прилади не є готові
        JMP M5
M3:     JSR DELAY               ; Звернення до підпрограми DELAY
        MOVEA.L #$ADDR1,A0      ; Відновлення адреси першого пристрою
        MOVEA. L #$ADDRR,A3     ; Відновлення початкової адреси
                                ; масиву адрес готових до роботи
                                ; пристроїв
M4:     DBF D1,M1               ; Завершення циклу за кількістю
                                ; перевірок
M5:     NOP                     ; Завершення програми
DELAY:  MOVE #$K,D4             ; Завантаження до D4 числа K, яке
                                ; визначає тривалість затримки
M6:     SUBQ #1,D4              ; Організація
        BNE M6                  ; програмної затримки
        RTS                     ; Повернення з підпрограми
```

*Приклад 12.4.4* Керувальна МПС обробляє дані, які надходять у циклі від *N* датчиків через пристрій введення ПВВ у вигляді байтів за алгоритмом

$Y = \overline{x_7} \vee x_6 \vee \overline{x_4} \vee x_3 \vee x_2 \vee \overline{x_0}$. Якщо для поточного даного $Y = 1$, то через час $T$ через пристрій виведення ПВИВ МПС виводить керувальний сигнал на відповідний виконавчий механізм і переходить до опитування наступного датчика. Якщо ж $Y = 0$, то МПС відразу переходить до приймання наступного даного.

Задамо номери $N$ датчиків з $N$-2 по -1. Ці номери МПС видаватиме на зовнішній комутатор, який підмикає і датчики, і відповідні виконавчі механізми синхронно до ПВВ та ПВИВ.

Алгоритм оброблення даного, який є заданий у вигляді логічної функції, описано у [8]. Задля його реалізації треба виконати такі дії:

1 Помножити логічно дане на маску *mask*1, яка вміщує 1 у розрядах, відповідним наявним у даному розрядам, і 0 – у розрядах, які є відсутні, вилучаючи їх з розгляду, і здобути результат *Y*1.
2 Зробити операцію виключного АБО над результатом *Y*1 та маскою *mask*2, яка матиме 1 у розрядах, де $x$ є інвертований, і 0 – у всієї решти, і здобути результат *Y*2.
3 Якщо $Y2 \neq 0$, то $Y = 0$, і, навпаки, якщо $Y2 = 0$, $Y = 1$.

Фрагмент програми

```
        MOVEA.L #$ADDR1,A1      ; Задання адреси ПВВ
        MOVEA.L #$, ADDR2,A2    ; Задання адреси ПВИВ
LOOP:   MOVE #$N-1,D0           ; Задання лічильника циклів
M1:     MOVE D0,(A2)            ; Виведення адреси N-го датчика
        MOVE (A1),D1            ; Введення даного Yi
        ANI.B #mask1,D1         ; Виключення розрядів, які є
                                ; відсутні у Yi, знаходженняY1i
        EORI.B #$mask2,D1       ; Знаходження Y2i
        BEQ M3                  ; Yi = 1
M2:     DBF D0,M1               ; Y = 0, звернення за наступним даним
        JMP LOOP                ; Циклування програми
M3:     JSR DELAY               ; Звернення до підпрограми DELAY
        MOVE D1,(A2)            ; Виведення керувального слова
        JMP M2
DELAY:  MOVE #$K,D3             ; Завантаження до D3 числа К, яке
                                ; визначає тривалість затримки
M4:     SUBQ #1,D4              ; Організація
        BNE M4                  ; програмної затримки
        RTS                     ; Повернення з підпрограми
```

Програма виконується у циклічному режимі неперервно.

**Контрольні питання:**

1 Які фрагменти має вміщувати робоча програма, яка керує МПС?

2 Чи завжди МПС працює під керуванням операційної системи?

3 Які пристрої введення-виведення можуть входити до складу МПС?

4 Як називається програмний модуль, призначений керувати пристроями введення-виведення?

**Контрольні питання підвищеної складності:**

1 Напишіть фрагмент програми оброблення даних, які надходять у циклі від датчика через пристрій введення у вигляді байтів за алгоритмом $Y = x_6 \wedge \overline{x_5} \wedge x_3 \wedge \overline{x_1} \wedge x_0$. Якщо для поточного даного $Y = 1$, то через час $T$ через пристрій виведення ПВИВ МПС виводить керувальний сигнал на відповідний виконувальний механізм і переходить до опитування наступного датчика; якщо ж $Y = 0$, то МПС відразу переходить до приймання наступного даного. Кількість датчиків і значення даних задати самостійно. Програма повинна працювати в циклі.

## СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ДО Частини І 2-го МОДУЛЯ


1 Шагурин И. И. Микропроцессоры и микроконтроллеры фирмы *Motorola*: Справ. пособие. – М.: Радио и связь, 1998. – 560 с.: ил.

2 Шагурин И. И. Современные микроконтроллеры и микропроцессоры *Motorola*: Справ. пособие. – М.: Горячая линия – Телеком, 2004. – 952 с.: ил.

3 *MC*68681 *DUAL ASYNCHRONOUS RECEIVER/TRANSMITTER (DUART) (INCLUDES DATA ON MC*2681 *AND MC*2682) – *Motorola Semiconductors. – September*, 1985.

4 *PROGRAMMER'S REFERENCE MANUAL (Includes CPU*32 *Insructions*) *M*68000 *Family Programmer's Reference Manual – ©MOTOROLA INC.*, 1992.

5 *M*68*EC*000 *CPU Module Users Manual. Rev* 3.0 – ©*MOTOROLA*, 1993.

6 *M*68*EC*0*x*0 *Integrated Development Platform Users Manual. Rev* 3.0 – ©*MOTOROLA*, 1993.

7 *THE FLIGHT*-68*EC*020 *EVM. Users Manual by: R.F. Coates. Revised and Edited by: J.H.R. Selby. – Flight Tltctronics LTD., Southamton.*

8 Проектирование импульсных и цифровых радиотехнических систем: Учеб. пособие для радиотехн. спец. вузов / Ю. П. Гришин, Ю. М. Казаринов, В. М. Катипов и др.; Под ред. Ю. М. Казаринова. – М.: Высшая школа, 1985. – 340 с.: ил.

# Частина ІІ МІКРОПРОЦЕСОРНІ СИСТЕМИ НА МІКРОКОНТРОЛЕРАХ, *DSP* ФІРМИ *MOTOROLA* ТА ЇХНЄ ПРОГРАМУВАННЯ

## 13 МІКРОПРОЦЕСОРНІ СИСТЕМИ НА МІКРОКОНТРОЛЕРАХ ФІРМИ *MOTOROLA* ТА ЇХНЄ ПРОГРАМУВАННЯ

**Вхідний контроль:**

1 Які пристрої називають мікроконтролерами?

2 Які пристрої можуть входити до складу мікроконтролера?

3 У складі якого обладнання можуть використовуватися мікроконтролери?

Фірма *Motorola* є провідним виробником мікроконтролерів (МК) у світі. Вона випускає 8-розрядні мікроконтролери, 16-, 32-розрядні модульні комунікаційні мікроконтролери та 32-розрядні *RISC* мікроконтролери *PowerPC*.

8-розрядні МК є найбільш поширеними представниками мікропроцесорної техніки. Їхня вартість не є високою, а доволі широкий спектр функціональних можливостей дозволяє використовувати ці контролери у різноманітних пристроях та приладах. Усі 8-розрядні МК входять до складу трьох сімейств – 68*HC*05, 68*HC*08 та 68*HC*11.

МК *M*68*HC*05 є найбільш прості й, відповідно, такі, що мають обмежені функціональні можливості, але є доволі дешевими. До складу цього сімейства входять різні типи мікроконтролерів, які відрізняються поміж собою обсягом та типом пам’яті, номенклатурою периферійних пристроїв, що їх розміщено на кристалі, а також іншими експлуатаційними характеристиками.

МК сімейства 68*HC*08 мають більшу продуктивність, більший обсяг внутрішньої пам’яті та розширені функціональні можливості, ніж інші МК

На кристалі МК сімейства 68*HC*11 зреалізовано велику номенклатуру периферійних пристроїв, вони також забезпечують можливість розширення обсягу пам’яті за рахунок підключення зовнішніх запам’ятовувальних пристроїв.

Загальна кількість різних моделей 8-розрядних МК, які сьогодні продукуються, становить близько 100.

16-розрядні МК фірми *Motorola* представлено двома сімействами – 68*HC*12/912 та 68*HC*16. МК 68*HC*12/912 мають повну архітектурну сумісність з сімейством 68*HC*11, в них вбудовано внутрішню *Flash*-пам’ять з великим обсягом – до 128 кбайт. Іхньою особливістю є виконування низки операцій, які забезпечують керування об’єктами з використанням “нечіткої логіки”. Вони мають доволі великий набір периферійного обладнання, високу продуктивність і відносно низьке енергоспоживання. МК 68*HC*16 є подальшим розвитком архітектури 8-розрядних мікроконтролерів 68*HC*11 і відрізнюється розширеним набором регістрів, доповненням системи команд і способів адресування. Їхнє використовування, у певних випадках, обмежено потребою підключення зовнішньої пам’яті.

Сімейство 32-розрядних МК *MC*68300 побудовано з використанням 32-розрядного процесора, який доповнено низкою високоефективних периферійних пристроїв, що дозволяє застосовувати їх в системах керування складними об'єктами та процесами при мінімальної кількості додаткових компонентів. Вони використовуються у робототехніці, авіаційно-космічній техніці, у різноманітному телекомунікаційному обладнанні.

Розпочинаючи з 90-х років XX сторіччя фірма *Motorola* також випускає комунікаційні 32-розрядні мікроконтролери, призначені для використовування у складі цифрових телекомунікаційних систем і забезпечують обробку даних та організацію швидкого обміну масивами даних по каналах зв'язку. До них належать мікроконтролери *MC*68302, *MC*68360, *MC*68356 та їхні модифікації.

## 13.1 Типові мікроконтролери фірми *Motorola*

**Вхідний контроль:**

1 Які пристрої входять до програмної моделі мікропроцесора *M*680*x*0?

2 Які відмінності мають фоннеймановська і гарвардська архітектури МП?

3 До якої з архітектур належать МП фірми *Motorola*, які вивчалися в попередніх розділах?

4 Призначення регістра прапорців (ознак результату)?

5 У який спосіб змінюється вміст регістра прапорців?

6 Стекова пам'ять та її використовування у МП.

7 Призначення регістра-лічильника команд.

### *13.1.1 8-розрядні мікроконтролери*

***Сімейство M68HC05/705***

Всі моделі найбільш поширеного сімейства 8-розрядних МК *M*68*HC*05/705 мають однакове процесорне ядро *CPU*05 і відрізнюються за наявністю, обсягом та типом пам'яті, яка входить до їхнього складу, певними характеристиками, а також номенклатурою периферійних пристроїв, розміщених на кристалі. Загальна кількість 8-розрядних контролерів, рекомендованих сьогодні до використовування, становить близько 40 моделей.

Мікроконтролери *M*68*HC*05 та *M*68*HC*705 відрізнюються лише наявністю у складі останнього репрограмованого ПЗП з електричним стиранням – РПЗП-ЕС *(EEPROM*).

Загальну структурну схему МК сімейства *M*68*HC*05/705 наведено на рис. 13.1.

До складу всіх 8-розрядних контролерів входять: процесор *CPU*05, блок програмування, блок контролю функціонування, генератор тактової частоти і блок внутрішнього запам'ятовувального пристрою. Склад паралельних портів і периферійних пристроїв відрізнюється у МК різних типів . Блок внутрішнього запам'ятовувального пристрою складається з оперативного запам'ятовувального пристрою даних, обсягом від 32 до 920 байт (для різних

моделей), постійного запам'ятовувального пристрою програм обсягом до 32 кбайт та службового постійного запам'ятовувального пристрою обсягом 240 байт. До складу деяких моделей входять репрограмовані ПЗП з електричним стиранням – РПЗП-ЕС *(EEPROM*), обсягом до 920 байт.

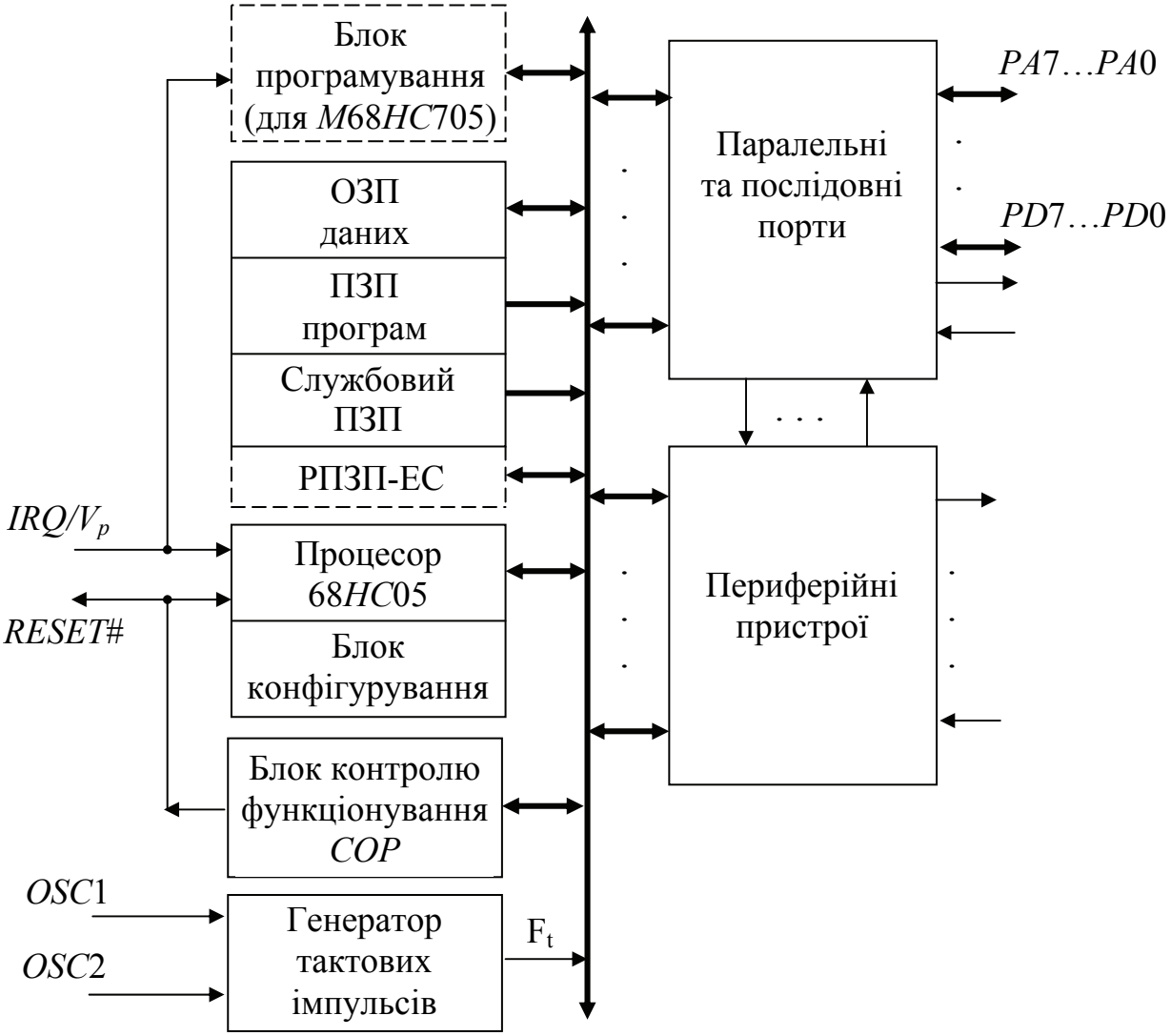



Рисунок 13.1 – Загальна структурна схема МК сімейства *M*68*HC*05/705

**Генератор тактової частоти** формує імпульси, частота котрих визначається кварцовим або керамічним резонатором, RC-ланцюгом або зовнішнім генератором, які підмикаються до виводів *OSC*1, *OSC*2. Частота імпульсів тактової частоти МК для більшості моделей становить $F_t$ = 2,1 МГц за напруги живлення $V_n$ = 5 В і $F_t$ = 1,0 МГц за $V_n$ = 3,3 В. Деякі моделі мають підвищену тактову частоту $F_t$ = 3,0 або 4,0 МГц за $V_n$ = 5 В.

**Блок програмування** використовується задля завантаження інформації до РПЗП-ЕС. Завантаження здійснюється через асинхронний послідовний порт відповідно до програми завантаження, яка зберігається у службовому ПЗП. При цьому здійснюється контроль правильності запису кожного байта.

**Блок конфігурування** використовується задля визначення режиму роботи пристроїв, які входять до складу МК.

**Блок контролю** функціонування забезпечує контроль за виконанням програми за допомогою вартового таймера *WDT* (*Watch-Dog Timer*), а також контроль частоти імпульсів тактової синхронізації.

**Блок паралельних та послідовних портів** у різних моделях МК вміщує від двох до чотирьох 8-розрядних паралельних портів, всі виводи котрих можуть використовуватися задля двоспрямованого обміну. В деяких моделях окремі виводи може бути призначено лише для виведення або введення, а також задля приймання сигналів запитів переривань, входів аналого-цифрового перетворювача, входу таймера, виходів сигналів контролера рідинно-кристалічного індикатора і для організації виводів синхронних та асинхронних портів.

Задля послідовного обміну використовуються асинхронні порти *SCI (Serial Communication Interface*) або *SCI+* і синхронні порти *SPI (Serial Peripheral Interface*) або *SIOP (Simple Synchronous Serial Input-Output Port*) або *SSPI (Simple Serial Peripheral Interface*). Деякі серії мають у своєму складі спеціалізовані послідовні порти, призначені для організації мікроконтролерних мереж. Такі МК широко використовуються у складі пристроїв автоматики, вимірювальної апаратури тощо.

До складу **блока периферійних пристроїв** можуть входити таймери, широтно-імпульсний модулятор, спеціальні вихідні схеми задля підключення пристроїв індикації тощо.

МК сімейства *M*68*HC*05/705 можуть працювати у двох режимах зі зниженим енергоспоживанням: очікування і зупину.

У режимі очікування (*Wait mode*) зупиняється робота процесора, але продовжується робота послідовних портів, таймерів, інших периферійних пристроїв та блока контролю функціонування. Перехід до цього режиму виконується при надходженні команди *WAIT*. Повернення до робочого режиму здійснюється при надходженні зовнішнього запиту переривання, сигналів переривання від таймера, сигналу запуску від блока контролю функціонування і зовнішнього сигналу *RESET*#. Повернення відбувається за один період тактової частоти. Залежно від джерела надходження сигналу повернення до робочого режиму до програмного лічильника *PC* буде завантажено або адресу відповідного вектора переривання, або вектор початкового завантаження. В цьому режимі споживана потужність зменшується в кілька разів.

Перехід до режиму зупину відбувається при надходженні команди *STOP*. В цьому режимі припиняється робота генератора тактових імпульсів, процесора, послідовних портів, таймерів, інших периферійних пристроїв та блока контролю функціонування. В цьому режимі споживаний струм зменшується до одиниць і навіть часток мікроамперів. Вихід з цього режиму відбувається при надходженні зовнішнього запиту на переривання (на вхід *IRQ*# або на вивід МК, який запрограмовано як вхід сигналів переривань) або зовнішнього сигналу *RESET*#. До програмного лічильника *PC* буде завантажена адреса зовнішнього переривання або вектор початкового завантаження. Повернення до робочого режиму відбувається за час, що дорівнює $4064T_t$.

При переході до режиму зниженого енергоспоживання у регістрі ознак *CCR* встановлюється значення ознаки $I = 0$, щоби забезпечити дозвіл переривання і повернення до робочого режиму.

В деяких моделях МК сімейства передбачено режим неповного зупину (*Halt mode*), який відрізнюється від режиму очікування лише тим, що генератор тактових імпульсів відключається від процесора, що зменшує споживану потужність. Задля повернення до робочого режиму в такому разі потрібен більший час, щоби забезпечити встановлення заданих параметрів тактових імпульсів.

**Регістрову модель процесора *CPU*05** подано на рис. 13.2.

Процесор виконує обробку 8-розрядних операндів і зреалізовує 65 команд. До складу регістрової моделі входять:

– 8-розрядний акумулятор, використовуваний задля зберігання одного з операндів та результату виконування арифметичних чи логічних операцій;

– 8-розрядний індексний регістр *X*, використовуваний задля формування адреси при індексному адресуванні. Цей регістр також є джерелом одного з операндів при виконуванні операції множення, а також може використовуватися задля тимчасового зберігання операндів та результатів;

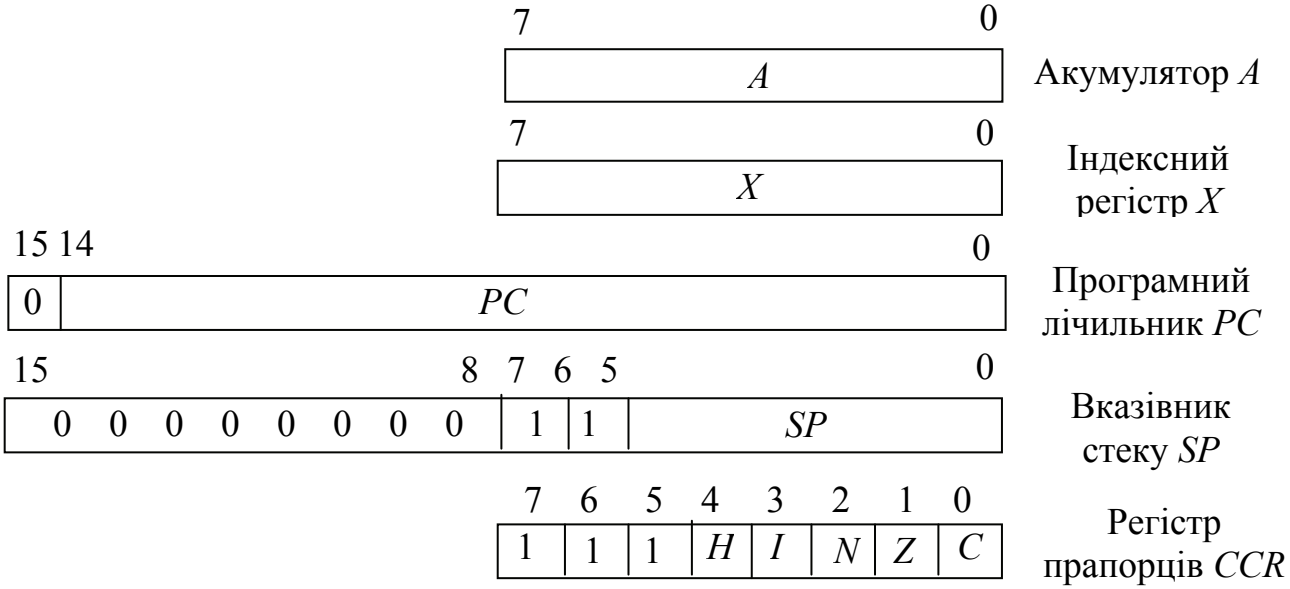



Рисунок 13.2 – Регістрова модель процесора *CPU*05

– 8-розрядний регістр прапорців *CCR*, який зберігає значення п’яти ознак результату, як показано на рис. 13.2:

– *C (Carry/Borrow Flag*)– ознака перенесення (набирає значення $C = 1$, якщо при виконуванні операції відбувається перенесення зі старшого розряду результату);

– *Z (Zero Flag*) – ознака нульового результату ($Z = 1$, якщо результат операції дорівнює нулю);

– *N (Negative Flag*) – ознака знаку результату ($N = 0$, якщо результат додатний, та $N = 1$ за від’ємного результату);

– *I (Interrupt Mask*) – ознака дозволу переривання ($I = 0$ – обробляння переривань дозволено, $I = 0$ – заборонено);

– *H (Half-Carry Flag*) – допоміжне (міжтетрадне) перенесення. Використовується при виконуванні операцій над двійково-десятковими числами;

– 16-розрядний програмний лічильник *PC* вміщує адресу поточної виконуваної команди. Кількість використовуваних розрядів залежить від обсягу внутрішньої пам'яті МК. Максимальний обсяг, що може адресуватися, становить 32 кбайти (розряди 0...14). При включенні МК (виконання команди *RESET*) до програмного лічильника автоматично завантажується адреса першої команди програми (вектор початкового завантаження) з двох останніх комірок пам'яті (\$*FFFE*...\$*FFFF*);

– 16-розрядний вказівник стека *SP*, який використовується для роботи зі стеком. Стек в МК сімейства *M*68*HC*05/705 має обсяг 64 байти і розміщується наприкінці молодших 256 байтів ОЗП даних, тому задля адресування використовується лише 6 молодших розрядів регістра *SP*, куди при запуску МК автоматично завантажується \$00*FF*. Стан старших 10 розрядів вказівника стека при роботі не змінюється.

**Блок конфігурування** призначено задля визначення певних характеристик та режимів роботи МК. Він складається з регістрів конфігурування. Ці регістри в МК підсімейства 68*HC*05 є комірками пам'яті ПЗП, програмованими маскою на етапі виготовлення мікросхеми, тому їхній вміст при експлуатації не змінюється. В МК підсімейства 68*HC*705 ці регістри є комірками одноразово програмованих ПЗП, які програмуються при першому вмиканні мікросхеми і за подальшої роботи не змінюються. Лише при використовуванні МК з РПЗП-ЕС вміст цих комірок можна змінювати. Кількість регістрів і призначення окремих бітів в них зумовлюється типом МК. Так, в МК *MC*68*HC*705*J*1*A*, який досліджується при виконуванні лабораторного практикуму з дисципліни, блок конфігурування представлено одним регістром *MOR*. Формат вмісту цього регістра подано на рис. 13.3.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| *SOSCD* | *EPMCEC* | *OSCRES* | *SWAIT* | *SWPDI* | *PIRQ* | *LEVEL* | *COPEN* |

Рисунок 13.3 – Формат вмісту регістра *MOR*

Біти цього регістра мають таке призначення:

– *SOSCD* – визначає значення затримки вмикання при запуску МК, яка потрібна задля входження до робочого режиму генератора тактових імпульсів. Якщо біт *SOSCD* = 0, то затримка становить 4064$T_t$, а за *SOSCD* = 1 – 128$T_t$. Цей варіант зреалізовується при використовуванні зовнішнього генератора тактових імпульсів;

– *EPMCEC* – біт захисту. Значення біта *EPMCEC* = 1 забороняє читання вмісту РПЗП-ЕС;

– *OSCRES* – при значенні цього біта *OSCRES* = 1 зреалізовується підмикання внутрішнього резистора поміж виводами *OSC*1 та *OSC*2 при

використовуванні високочастотного кварцового резонатора в якості елемента, який визначає часові характеристики;

– *SWAIT* – значення цього біта *SWAIT* = 1 забезпечує переключення МК до режиму неповного зупину при надходженні команди *STOP*;

– *SWPDI* – значення цього біта *SWPDI* = 1 забезпечує підключення резисторів, які "підтягують" їхній потенціал до рівня логічного 0, для виводів портів, запрограмованих на введення даних;

– *PIRQ* – значення цього біта *PIRQ* = 1 забезпечує роботу виводів молодшої тетради порту *A* (*PA*3 – 0) в якості входів для зовнішніх запитів переривання;

– *LEVEL* – визначає вид сигналу зовнішнього переривання. *LEVEL* = 1 відповідає сигналові переривання, котрий має низький рівень (логічний 0), *LEVEL* = 0 визначає сигнал переривання як зріз потенціалу;

– *COPEN* – дозволяє при значенні *COPEN* = 1 роботу блока функціонування.

Найбільш складну організацію має регістр конфігурування МК *MC*68*HC*705*C*8*A*, котрий зреалізовується за допомогою трьох регістрів: *OPTION*, *MOR*1 та *MOR*2.

Регістр *OPTION* входить до складу блока пам'яті МК і має адресу $1*FDF*, регістри *MOR*1 та *MOR*2 входять до складу блока конфігурування. Формат цих регістрів подано на рис. 13.4.

а)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| *RAM*1 | *RAM*0 | 0 | 0 | *SEC* | 0 | *IRQ* | 0 |

б)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| *PBPU*7 | *PBPU*6 | *PBPU*5 | *PBPU*4 | *PBPU*3 | *PBPU*2 | *PBPU*1 | *PBPU*0<br>*COPC* |

в)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | *NCOPE* |

Рисунок 13.4 – Формат вмісту регістрів *OPTION* (а), *MOR*1 (б) та *MOR*2(в)

Регістр *OPTION* визначає тип розділів пам'яті, що конфігуруються, і вид сигналів зовнішнього переривання. Його біти мають такі значення:

– *RAM*0 визначає тип першого розділу пам'яті, що конфігурується, (адреси комірок $0020...$004*F*). Значення *RAM*0 = 0 свідчить, що у цьому розділі розташовано комірки РПЗП-ЕС, а за *RAM*0 = 1 у цьому розділі розташовано комірки ОЗП;

– *RAM*1 визначає тип другого розділу пам'яті, що конфігурується, (адреси комірок $0020...$015*F*). Значення *RAM*1 = 0 свідчить, що у цьому розділі розташовано комірки РПЗП-ЕС, а за *RAM*1 = 1 у цьому розділі розташовано комірки ОЗП;

– *IRQ* визначає вид сигналу зовнішнього переривання, який сприймається як запит переривання. *IRQ* = 1 відповідає сигналові переривання, що має низький рівень (логічний 0), а *IRQ* = 0 схарактеризовує сигнал переривання як зріз імпульсу;

– *SEC* – біт захисту. Значення біта *SEC* = 1 забороняє читання вмісту РПЗП-ЕС.

Регістр *MOR*1 (адреса $1*FF*0) визначає призначення виводів порту *B*: при значенні *PBPUx* = 0 відповідний вивід *PUx* використовується як вхід зовнішніх запитів переривання, за *PBPUx* = 0 цей вивід використовується задля обміну даними. Молодший біт (*PBPU*0 *COPC*) цього регістра також використовується задля скидання вартового таймера у блоці контролю функціонування.

Регістр *MOR*2 (адреса $1*FF*1) використовується в режимі емуляції моделі *MC*68*HC*0512*A* і виконує функції скидання вартового таймера у блоці контролю функціонування цієї моделі.

**Блок контролю функціонування** складається з блоків контролю функціонування *COP (Computer Operating Property*), до складу котрих входить вартовий таймер, а також монітор тактових імпульсів (в деяких моделях).

Вартовий таймер блока контролю функціонування (*WDT*) МК сімейства *M*68*HC*05/705 контролює правильність виконування програми. Правильність виконування перевіряється за результатами запису певних значень до відповідних регістрів. Якщо такий запис не виконується за певний час $T_w$, то блок контролю формує сигнал *RESET*# – і відбувається перезавантаження контролера. Період контролю визначається при конфігуруванні МК.

В МК *MC*68*HC*705*J*1*A* робота *WDT* дозволяється при встановленні біта *COPEN* = 1 до регістра конфігурування *MOR*. Скидання вартового таймера відбувається під час запису нуля до молодшого біта регістра *COPR*, адреса якого є $07*F*0. Період скидання має не перевищувати

$$T_w = 262144\ K_w / F_t,$$

де $K_w$ – коефіцієнт задля задавання можливих інтервалів часу контролю, визначається значенням бітів *RT*1-0 у регістрі керування таймером *MFT*. Визначається за табл. 13.1; $F_t$ – тактова частота МК.

Таблиця 13.1 – Відповідність значень $K_w$ бітам *RT*1-0

| *RT*1-0 | | $K_w$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 2 |
| 1 | 0 | 4 |
| 1 | 1 | 8 |

Така процедура використовується у більшості моделей і має назву – непрограмоване скидання. Вона зреалізовується періодичним завантаженням нуля до молодшого біта регістра *COPR*:

```
LDA #$x0      ; Завантаження до акумулятора числа, яке має значення 0
              ; у молодшому розряді
STA $07F0     ; Перезапис числа з акумулятора до регістра COPR
```

Ці команди слід включити до тексту робочої програми, і вони мають виконуватися не рідше одного разу за час контролю $T_w$.

Режим непрограмованого скидання для МК *MC*68*HC*705*C*8*A* зреалізовується при запису 1 до біта *NCOPE* регістра *MOR*2, а скидання вартового таймера відбувається під час запису 0 до молодшого біта регістра *MOR*1. Задля визначення періоду скидання в цьому разі значення коефіцієнта $K_w = 1$.

В деяких моделях МК, в тому числі *MC*68*HC*705*C*8*A*, можна зреалізовувати програмне скидання замість описаного вище. При цьому для керування роботою вартового таймера використовується вміст двох регістрів у блоці контролю функціонування: регістра керування *COPCR* (адреса $001*E*) і регістра скидання *COPR*. Вміст регістра *COPCR* подано на рис. 13.5.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | *COPF* | *CME* | *COPE* | *CM*1 | *CM*0 |

Рисунок 13.5 – Формат вмісту регістра *COPCR*

Біти регістра *COPCR* мають таке призначення:

– *COPF* – ознака перезапуску блока контролю функціонування; біт набирає значення *COPF* = 1, якщо відбувся перезапуск МК після спрацьовування вартового таймера чи зниження частоти тактових імпульсів (значення цього біта можна лише читати);

– *CME* – значення *CME* = 1 дозволяє роботу схеми контролю частоти тактових імпульсів, протилежне значення – забороняє;

– *COPE* – значення *COPE* = 1 дозволяє контроль за виконанням програми за допомогою вартового таймера, котрий зреалізовує непрограмне скидання;

– *CM*1, *CM*0 – визначають значення коефіцієнта $K_w$ аналогічно до значень *RT*1-0 регістра керування таймером *MFT*.

Читання вмісту регістра *COPCR* призводить до скидання значення біта *COPF* = 0. При вмиканні МК всі біти, окрім *COPF*, встановлюються в 0. Після запуску біти *CME*, *COPE*, *CM*1–0 можуть встановлюватися в 1 лише один раз, а потім лише читатися.

Задля контролю виконування програми в цьому разі встановлюється значення біта *PCOPE* = 1, а до регістра *COPR* (адреса $001*D*) здійснюється періодичний запис чисел $55, потім $*AA*. Часовий інтервал поміж цими записами не повинен перевищувати значення $T_w$, яке здобувається як

$$T_w = 32768\, K_w / F_t,$$

де значення $K_w$ та $F_t$ віднаходяться в наведений вище спосіб.

Якщо за час $T_w$ запису обох чисел не відбувається, то формується сигнал RESET# і здійснюється перезавантаження МК.

Блок контролю функціонування МК *MC*68*HC*705*C*8*A* також вміщує монітор тактових імпульсів, котрий контролює значення частоти генератора тактових імпульсів. Ця схема формує сигнал *RESET#* , якщо за час $T_x$ не надходить тактового імпульсу. Цю схему слід виключати, якщо передбачається робота з низькими тактовими частотами.

**Блок паралельних та послідовних портів** МК *MC*68*HC*705*J*1*A* вміщує лише два паралельних порти *A* та *B* з можливих п’яти, передбачених у складі блока паралельних та послідовних портів сімейства *M*68*HC*05/705.

Вісім виводів порту *A* МК *MC*68*HC*705*J*1*A* може бути запрограмовано на введення-виведення даних у паралельному форматі або чотири з них можуть використовуватися в якості входів для сигналів переривань. Порт *B* має лише шість виводів для організації обміну у паралельному форматі.

Робота портів зорганізовується за допомогою регістрів даних *PORTA* та *PORTB*, котрі мають адреси відповідно \$00 та \$01, а також двох регістрів напрямку пересилання *DDRA* та *DDRB*. Регістри напрямку пересилання визначають режим роботи відповідних виводів, котрі після початкового завантаження запрограмовані на приймання даних (в регістрах *DDRx* записано 0 у всіх розрядах). Для використання виводу в якості виходу даних, відповідні розряди регістрів напрямку слід встановити у стан 1.

Встановлення біта *SWPDI* = 1 дозволяє підключення (”підтягування”) входів портів до рівня логічного нуля задля виключення завад, які можуть виникати у вхідних колах МК, при переході виходів джерела сигналів до високоімпедансного стану.

Виходи портів забезпечують значення струмів навантаження $I_{н0}$ = 1,6 мА при формуванні сигналу логічного 0 і $I_{н1}$ = 0,8 мА – при формуванні сигналу логічної 1.

У складі МК *MC*68*HC*705*J*1*A* блока послідовних портів немає.

У складі інших МК сімейства *M*68*HC*05/705 він складається з таких пристроїв.

**Асинхронний послідовний порт (*SCI*),** який входить до складу МК типу MC68HC705C8A, забезпечує стандартний асинхронний формат приймання-передавання даних у вигляді кадру, який вміщує один стартовий і один стоповий біти, вісім інформаційних бітів (*D*0...*D*7) і, за потреби, може доповнюватися одним додатковим контрольним бітом (*D*8*). Формат кадру подано на рис. 13.6.

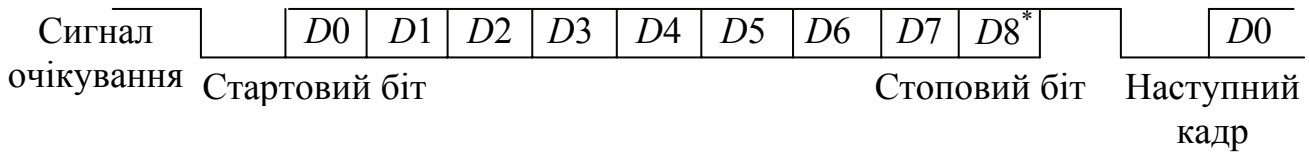



Рисунок 13.6 – Формат кадру даних порту *SCI*

Обмін даними здійснюється через буферний регістр *SCDR*, вхід і вихід котрого з'єднуються з відповідними виводами паралельного порту *D*, які використовуються для послідовного обміну (*PD*0 та *PD*1).

Перед передаванням даних на лінії встановлюється сигнал очікування (сигнал логічної 1) з часом передавання не менше за період тактової синхронізації $T_s = 1/F_s$. Відсутність цього сигналу упродовж часу понад $(10...11)T_s$ сприймається як сигнал завершення обміну – *BREAK*.

Якщо кадр вміщує в усіх інформаційних бітах, а також у контрольному біті сигнал, що дорівнює 1, то такий кадр сприймається як сигнал завершення передавання – *BREAK*, а в разі передавання сигналу логічного 0 – як сигнал очікування наступного циклу передавання даних – *IDLE*.

Керування обміном даними через порт *SCI* здійснюється відповідно до вмісту 8-розрядних регістрів керування *SCCR*1 (адреса \$0*E*) і *SCCR*2 (адреса \$0*F*), регістра стану *SCSR* (адреса \$10) і регістра *BRR* (адреса \$0*D*), формат вмісту котрих подано на рис. 13.7.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| а) | *R*8 | *T*8 | – | *M* | *WAKE* | – | – | – |
| б) | *TIE* | *TCIE* | *RIE* | *TLIE* | *TE* | *RE* | *RWU* | *SBK* |
| в) | *TDRE* | *TC* | *RDFR* | *IDLE* | *OR* | *NF* | *RE* | – |
| г) | – | – | *SCP*1 | *SCP*0 | – | *SCR*2 | *SCR*1 | *SCR*0 |

Рисунок 13.7 – Формат вмісту регістрів *SCCR*1 (а), *SCCR*2 (б), *SCSR* (в), *BRR* (г)

Регістр *SCCR*1 (рис. 13.7, а) вміщує такі біти:

– *WAKE* – сигнал активізації приймача;

Значення *WAKE* = 0 активізує приймач при надходженні на вхід даних *RDI* символа *IDLE*, а значення *WAKE* = 1 – якщо старший (адресний) розряд даних, що надійшли на вхід, має одиничне значення;

– *M* – сигнал, що визначає розрядність даних;

*M* = 0 визначає, що дані мають розрядність 1 байт (контрольного біта немає), а значення *M* = 1 відповідає даним, що мають довжину 9 розрядів (дані вміщують контрольний біт);

– *R*8 – розряд призначено для запису значення біта контролю до прийнятого сигналу;

– *T*8 – біт, значення котрого записується у кадр і передається як біт контролю.

Біти регістра *SCCR*2 (рис. 13.7, б) мають таке призначення:

– *TIE* – біт дозволу формування запиту контролю переривання; значення *TIE* = 1 дозволяє формування сигналу при встановленні в регістрі SCSR ознаки звільнення буферного регістра при передаванні даних – *TDRE* = 1;

– *TCIE* – значення *TCIE* = 1 дозволяє формування запиту контролю переривання при встановленні в регістрі *SCSR* ознаки завершення передавання даних – *TC* = 1;

– *RIE* – значення *RIE* = 1 дозволяє формування запиту контролю переривання при встановленні в регістрі *SCSR* ознаки заповнення буфера або переповнення приймача – *RDRF* = 1 або *OR* = 1;

– *ILIE* – біт дозволу формування запиту контролю переривання (*ILIE* = 1) при надходженні на вхід даних *RDI* сигналу завершення обміну (встановлення біта *IDLE* = 1 в регістрі *SCSR*);

– TE – біт включення передавача (*TE* = 1), а також його відключення (*TE* = 0);

– RE – біт включення приймача (*RE* = 1), а також його відключення (*RE* = 0);

– *RWU* – біт керування приймачем; значення *RWU* = 1 переводить приймач до режиму очікування, а також дозволяє (відповідно до значення *WAKE*) активізацію приймача;

– *SBK* – біт завершення обміну; при значенні *SBK* = 1 формується і передається на вихід передавача символ завершення обміну *BREAK* (послідовність 0).

Перед передаванням даних, записаних до регістра *SCDR*, біт *SBK* встановлюється в 1, потім це значення скидається і передавач передає символ завершення обміну (10 або 11 нулів) – встановлює на виході рівень 1 упродовж одного періоду сигналу синхронізації; після цього розпочинається передавання кадру даних.

Регістр стану *SCSR* (рис. 13.7, в) вміщує ознаки, формовані в перебігу передавання чи приймання даних. Для користувача він є доступний лише для читання. Біти регістра стану мають таке призначення:

– *TDRE* – ознака звільнення буфера передавача; цей біт набирає значення *TDRE* = 1 після завершення перезапису вмісту регістра *SCDR* до вихідного зсувового регістра для подальшого передавання лінією зв’язку;

– *TC* – ознака завершення передавання; набирає значення *TC* = 1 після передавання останнього біта даних з вихідного зсувного регістра;

– *RDRF* – ознака заповнення буфера приймача; значення *RDRF* = 1 встановлюється після завершення перевантаження даних з вхідного зсувного регістра до буферного регістра;

– *IDLE* – ознака надходження сигналу очікування; якщо на вході даних сигнал дорівнює 1 упродовж 10...11 тактів сигналу синхронізації, то біт набирає значення *IDLE* = 1;

– *OR* – ознака переповнювання приймача. Встановлюється значення *OR* = 1, якщо поточний символ надходить до зсувного регістра приймача до

зчитування з регістра *SCDR* попереднього. В такому разі вміст SCDR зберігається, а символ, що надійшов, втрачається;

– *NF* – ознака наявності шумів на лінії приймання. Встановлюється значення *NF* = 1, в разі зміни стану входу до завершення приймання поточного біта;

– *FE* – ознака порушення кадру. Значення *FE* = 1 встановлюється, якщо тривалість стопового біта є меншою за період сигналу синхронізації $T_s$.

Ознаки запитів переривань *TDRE*, *TC*, *RDRF*, *IDLE*, *OR* формуються, якщо це дозволено відповідними бітами регістра *SCCR*2. Програми обслуговування цих переривань повинні зчитати і проаналізувати вміст регістра SCSR задля правильної обробки переривання. Ознаки *NF* та *FE* не формують запитів на переривання, читаються і аналізуються за результатами програмного опитування.

Значення ознак *TDRE* і *TC* скидаються в 0, якщо при передаванні даних читання вмісту *SCSR* виконується після запису даних до регістра *SCDR*, а значення ознак *RDRF*, *IDLE* та *OR* скидаються в 0, якщо при прийманні даних читання вмісту *SCSR* виконується після зчитування даних з регістра *SCDR*.

Вміст регістра *BRR* (рис. 13.7, г) зумовлює швидкість обміну даними встановленням значення тактової частоти $F_s$ відповідно до виразу $F_s = F_t / (K_d \cdot K_s)$. Значення коефіцієнтів $K_d$ та $K_s$ обираються встановленням відповідних значень бітів регістра *BRR*. Значення $K_d$ зумовлюється значенням бітів *SCP*1-0, а коефіцієнта $K_s$ – *SCP*2-0. Це надає змогу зреалізовувати великий діапазон значень швидкостей обміну від 10 біт/с до 100 кбіт/с.

В деяких моделях МК сімейства використовується модифікований асинхронно-синхронний порт *SCI+*. Використовування такого порту надає змогу визначати швидкість обміну окремо для лінії передавання і приймання даних, а також забезпечує можливість синхронного передавання даних. Для забезпечення синхронного передавання даних порт має окремий вихід синхросигналу, який не суміщено з виходами паралельних портів.

**Синхронний послідовний порт (*SPI*),** який входить до складу порту *D* МК *MC*68*HC*705*C*8*A*, складається з регістра даних *SPDR* (адреса \$0*C*), регістра керування *SPCR* (адреса \$0*A*) і регістра стану *SPSR* (адреса \$0*B*).

Обмін здійснюється 8-розрядними даними, які зчитуються з регістра даних SPDR або записуються до нього.

Керування процесом обміну здійснюється відповідно до стану бітів регістра *SPCR*, формат вмісту якого подано на рис. 13.8, а схемна реалізація організації послідовного обміну поміж двома абонентами через синхронний послідовний порт *SPI* – на рис. 13.9.

Вивід *SCK* (рис. 13.9) призначено для передавання сигналу синхронізації. На боці ведучого *SPI* цей вивід використовується як вихід сигналу, а на боці веденого – як вхід. Лінія *MOSI* використовується для передавання інформації від ведучого до веденого *SPI*. Лінією *MISO* здійснюється передавання інформації від веденого *SPI* до ведучого. Виводи *SS#* використовуються для вибору режиму функціонування; рівень 1, який надходить на цей вивід, визначає відповідний *SPI* як ведучий, а рівень 0 – як ведений.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| *SPIE* | *SPE* | – | *MSTR* | *CPOL* | *CPHA* | *SPR*1 | *SPR*0 |

Рисунок 13.8 – Формат вмісту регістра керування *SPCR*

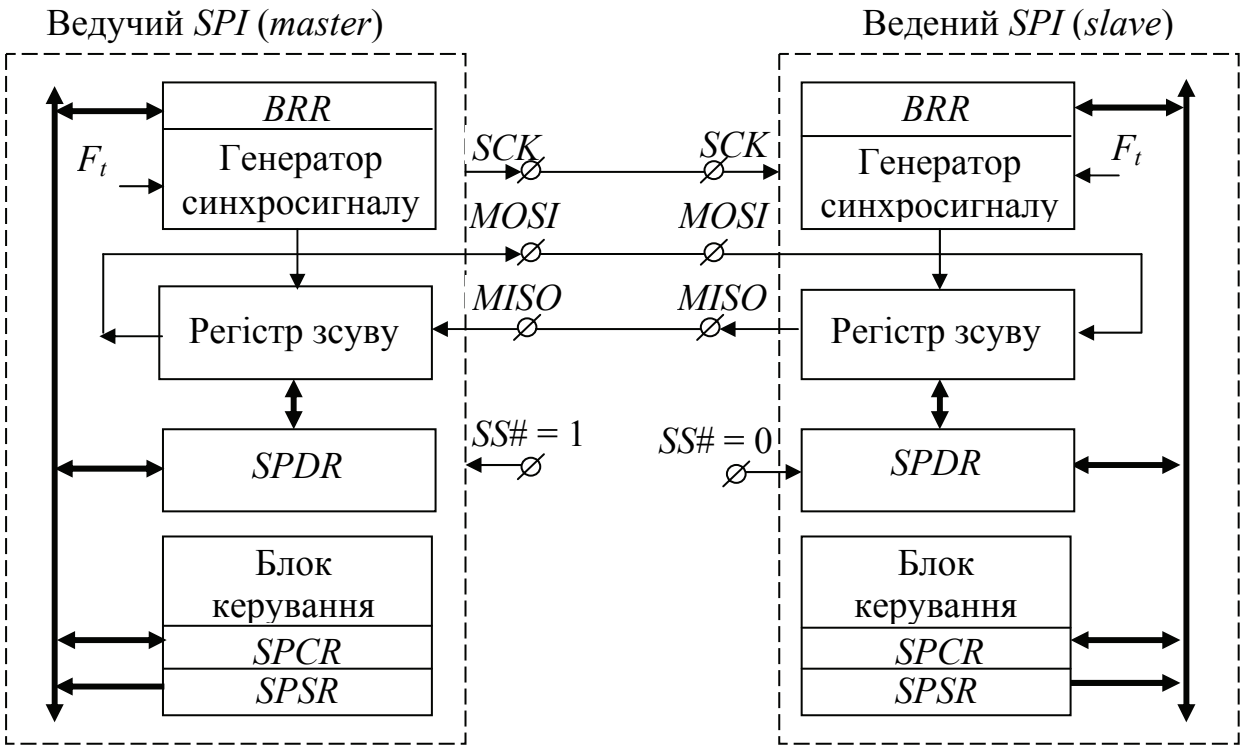



Рисунок 13.9 – Організація синхронного обміну даними через порт *SPI*

Обмін проводиться по лініях *MOSI* або *MISO* і в кожному такті здійснюється запис одного біта до молодшого розряду зсувного регістра і вивід одного біта зі старшого. Обмін виконується байтами, які зчитуються або записуються до регістра *SPDR*.

Обмін здійснюється відповідно до коду, який записано у регістрі *SPCR* (див. рис. 13.8):

– значення *SPE* = 1 призводить до включення порту і використання його виводів (див. рис. 13.9);

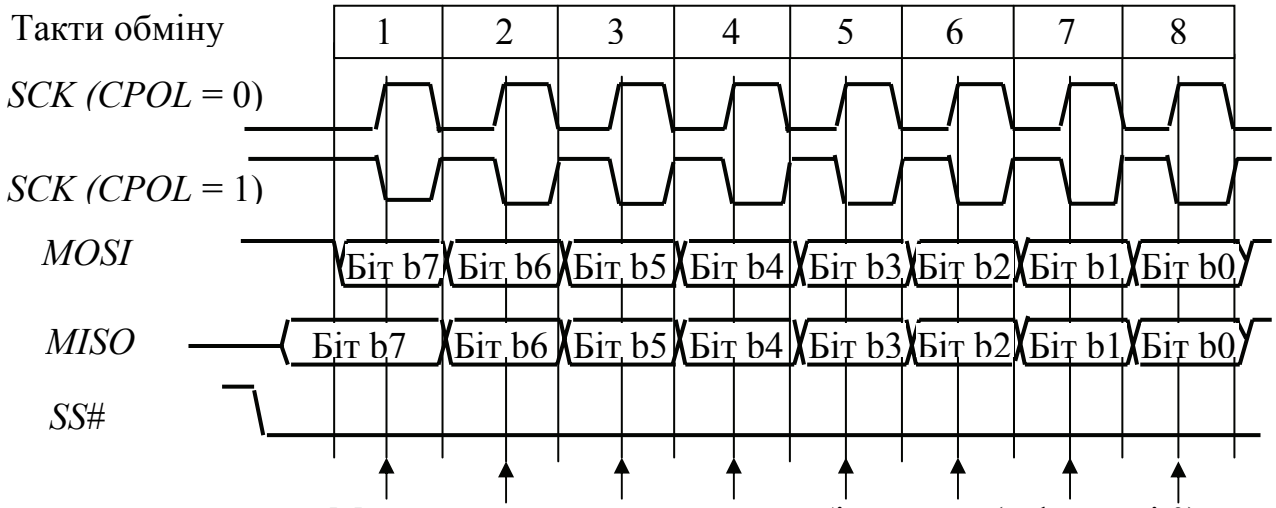
– значення біта *MSTR* визначає порт для програмування в якості веденого або ведучого; *MSTR* = 1 визначає відповідний порт як ведучий, а значення 0 – як ведений;

– значення бітів *SPR1*-0 ведучого *SPI* зумовлює швидкість обміну; значення цих бітів задають значення коефіцієнта ділення $K_d$ у виразі $F_s = F_t/K_d$;

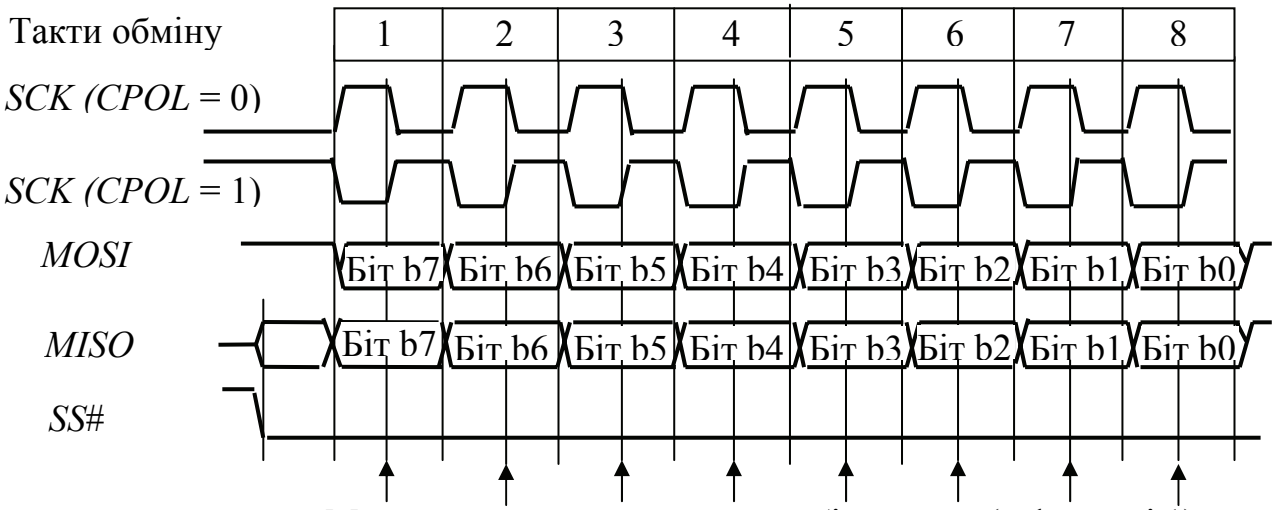
– біт *CPOL* визначає полярність синхросигналу. Значення *CPOL* = 0 підтримує на лінії *SCK* незмінний рівень 0 до початку циклу обміну, а значення *CPOL* = 1 – рівень 1. Після завантаження даних до регістра *SPDR* ведучого порту на його вихід *SCK* надходять сигнали тактової синхронізації. Фронт або зріз цих сигналів, залежно від значення біта *CPOL*, зумовлюють момент

початку передавання поточного біта або початок запису його до зсувного регістра;

– *CPHA* – біт фази синхронізації. Визначає активний перепад сигналу синхронізації, який зумовлює формат передавання даних, як показано на рис. 13.10. Робота порту *SPI* може здійснюватися у форматі 0 (рис. 13.10, а) або у форматі 1 (рис. 13.10, б).




а)




б)

Рисунок 13.10 – Часові діаграми процесу передавання даних через *SPI*

Порти *SPI* працюють у форматі 0 при встановленні у регістрі *SPCR* значення біта *CPHA* = 0. При цьому ведений *SPI* формує старший біт сигналу, що передається після отримання від ведучого *SPI* сигналу *SS#* = 0 (як показано на рис. 13.10, а). Значення біта *CPOL* = 0 означає, що після запису даних до регістра SPDR ведучого *SPI* розпочинається процес обміну даними і видавання даних з ведучого та веденого портів здійснюється при встановленні на виводі *SCK* рівня 0, а введення даних до зсувних регістрів – при встановленні на

виводі *SCK* рівня 1. Якщо значення біта *CPOL* = 1, то виведення даних відбувається за значення SCK = 1, а введення – за рівня 0.

Після завершення передавання символу на вхід *SS#* веденого порту треба подати сигнал рівня 1, а перед початком передавання наступного символу встановити рівень 0.

При роботі у форматі 1 ведучий і ведений порти розпочинають передавання даних водночас. Якщо значення біта *CPOL* = 0, то початок передавання визначається надходженням фронту сигналу синхронізації і надходженням зрізу, якщо значення *CPOL* = 1.

Введення даних здійснюється при надходженні фронту, якщо *CPOL* = 1, і зрізу, – якщо *CPOL* = 0. При цьому значення сигналу *SS#* на вході веденого порту може зберігати своє значення поміж циклами передавання різних символів.

Контроль за процесом обміну здійснюється за допомогою вмісту регістра стану SPSR, формат котрого подано на рис. 13.11. Значення бітів – ознаки виконування поточного циклу обміну – встановлюються після завершення кожного циклу обміну.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| *SPIF* | *WCOL* | – | *MODF* | – | – | – | – |

Рисунок 13.11 – Формат вмісту регістра стану *SPSR*

Біти регістра стану *SPSR* мають таке призначення:

– *SPIF* – ознака завершення обміну. Після передавання байта даних цей біт набирає значення *SPIF* = 1. При цьому формується запит переривання *SPI*, якщо в регістрі *SPCR* біт дозволу переривання встановлено *SPIE* = 1;

– *WCOL* – ознака помилки запису. Встановлюється значення *WCOL* = 1, якщо здіснюється спроба запису до регістра даних *SPDR* до завершення процесу зчитування даних з нього;

– *MODF* – ознака помилки режиму. Встановлюється значення *MODF* = 1 за спроби перевести порт, який працює в режимі ведучого, до режиму веденого.

Вміст регістра *SPSR* може бути лише зчитано. Обнулення бітів цього регістра відбувається після зчитування його поточного стану і запису до регістра *SPDR* даних для наступного циклу обміну.

Послідовний периферійний порт (*SIOP*)**,** котрий використовується в МК *MC*68*HC*705*P*6*A*, може використовуватися для обміну даними, який буде відбуватися розпочинаючи зі старшого або молодшого розряду.

Задля обміну використовуються три виводи: *SDI* – для передавання інформації від веденого порту до ведучого (аналог *MISO*), *SDO* – для передавання інформації від ведучого порту до веденого (аналог *MOSI*) і лінії *SCK* – для передавання сигналу синхронізації. Виводи порту *SIOP* зорганізовуються на виводах *PB*5-7 паралельного порту B. Сигнал *SS#* – відсутній, тому що в цьому порті не формується запит переривання після

завершення передавання і стан порту після завершення поточного циклу обміну можна контролювати лише програмно.

Послідовний периферійний порт (*SSPI*) переважно є аналогічний до порту *SIOP*.

**Блок периферійних пристроїв.**

Таймер є одним з головних пристроїв, які входять до складу цього блока і призначений задля формування певних часових інтервалів. Необхідність введення таймера до складу блока периферійних пристроїв зумовлено призначенням МК для керування певними об'єктами та процесами.

До складу МК *MC*68*HC*705*J*1*A* входить 15-розрядний багатофункціональний таймер *MFT*, структурну схему котрого подано на рис. 13.12.

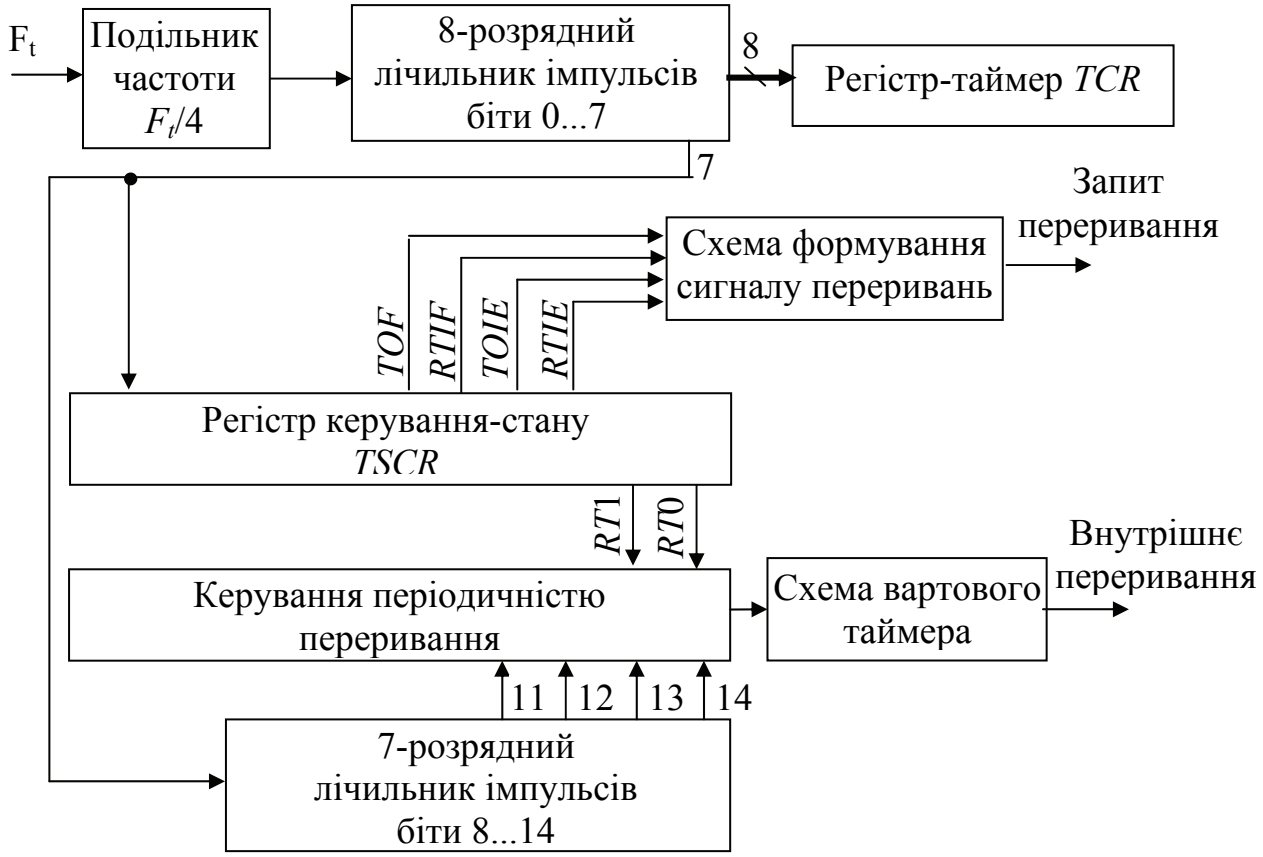



Рисунок 13.12 – Структурна схема багатофункціонального таймера

Багатофункціональний таймер побудований на базі 15-розрядного лічильника, 8 молодших розрядів котрого являють собою програмно-доступний регістр-таймер *TCR* (адреса $09), а 7 старших розрядів використовуються як подільник частоти при формуванні періодичних переривань. Для контролю за роботою таймера використовується вміст регістра-таймера, який може бути прочитано за будь-якого моменту часу. Вихід старшого, 7-го, біта може використовуватися задля переривання роботи таймера через кожні 1024 періоди тактової частоти. Частота роботи лічильника становить $F_c = F_t / 4$.

Вартовий таймер – *COP (Computer operation property)*, який входить до складу багатофункціонального таймера, призначено задля контролю за роботою програмного забезпечення і реалізації переривань через певні відліки часу.

До складу багатофункціонального таймера *MFT* також входить регістр керування-стану *TSCR* (адреса $08), формат котрого подано на рис. 13.13.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| *TOF* | *RTIF* | *TOIE* | *RTIE* | *TOFR* | *RTIFR* | *RT*1 | *RT*0 |

Рисунок 13.13 – Формат регістра керування-стану *TSCR*

Біти цього регістра відповідають певним ознакам, використовуються для керування роботою таймера і мають таке призначення:

– *TOIE* (*Timer Overflow Interrupt Enable*) – ознака дозволу переривання переповнювання;

– *TOF (Timer Overflow Flag*) – ознака переповнювання. Значення *TOF* = 1 встановлюється при зміні вмісту регістра-таймера з $*FF* на $00. Якщо, при цьому ознака дозволу переривання переповнювання встановлена *TOIE* = 1, то відбувається переривання роботи МК. Отже, переривання переповнювання може відбуватися через кожні $2^{10} = 1024$ періоди тактової частоти;

– *RTIF (Real-Time Interrupt Flag*) – ознака періодичного переривання;

– *RTIE (Real-Time Interrupt Enable*) – ознака дозволу періодичного переривання;

– *TOFR (Timer Overflow Flag Reset*) – біт скидання ознаки переповнювання. Запис 1 до цього біта після переривання дозволяє встановлення ознаки *TOF* = 0;

– *RTIFR (Real-Time Interrupt Flag Reset*) – біт скидання ознаки періодичного переривання. Запис 1 до цього біта після переривання дозволяє встановлення ознаки *RTIF* = 0;

– *RT*1-0 (*Real-Time Interrupt)* – біти, що встановлюють часові інтервали встановлення ознаки періодичного переривання. Значення *RTIF* = 1 встановлюється через інтервали, визначувані відповідно до вмісту цих бітів:

| *RT*1-0 | Період встановлення ознаки *RTIE* |
|---|---|
| 00 | $T_p = 2 \cdot 14\ T_t$ |
| 01 | $T_p = 2 \cdot 15\ T_t$ |
| 10 | $T_p = 2 \cdot 16\ T_t$ |
| 11 | $T_p = 2 \cdot 17\ T_t$ |

Використовування багатофункціонального таймера *MFT* дозволяє виконувати вимірювання часових інтервалів поміж певними подіями, формування сигналів з певною затримкою, періодичний виклик потрібних підпрограм, формування імпульсів потрібної частоти та довжини, а також формувати сигнали внутрішніх переривань від вартового таймера.

До складу інших МК сімейства можуть входити таймерні блоки з різною організацією для керування певними об'єктами. Більшість з них побудовано на базі 16-розрядного таймерного блока, котрий доповнюється 8-розрядним лічильником-таймером та 14-розрядним базовим таймером. До складу МК *MC*68*HC*705*B*16 також входять два широтно-імпульсних модулятори, котрі можуть формувати послідовність імпульсів з програмованою шпаруватістю.

***Сімейство M68HC08/908***

Сімейство *M*68*HC*08/908 є подальшим розвиненням МК *M*68*HC*05/705. Вони зберігають їхні архітектурні особливості, однак дозволяють підвищувати техніко-економічні характеристики пристроїв, до складу котрих вони входять. До складу МК підсімейства *M*68*HC*908 входить *Flash*-пам'ять, що дозволяє широко використовувати його у пристроях, які випускаються малими серіями. Програмна сумісність з МК *M*68*HC*05/705 дозволяє використовувати програмне забезпечення, розроблене для них задля програмного керування новими пристроями.

Сьогодні випускається і рекомендовано для використовування понад 30 моделей МК цього сімейства, які відрізнюються складом паралельних і послідовних портів, а також периферійного обладнання.

До складу МК сімейства *M*68*HC*08/908 входять: процесорне ядро *CPU*08; внутрішня пам'ять програм – ПЗП, програмований маскою, обсягом до 32 кбайт або *Flash*-пам'ять, обсяг котрої становить 60 кбайт; ОЗП даних, який має ємність від 128 байт до 2 кбайт. В деяких моделях є РПЗП-ЕС обсягом 512 байт або 1 кбайт.

Залежно від функціонального призначення МК, до їхнього складу може входити від 5-ти до 8-ми паралельних портів, послідовні порти *SCI* та *SPI*. До складу МК деяких серій входять спеціалізовані послідовні порти, призначені для організації мікроконтролерних мереж. Такі МК зреалізовують обмін даними по мультиплексованій шині *J*1850, зреалізовують інтерфейс з послідовною шиною *USB* і спеціалізовані інтерфейсні модулі *CAN* та *I*2*C*. МК серії *EY* вміщує послідовний порт *ESCI*, яки зреалізовує протокол *LIN* задля обміну даними однопровідною лінією зв'язку.

До складу периферійних пристроїв всіх МК сімейства *M*68*HC*08/908 входять 16-розрядні таймери. Більшість моделей вміщує 8- або 10-розрядні АЦП.

Окрім того, МК серії *LD* мають спеціальні виходи сигналів синхронізації і використовуються для керування цифровими моніторами, а МК серії *RF* мають у своєму складі радіопередавач.

Характерною особливістю побудови МК цього сімейства є побудова МК зі стандартним набором модулів (модульний принцип). Поєднання окремих модулів на одному кристалі надає змогу формувати МК різноманітного функціонального призначення.

Загальну структурну схему МК цього сімейства наведено на рис. 13.14. До складу цієї схеми входять стандартні модулі, кожен з котрих має власне функціональне призначення.

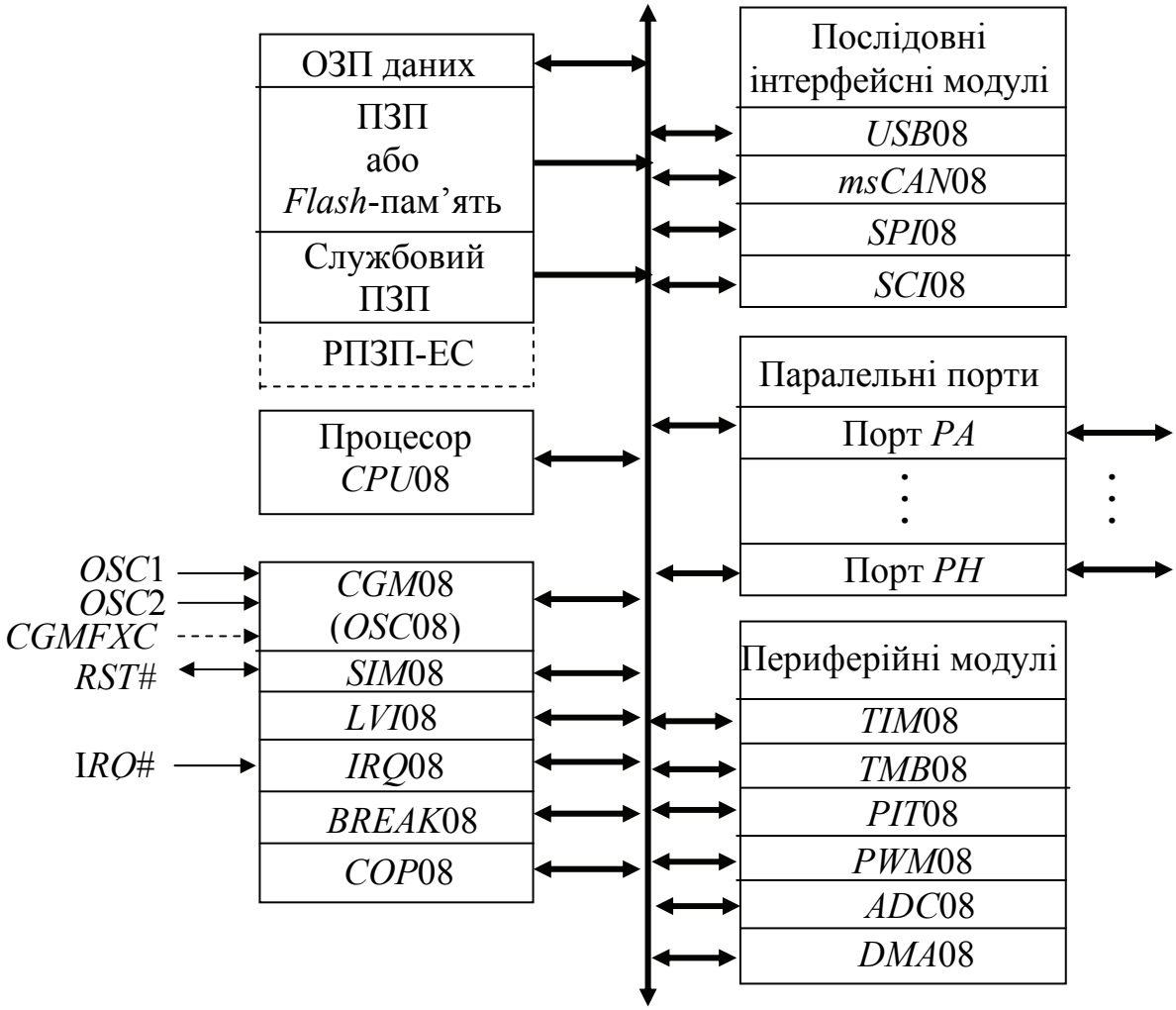



Рисунок 13.14 – Загальна структурна схема МК сімейства *M*68*HC*08/908

Конфігурування МК визначається вмістом регістрів конфігурування, котрі зумовлюють характеристики мікроконтролера. МК підсімейства *M*68*HC*08 у своєму складі мають регістри конфігурування *MOR*, які є комірками ПЗП і їхнє програмування здійснюється у перебігу виготовляння мікросхеми.

Регістри конфігурування *CONFIG* підсімейства *M*68*HC*908 треба програмувати при кожному запускові МК, а упродовж сеансу роботи їхній вміст залишається незмінним.

Формат вмісту регістрів *CONFIG*1 та *CONFIG*2 для МК серій *M*68*HC*908*JK*1, *JK*3 та *JL*3 подано на рис. 13.15.

Біти регістра *CONFIG*1 призначено для виконування таких функцій:

– *COPRS* – використовується для керування роботою вартового таймера і визначає період його роботи;

– *LVID* – використовується для дозволу роботи модуля контролю живлення; значення *LVID* = 1 забороняє роботу цього модуля;

– *SSREC* – визначає час затримки *Td* при виході МК з режиму зупину; при значенні *SSREC* = 1 час затримки дорівнює $Td = 32 \cdot Tq$ і $Td = 4096 \cdot Tq$ – при встановленні *SSREC* = 0;

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| *CONFIG*1 ($001*F*) | *COPRS* | Резерв | Резерв | *LVID* | Резерв | *SSREC* | *STOP* | *COPD* |
| *CONFIG*2 ($001*E* | *IRQPUD* | Резерв | Резерв | *LVIT*1 | *LVIT*0 | Резерв | Резерв | Резерв |

Рисунок 13.15 – Формат вмісту регістрів конфігурування

– *STOP* – значення біта *STOP* = 1 дозволяє перехід МК до режиму зупину при надходженні команди *STOP*; значення *STOP* = 0 при виконуванні команди STOP здійснює перезавантаження процесора;

– *COPD* – використовується для керування роботою вартового таймера; значення *COPD* = 1 забороняє роботу цього модуля.

Біти регістра *CONFIG*2 також використовуються для конфігурування і мають таке призначення:

– *IRQPUD* – значення цього біта *IRQPUD* = 1 забезпечує підключення до входу *IRQ#* резистора, який “підтягує” потенціал цього входу до потенціалу напруги живлення;

– *LVIT*1-0 – визначають значення номінальної напруги живлення; значення *LVIT*1-0 = 00 або 01 відповідає значенню 5,0 В, а значення *LVIT*1-0 = 10 – 3,0 В.

**Модуль формування тактових сигналів *CGM*08** (*Clock Generator Module*) призначено для формування послідовностей імпульсів, потрібних для роботи процесора та периферійних модулів.

Модуль *CGM*08 вміщує два генератори імпульсів: *CG*, котрий формує послідовність імпульсів, частота яких $F_q$ визначається зовнішнім кварцовим резонатором, і генератор *PG* – частота сигналу котрого $F_p = N \cdot F_q$ визначається роботою схеми фазового автоналаштовування частоти.

Сигнал з виходу одного з генераторів після ділення його на 2 надходить на модуль системної інтеграції *SIM*08, де виконується формування тактових імпульсів. Потреба використовування двох генераторів зумовлена вимогами зменшення рівня високочастотних завад при роботі з високими частотами. Для пристроїв, які працюють з тактовою частотою понад 1 МГц, рекомендовано використовувати генератор *PG*. Сигнал на виході такого генератора формується внаслідок множення відносно низькочастотного сигналу $F_q$ і роботи схеми фазового автоналаштовування частоти *PLL (Phase-Locked Loop*).

Схема *PLL* має два режими роботи:

– режим захоплення частоти, за роботи в якому після початкового завантаження забезпечується перехід до генерування сигналу в заданому частотному діапазоні;

– режим утримування. В цьому режимі схема підтримує значення вихідної частоти і компенсує можливі відхилення її значення в межах робочої смуги частот.

Керування роботою модуля *CGM*08 здійснюється за допомогою вмісту регістрів керування *PCTL* (адреса \$001*C*), *PBWC* (адреса \$001*D)*, *PPG* (адреса \$001*E*). Формат цих регістрів подано на рис. 13.16.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| *PCTL* | *PLLIE* | *PLLF* | *PLLON* | *BCS* | 1 | 1 | 1 | 1 |
| *PBWC* | *AUTO* | *LOCK* | *ACQ#* | *XLD* | 0 | 0 | 0 | 0 |
| *PPG* | *MUL*7 | *MUL*6 | *MUL*5 | *MUL*4 | *VRS*7 | *VRS*6 | *VRS*5 | *VRS*4 |

Рисунок 13.16 – Формат вмісту регістрів керування модуля *CGM*08

**Регістр *PCTL*** – регістр керування модулем *CGM*08 – вміщує такі біти:

– *PLLIE* – біт дозволу переривання за запитом від модуля *CGM*08. При запуску МК набирає значення *PLLIE* = 0 (переривання заборонені) і може встановлюватися *PLLIE* = 1 лише за автоматичного керування генератором *PG*;

– *PLLF* – ознака запиту переривання від модуля *CGM*08. Набирає значення *PLLF* = 1 за встановленого біта *PLLIE* і автоматичного керування;

– *PLLON* – біт дозволу роботи (за *PLLON* = 1) генератора *PG*.При запуску МК встановлюється PLLON = 0;

– *BCS* – біт вибору сигналу задля формування тактових частот. Значення *BCS* = 0 передбачає використовування генератора *CG*, а значення *BCS* = 1 – генератора *PG*. При включенні МК встановлюється значення *BCS* = 0, при зміні цього значення – біт *PLLON* =1 і стає доступним лише для читання;

**Регістр *PBWC*** – регістр керування формуванням частоти має в своєму складі біти:

– *AUTO* – біт керування режимом роботи генератора *PG*. Значення біта *AUTO* = 1 встановлюється за автоматичного режиму та *AUTO* = 0 – за програмного;

– *LOCK* – біт точного встановлювання частоти за автоматичного режиму роботи. Значення *LOCK* = 1 встановлюється, якщо частота сигналів, які формуються, перебуває в межах робочої смуги частот, і *LOCK* = 0 – якщо частота виходить за ці межі;

– *ACQ#* – біт керування режимом роботи генератора. За автоматичного керування значення *ACQ#* =0 засвідчує, що робота відбувається в режимі захоплення частоти, а значення *ACQ#* = 1 – в режимі утримування. За програмного режиму значення цього біта зумовлюють режим роботи: *ACQ#* = 0 переводить генератор до режиму захоплення частоти, а *ACQ#* = 1 – до режиму утримування;

– *XLD* – біт контролю функціонування кварцового резонатора. При встановленні *XLD* = 1 через 4·*N* тактів зчитується його значення; якщо значення *XLD* = 0, то резонатор є активний і формує точне значення частоти; якщо після зчитування значення *XLD* = 1, то це відповідає неактивному станові резонатора.

При запуску всі біти цього регістра набирають значення 0.

**Регістр *PPG*** – регістр завдавання коефіцієнтів, які визначають часові характеристики сигналів, вміщує такі біти:

– *MUL*7–4 визначають значення коефіцієнта множення частоти $N = 1...15$;

– *VRS*7– 4 задають коефіцієнт $L = 1…15$, котрий визначає значення центральної частоти в робочій смузі частот системи *PLL*.

**Модуль системної інтеграції *SIM*08** виконує початкове завантаження МК при включенні живлення та перезавантаження при надходженні зовнішнього сигналу скидання *RST#* або сигналу від модуля контролю функціонування *COP*08 або при формуванні помилкового коду команди чи звернення до неіснуючої адреси. Окрім того, цей модуль формує тактові сигнали для процесора й інших модулів, керує передаванням команд і даних внутрішньою шиною, обслуговує запити переривання, зреалізовує різні режими роботи МК. До складу модуля входять регістри, вміст котрих відповідає виконуванню перелічених функцій і формат яких подано на рис. 13.17.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| *SRSR* | *POR* | *PIN* | *COP* | *ILOP* | *ILAD* | 0 | *LVI* | 0 |
| *SBSR* | Резерв | Резервl | Резерв | Резерв | Резерв | Резерв | *SBSW* | Резерв |
| *SBFCR* | *BCFE* | Резерв | Резерв | Резерв | Резерв | Резерв | Резерв | Резерв |

Рисунок 13.17 – Формат вмісту регістрів керування модулем системної інтеграції

**Регістр *SRSR*** (адреса $*FE*01) – регістр стану модуля системної інтеграції *SIM*08. Його вміст дозволяє встановити причину запуску МК. Кожен з його бітів має відповідне призначення і встановлюється в 1 в разі, якщо:

– *POR* – відповідає запуску МК при включенні живлення;

– *PIN* – запуск МК за сигналом скидання *RST#*;

– *COP* – запуск за сигналом вартового таймера *COP*08;

– *ILOD* – перезавантаження при формуванні помилкового коду команди;

– *ILAD* – перезавантаження при зверненні до неіснуючої адреси;

– *LVI* – запуск за сигналом модуля контролю живлення *LVI* 08.

Вміст регістра стану є доступний лише для зчитування, й після його виконання всі біти встановлюються в 0. При роботі програми ініціалізації вміст цього регістра слід зчитати і програма має виконати аналіз причин, що призвели до запуску чи перезавантаження.

**Регістр *SBSR*** (адреса $*FE*00) – регістр коригування адреси повернення з підпрограми обслуговування переривань. До його складу входить лише один

біт *SBSW*, до якого є можливе звернення, решту бітів зарезервовано задля тестування в умовах виробника. Біт *SBSW* встановлюється, якщо переривання в контрольній точці відбулось при виконуванні команд *WAIT* або *STOP*. В цьому разі МК виходить з режиму очікування чи зупину і розпочинає виконування підпрограми обробки переривання, в котрій до стека завантажується адреса команди, яка слідує за командою *WAIT* чи *STOP*. Тому при поверненні до основної програми, після завершення виконування підпрограми обробки переривання (команда *RTI*), МК не повернеться до режиму очікування чи зупину. Задля запобігання зміни режиму функціонування підпрограма обробки переривання перед виконуванням команди *RTI* має перевірити значення цього біта і, в разі потреби, скоригувати значення адреси повернення. Задля скидання цього біта слід записати 0 до відповідного біта регістра *SBSR*.

**Регістр *SBFCR*** (адреса $*FE*03) – регістр дозволу змінювання стану периферійних модулів. Також має лише один активний біт. Встановлення біта *BCFE* = 1 дозволяє при обробці переривань в контрольній точці змінювати стан всіх периферійних модулів. При значенні біта *BCFE* = 0 стан периферійних модулів в перебігу налагодження програми буде зберігатися.

**Модуль керування зовнішнім перериванням *IRQ*08** обслуговує зовнішні запити переривання, які надходять на вхід *IRQ#*, відповідно до обраного режиму. До складу модуля входить регістр керування зовнішніми перериваннями *ISCR* (адреса $001*A*), формат якого подано на рис. 13.18.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | *IRQF* | *ACK* | *IMASK* | *MODE* |

Рисунок 13.18 – Формат вмісту регістрів керування зовнішніми перериваннями *ISCR*

До цього регістра входять чотири біти, які мають таке призначення:

– *IRQF* – ознака зовнішнього запиту переривання (є доступний лише для читання), значення *IRQF* = 1 встановлюється при надходженні запиту переривання на вхід *IRQ#*;

– *ACK* – біт підтвердження приймання запиту переривання (є доступний лише для запису), при запису до нього 1 відбувається скидання біта *IRQF*;

– *IMASK* – біт маски зовнішнього переривання. При встановленні *IMASK* = 1 надходження зовнішнього запиту не призводить до переривання виконуваної програми;

– *MODE* – біт визначення виду сигналу переривання; встановлення значення *MODE* = 1 спричинює переривання при надходженні на вхід *IRQ#* низького рівня сигналу.

**Модуль переривання в контрольній точці *BREAK*08** використовується при налагодженні програмного забезпечення для реалізації режиму зупину у контрольній точці.

Переривання такого типу відбувається, якщо при налагодженні програмного забезпечення сформована процесором адреса збігається з 16-розрядною адресою контрольної точки, записаної в регістрах *BRKH-BRKL* модуля *BREAK*08**.** При цьому, замість адреси наступної команди, формується адреса програмного переривання *SWI*, при обслуговуванні якого зупиняється робота таймерних модулів та вартового таймера. Виконання програмного переривання *SWI* є немаскованим, тому відбувається незалежно від значення біта I регістра прапорців.

Для керування модулем використовується регістр *BRKSR*, який входить до складу модуля. Формат регістра подано на рис. 13.19.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| *BRKE* | *BRKA* | 0 | 0 | 0 | 0 | 0 | 0 |

Рисунок 13.19 – Формат вмісту регістра *BRKSR* модуля *BREAK*08

До складу цього регістра входять два активних біти – *BRKE* та *BRKA*, котрим притаманні такі особливості роботи:

– *BRKE* – ознака дозволу переривання в контрольній точці; значення BRKE = 1 дозволяє переривання, а *BRKE* = 0 – забороняє;

– *BRKA* – ознака збігу адреси поточної команди з вмістом регістрів *BRKH-BRKL*$, встановлення значення *BRKA* = 1 призводить до переривання в контрольній точці.

При функціонуванні модуля також використовуються значення бітів *BCFE* та *SBSW* в модулі системної інтеграції.

**Модуль контролю напруги живлення *LVI*08** використовується для контролю напруги живлення. Якщо при роботі МК напруга живлення знижується понад задане значення, то модуль контролю напруги живлення *LVI*08 переводить МК до початкового стану, котрий зберігається до встановлення нормального рівня напруги.

У більшості серій сімейства до складу модуля *LVI*08 входить регістр стану *LVISR*, вміст якого можна лише зчитувати. Формат регістра подано на рис. 13.20.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| *LVIOUT* | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Рисунок 13.20 – Формат вмісту регістра стану *LVISR*

Біт *LVIOUT* цього регістра набирає значення *LVIOUT* = 1, якщо напруга живлення зменшується понад рівень 4,0...4,3 В за номінальної напруги живлення 5 В і до рівня 2,4...2,7 В – за номінального значення 3 В.

Модуль продовжує роботу в режимі очікування і відключається в режимі зупину.

**Модуль контролю функціонування *COP*08** контролює виконування програми за допомогою вартового таймера.

Робота вартового таймера полягає в тому, що до регістра керування *COPCTL* (адреса $*FFFF*) періодично записується довільне число. Запис повинен відбуватися не рідше одного разу за час $T_w$. Цей час визначається значенням біта *COPRS* у регістрі конфігурування *CONFIG*1 і становить:

$T_w = 262128 \cdot T_q$ – за значення біта *COPRS* = 0;

$T_w = 8176 \cdot T_q$ – за значення біта *COPRS* = 1.

Якщо такий запис не буде виконуватися, то вартовий таймер виконає перезапуск МК. Такий запис зручно виконувати командою

```
STA $FFFF          ; запис вмісту акумулятора до регістра з адресою $FFFF
```

Цю команду слід долучати до тексту програми задля періодичного виконування.

Вартовий таймер продовжує свою роботу в режимі очікування, і тому, залежно від його поточного стану перед надходженням команди *WAIT*, може статися його незапланованe спрацьовування.

В режимі зупину вартовий таймер припиняє роботу, але зберігає значення поточного стану, тому має сенс перед виконуванням команди STOP виконувати скидання цього модуля.

До складу послідовних інтерфейсних модулів входять:

**Модуль інтерфейса з послідовною шиною *USB*08** забезпечує обмін даними шиною *USB*, яка використовується в обчислювальній техніці.

**Модуль послідовного інтерфейса *msCAN*08** забезпечує обмін даними в стандартному та розширеному форматах відповідно до специфікацій *CAN* 2.0 *A* та *CAN* 2.0 *B*.

**Модуль синхронного периферійного інтерфейса *SPI*08** забезпечує синхронний обмін даними зі швидкістю до 4 Мбіт/с. Цей модуль використовується для організації зв’язку поміж МК та іншими пристроями на незначній відстані.

**Модуль асинхронного інтерфейса зв’язку *SCI*08** зреалізовує стандартний асинхронний протокол обміну 8- або 9-бітними даними з одним стартовим та одним стоповим бітами зі швидкістю до 130 кбіт/с.

Блок паралельних портів вміщує від 2-х до 8-ми паралельних 8-розрядних портів (*PA, PB, ..., PG, PH*). Виводи деяких портів можуть використовуватися задля виконання альтернативних функцій: організації послідовного обміну даними, обміну сигналами таймерних модулів, введення аналогових сигналів тощо.

До блока периферійних модулів для різних серій можуть входити:

**Таймерний модуль *TIM*08** призначений для формування часових інтервалів, потрібних для керування МК та іншими пристроями, які входять до складу системи, в цілому. Він побудований на базі 16-розрядного лічильника, який, в різних моделях, може мати 2, 4 чи 6 каналів, які можуть працювати в режимі захоплення чи збігу з наперед окресленим значенням часу

спрацьовування. Канали мають входи сигналів захоплення ІС і відповідні регістри захоплення та порівняння, а також виходи сигналів збігання. Лічильник може працювати з вхідними сигналами, які надходять від тактового генератора модуля *CGM*08 – режим таймера (з можливістю його зупину та запуску) або з сигналами від зовнішніх пристроїв – режим лічби зовнішніх подій. Окрім того, пари каналів таймера можуть використовуватися задля формування сигналів широтно-імпульсної модуляції (ШІМ). Більшість моделей сімейства вміщує два незалежних таймерних модуля.

**Модуль базового таймера *TBM*08** забезпечує періодичне формування сигналів запиту переривання при переповненні базового лічильника. Він вміщує 15-розрядний лічильник, на вхід якого надходять сигнали кварцового резонатора. В режимі очікування цей таймер продовжує функціонувати і забезпечує перехід МК до робочого режиму. В режимі зупину таймер може продовжувати працювати, якщо дозволено роботу генератора *CGM*08 і формування сигналу запиту переривання.

**Модуль таймера періодичних переривань *PIT*08** використовується для періодичного формування сигналів запиту переривання. Модуль побудовано на базі 16-розрядного лічильника і забезпечує формування сигналів запиту переривань в широкому діапазоні значень періоду цих сигналів. В режимі очікування цей таймер продовжує функціонувати і забезпечує перехід МК до робочого режиму при переповненні лічильника, а в режимі зупину цей модуль вимикається.

**Модуль широтно-імпульсного модулятора *PWM*08** вміщує 12-розрядний 6-канальний ШІМ модулятор.

**Модуль аналого-цифрового перетворення *ADC*08** використовується для аналого-цифрового перетворення вхідних аналогових сигналів. В більшості моделей зреалізовано 8-розрядне перетворення аналогового сигналу, а в деяких моделях – 10-розрядне. Кількість аналогових входів в різних моделях може становити від 4 до 15.

**Модуль прямого доступу до пам’яті *DMA*08** забезпечує роботу зі швидкодіючими зовнішніми пристроями.

**Процесорний модуль *CPU*08** є модифікованим варіантом процесора *CPU*05. Він має розширений набір команд та способів адресування і програмно цілковито є сумісний з *CPU*05. Регістрову модель процесора *CPU*08 подано на рис. 13.21.

До регістрової моделі входять:

– 8-розрядний акумулятор *A*;

– 16-розрядний індексний регістр *H:C*. Для забезпечення програмної сумісності з *CPU*05 індексний регістр складається з двох частин, роздільне використовування яких дозволяє забезпечувати функціонування програм МК сімейства M68*HC*05/705;

– 16-розрядний програмний лічильник *PC*;

– 16-розрядний вказівник стека *SP*;

– 8-розрядний регістр прапорців *CCR*. Вміщує такі ознаки, які формуються у процесорі *CPU*05. До них додано ознаку *V* – ознаку

переповнювання при обробці чисел зі знаком. Ознака встановлюється $V = 1$, якщо результат має кількість розрядів більшу, ніж можливо розмістити їх в розрядній сітці МК.

–

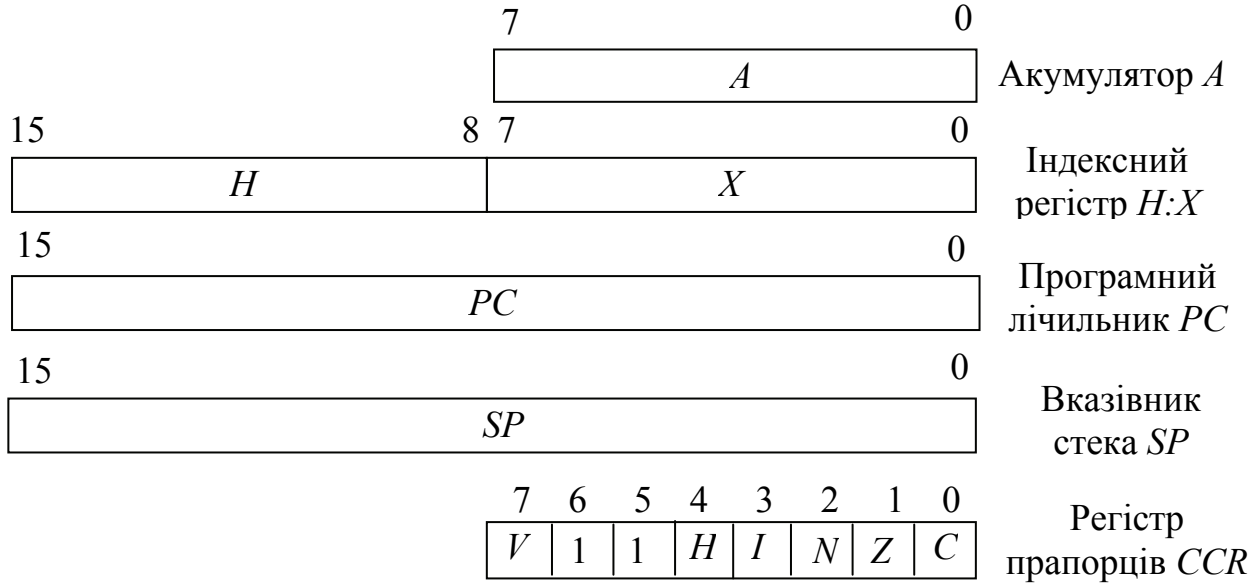



Рисунок 13.21 – Регістрова модель процесора *CPU*08

Обсяг адресного простору, який може адресувати МК сімейства *M*68*HC*08/908, становить 64 кбайти. Деякі МК сімейства мають менший обсяг адресного простору, тому в їхньому адресному просторі є області адрес, які не відповідають певним пристроям. При зверненні до таких адрес зреалізовується переривання, яке відповідає наявності помилки у програмі – зверненні до неіснуючої комірки. Внаслідок цього відбувається перезавантаження МК, до програмного лічильника *PC* автоматично завантажується адреса першої команди обробки переривання з двох останніх комірок адресного простору (\$*FFFE* – \$*FFFF*).

При перезавантаженні МК у вказівник стека автоматично записується адреса вершини стека (\$00*FF*), яка забезпечує використовування в якості стека комірок пам’яті ОЗП та регістрів, адреси яких містяться у діапазоні (\$0000...\$00*FF*). При роботі користувач має змогу замінити цю адресу вершини стека на іншу в межах адресного простору ОЗП.

***Сімейство MC68HC11/711***

Найбільш досконалим сімейством 8-розрядних МК є *MC*68*HC*11/711. Сьогодні випускається близько 20 різних моделей МК цього сімейства. До складу структурної схеми МК входять модулі, які за функціональним призначенням є аналогічні до описаних вище. Різні моделі сімейства мають однакове процесорне ядро і відрізнюються обсягом та типом внутрішнього запам’ятовувального пристрою, складом периферійного обладнання й деякими іншими характеристиками. Особливістю цього сімейства є можливість збільшення зовнішньої пам’яті обсягом від 64 кбайт до 4 Мбайт.

МК сімейства *MC*68*HC*11/711 у своєму складі мають внутрішню пам’ять програм – ПЗП – у складі MC68HC11 або РПЗП – у *MC*68*HC*711, обсягом до 32 кбайт; ОЗП даних обсягом від 192 до 1024 байт. Деякі моделі мають внутрішній РПЗП-ЕС обсягом до 540 байт.

При використовуванні МК цього сімейства можна зорганізовувати роботу в двох режимах – автономному і розширеному. При роботі в автономному режимі використовується лише внутрішня пам’ять МК, а в розширеному дозволяється підключення зовнішньої пам’яті, робота з якою відбувається за допомогою мультиплексованої чи окремої зовнішньої шини адреси/даних. В деяких моделях передбачено розширення зовнішньої пам’яті і можливість зорганізовувати банки пам’яті.

Всі моделі мають в своєму складі 16-розрядний таймер, котрий може мати 3-4 входи сигналів захоплення частоти (*IC*) і 4-5 входів сигналів збігу частот. Цей таймер також використовується задля формування сигналів періодичних переривань. Окрім таймера, до складу МК входить 8-розрядний лічильник зовнішніх подій.

Деякі МК вміщують 8-розрядні широтно-імпульсні модулятори, які мають 4 входи, що вони можуть працювати в режимі 16-розрядних широтно-імпульсних модуляторів з двома входами.

МК сімейства вміщують від 4-х до 10-ми паралельних 8-розрядних паралельних портів, асинхронний і синхронний послідовні порти *SCI* (*SCI*+) і *SPI*.

До складу більшості моделей входить 8-розрядний АЦП з 8-мома аналоговими входами.

Процесорне ядро сімейства складається з процесора 68*HC*11, регістрову модель котрого зображено на рис. 13.22.

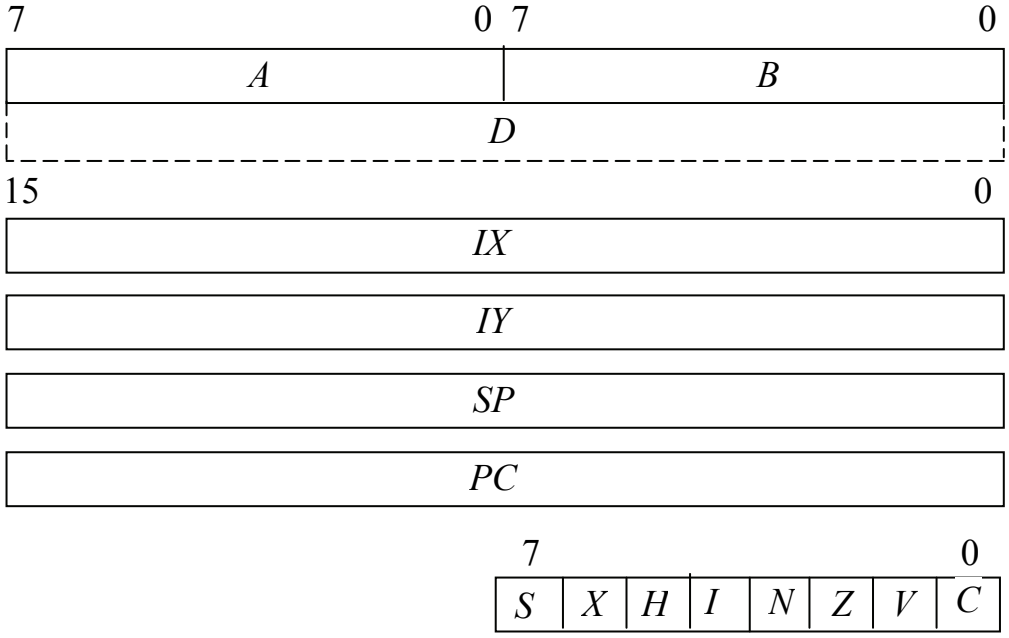



Рисунок 13.22 – Регістрова модель процесора 68*HC*11

До складу регістрової моделі процесора 68*HC*11 входять:

– два 8-розрядні акумулятори *A* і *B*, котрі при виконуванні деяких команд об'єднуються у 16-розрядний регістр *D*;

– два 16-розрядні індексні регістри *IX* і *IY*, котрі використовуються для формування адреси при індексному адресуванні операндів;

– 16-розрядний вказівник стека *SP*;

– 16-розрядний програмний лічильник *PC*;

– 8-розрядний регістр прапорців *CCR*, котрий вміщує ознаки *H, I, N, Z*, *V* і *C*, призначення й використовування котрих є аналогічні до 8-розрядних МК. До складу цього регістра до *CPU*12 додатково введено ознаку *X*, встановлення якої *X* = 1 забороняє обслуговування зовнішнього запиту переривання по входу *XIRQ*#, і ознаку *S*, яка при встановленні *S* = 1 забороняє переключення МК до режиму зупину при надходженні команди *STOP*.

Процесор виконує обробку 8- і 16-розрядних операндів і зреалізовує 108 команд. При роботі з операндами він використовує такі способи адресування як сімейство 68*HC*05/705; більшість команд, виконуваних МК, є аналогічні до команд сімейства 68*HC*05/705. Відмінності зумовлено наявністю двох акумуляторів. Додатково до системи команд введено команди ділення і розширено можливості команд бітових операцій.

МК сімейства можуть працювати в чотирьох режимах, які зумовлюються зовнішніми сигналами *MODB* і *MODA*, які приймаються при початковому завантаженні МК, а також вмістом регістра *HPRIO*. Режими функціонування визначаються відповідно до табл. 13.2.

Таблиця 13.2 – Режими функціонування МК сімейства *MC*68*HC*11/711

| Режим | Зовнішні сигнали | | Вміст регістра *HPRIO* | | | | |
|---|---|---|---|---|---|---|---|
| | *MODB* | *MODA* | *RBOOT* | *SMOD* | *MDA* | *IRV* | *PSEL*3-0 |
| Робочий автономний | 1 | 0 | 0 | 0 | 0 | 0 | 0110 |
| Робочий розширений | 1 | 1 | 0 | 0 | 1 | 0 | 0110 |
| Спеціальний – завантаження | 0 | 0 | 1 | 1 | 0 | 1 | 0110 |
| Спеціальний – тестування | 0 | 1 | 0 | 1 | 1 | 1 | 0110 |

Автономний (однокристальний) робочий режим передбачає функціонування МК з використанням лише внутрішньої пам'яті, а в розширеному режимі передбачено підмикання через порти *B* та *C* зовнішнього запам'ятовувального пристрою. При роботі у спеціальному режимі відбувається завантажування внутрішнього запам'ятовувального пристрою через послідовний порт *SPI* під керуванням спеціальної програми-монітора, яку розміщено у службовому ПЗП, або тестування МК заводом-виробником.

Режим функціонування і конфігурація МК визначаються вмістом службових регістрів *HPRIO, CONFIG, OPTION* та *INIT*, котрі входять до складу процесора. Формат вмісту цих регістрів подано на рис. 13.23. Для різних серій МК адреси цих регістрів та їхній склад не збігаються, але призначення бітів є однаковим.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *HPRIO* | *RBOOT* | *SMOD* | *MDA* | *IRVNE* | *PSEL*3 | *PSEL*2 | *PSEL*1 | *PSEL*0 |
| *OPTION* | *ADPU* | *CSEL* | *IRQE* | *DLU* | *CME* | *FCME* | *CR*1 | *CR*0 |
| *INIT* | *RAM*3 | *RAM*2 | *RAM*1 | *RAM*0 | *REG*3 | *REG*2 | *REG*1 | *REG*0 |
| *CONFIG* | 0 | 0 | 0 | 0 | *NOSEC* | *NOCOP* | *ROMON* | *EEON* |

Рисунок 13.23 – Формат вмісту регістрів конфігурування МК *MC*68*HC*11/711

До складу регістра *HPRIO* входять біти:

– *RBOOT* – біт дозволу звернення до програми-монітора, яка виконує завантажування ОЗП із зовнішнього джерела. З табл. 13.2 видно, що при роботі у спеціальному режимі завантаження значення цього біта встановлюється *RBOOT* = 1;

– *SMOD* разом з бітом *MDA*, відповідно до табл. 13.2, визначають режим функціонування МК після початкового завантаження;

– *IRVNE* – в розширеному режимі значення *IRVNE* = 1 дозволяє видавання даних на виводи порту *C* при зверненні до внутрішньої пам’яті; у автономному режимі та спеціальних режимах значення *IRVNE* = 1 забороняє виведення тактових сигналів на вивід *E* (задля зменшення рівня завад в системі);

– *PSEL*3-0 – визначають пріоритет обслуговування маскованих запитів переривань.

При початковому завантажуванні біти *RBOOT, SMOD, MDA, IRVNE* та *PSEL*3-0 набирають значень відповідно до табл. 13.2. Після запуску значення біта *RBOOT* і бітів *SMOD* та MDA у спеціальному режимі можна змінювати. Значення біта *IRVNE* можна одноразово змінювати у будь-якому режимі. До бітів *PSEL*3-0 можливе звернення в робочому режимі. До біта *MDA* також є можливе одноразове звернення в робочому режимі.

Регістр *OPTION* вміщує біти, які визначають функціонування окремих модулів МК:

– *ADPU* – значення *ADPU* =1 вмикає живлення модуля АЦП;

– *CSEL* – значення біта *CSEL* = 1 дозволяє використовування внутрішнього *RC*-генератора задля формування синхросигналів забезпечення програмування РПЗП-ЕС та роботу АЦП; значення *CSEL* = 0 відповідає використовуванню задля синхронізації тактових імпульсів МК;

– *IRQE* – визначає вид зовнішнього сигналу запиту переривань *IRQ#*; при значенні *IRQE* = 0 за активний вважається низький рівень вхідного сигналу *IRQ#; IRQE* = 1 визначає, що запит переривання здійснюється від зрізу сигналу *IRQ#*;

– *DLU* – значення *DLU* = 1 визначає затримку початку функціонування після виходу МК з режиму зупину на час близько 4000 тактів, при значенні біта *DLU* = 0 затримка становить 4 такти;

– *CME* – дозволяє функціонування, за *CME* = 1, схеми контролю тактової частоти;

– *FCME* – дозволяє функціонування, за *FCME* = 1, схеми контролю тактової частоти, за будь-якого значення біта *CME*; за значення *FCME* = 0 функціонування блока контролю тактової частоти визначається вмістом біта *CME*;

– *CR*1-0 – біти визначають значення *Kw* – коефіцієнта задля задавання можливих інтервалів часу контролю для вартового таймера. Визначається відповідно до табл. 13.1.

При початковому завантаженні всі біти регістра *OPTION* скидаються і набирають значення 0, окрім біта *DLU*, який встановлюється в 1. За подальшої роботи у спеціальних режимах всі біти є доступними і в них може проводитися запис і відбуватися зчитування їхнього вмісту. В робочих режимах значення бітів *ADPU, CSEL, CME* може змінюватися, а значення інших бітів можна одноразово змінювати упродовж 64 тактів після запуску МК.

Вміст регістра *INIT* визначає розміщення в адресному просторі ОЗП даних і внутрішніх регістрів периферійних модулів. За початкового завантаження біти *RAM*3-0, котрі визначають адресу ОЗП, набирають значення 0000, що забезпечує звернення до сторінки з адресою 0. Біти *REG*3-0 набирають значення 0001 і адресують сторінку з адресою 1, в якій розміщено адреси всіх внутрішніх регістрів периферійних модулів. За роботи в одному з робочих режимів дозволяється одноразовий запис до цього регістра упродовж 64-х тактів після запуску.

Регістр *CONFIG* визначає конфігурацію МК, і його біти мають таке призначення:

– *NOSEC* – захист внутрішніх ОЗП та РПЗП-ЕС від зовнішнього зчитування; значення біта *NOSEC* = 0 вказує на наявність такого захисту (встановлюється виробником, за замовленням користувача), значення біта *NOSEC* = 1 визначає відсутність такого захисту;

– *NOCOP* – керування роботою модуля контролю функціонування; значення *NOCOP* = 0 дозволяє функціонування цього модуля;

– *ROMON* – керування роботою внутрішнього ПЗП; значення біта *ROMON* = 1 дозволяє функціонування ПЗП, а значення *ROMON* = 0 спричинює звернення до зовнішньої пам’яті;

– *EEON* – керування роботою внутрішнього РПЗП-ЕС; значення біта *EEON* = 1 дозволяє функціонування РПЗП-ЕС, а значення *EEON* = 0 спричинює звернення до ОЗП або до зовнішньої пам’яті.

Регістр *CONFIG* програмується користувачем як РПЗП-ЕС, і тому він зберігає значення власних бітів до перепрограмування.

### *13.1.2 16-розрядні мікроконтролери*

#### *Сімейство 68HC12/912*

16-розрядні МК сімейства 68*HC*12/912 є основним промисловим стандартом 16-розрядних МК фірми *Motorola*. З 2002 року фірма розпочала випуск нового покоління цього сімейства – 68*HCS*12/912. Ці МК схарактеризовуються значним підвищенням продуктивності (тактову частоту збільшено до 40 МГц) і збільшенням обсягу внутрішньої *Flash*-пам'яті (до 512 кбайт).

До складу сімейства 68*HCS*12/912 входить низка МК, які відрізнюються один від одного типом та обсягом внутрішньої пам'яті, кількістю та типами периферійних пристроїв, які розміщено у мікросхемі.

В системах та пристроях ці МК можуть працювати в автономному та розширеному режимах. При роботі в автономному режимі (*Single Chip)* МК використовує лише внутрішню пам'ять задля зберігання програм і даних, а в розширеному режимі передбачено підключення зовнішнього запам'ятовувального пристрою і організація 8- або 16-розрядної системної шини (в більшості моделей – мультиплексованої).

МК сімейства 68*HC*12 мають модульну структуру, яка складається з низки стандартних функціональних блоків, взаємодія поміж якими зреалізовується по стандартизованій міжмодульній шині. Загалом до складу МК входять: 16-розрядний процесор *CPU*12, внутрішній оперативний запам'ятовувальний пристрій (ОЗП даних), РПЗП-ЕС, Flash-пам'ять або ПЗП, програмований маскою, модуль інтеграції *LIM* та набір периферійних модулів. Процесор *CPU*12 взаємодіє з цими модулями через їхні регістри, адреси котрих після запуску розміщуються у перших 512-ти комірках адресного простору.

Модуль інтеграції *LIM* використовується для реалізації службових функцій, до котрих належать:

– генерування потрібних тактових сигналів;

– початковий запуск і конфігурування МК;

– реалізація переривань при надходженні запитів від зовнішніх пристроїв, внутрішніх переривань процесора, періодичних переривань, а також зовнішніх запитів;

– реалізація інтерфейса із зовнішньою системною шиною у розширеному режимі;

– контроль функціонування МК за допомогою вартового таймера і моніторингу тактових імпульсів;

– налагодження програм за допомогою режиму *BDM* (*Back-ground Debug Mode*) і апаратного встановлювання контрольних точок.

До складу блока периферійних модулів можуть входити:

– 8 – 12 паралельних портів задля обміну 8-розрядними даними. Виводи більшості портів можуть використовуватися для виконування інших функцій,

наприклад, для передавання адреси та даних при роботі із зовнішньою пам’яттю, організації послідовних портів *SCI, SPI* таймера, АЦП, контролерів шин тощо;

– таймерні модулі *TIM* та *ECT*, котрі побудовано на базі 16-розрядного лічильника, а також 16-розрядний лічильник подій, який входить до складу деяких моделей;

– модулі аналого-цифрового перетворювання входять до складу всіх МК сімейства, мають 8 входів для приймання аналогових сигналів. В більшості моделей зреалізовано 10-бітове перетворювання аналогового сигналу;

– модуль формування сигналів широтно-імпульсної модуляції має чотири вихідних канали, значення парності для кожного з котрих можуть визначатися окремо;

– модулі послідовного асинхронного та синхронного інтерфейсів *SCI* та *SPI*, котрі працюють аналогічно до відповідних пристроїв 8-розрядних МК;

– спеціалізовані інтерфейсні модулі *BDLC, CAN*, *I*2*C* входять до складу різних моделей і зреалізовують обмін даними відповідно до мережних протоколів *J*1850, *CAN* 2.0*A*/*B*, *I*2*C*.

16-розрядний процесор *CPU*12, який входить до складу МК сімейства дозволяє виконувати обробку 8- і 16-розрядних операндів. Регістрова модель цього процесора є аналогічна до процесора *CPU*11, яку подано на рис. 13.22.

До складу регістрової моделі входять ті ж самі регістри, як у *CPU*11, які мають аналогічне функціональне призначення.

Програмний код *CPU*12 є суміщеним з кодом 68*HC*11 на рівні вхідного тексту.

Процесор *CPU*12 виконує всі способи адресування, які використовуються сімейством 68*HC*11/711 і, окрім того, може додатково зреалізовувати ще 7 додаткових варіантів індексного адресування.

Система команд процесора *CPU*12 вміщує 208 команд над операндами, які розміщуються у комірках пам’яті та регістрах, і є розширеним набором команд сімейства 68*HC*11/711.

***Сімейство 68HC16/916***

МК сімейства 68*HC*16/916 є подальшим розвиненням тенденцій, які було зреалізовано в 8- та 16-розрядних контролерах. Сімейство вміщує низку моделей, котрі відрізнюються типом та обсягом внутрішньої пам’яті та номенклатурою периферійного обладнання, які входять до складу МК. У порівнянні з сімейством 68*HC*11/711 ці МК схарактеризовуються такими перевагами:

– розширений набір команд та способів адресування;

– введення спеціальних регістрів та команд задля зреалізовування операцій цифрової обробки сигналів;

– підвищена продуктивність за рахунок збільшення тактової частоти до 25 МГц;

– збільшення до одного Мбайта обсягу адресного простору (через те що є дозволено роздільне звернення до пам'яті команд та пам'яті даних, то сумарний обсяг пам'яті може сягати 2 Мбайти);

– використовування периферійних пристроїв з розширеними функціональними можливостями;

– наявність вбудованих засобів налагодження програмного забезпечення.

МК сімейства 68*HC*16/916 мають модульну структуру і будуються зі стандартних функціональних модулів, до складу яких входять: процесор *CPU*16, модуль внутрішньої пам'яті, модуль системної інтеграції *SIM* або *SCIM*, модуль послідовного інтерфейсу *QSM*, багатоканальний комунікаційний інтерфейс *MCCI*, процесор часових інтервалів *TPU*, таймерні модулі *GPT* або *CTM*, аналого-цифровий перетворювач *ADC*. Деякі з цих модулів використовуються також у 32-розрядних МК фірми. Узагальнена структурна схема МК сімейства 68*HC*16/916 подана на рис. 13.24.

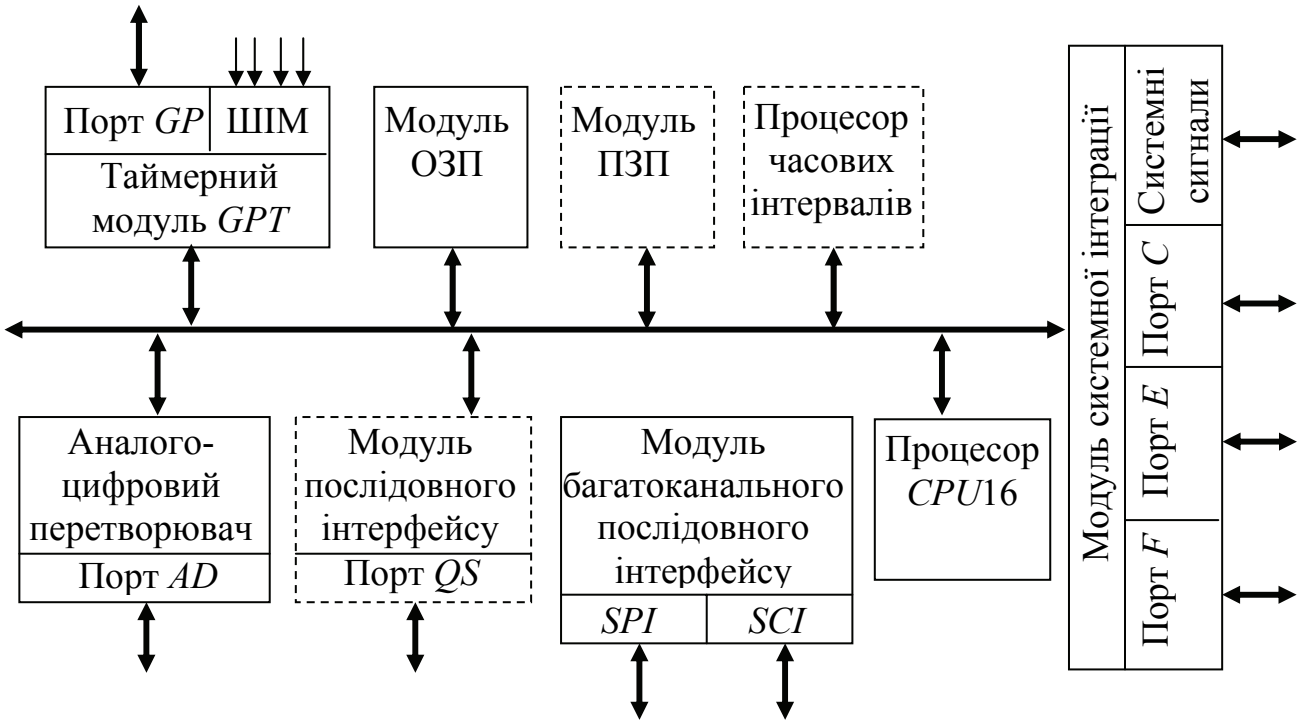



Рисунок 13.24 – Узагальнена структурна схема МК сімейства 68*HC*16/916

До складу цієї схеми входять:

– модуль системної інтеграції *SIM* або *SCIM*, який призначено для забезпечення функціонування процесора і його сумісної роботи з іншими пристроями системи. Він генерує імпульси тактової синхронізації; контролює за допомогою вартового таймера функціонування системи; зреалізовує інтерфейс з периферійним устаткуванням; формує періодичні запити переривання;

– таймерний модуль *GPT* або *CTM*. Його побудова і функціональне призначення є аналогічні до таймерних модулівм сімейства 68*HC*11/711 та 68*HC*12/912. До складу модуля *GPT* входить пристрій для формування сигналів з широтно-імпульсною модуляцією;

– процесор часових інтервалів *TPU*, який входить до складу деяких моделей, призначений для виконування низки функцій формування часових інтервалів, вміщує 16 таймерних каналів, котрі можуть програмуватися і працювати без участі процесора;

– модуль багатоканального послідовного інтерфейсу *MCCI*, вміщує синхронний порт *SPI* та два асинхронних порти *SCI*. Їхнє функціонування є аналогічне до однойменних портів МК сімейства 68*HC*05/705 та 68*HC*11/711;

– модуль послідовного інтерфейсу *QSM* використовується в деяких моделях сімейства і вміщує один асинхронний порт *SCI* і модифікований варіант синхронного порту *QSPI*, котрий має в своєму складі буферну пам'ять, що надає змогу зорганізовувати передавання блоків даних обсягом до 32-х байт без участі процесора;

– модуль аналого-цифрового перетворювача виконує 8- або 10-бітне перетворювання сигналів, які надходять з 8-ми аналогових входів.

Процесор *CPU*16 виконує обробку 8-, 16- та 32-розрядних операндів і має систему команд, до складу якої входить 264 команди. Регістрову модель процесора наведено на рис. 13.25.

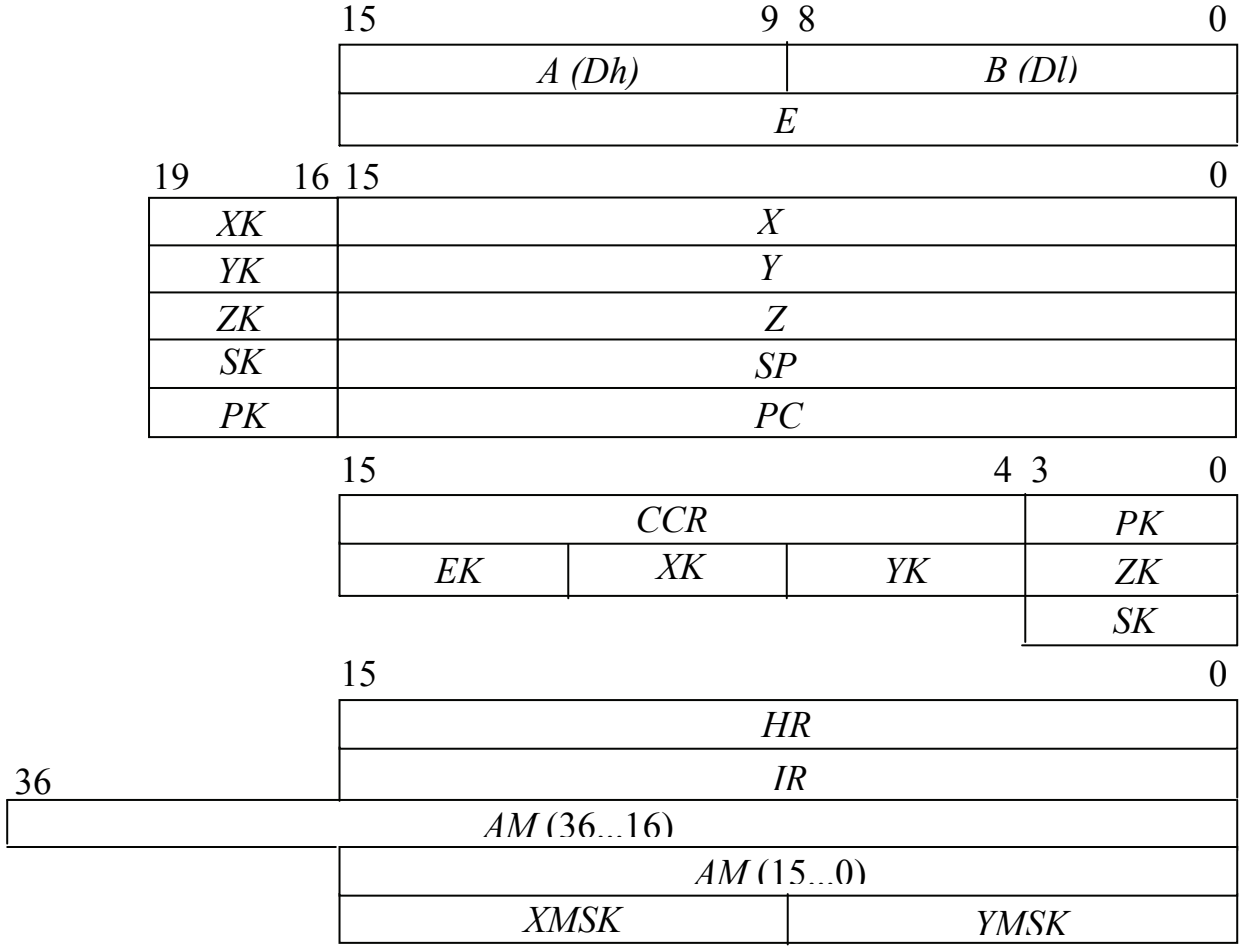



Рисунок 13.25 – Регістрова модель процесора *CPU*16

До регістрової моделі входять:

– два 8-розрядних акумулятора – *A* та *B*, котрі при обробці 16- та 32-розрядних операндів використовуються як один 16-розрядний акумулятор *D*;

– додатковий 16-розрядний акумулятор E;

– три 16-розрядних індексних регістри – *X, Y, Z,* котрі використовуються для адресування операндів;

– 16-розрядний вказівник стека *SP*;

– 16-розрядний лічильник команд *PC*.

Всі індексні регістри, вказівник стека і лічильник команд мають 4-розрядне розширення, відповідно *XK, YK, ZK, SK, PK*. Ці розширення використовуються для адресування 16-ти банків пам'яті, на котрі поділено адресний простір МК. Вибір банку здійснюється за допомогою 4-розрядного розширення адреси *EK* і значень відповідних розширень. Розширення *EK, XK, YK, ZK* поєднано в один 16-розрядний регістр *K*. Розширення лічильника команд *PK* розміщено у молодших розрядах регістра ознак *CCR*, а розширення вказівника стека SP – в окремому 4-розрядному регістрі *SK*.

До регістрової моделі також входять спеціалізовані регістри *HR, IR, AM, MR,* котрі використовуються при виконуванні послідовного множення-додавання дробових чисел при виконуванні цифрової обробки сигналів і два 8-розрядних регістри – *XMSK* та *YMSK* – задля зберігання масок, котрі визначають обсяг буферного модуля пам'яті при виконуванні команд множення з накопиченням *MAC*.

Вміст регістра умов *CCR* подано на рис. 13.26.

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *S* | *MV* | *H* | *EV* | *N* | *Z* | *V* | *C* | *IP* | | | *SM* | *PK* | | | |

Рисунок 13.26 – Вміст регістра умов *CCR*

До складу регістра умов входять:

– *PK* – 4 розряди розширення програмного лічильника;

– *SM* – ознака "насичення" результату в акумуляторі *AM*;

– *IP* – маска запиту переривання;

– *EV* – ознака поширення результату в акумуляторі *AM*;

– *MV* – ознака переповнювання результату в акумуляторі *AM*.

Інші біти – *S, H, N, Z, V, C* – мають призначення, аналогічне до таких самих ознак процесора *CPU*11, 12.

Біти *SM, EV, MV* використовуються при виконуванні операцій цифрової обробки сигналів.

МК сімейства 68*HC*16/916 забезпечують конвеєризацію при виконуванні програм, і черга з 6-ти команд зберігається у пристрої керування процесора. Тому поточне значення вмісту програмного лічильника разом з розширенням *PK:PC*, що визначає адресу наступної команди, має значення на 6 більше за значення старшого байта поточної команди.

### *13.1.3 32-розрядні мікроконтролери*

Всі МК сімейства 683*xx* побудовано з використанням 32-розрядного процесорного ядра *CPU*32 і периферійних модулів. Ці пристрої поєднуються за допомогою стандартної міжмодульної шини. Більшість периферійних модулів є аналогічні до периферійних модулів МК сімейства 68*HC*16/916.

Архітектура МК 683*xx* базується на принципах, що їх було закладено в сімейство мікропроцесорів *MC*68000, що дозволяє мати обсяг адресного простору до 16 Мбайт і використовувати програмне забезпечення, яке розроблено для процесорів цього сімейства. Відповідно до особливостей архітектури, є певна різниця при виконуванні програм операційної системи і програм користувача. Тобто МК може функціонувати у двох режимах: режимі супервізора (виконування програм операційної системи) і режимі користувача (виконування програм користувача). За роботи в режимі супервізора при виконуванні програм дозволяється доступ до всіх ресурсів системи; команди, які виконуються лише в цьому режимі, називають привілейованими. В режимі супервізора користувач має можливість звертатися до регістрів і комірок пам'яті, які входять до складу периферійних модулів. При включенні МК розпочинає функціонувати в режимі супервізора, і за ініціалізації операційна система визначає режим подальшої роботи. Переведення до режиму супервізора здійснюється при встановленні відповідного біта в регістрі ознак процесора і при подальшій роботі режим не змінюється. Повернення до режиму користувача виконується лише при обробці виключних ситуацій або повторному перезапуску. Ці особливості відображено в регістровій моделі процесора *CPU*32.

Регістрову модель процесора *CPU*32 наведено на рис. 13.27.

До складу регістрової моделі входять:

– 2 групи 32-розрядних регістрів:

8 регістрів даних, які можуть працювати з даними, що є байтами, словами та довгими словами;

8 регістрів адреси, в котрих регістр *A*7 є дубльованим; він використовується в якості вказівника стека, для роботи в режимі супервізора має назву *SSP*, а при роботі в режимі користувача – *USP*.

– *PC* – 32-розрядний програмний лічильник; в процесорі *CPU*32 використовуються лише 24 розряди, відповідно до розрядності шини даних і обсягу адресного простору;

– *SR* – 16-розрядний регістр стану; складається з двох частин, кожна з котрих має розрядність: системного байта і байта користувача (регістру ознак – *CCR*). Регістр *SR* є повністю доступним в режимі супервізора. В режимі користувача доступ є можливий лише до байта користувача. Вміст регістра *SR* подано на рис. 13.28;

– регістри *VBR, SFC, DFC* є доступними лише в режимі супервізора. 32-розрядний регістр *VBR* вміщує базову адресу таблиці векторів виключень, завантаження цього регістра проводиться привілейованою командою *MOVEC*. Регістри *SFC, DFC* використовуються при виконуванні процесором

привілейованої команди *MOVEC*, котра зреалізовує пересилання даних поміж регістрами *D*7...0 або *A*7...0 і зовнішньою пам'яттю. Під час записування даних до пам'яті вміст регістра *DFC* використовується в якості функціонального коду *FC*2...0, а при завантажуванні регістрів в якості функціонального коду *FC*2...0 використовується вміст *SFC*. В такий спосіб відбувається організація додаткових банків пам'яті і розширюється загальний адресний простір.

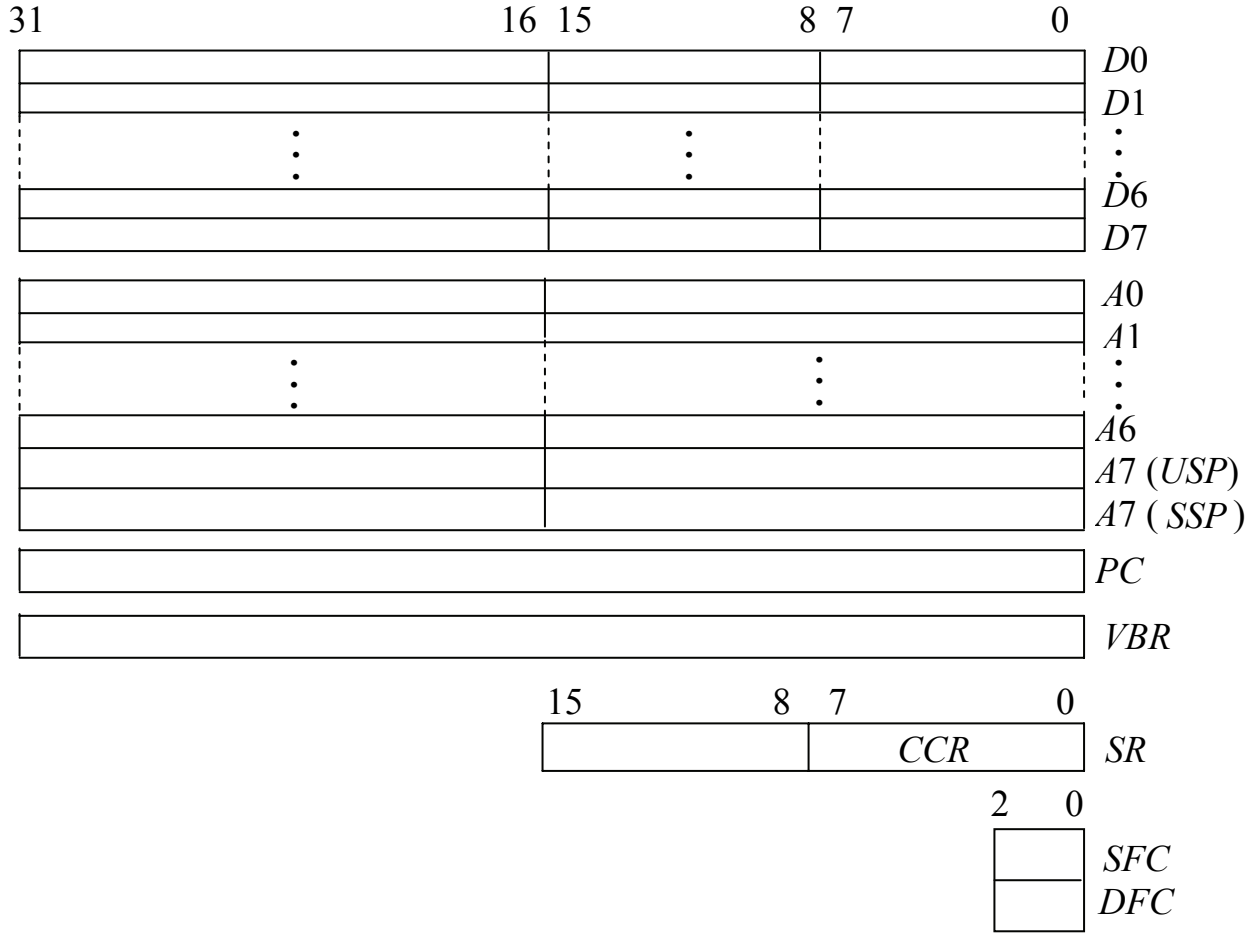



Рисунок 13.27 – Регістрова модель процесора *CPU*32

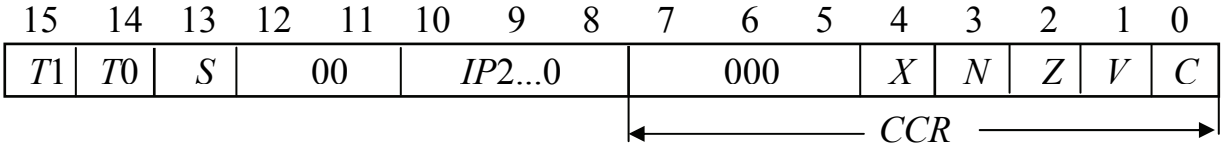



Рисунок 13.28 – Формат вмісту регістра стану *SR*

До регістра стану *SR* входять біти:

*системні ознаки:*

– *T*1 – ознака трасування по програмних переходах; значення *T*1 = 1 зупиняє виконування після кожної команди передавання керування (умовних, безумовних переходів);

– *T*0 – ознака трасування; встановлення біта *T*0 = 1 переключає процесор до покрокового режиму, програма зупиняється після виконання кожної команди;

– *S* – ознака супервізора, значення цієї ознаки *S* = 1 відповідає функціонуванню процесора в режимі супервізора;

– *IP*2-0 – маска запитів переривання, визначає рівень пріоритету обслуговування поточного запиту переривання;

*ознаки користувача:*

– *X* – ознака розширення результату, при виконуванні більшості операцій копіює значення біта *C*; у деяких випадках значення цього біта формується в залежності від виконуваної операції;

– *N* – ознака знаку; зберігає копію знакового розряду результату операції; значення ознаки *N* = 1 відповідає від'ємному результатові;

– *Z* – ознака нульового результату; при виниканні результату, який дорівнює 0, ознака набирає значення 1;

– *V* – ознака переповнювання; набирає значення *V* = 1 в разі переповнювання розрядної сітки при обробці операндів зі знаком;

– *C* – ознака перенесення; здійснює перенесення зі старшого розряду при виконуванні арифметичних операцій та зберігає вміст розряду, який було висунуто, при виконуванні операцій зсувів; набирає значення *C* = 1 при виниканні перенесення зі старшого розряду результату виконуваної операції.

**Контрольні питання:**

1 Чим відрізнюються мікроконтролери різних моделей фірми *Motorola*?

2 Які пристрої можуть входити до складу мікроконтролерів?

3 Які регістри входять до складу процесорного ядра *CPU*05 і яке є функціональне призначення кожного з них?

4 Які ознаки результату формуються при роботі МК сімейства *M*68*HC*05/705?

5 Які послідовні і паралельні порти входять до складу МК сімейства *M*68*HC*05/705?

6 Який обсяг має стек в МК сімейства *M*68*HC*05/705 і яку адресу має вказівник вершини при початковому завантаженні?

7 Яке призначення має блок конфігурації і які регістри до нього входять в МК сімейства *M*68*HC*05/705?

8 В чому полягає функціональне призначення бітів регістра *MOR* МК *MC*68*HC*705*J*1*A* і в який спосіб відбувається змінювання їхнього вмісту?

9 Які регістри входять до складу блока конфігурування МК *MC*68*HC*705*C*8*A*?

10 Яке призначення має блок контролю функціонування МК сімейства *M*68*HC*05/705?

11 В чому полягає виконання процедури непрограмованого скидання МК сімейства *M*68*HC*05/705?

12 У який спосіб відбувається керування процесом обміну даними через асинхронний порт МК сімейства *M*68*HC*05/705?

13 У який спосіб відбувається керування процесом обміну даними через синхронний порт МК сімейства *M*68*HC*05/705?

14 Які сигнали формуються при організації синхронного обміну через порт *SPI* МК сімейства *M*68*HC*05/705?

15 У який спосіб побудовано багатофункціональний таймер в МК сімейства *M*68*HC*05/705?

16 Призначення бітів регістра керування-стану блока *MFT*?

17 У який спосіб визначаються часові інтервали встановлення ознаки періодичного переривання в блоці *MFT*?

18 Де зберігається адреса повернення до головної програми в період виконування підпрограми?

19 Чим відрізнюються системи синхронного та асинхронного послідовного обміну?

20 Які сигнали використовуються для організації синхронного обміну даними через порт *SPI*?

21 Які два генератори імпульсів входять до складу модуля формування тактових сигналів *CGM* 08?

22 Яке призначення має модуль системної інтеграції *SIM* 08?

23 Чим відрізнюються регістрові моделі процесорів *CPU*05 та 68*HC*11?

24 Які особливості побудови властиві 16-розрядним МК?

25 Які пристрої входять до складу МК сімейства 68*HC*16/916 і яке вони мають призначення?

26 Які регістри входять до складу регістрової моделі процесора *CPU*16?

27 Які регістри входять до складу регістрової моделі процесора *CPU*32?

28 Які МК можуть виконувати операції цифрової обробки сигналів?

**Контрольні питання підвищеної складності:**

1 У який спосіб можна запрограмувати МК *MC*68*HC*705*J*1*A* для роботи із зовнішним генератором тактових імпульсів?

2 Запрограмуйте МК *MC*68*HC*705*J*1*A* на приймання зовнішніх запитів на переривання по каналах *PA*3...*PA*0.

3 Запрограмуйте режим непрограмного скидання у МК сімейства M68HC05/705.

4 Сформуйте керувальне слово, яке слід записати до регістра *SCCR*1 МК *MC*68*HC*705*C*8*A* задля вмикання приймача при надходженні даних довжиною 1 байт зі значенням одиниці в старшому розряді і для запису значення біта контролю з прийнятого сигналу.

5 Що станеться з прийнятою інформацією, якщо в регістрі стану *SCSR* МК *MC*68*HC*705*C*8*A* буде записано:

| *TDRE* | *TC* | *RDFR* | *IDLE* | *OR* | *NF* | *RE* | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | – |

6 У який спосіб відбувалася робота вартового таймера МК *MC*68*HC*705*J*1*A*, якщо в регістрі керування-стану *TSCR* буде записано:

| *TOF* | *RTIF* | *TOIE* | *RTIE* | *TOFR* | *RTIFR* | *RT*1 | *RT*0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |

7 Назвіть причину перезавантаження МК сімейства *M*68*HC*08/908, якщо у регістрі стану *LVISR* буде записано:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

8 Сформуйте керувальне слово для МК MC68HC11/711 задля забезпечення:

– включення модуля АЦП та його синхронізації від генератора тактових імпульсів МК;
– переривання при надходженні зрізу імпульса на вході *IRQ*#;
– дозволу функціонування схеми контролю тактової частоти;
– встановлення значення коефіцієнта $K_w = 4$.

## 13.2 Система команд мікроконтролерів фірми *Motorola*

**Вхідний контроль:**

1 Які особливості має мова Асемблер порівняно з мовами програмування високого рівня?

2 Які способи адресування операндів мають мікропроцесори фірми *Motorola*?

3 У який спосіб зберігаються слова в комірках пам’яті, які адресують мікропроцесори фірм *Intel* та *Motorola*?

4 Вміст якого регістра змінюється у командах передавання керування?

5 Для чого використовується вміст бітів регістра ознак у командах передавання керування?

### Способи адресування і система команд сімейства *M*68*HC*05/705

Мікроконтролери цього сімейства, відповідно до регістрової моделі і структурної схеми МК (розд. 13.1), можуть зреалізовувати набір з 65-ти команд, які виконують обробку 8-розрядних операндів. Операнди можуть розміщуватися в регістрах *A, X* та пам’яті. Команди мають розмір від одного до 3-х байт: у першому байті розміщується код операції, а другий та третій

забезпечують адресування операндів, тобто можуть бути команди, які не вміщують операндів. МК зреалізовує такі способи адресування:

– регістрове – операнд розміщується в регістрі *A* або *X*;

– непряме регістрове (індексне) – за адресу операнда слугує вміст регістра *X*;

– індексне зі зміщенням – адреса операнда є сумою вмісту регістра *X* та 8- або 16-розрядного зміщення (число без знаку), що задано в другому та третьому байтах команди;

– пряме – адреса операнда визначається значеннями другого та третього байтів команди;

– безпосереднє – 8-розрядний операнд (число) вказується у другому байті команди;

– відносне – використовується лише в командах розгалуження. Адреса наступної команди визначається як сума поточного вмісту програмного лічильника *PC* і 8-розрядного зміщення (число зі знаком), яке розміщено у другому байті команди.

При використовуванні непрямого регістрового і прямого адресування можливе є звернення лише до 256-ти початкових позицій адресного простору, у котрому розміщено всі регістри МК і значна частина ОЗП. Індексне зі зміщенням адресування при використовуванні 8-розрядного зміщення дозволяє звернення до 512-ти початкових позицій адресного простору, при використовуванні 16-розрядного зміщення – до всього адресного простору, але треба, щоби максимальне значення адреси не перевищувало $1*FFF*. При використовуванні відносного адресування можна звертатися до комірки, розташованої на 127 позицій вище або нижче за поточну. Способи адресування, залежно від сформованої адреси, поділяються на довгі та короткі. До коротких належать способи адресування: непряме регістрове, пряме та індексне зі зміщенням при використовуванні 8-розрядного зміщення.

Всі команди, що входять до системи команд сімейства *M*68*HC*05/705, можна поділити на групи відповідно до операцій, які вони виконують. Тобто є команди пересилань, арифметичних та логічних операцій, зсувів, бітових операцій і встановлення ознак, керування програмою та процесором.

До **команд пересилань** належать команди, які виконують завантажування операндів з пам’яті до регістрів *A, X* або завантажування комірок пам’яті з цих регістрів, пересилання операндів поміж регістрами *A* та *X*, а також обнулення вмісту регістрів *A, X* і комірок пам’яті, що адресуються короткими способами адресування. Перелік команд пересилань і способів подання операндів подано у табл. 13.3.

До **команд арифметичних операцій** належать команди, які виконують відповідні арифметичні операції над 8-розрядними операндами, один з котрих розміщується в акумуляторі *A*, а другий міститься у комірці пам’яті, адреса котрої визначається способом адресування в команді. Результат виконання операції неодмінно записується до акумулятора. Операції додавання та віднімання при виконуванні можуть використовувати значення біта перенесення *C*. До таких операцій також належать команди інкрементування та

декрементування, котрі, відповідно, зреалізовують додавання (віднімання) 1 до (від) вмісту акумулятора, індексного регістра або комірки пам’яті.

Таблиця 13.3 – Перелік команд пересиланнь

| Назва | Мнемокод | Операція | Спосіб адресування |
|---|---|---|---|
| Команди завантаження регістра *A* | *LDA #opr* | *#opr → A* | Безпосереднє |
| | *LDA $addr* | *(M) → A* | Пряме коротке |
| | *LDA $addr* | *(M) → A* | Пряме довге |
| | *LDA ,X* | *(M) → A* | Непряме регістрове |
| | *LDA $addr,X* | *(M) → A* | Індексне зі зміщенням коротке |
| | *LDA $addr,X* | *(M) → A* | Індексне зі зміщенням довге |
| Команди завантаження регістра *X* | *LDX #opr* | *#opr → X* | Безпосереднє |
| | *LDX $addr* | *(M) → X* | Пряме коротке |
| | *LDX $addr* | *(M) → X* | Пряме довге |
| | *LDX ,X* | *(M) → X* | Непряме регістрове |
| | *LDX $addr,X* | *(M) → X* | Індексне зі зміщенням коротке |
| | *LDX $addr,X* | *(M) → X* | Індексне зі зміщенням довге |
| Команди завантаження комірки пам’яті з регістра *A* | *STA $addr* | *(A) → M* | Пряме коротке |
| | *STA $addr* | *(A) → M* | Пряме довге |
| | *STA ,X* | *(A) → M* | Непряме регістрове |
| | *STA $addr,X* | *(A) → M* | Індексне зі зміщенням коротке |
| | *STA $addr,X* | *(A) → M* | Індексне зі зміщенням довге |
| Команди завантаження комірки пам’яті з регістра *X* | *STX $addr* | *(X) → M* | Пряме коротке |
| | *STX $addr* | *(X) → M* | Пряме довге |
| | *STX ,X* | *(X) → M* | Непряме регістрове |
| | *STX $addr,X* | *(X) → M* | Індексне зі зміщенням коротке |
| | *STX $addr,X* | *(X) → M* | Індексне зі зміщенням довге |
| Пересилання вмісту регістра A до регістра *X* | *TAX* | *(A) → X* | Регістрове |
| Пересилання вмісту регістра X до регістра *A* | *TXA* | *(X) → A* | Регістрове |
| Обнулення вмісту регістра A | *CLRA* | $00 → *A* | Регістрове |
| Обнулення вмісту регістра *X* | *XLRX* | $00 → *X* | Регістрове |
| Обнулення вмісту комірки пам’яті | *CLR $addr* | $00 → *M* | Пряме коротке |
| | *CLR, X* | $00 → *M* | Непряме регістрове |
| | *CLR $addr,X* | $00 → *M* | Індексне зі зміщенням коротке |

МК також зреалізовує операцію беззнакового множення 8-розрядних даних, які розміщуються в акумуляторі *A* та індексному регістрі *X*; результат

множення записується до пари регістрів *X:A,* отже, старший байт неодмінно записується до регістра *X*.

До команд арифметичних операцій також належать команди порівняння і тестування операндів. Існують дві групи таких команд: порівняння вмісту акумулятора *A* і вмісту комірки пам'яті та вмісту регістра *X* і комірки пам'яті. Команди обох груп виконуються як віднімання від вмісту відповідного регістра вмісту комірки пам'яті, при цьому прапорці змінюються відповідно до результату, а результат не формується.

Тестування виконується для операндів, які можуть розміщуватися в пам'яті, акумуляторі та індексному регістрі. Їхнє виконування полягає у відніманні числа $00 від операнда. При цьому формуються ознаки, що відповідають самому операндові.

Команди зміни знаку (формування доповнювального коду) також належать до цієї групи. В них можуть використовуватися лише короткі способи адресування операндів.

Перелік команд арифметичних операцій подано у табл. 13.4.

Таблиця 13.4 – Арифметичні команди

| Назва | Мнемокод | Операція | Спосіб адресування |
|---|---|---|---|
| Додавання з перенесенням | *ADC opr* | *(A)+(M)+C→A* | Використовуються всі способи адресування і подання операндів, як показано в табл. 13.3 |
| Додавання | *ADD opr* | *(A)+(M)→A* | |
| Віднімання з позикою | *SBC opr* | *(A)–(M)–C→A* | |
| Віднімання | *SUB opr* | *(A)–(M)→A* | |
| Інкрементування комірки пам'яті | *INC $addr* | (*M*)+1→*M* | Пряме коротке |
| | *INC ,X* | | Непряме регістрове |
| Інкрементування регістра *A* | *INCA* | (*A*)+1→*A* | Регістрове |
| Інкрементування регістра *X* | *INCX* | (*X*)+1→*X* | Регістрове |
| Декрементування комірки пам'яті | *DEC $addr* | (*M*)–1→*M* | Пряме коротке |
| | *DEC ,X* | | Непряме регістрове |
| Декрементування регістра *A* | *DECA* | (*A*)–1→*A* | Регістрове |
| Декрементування регістра *X* | *DECX* | (*X*)–1→*X* | Регістрове |
| Беззнакове множення | *MUL* | *(X)×(A)→X:A* | Як показано в табл. 13.3 |

Закінчення табл. 13.4

| Назва | Мнемокод | Операція | Спосіб адресування |
|---|---|---|---|
| Порівняння операндів | *CMP opr* | *(A)–(M)* | Використовуються всі способи адресування і подання операндів, як показано в табл. 13.3 |
| | *CPX opr* | *(X)–(M)* | |
| Тестування операндів | *TST $addr* | (*M*)–$00 | Пряме коротке |
| | *TST ,X* | | Непряме регістрове |
| | *TST $addr,X* | | Індексне зі зміщенням коротке |
| | *TSTA* | (*A*)–$00 | Регістрове |
| | *TSTX* | (*X*)–$00 | Регістрове |

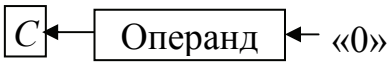
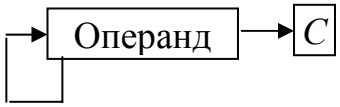
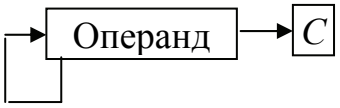
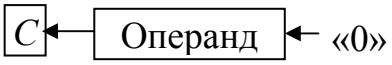
До **логічних операцій,** які виконує МК *M*68*HC*05/705, належать команди логічного множення (кон'юнкції), логічного додавання (диз'юнкції), виключного АБО та логічного інвертування, які виконуються з 8-розрядними операндами. Команда бітового тестування, котра також належить до цієї групи, виконує логічне множення операндів, але не формує результату. Перелік команд логічних операцій подано у табл. 13.5.

Таблиця 13.5 – Команди логічних операцій

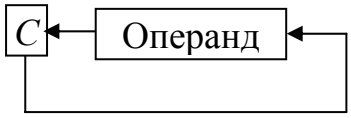
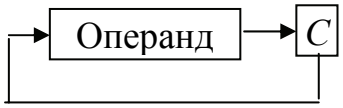
| Назва | Мнемокод | Операція | Спосіб адресування |
|---|---|---|---|
| Логічне множення | *AND opr* | $A \wedge (M) \rightarrow A$ | Використовуються всі способи адресування і подання операндів, як показано в табл. 13.3 |
| Логічне додавання | *OR opr* | $A \vee (M) \rightarrow A$ | |
| Виключне АБО | *EOR opr* | $A \veebar (M) \rightarrow A$ | |
| Логічна інверсія операнда | *COM $addr* | *$FF–(M)→M* | Пряме коротке |
| | *COM ,X* | | Непряме регістрове |
| | *COM $addr,X* | | Індексне зі зміщенням коротке |
| | *COMA* | *$FF–(A)→A* | Регістрове |
| | *COMX* | *$FF–(X)→X* | Регістрове |

До групи **команд зсувів,** які зреалізовує МК *M*68*HC*05/705, належать команди арифметичних, логічних та циклічних зсувів, котрі виконуються над вмістом регістрів *A, X* та комірок пам'яті, що адресуються короткими способами адресування. Перелік команд зсувів та особливості їхнього виконування подано в табл. 13.6.

Таблиця 13.6 – Перелік команд зсувів

| Назва | Синтаксис Асемблера | Способи адресування | Операція |
|---|---|---|---|
| Арифметичний зсув ліворуч | *ASL $addr* | Пряме коротке | |
| | *ASL ,X* | Непряме регістрове | |
| | *ASL $addr,X* | Індексне зі зміщенням коротке | |
| Арифметичний зсув ліворуч вмісту *A* | *ASLA* | Регістрове |   |
| Арифметичний зсув ліворуч вмісту *X* | *ASLX* | Регістрове | |
| Арифметичний зсув праворуч | *ASR $addr* | Пряме коротке | |
| | *ASR ,X* | Непряме регістрове | |
| | *ASR $addr,X* | Індексне зі зміщенням коротке |   |
| Арифметичний зсув праворуч вмісту *A* | *ASRA* | Регістрове | |
| Арифметичний зсув праворуч вмісту *X* | *ASRX* | Регістрове |   |
| Логічний зсув ліворуч | *LSL $addr* | Пряме коротке | |
| | *LSL, X* | Непряме регістрове | |
| | *LSL $addr,X* | Індексне зі зміщенням коротке |   |
| Логічний зсув ліворуч вмісту *A* | *LSLA* | Регістрове | |
| Логічний зсув ліворуч вмісту *X* | *LSLX* | Регістрове | |
| Логічний зсув праворуч | *LSR $addr* | Пряме коротке | |
| | *LSR ,X* | Непряме регістрове | |
| | *LSR $addr,X* | Індексне зі зміщенням коротке | |
| Логічний зсув праворуч вмісту *A* | *LSRA* | Регістрове | |
| Логічний зсув праворуч вмісту *X* | *LSRX* | Регістрове | |

Закінчення табл. 13.6

| Назва | Синтаксис Асемблера | Способи адресування | Операція |
|---|---|---|---|
| Циклічний зсув ліворуч | *ROL $addr* | Пряме коротке |   |
| | *ROL ,X* | Непряме регістрове | |
| | *ROL $addr,X* | Індексне зі зміщенням коротке | |
| Циклічний зсув ліворуч вмісту *A* | *ROLA* | Регістрове | |
| Циклічний зсув ліворуч вмісту *X* | *ROLX* | Регістрове | |
| Циклічний зсув праворуч | *ROR $addr* | Пряме коротке |   |
| | *ROR ,X* | Непряме регістрове | |
| | *ROR $addr,X* | Індексне зі зміщенням коротке | |
| Циклічний зсув праворуч вмісту *A* | *RORA* | Регістрове | |
| Циклічний зсув праворуч вмісту *X* | *RORX* | Регістрове | |

**Команди бітових операцій**, які входять до системи команд МК M68HC05/705, встановлюють потрібне значення (0 або 1) відповідного біта $b_n$ у 8-розрядному операнді, адреса котрого вміщується у другому байті команди (використовується лише пряме адресування). Номер біта *n* задається в команді. До цієї групи команд також належать команди встановлення необхідного значення прапорців *C* та *I* в регістрі ознак *CCR*. Перелік команд бітових операцій подано в табл. 13.7.

Таблиця 13.7 – Команди бітових операцій

| Назва | Синтаксис Асемблера | Операція |
|---|---|---|
| Встановлення потрібного значення біта з номером n = (0...7) | *BCLR n,$addr* | $0 \rightarrow b_n$ |
| | *BSET n,$addr* | $1 \rightarrow b_n$ |
| Встановлення потрібного значення прапорця *C* | *CLC* | $0 \rightarrow C$ |
| | *SEC* | $1 \rightarrow C$ |
| Встановлення потрібного значення прапорця *I* | *CLI* | $0 \rightarrow I$ |
| | *SEI* | $1 \rightarrow I$ |

До групи **команд керування програмою** входять команди, які зреалізовують:

– безумовний перехід у програмі;

– умовні переходи (умовні та безумовні розгалуження);
– перехід (розгалуження) до підпрограми;
– вихід з підпрограми;
– програмне переривання;
– повернення з програми обслуговування переривання.

Перелік команд цієї групи наведено у табл. 13.8.

Таблиця 13.8 – Команди керування програмою

| Назва | Мнемокод | Операція |
|---|---|---|
| Безумовний перехід | *JMP $addr* | $(EA)_{\text{корот.}} \rightarrow PC$ |
| | *JMP $addr* | $(EA)_{\text{довг.}} \rightarrow PC$ |
| | *JMP ,X* | $(X) \rightarrow PC$ |
| | *JMP $addr,X* | $(X) + \$addr_{\text{корот.}} \rightarrow PC$ |
| | *JMP $addr,X* | $(X) + \$addr_{\text{довг.}} \rightarrow PC$ |
| Умовний перехід | $B_{CC}$ *$rel8* | $(PC)+2+\$rel8 \rightarrow PC$, якщо умова виконується |
| Безумовне розгалуження | *BRA $rel8* | $(PC)+2+\$rel8 \rightarrow PC$ |
| Відсутність розгалуження | *BRN $rel8* | $(PC)+2 \rightarrow PC$ |
| Розгалуження при встановленні відповідного біта n | *BRSET n,$addr,rel8* | $(PC)+2+rel8 \rightarrow$ PC, якщо $b_n = 1$ |
| | *BRCLR n,$addr,rel8* | $(PC)+2+$rel8 $\rightarrow PC$, якщо $b_n = 0$ |
| Перехід до підпрограми | *JSR $addr* | $(PC)+n \rightarrow PC$<br>$(PCl) \rightarrow SP;\ SP - 1 \rightarrow SP$<br>$(PCh) \rightarrow SP;\ SP - 1 \rightarrow SP$<br>$(EA) \rightarrow PC$ |
| | *JSR $addr* | |
| | *JSR ,X* | |
| | *JSR $addr,X* | |
| | *JSR $addr,X* | |
| Розгалуження до підпрограми | *BSR $rel8* | $(PC)+2 \rightarrow PC$<br>$(PCl) \rightarrow SP;\ SP - 1 \rightarrow SP$<br>$(PCh) \rightarrow SP;\ SP - 1 \rightarrow SP$<br>$(PC)+2+rel8 \rightarrow PC$ |
| Вихід з підпрограми | *RTS* | $SP+1 \rightarrow SP;\ (SP) \rightarrow PCh$<br>$SP+1 \rightarrow SP;\ (SP) \rightarrow PCl$ |
| Програмне переривання | *SWI* | $(PC)+1 \rightarrow PC$<br>$(PCl) \rightarrow SP;\ (SP) - 1 \rightarrow SP$<br>$(PCh) \rightarrow SP;\ (SP) - 1 \rightarrow SP$<br>$(X) \rightarrow SP;\ (SP) - 1 \rightarrow SP$<br>$(A) \rightarrow SP;\ (SP) - 1 \rightarrow SP$<br>$(CCR) \rightarrow SP;\ (SP) - 1 \rightarrow SP$<br>$1 \rightarrow I(CCR)$;<br>Ст. байт $Ve \rightarrow PCh$<br>Мл. байт $Ve \rightarrow PCl$ |

Закінчення табл. 13.8

| Назва | Мнемокод | Операція |
|---|---|---|
| Вихід з програми обслуговування переривання | *RTI* | *SP*+1→*SP; (SP)→CCR*<br>*SP*+1→*SP; (SP)→A*<br>*SP*+1→*SP; (SP)→X*<br>*SP*+1→*SP; (SP)→PCh*<br>*SP*+1→*SP; (SP)→PCl* |

Команда *JMP* завантажує до програмного лічильника адресу переходу, відповідно до використовуваного способу адресування.

В командах умовного переходу і розгалужень використовується лише відносний спосіб адресування. Адреса переходу формується як сума поточного вмісту програмного лічильника, збільшеного на 2 (задля звернення до наступної виконуваної команди в програмі), і 8-розрядного зміщення ($*rel*8) відносно початкової адреси розміщення програми, яке є числом зі знаком. Умови, що перевіряються, наведено в табл. 13.9. Команда безумовного розгалуження *BRA* є аналогом команди безумовного переходу з використанням відносного адресування, а команда відсутності розгалуження BRN є аналогічна до команди *NOP*. Вона пропускає у програмі два байти, після чого триває виконування програми. Команду *BRN* зручно використовувати при налагодженні програм замість команди $B_{CC}$ задля виконування програми без розгалуження.

Таблиця 13.9 – Мнемокоди і умови виконування команд умовних переходів

| Мнемокод умови *CC* | Умова, що перевіряється | Логічний вираз |
|---|---|---|
| *NE* | Не дорівнює (ненульовий результат) | *Z* = 0 |
| *EQ* | Дорівнює (нульовий результат) | *Z* = 1 |
| *HI* | Вище | (*Z* + *C*) = 0 |
| *LS* | Нижче або дорівнює | (*Z* + *C*) = 1 |
| *HS* | Вище або дорівнює (немає перенесення) | *C* = 0 |
| *LO* | Нижче (є перенесення) | *C* = 1 |
| *PL* | Додатний результат | *N* = 0 |
| *MI* | Від’ємний результат | *N* = 1 |
| *HCC* | Немає міжтетрадного перенесення | *H* = 0 |
| *HCS* | Є міжтетрадне перенесення | *H* = 1 |
| *MC* | Переривання дозволено | *I* = 0 |
| *MS* | Переривання заборонено | *I* = 1 |
| *IH* | Відсутність запиту переривання | *IRQ*# = 1 |
| *IL* | Надходження запиту переривання | *IRQ*# = 0 |

Команди розгалуження при встановленні відповідного біта n *BRSET* і *BRCLR* перевіряють значення цього біта в операнді, визначеному за допомогою

8-розрядного прямого адресування. Зміщення в цих командах (\$*rel*8) визначається відносно початкової адреси розміщення програми.

Команди переходу і умовного переходу до підпрограм *JSR* та *BSR* зберігають у стеку адресу команди повернення до головної програми і завантажують до програмного лічильника *PC* початкову адресу підпрограм, котра визначається за допомогою прямого, індексного чи індексного зі зміщенням адресування. За виконування команди *JSR* адреса, що завантажується до стека, має бути більшою за вміст регістра *PC* на 1, 2 або 3, залежно від способу адресування. За використовування індексного адресування це є 1, за індексного адресування з 8-розрядним зміщенням чи короткого прямого – 2 і за адресування з 16-розрядним зміщенням і довгого прямого – 3. В команді *BSR* використовується відносне адресування. При поверненні до головної програми вміст регістра *PC* відновлюється зі стека.

Команда програмного переривання *SWI* до завантаження початкової адреси підпрограми обслуговування програмного переривання зберігає в стеку: значення програмного лічильника *PC*, акумулятора *A*, індексного регістра *X* та регістра ознак *CCR*. Потім встановлює значення біта дозволу переривання $I = 1$ і побайтно завантажує до програмного лічильника значення вектора переривань $V_e$. Команда повернення відновлює всі значення, які було завантажено до стека.

До **команд керування процесором** належать команди, які подано у табл. 13 10.

Таблиця 13.10 – Команди керування процесором

| Назва | Мнемокод | Операція |
|---|---|---|
| Встановлення початкового значення вказівника стека | *RSP* | \$00*FF*→*SP* |
| Відсутність операцій | *NOP* | (*PC*)+1 → *PC* |
| Перехід до режиму очікування | *WAIT* | Зупинення процесора 0→ *I* |
| Перехід до режиму зупину | *STOP* | Зупинення генератора тактових імпульсів 0→ *I* |

Команда встановлення початкового значення вказівника стека завантажує адресу \$00*FF* до регістра *SP*.

Команди *WAIT* та *STOP* переводять МК до режиму енергозберігання з можливістю повернення до нормального режиму, для чого при їхньому виконуванні встановлюються значення ознаки дозволу переривань $I = 0$.

**Контрольні питання:**

1 Які прапорці виставляються при виконуванні арифметичних операцій?

2 Які прапорці виставляються при виконуванні логічних операцій?

3 Які способи адресування використовуються в мові Асемблер МК фірми *Motorola*?

4 У який спосіб поділяються способи адресування залежно від сформованої адреси? Чим вони відрізнюються?

5 За допомогою яких команд можна виконувати завантаження регістра акумулятора?

6 Які способи адресування можна використовувати задля зберігання вмісту індексного регістра *X* в пам'яті?

7 Чим відрізнюється використовування команд *CMP* та *TEST*?

8 Наведіть приклади виконування команд *ASRA* та *LSRA* з операндом, який дорівнює $82 на 2 розряди, і поясніть результати.

9 До якого регістра буде завантажено адресу переходу при виконуванні команди *JMP ,X?*

10 У який спосіб формується адреса нового вмісту програмного лічильника *PC* при виконуванні команди *BRSET n,$addr,rel*18?

**Контрольні питання підвищеної складності:**

1 Які дії відбуваються в МК при виконуванні команди програмного переривання *SWI*?

2 Які дії відбуваються в МК при виконуванні команди повернення з програми обслуговування переривання *RTI*?

3 Які умови можуть використовуватись у команді умовного переходу $B_{CC}$?

4 Які команди додавання операндів у мові Асемблер МК фірми *Motorola* Ви знаєте? Чим вони відрізнюються одна від одної?

5 Чим різняться команди *SBC* і *SUB*? Поясніть на прикладі.

6 В яких регістрах розміщуються операнди при виконуванні команди *MUL* і де розміщується результат після її виконання?

## 13.3 Налаштовування вбудованих засобів мікроконтролерів

**Вхідний контроль:**

1 Які периферійні пристрої входять до складу мікроконтролера *MC*68*HC*705*J*1*A*?

2 Для чого використовується регістр *MOR* блока конфігурування МК *MC*68*HC*705*J*1*A* і яке призначення мають його окремі розряди?

3 Яке призначення має блок контролю функціонування МК *MC*68*HC*705*J*1*A* і які пристрої входять до його складу?

4 Яка різниця існує поміж послідовними і паралельними портами мікроконтролера?

5 Які порти входять до складу МК *MC*68*HC*705*J*1*A*?

6 Які регістри входять до складу паралельних портів МК *MC*68*HC*705*J*1*A*?

7 За допомогою яких команд можна здійснювати запис до регістра *DDRA*?

8 В який спосіб визначається напрямок обміну через паралельний порт МК *MC*68*HC*705*J*1*A*?

Виводи паралельних портів МК *MC*68*HC*705*J*1*A* можна використовувати для обміну будь-якими сигналами. При використовуванні окремих розрядів можна формувати на них сигнали у послідовному вигляді, тобто МК можна використовувати в якості генератора послідовності імпульсів потрібної частоти і щілинності. Блок-схему алгоритму функціонування МК *MC*68*HC*705*J*1*A* в якості генератора імпульсів подано на рис. 13.29.

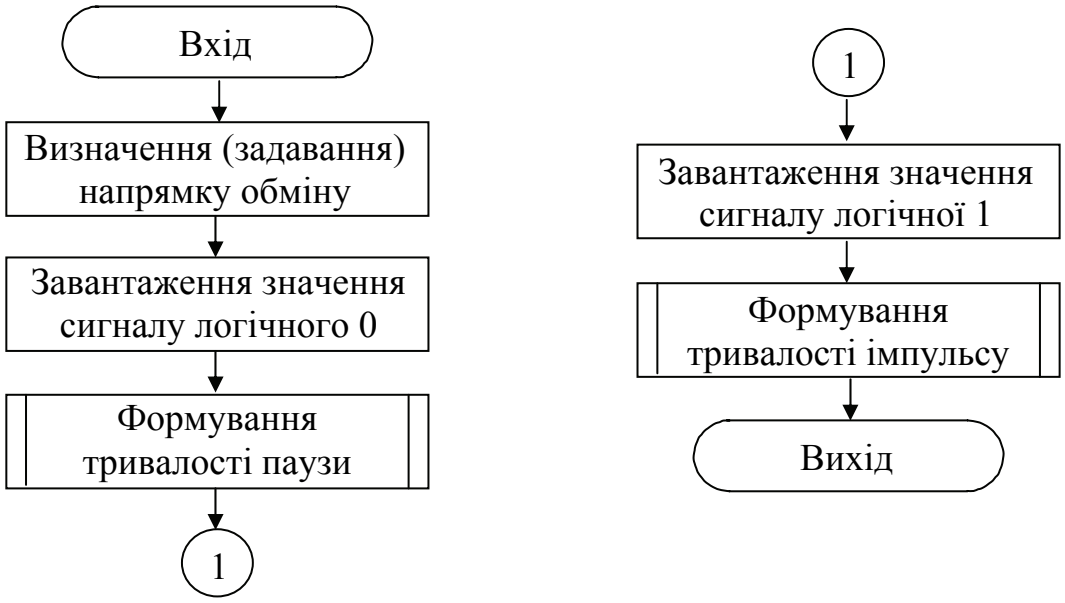



Рисунок 13.29 – Блок-схема алгоритму роботи *MC*68*HC*705*J*1*A* в якості генератора імпульсів

Визначення напрямку обміну зумовлюється встановленням відповідного біта в регістрі напрямку пересилання *DDRA* (адреса $0004) або *DDRB* (адреса $0005), як було показано в розділі 13.1.1. Вважаємо, що це будуть всі 8 виводів порту *A*.

Виведення здійснюється з регістра даних порту *PORTA*, до якого треба завантажити значення сигналу з акумулятора *A*. Виведення даних триватиме упродовж такого відтинку часу, допоки сигнали на виходах порту не змінять свого значення. Тому треба сформувати часовий інтервал, котрий визначатиме час формування імпульсу чи то паузи поміж імпульсами. Цей часовий інтервал формується за допомогою підпрограми формування тривалості; вважаємо, що послідовність розпочинається з паузи (рівень логічного 0). Зазвичай формування часових інтервалів може відбуватися за допомогою таймера чи то у програмний спосіб. При формуванні часового інтервалу в програмний спосіб розроблюється циклічна підпрограма, час виконування якої дорівнює заданому часовому інтервалові. При розв’язуванні нашого завдання ця підпрограма може бути побудована за алгоритмом, який подано на рис. 13.30. На цьому рисунку не зазначено блоки входу та виходу, тому що вважається, що робота цієї підпрограми здійснюється згідно з алгоритмом рис. 13.29.

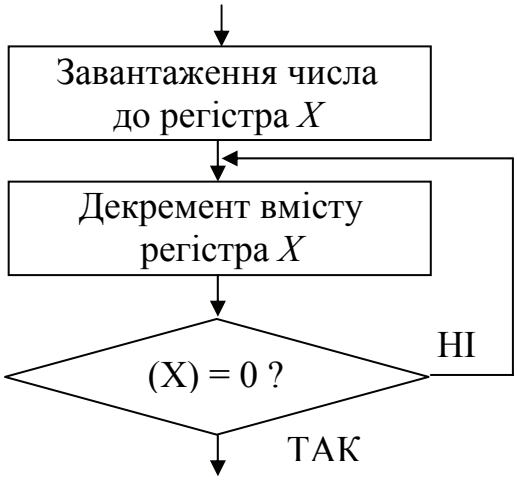



Рисунок 13.30 – Блок-схема алгоритму підпрограми часової затримки

Число, завантажуване до регістра, визначає час затримки підпрограми.

Після виконання цієї підпрограми слід завантажити до регістра даних нове значення сигналу і сформувати часову затримку, що дорівнювала б часові виведення цього значення.

За алгоритмом рис. 13.29, з урахуванням алгоритму рис. 13.30, розроблюємо програму генератора імпульсної послідовності мовою Асемблер:

```
      LDA  #FF      ; Визначення напрямку обміну на виведення
      STA  $0004    ; даних каналом А
M3:   LDA  #00      ; Завантаження рівня сигналу, який
                    ; виводитиметься каналом А
      STA  $0000    ; Виведення каналом А сигналу логічного 0
      JSR  M1       ; Безумовний перехід до підпрограми затримки,
                    ; розташованої за міткою М1
      LDA  #FF      ; Встановлення рівня логічної 1 і виведення цього
      STA  $0000    ; сигналу каналом А
      JSR  M2       ; Безумовний перехід до підпрограми,
                    ; розташованої за міткою М3
      BRA  M3       ; Безумовний перехід до формування нового
                    ; періоду сигналу
M1:   LDX #8        ; Завантаження до індексного регістра Х кількості
                    ; циклів виконування підпрограми формування
                    ; сигналу логічного 0
M4:   DECX          ; Підпрограма формування часового інтервалу
                    ; паузи поміж імпульсами. Декремент вмісту
                    ; регістра Х
      BNE  M4       ; Умовний перехід до команди DECX
      RTS           ; Вихід з підпрограми формування часового
                    ; інтервалу паузи поміж імпульсами. Повернення
                    ; здійснюється до команди LDA #FF, що є
```

```
                    ; наступною за командою звернення до цієї
                    ; підпрограми
M2:  LDX #4         ; Завантаження до індексного регістра X кількості
                    ; циклів виконування підпрограми формування
                    ; сигналу логічної 1
M5:  DECX           ; Підпрограма формування одного імпульсу
                    ; з послідовності
                    ; Декремент вмісту регістра X
     BNE  M5        ; Умовний перехід до команди DECX
     RTS            ; Вихід з підпрограми формування часового
                    ; інтервалу паузи поміж імпульсами. Повернення
                    ; здійснюється до команди BRA
```

Ця програма формуватиме імпульсну послідовність, частота слідування імпульсів в котрій визначатиметься тактовою частотою МК.

Використовуючи таймер, який входить до складу МК, можна також формувати сигнали з потрібною часовою затримкою і використовувати ці сигнали задля керування потрібними об’єктами. Для формування потрібних часових інтервалів можна використовувати сигнали переривання від таймера або від периферійних пристроїв.

Припустімо, що треба розробити програму керування станом об’єкта (періодичне включення/виключеня) по переповнюванні 15-розрядного лічильника багатофункціонального таймера *MFT*, якщо не надходить сигнал зовнішнього переривання. В разі надходження сигналу зовнішнього переривання, який надходить на вивід 0 порту *A*, програма формує сигнал логічної 1 на виводі 7 порту *A*.

Програма, яка функціонуватиме за цим алгоритмом, матиме такий вигляд:

```
     CLRX           ; Обнулення індексного регістра X
     LDA #$80       ; Завантаження сигналу включення, який
                    ; виводитиметься у 8-му розряді порту A
     STA $0000      ; Включення зовнішнього пристрою по каналу A
     LDA  #$FE      ; Завантаження до регістра A коду, який
                    ; визначатиме виводи порта A PA7...PA1 як
                    ; виходи, а вивід PA0 – як вхід
     STA $0004      ; Завантаження коду з регістра A до регістра
                    ; напрямку пересилання DDRA
     BSET 5,$0008   ; Встановлення біта дозволу переривання
                    ; переповнювання лічильника у регістрі TSCR
     CLI            ; Скидання ознаки I в регістрі CCR (дозвіл
                    ; переривання)
M1:  JMP M1         ; Безумовний перехід на ту саму команду задля
                    ; формування затримки формування вихідного
                    ; сигналу до моменту переповнювання лічильника
                    ; у багатофункціональному таймері
```

```
M2:  BSET 3,$0008     ; Встановлення біта TOFR (біт скидання ознаки
                      ; переповнювання) у регістрі TSCR
     LDA $0000        ; Завантаження до акумулятора А вмісту регістра
                      ; даних PORTA (число #$80)
     EOR #$80         ; Обнулення вмісту акумулятора
     STA $0000        ; Відключення зовнішнього пристрою по каналу А
                      ; (виведення рівня логічного 0 каналом А)
     RTI              ; Вихід з підпрограми обслуговування
                      ; переривання (повернення до команди JMP M1)
M3:  BSET 1,$000A     ; Скидання ознаки IRQR (скидання запиту
                      ; на переривання) і скидання ознаки IRQF
                      ; (відсутність запиту на переривання) в регістрі
                      ; ISCR
     BSET 7,$0000     ; Виведення сигналу через вивід 7 PORTA
     RTI              ; Вихід з підпрограми формування сигналу на
                      ; виводі 7 PORTA
```

До складу цієї програми входять дві підпрограми, які позначено мітками *M*2 та *M*3. Підпрограма *M*2 призначена для формування сигналу вимикання зовнішнього пристрою, який підімкнено до порту *A*. Повернення з цієї підпрограми відбувається до команди, позначеної міткою *M*1, яка формує часову затримку до моменту переповнювання лічильника багатофункціонального таймера *MFT*. Переповнювання лічильника призводить до встановлення внутрішнього переривання, яке забезпечує циклічне змінювання стану виводу 8 порту *A*. Для правильного функціонування програми до пам’яті слід завантажувати адреси, до яких здіснюватиметься звернення при виконуванні відповідних переривань.

```
$07F8    03   0F   03   18   03   00   03   00
```

У комірках пам’яті з адресами $07F8 та $07F9 розміщено початкову адресу підпрограми обробки переривання по переповнюванні лічильника – $030*F*, в комірках $07*FA* та $07*FB* – початкову адресу підпрограми обробки зовнішнього переривання – $0318 і в наступних комірках – початкову адресу розміщення всієї програми.

Звернення до підпрограми, позначеної міткою *M*3, здійснюється лише за наявності зовнішнього переривання.

Програма формує часові інтервали, що вони дорівнюють часові виконування 1025 та 1022 машинних тактів МК. Тривалість цих інтервалів можна змінити, встановлюючи потрібні значення *RT*1-0 в регістрі *TSCR*.

Протокол виконування програми в режимі обробки переривання по переповнюванні лічильника наведено в табл. 13.11. В протоколі не зазначено вміст регістрів, які не використовуються при виконуванні цієї програми: регістра даних порту *B*, регістра напрямку пересилання порту B та регістра *PDRB*, а також регістрів *ISCR* та *PDRA*, які не змінюють своїх значень, $80 та

$00 відповідно. В колонці *CYCLES* наведено кількість машинних циклів, яка відповідає виконуванню поточної команди.

Таблиця 13.11 – Протокол виконування програми періодичного включення/відключення периферійного пристрою за сигналом переповнення лічильника

| Команда | *A* | *X* | *PORTA* | *DDRA* | *TSCR* | *TIMER* | *CCR* | *CYCLES* |
|---|---|---|---|---|---|---|---|---|
| CLRX | *XX* | $00 | *XX* | $00 | $03 | $00 | 111.I… | $00 |
| LDA #$80 | $80 | $00 | *XX* | $00 | $03 | $00 | 111.*I.Z.* | $03 |
| STA $0000 | $80 | $00 | *XX* | $00 | $03 | $01 | 111.*IN*.. | $05 |
| LDA #$FE | $80 | $00 | *XX* | $00 | $03 | $02 | 111.*IN*.. | $09 |
| STA $0004 | $FE | $00 | *XX* | $00 | $03 | $02 | 111.*IN*.. | $0*B* |
| Формування сигналу включення | | | | | | | | |
| BSET 5,$0008 | *$FE* | $00 | $80 | *$FE* | $03 | $03 | 111.*IN*.. | $0*F* |
| CLI | *$FE* | $00 | $80 | *$FE* | $23 | $05 | 111.*IN*.. | $14 |
| JMP M1 | *$FE* | $00 | $80 | *$FE* | $23 | $05 | 111..*N*.. | $16 |
| … | | | | | | | | |
| JMP M1 | *$FE* | $00 | $80 | *$FE* | $23 | $*FF* | 111..*N*.. | $3*FD* |
| JMP M1 | *$FE* | $00 | $80 | *$FE* | $*A*3 | $00 | 111..*N*.. | $400 |
| BSET 3,$0008 | *$FE* | $00 | $80 | *$FE* | $*A*3 | $03 | 111.*IN*.. | $40*D* |
| LDA $0000 | *$FE* | $00 | $80 | *$FE* | $23 | $04 | 111.*IN*.. | $412 |
| EOR #$80 | $80 | $00 | $80 | *$FE* | $23 | $05 | 111.*IN*.. | $415 |
| STA $0000 | $00 | $00 | $80 | *$FE* | $23 | $05 | 111.*I.Z.* | $417 |
| RTI | $00 | $00 | $00 | *$FE* | $23 | $06 | 111.*I.Z.* | $41*B* |
| Формування сигналу відключення | | | | | | | | |
| JMP M1 | $*FE* | $00 | $00 | $*FE* | $23 | $09 | 111..*N*.. | $424 |
| … | | | | | | | | |
| JMP M1 | *$FE* | $00 | $00 | *$FE* | $23 | *$FF* | 111..*N*.. | $7*FC* |
| JMP M1 | *$FE* | $00 | $00 | *$FE* | $23 | *$FF* | 111..*N*.. | $7*FF* |
| JMP M1 | *$FE* | $00 | $00 | *$FE* | $*A*3 | $00 | 111..*N*.. | $802 |
| BSET 3,$0008 | *$FE* | $00 | $00 | *$FE* | $*A*3 | $03 | 111.*IN*.. | $807 |
| LDA $0000 | *$FE* | $00 | $00 | *$FE* | $23 | $05 | 111.*IN*.. | $814 |
| EOR #$80 | $00 | $00 | $00 | *$FE* | $23 | $05 | 111.*I.Z.* | $817 |
| STA $0000 | $00 | $00 | $00 | *$FE* | $23 | $06 | 111.*IN*.. | $819 |
| RTI | $80 | $00 | $80 | *$FE* | $23 | $07 | 111.*IN*.. | $81*D* |

**Контрольні питання:**

1 У який спосіб можна задати час затримки, що він формується при виконуванні часової затримки?

2 У який спосіб можна здійснювати вихід з підпрограми часової затримки за різних способів її побудови?

3 Чим визначатиметься час затримки при використовуванні переривання за переповнювання лічильника?

4 Скільки розрядів має регістр-лічильник багатофункціонального таймера *MFT*?

**Контрольні питання підвищеної складності:**

1 В чому полягає призначення бітів *RT*1-0 і як значення їхнього вмісту можна використовувати задля програмування вбудованих засобів?

2 У який спосіб можна визначити час формування МК певного інтервалу?

3 Напишіть програму керування (включення/відключення) певним об’єктом. Включення об’єкта відбувається формуванням сигналу логічної 1 на виводі 7 порту *A*, а відключення – формуванням сигналу логічного 0 на тому ж самому виводі. Включення відбувається при надходженні сигналу зовнішнього переривання, а відключення через певний час затримки, який формується за допомогою багатофункціонального таймера *MFT*.

# 14 RISC-ПРОЦЕСОРИ ФІРМИ MOTOROLA

**Вхідний контроль:**

1 Скільки байтів може займати команда мовою Асемблер *CISC*-процесорів?

2 Які особливості має архітектура *CISC*- та *RISC*-процесорів?

3 За яких способів способи адресування операндів використовують 32-розрядні універсальні мікропроцесори фірми *Motorola*?

4 Які ознаки *CISC*- чи *RISC*-архітектури мають мікропроцесори *Pentium* фірми *Intel*?

5 Скільки регістрів входять до регістрового файла моделі користувача універсальних МП фірми *Motorola*?

## 14.1 *RISC*-процесори *PowerPC*

Назва *RISC (Reduced Instruction Set Computing*) означає “комп’ютер зі зменшеним набором команд”, але в сучасних розробках *RISC*-мікропроцесорів набір команд є значно розширений і може вміщувати команди оброблення даних з плаваючою точкою. Систему команд *RISC*-процесорів можна схарактеризувати як таку, яка має фіксований формат команд, що значно спрощує керувальний пристрій та зменшує обсяг мікропрограмної пам’яті і призводить до зменшення розміру кристала *RISC*-процесорів, зниження їхньої вартості та підвищення тактової частоти. Система команд є вільна від складних та рідко використовуваних команд і способів адресування. Введення фіксованого набору команд забезпечує високу ефективність роботи конвеєра, зменшує кількість тактів простоювання та очікування процесорів, що підвищує їхню ефективність.

Переваги та ефективність *RISC*-процесорів зумовили їхнє виробництво усіма провідними фірмами, які випускають мікропроцесори: *Motorola, Intel, Hewlett-Packard, IBM, Sun Microsystems, MIPS*.

За найоптимальніші *RISC*-процесори та *RISC*-контролери вважаються розробки консорціуму фірм *Motorola, IBM, Apple Computers – MPC*6*XX PowerPC*, а також розробки фірми *Motorola – MFC*5*XXX ColdFire*.

До основних особливостей *RISC*-процесорів можна віднести:

– розширений обсяг регістрової пам’яті: від 32-х до кількох сотень регістрів загального призначення;

– використовування в командах оброблення даних лише регістрового адресування; команди звернення до пам’яті використовуються лише при завантаженні та зберіганні вмісту регістрів, а також у командах керування програмою;

– відмову від апаратної підтримки складних способів адресування – з постінкрементом, предекрементом та деяких непрямих;

– фіксований формат команд, зазвичай 4 байти, замість змінного формату (від одного до 12-ти байт), що є характерним для *CISC*-мікропроцесорів;

– відсутність команд, які не відповідають прийнятому форматові.

За базову модель МП *PowerPC* можна вважати *MPC*604. У цій моделі, як і в сімействі *MX*68*XXX*, зберігається принцип виокремлювання певних ресурсів задля розв'язування завдань супервізора та користувача. Це означає, що архітектура процесора вміщує окремі регістри, які входять і до моделі користувача і до моделі супервізора, і має привілейовані команди, які виконуються лише в режимі супервізора.

До регістрової моделі користувача (рис. 14.1), яка є спільною для всього сімейства *PowerPC*, входять:

– тридцять два 32-розрядні регістри загального призначення *GPR*31...*GPR*0 задля зберігання цілочисельних операндів;

– тридцять два 64-розрядних регістри *FPR*31…*FPR*0 задля зберігання операндів з плаваючою точкою;

– 32-розрядні регістри прапорців (умов) *CR* та станів *FPSCP* (рис. 14.2, а) при обробці даних з плаваючою точкою;

– 32-розрядні регістри XER, LR, CTR, які використовуються при обробці виключень.

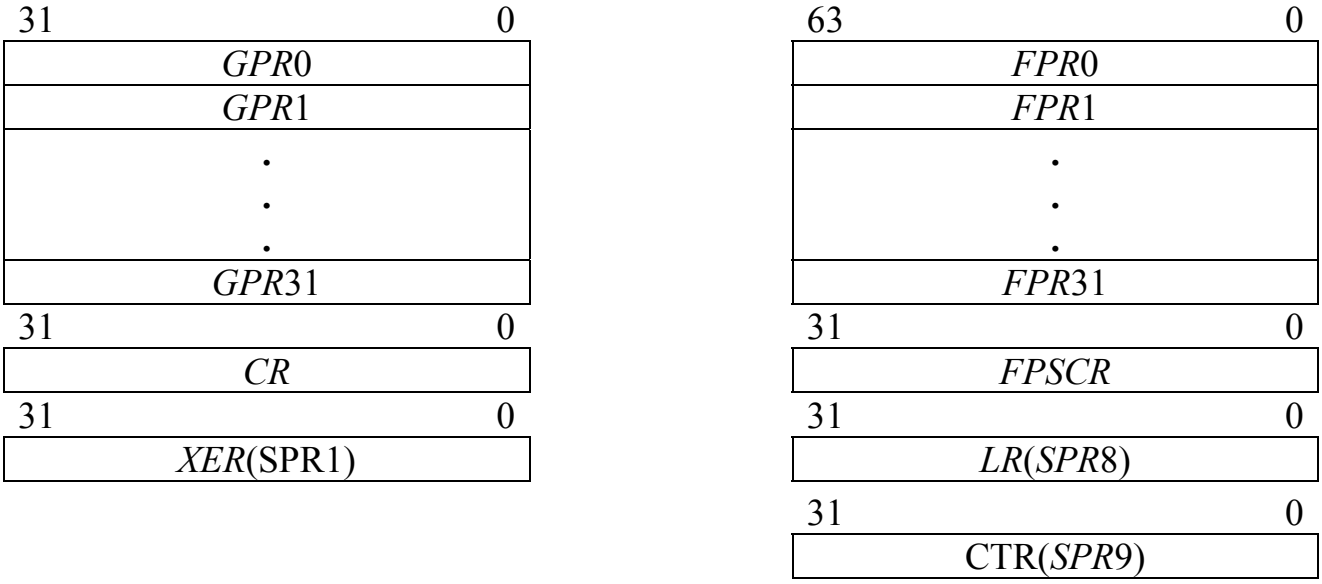
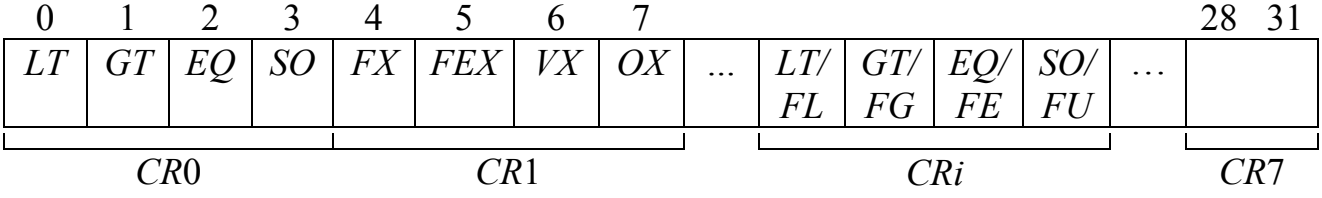
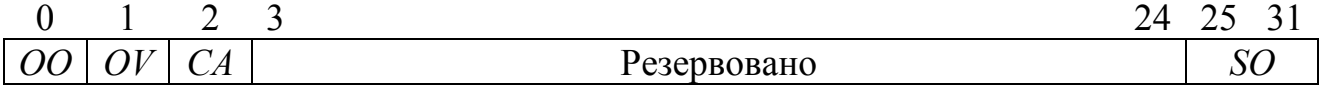



Рисунок 14.1 – Регістрова модель користувача
для мікропроцесорів *PowerPC*




а) CR




б) *XER*

Рисунок 14.2 – Формати вмісту регістрів *CR* та *XER*

Слід звернути увагу на те, що регістрова модель не вміщує ані програмного лічильника, ані вказівника стека.

Регістр прапорців *CR* складається з восьми 4-бітових полів *CR*7...*CR*0, молодший з яких *CR*0 вміщує прапорці, які формуються внаслідок цілочисельних операцій:

– *LT* = 1 за від'ємного результату;
– *GT* = 1 за додатного ненульового результату;
– *EQ* = 1 за нульового результату;
– *SO* – набирає значення аналогічного біта у регістрі *XER* (рис. 14.2, б).

Поле *CR*1 вміщує прапорці *FX, FEX, VX, OX*, які встановлюються при операціях з плаваючою точкою аналогічно регістру *FPSCR*. Вміст інших полів – *CR*7...*CR*2 – встановлюється за результатом операцій порівняння цілих чи дійсних, при операціях з плаваючою точкою, чисел. При порівнянні цілих чисел формуються прапорці: *LT* (менш), *GT* (понад), *EQ* (дорівнює), *SO*. При порівнянні дійсних чисел встановлюються аналогічні прапорці: *FL* (менш), *FG* (понад), *FE* (дорівнює), а також *FU*, який встановлюється у 1, коли один з порівнюваних операндів не є число. Наявність кількох результатів порівняння дозволяє водночас зберігати їх і використовувати за потреби.

Нумерацію розрядів *RISC*-процесорів *PowerPC* прийнято в напрямку ліворуч → праворуч, розпочинаючи з 0-го розряду, який є старшим, тобто *D*0 – старший розряд даних, а *D*31 – молодший. Відповідно, вміст регістрів мікропроцесора подається у форматі, коли старший біт має номер 0, а молодший – 31.

Регістр *XER* вміщує прапорці перенесення *CA*, переповнення *OV*, які встановлюються при виконуванні арифметичних цілочисельних операцій, та прапорець загального переповнення *S*0, який зберігається таким, що дорівнює 1, до виконування команд завантаження до регістра нового даного.

Регістр зв'язку *LR* зберігає адреси команд умовного та безумовного переходів чи звернення до підпрограми.

Регістр-лічильник *CTR* вміщує задану кількість циклів і при виконуванні чергового циклу його вміст зменшується на 1.

На рис. 14.1 у дужках подано також номери певних поіменованих регістрів, які використовуються у певних командах звернення до цих регістрів як до службових.

Регістрова модель супервізора, окрім регістрової моделі користувача, вміщує кілька груп службових регістрів, основним з яких є регістр керування *MSR*, який визначає режим функціонування процесора. Режими функціонування процесора можуть бути такими:

– зі зниженим енергоспоживанням;
– зі встановленням порядку *big-endian* або *little-endian* оброблення байтів при виключеннях;
– з обслуговуванням зовнішніх запитів переривань;
– режим користувача чи режим супервізора;
– режим виконування операцій над числами з плаваючою точкою;
– режим трасування;

– режим задавання базової адреси векторів переривань;

– режим дозволу трансляції адрес команд та даних за допомогою пристроїв керування пам’яттю *IMMU, DMMU*;

– режим з контролем продуктивності процесора тощо.

У процесорі передбачено заходи щодо контролю функціонування кешів команд та даних. Можливе є задавання режимів послідовного та паралельного виконування команд у суперскалярній структурі, статичного та динамічного передбачення подальшого перебігу програми.

Окремі службові регістри мікропроцесора визначають серійний номер та код ідентифікації процесора в мультипроцесорних системах.

Процесор вміщує таймер базового часу, який перемикається тактовою частотою.

На рис. 14.3 подано структуру МП *MPC*604. Процесор має суперскалярну структуру, яка складається з шести виконавчих пристроїв, які працюють паралельно:

– блок оброблення розгалужень *BPU*;

– два блоки задля виконування одноциклових цілочисельних операцій *SIU*1, *SIU*2;

– один пристрій задля виконування багатоциклових цілочисельних операцій *MIU*;

– пристрій оброблення даних з плаваючою точкою *FPU*;

– блок вибирання операндів з пам’яті *LSU*.

Суперскалярна структура забезпечує одночасне виконування чотирьох команд.

Через суперскалярну структуру процесора є можливе звернення виконавчих пристроїв одночасно до одних і тих самих регістрів. З метою зниження помилок за спроби запису до регістра введено буферні регістри, які затримують запис доти, допоки інший пристрій не зчитає старі дані. Вони слугують задля проміжного зберігання операндів, які дублюють регістри *GPR*, *FPR*, використовувані при виконуванні поточних операцій.

Після завершення цих операцій здійснюється перезапис результатів у *GPR* та *FPR*, так званий зворотний запис.

Конвеєр виконування команд має шість основних каскадів: вибирання команди (1), дешифрування (2), розподілення команд поміж виконавчими пристроями (3), виконування команд (4), запису результатів до буфера (5) та зворотного запису результатів до регістрів *SPR* чи *FPR*. Пристрій керування одночасно вибирає з кеша і декодує чотири команди, які розподіляються по відповідних виконавчих пристроях.

Більшість команд виконуються за один такт. Ці команди зреалізовуються за допомогою *SIU*1 та *SIU*2, які здійснюють додавання, віднімання, порівняння, зсуви, логічні операції. Цілочисельне множення здійснюється за 3 або 4 такти, ділення – за 20 тактів. Ці операції виконує пристрій *MIU*. Більшість команд з плаваючою точкою, окрім ділення, виконується за три такти у трьох виконавчих каскадах, які складають структуру *FPU*. Правильна послідовність

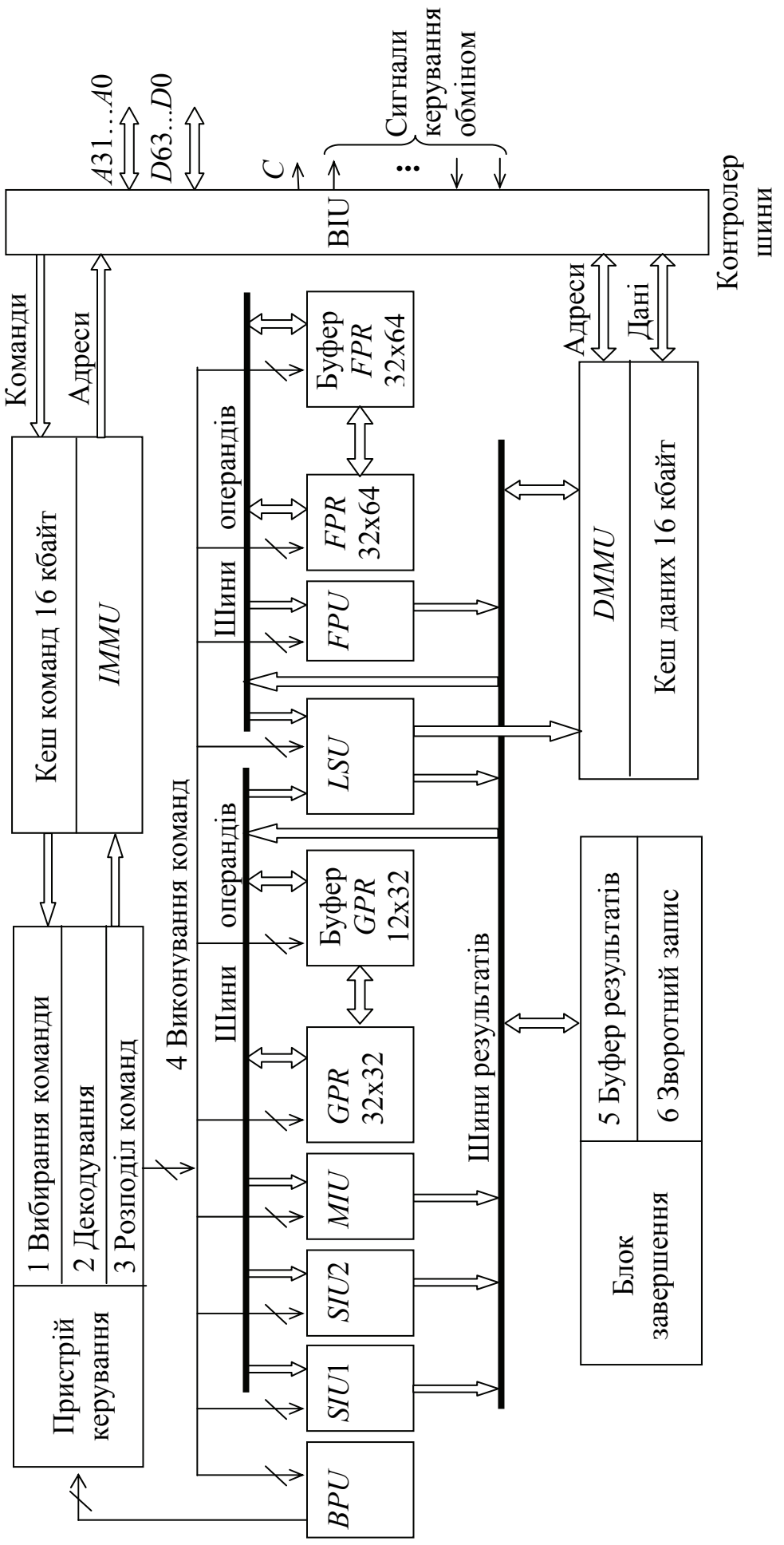



Рисунок 14.3 – Структура мікропроцесора *MPC*604

здобутих результатів виконування операцій збирається у спеціальному блоці завершення, який побудовано на підгрунті буферної пам'яті типу черги. Зворотний запис результатів з буфера до регістрів *GPR, FPR* здійснюється відповідно до порядку подавання команд.

Блок оброблення розгалужень *BPU* завбачає напрямок розгалуження за алгоритмами статичного та динамічного завбачання, які ґрунтуються відповідно на тексті програми або на результатах попередніх розгалужень. Алгоритми завбачання використовують для прийняття рішень кеш адрес розгалужень *BTAC* та таблицю історії розгалужень BHT. Кеш BTAC зберігає 64 адреси переходів, які виконувались попередніми командами. Якщо виконувана команда переходів вміщує адресу, яка збігається з вже записаною у BTAC, то процесор вказує на розгалуження за цією адресою, тобто перехід здійснюється, а якщо ж адреси у таблиці немає, – то не здійснюється.

У разі надходження команди умовного переходу *BPU* аналізує таблицю BHT, яка є кеш-пам'яттю, у ній зберігаються 512 адрес попередніх умовних переходів. Для кожної адреси у таблиці є два біти, які визначають ймовірність розгалуження за вказаною адресою: 00 – практично не виконується, 01 – виконується рідко, 10 – виконується часто, 11 – виконується надто часто. Значення цих бітів змінюються після кожної операції умовного переходу: якщо розгалуження відбулося за зазначеною адресою, то ймовірність збільшується на 1 (максимальне значення становить 11); якщо дана адреса не використовується, то ймовірність зменшується на 1 (мінімальне значення 00). Замість адрес з нульовою ймовірністю вводяться нові адреси з команд умовних переходів, які надходять. Розгалуження завбачається, якщо його ймовірність становить 01 або 11. Правильність завбачання перед виконуванням додатково перевіряється після визначення відповідних прапорців у регістрі *CR*. Завбачена команда за адресою, вказаною у команді умовного переходу, вибирається у конвеєр, тобто розгалуження здійснюється; якщо ж адресу переходу не записано в таблиці BTAC, то виконується наступна команда програми. При перевірці виконування умов переходу за прапорцями у регістрі *CR*, якщо умову не виконано, тобто завбачення було помилковим, процесор звільнює конвеєр від команд, які були помилково вибрані, і завантажує нові команди, розпочинаючи з правильної адреси. Помилково зроблені завбачання призводять до зниження продуктивності процесора.

Через суперскалярну структуру процесора водночас можуть виконуватись кілька команд, які змінюють значення прапорців у регістрі *CR*. Для розв'язування конфліктів у *BTU* є вісім буферних регістрів, які дублюють вміст *CR* і використовуються різними виконавчими пристроями. Після завершення виконування команди їхній вміст переноситься до основного регістра *CR*, тобто здійснюється зворотний запис.

Мікропроцесор *MPC*604 здійснює роздільне оброблення команд і даних за допомогою двох пристроїв керування пам'яттю – *IMMU* та *DMMU*, які можуть виконувати сегментне, сторінкове та блокове сегментування. Внутрішні кеші команд та даних мають обсяг 16 кбайт.

Інтерфейс мікропроцесора із зовнішніми пристроями забезпечується контролером шини *BIU*. Для кожного байта адреси та даних передаються та приймаються біти парності, за допомогою яких здійснюється контроль помилок звернення до шини. Шина адреси має 32 розряди, а шина даних – 64 розряди, що підвищує продуктивність мікропроцесора. Припустиме є підмикання зовнішнього кеша 2-го рівня.

Мікропроцесор вміщує спеціалізований послідовний порт задля виконування тестування за стандартом *JTAG* (*IEEE* 1149.1).

Задля живлення мікропроцесора використовується напруга $V_{ж} = 3{,}3$ В. Зниження енергоспоживання (400 мВт) забезпечується при зупині мікропроцесора, коли припиняється виконування команд, але продовжують працювати базовий таймер, система декрементування та блок обслуговування запитів на переривання.

**Контрольні питання:**

1 Які основні особливості RISC-процесорів Вам відомі?

2 Поясніть, з яких регістрів складається регістрова модель користувача процесора MPC604?

3 Які провідні фірми випускають RISC-процесори?

4 Які поля має регістр прапорців CR і в який спосіб вони використовуються?

5 Назвіть режими функціонування процесора *MPC*604.

6 Який обсяг регістрової пам’яті використовують сучасні *RISC*-процесори?

7 Яке нумерування розрядів прийнято у регістрах *RISC*-процесорів?

8 Які вбудовані методи захисту використовує МП *PowerPC*?

**Контрольні питання підвищеної складності:**

1 Доведіть, що процесор *MPC*604 має суперскалярну структуру.

2 Як апаратно оброблюються розгалуження за алгоритмами динамічного завбачання?

## 14.2 *RISC*-процесори *ColdFire*

**Вхідний контроль:**

1 Чи є довжина команд процесора *MPC*604 незмінна?

2 Яку розрядність мають шини адреси та шини даних процесора *MPC*604?

*RISC*-процесори *ColdFire* (*MCF*5*XXX*) мають таку саму модель користувача, як і сімейство *M*68*XXX*, зреалізовують основні команди та способи адресування цього сімейства. Завдяки цьому МП *MCF*5*XXX* можуть використовувати значний обсяг програмного забезпечення, розробленого для *M*68*XXX*. Для зменшення обсягу пам’яті команд використовуються команди змінної довжини: 2, 4, 6 байт. Низка моделей вміщують на кристалі ВІС

таймери, паралельні та послідовні порти, контролер переривань та інші периферійні пристрої; за це їх називають інтегрованими. МП *Cold Fire* слугують для побудови мікропроцесорних систем і можуть входити до складу спеціалізованих мікроконтролерів.

МП вміщують процесорне ядро *CFPU* з *RISC*-архітектурою, об'єднану кеш-пам'ять команд/даних обсягом 2 кбайти та блок зовнішнього інтерфейсу, який забезпечує зв'язок з 32-розрядною мультиплексованою системною шиною даних/адрес.

Регістрова модель процесора *CFPU* відрізнюється тим, що має один вказівник стека *A*7 і формує спільний стек для режимів користувача та супервізора (рис. 11.1). До регістрової моделі супервізора додатково входять регістр базової адреси таблиці векторів переривань та виключень, регістри керування кеш-пам'яттю та звернення до пам'яті.

Процесор *CFPU* працює в режимі користувача або супервізора, аналогічно до МП сімейства *M*68*XXX*, емулює основні команди і способи адресування цих МП. З базового набору команд *CFPU* не виконує деякі команди та операції з двійково-десятковими числами, що не робить систему команд недостатньо повною.

Задля налагодження МПС на МП *MCF*5202 надаються такі можливості:

– реалізація режиму налагодження, за якого процесор працює під керуванням зовнішнього налагоджувача;

– контроль внутрішнього стану процесора при виконуванні поточної програми;

– виконування програми із зупином у контрольних точках.

На рис. 14.4 подано структуру інтегрованого мікропроцесора *MCF*5204, до складу якого входять:

– процесорне ядро *CFPU* з *RISC*-архітектурою *ColdFire*;
– кеш-пам'ять команд *IC* обсягом 512 байт;
– статичне ОЗП обсягом 512 байт;
– модуль системного інтерфейсу *SIM-M* з 8-розрядним портом *A*;
– блок тестування та налагодження (БТН);
– таймерний модуль;
– асинхронний послідовний інтерфейс (АПІ) типу *UART*.

До регістрової моделі супервізора додано 32-розрядні регістри *RAMBAR* та *MBAR*, які задають базову адресу та режим роботи внутрішнього ОЗП, регістрів різних модулів та блоків МП.

БТН зреалізовує набір режимів та варіантів тестування за стандартом *JTAG* (IEEE 1149.1).

Внутрішнє статичне ОЗП має обсяг 512 байт. Це ОЗП може розміщуватися, розпочинаючи з будь-якої адреси, яка може задаватися програмно. Керування ОЗП здійснюється шляхом ініціалізації регістра *RAMBAR*, біти якого можуть маскувати доступ до нього різних типів звернень: в режимі користувача, супервізора при записуванні та зчитуванні даних та команд.

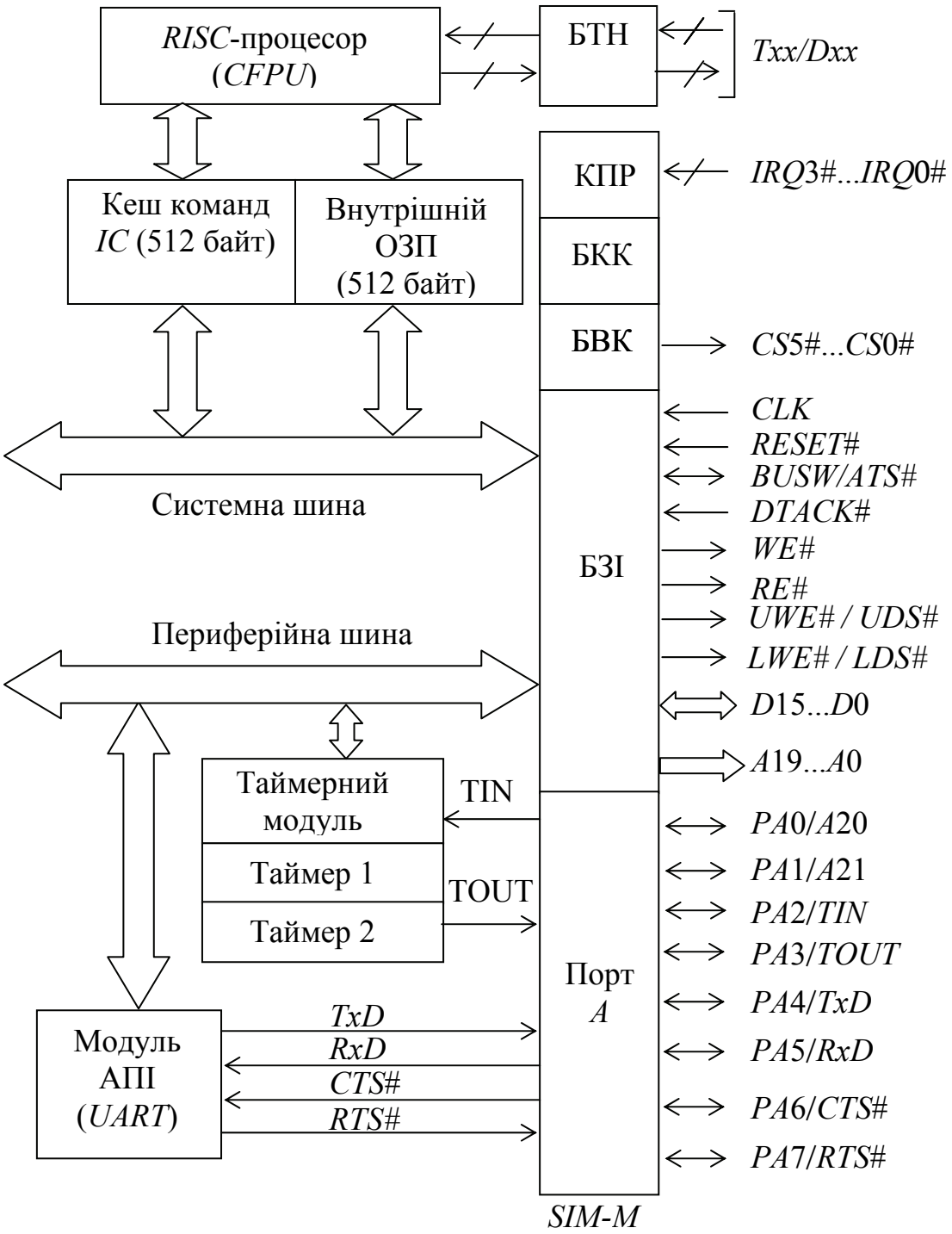



Рисунок 14.4 – Структура інтегрованого мікропроцесора *MCF*5204

Модуль системної інтеграції *SIM-M,* до якого входить блок зовнішнього інтерфейса (БЗІ), забезпечує доступ до шести різних банків даних обсягом по 4 Мбайти, для кожного з яких можна ініціалізувати власний протокол обміну.

БВК вміщує шість наборів по три регістри, які керують шістьма банками даних, призначених для зберігання даних у різних режимах роботи процесора.

Блок конфігурації та контролю (БКК) вміщує регістр *MBAR*, який задає базову адресу блока пам’яті, в якому зберігаються адреси службових регістрів внутрішніх модулів: *SIM-M*, таймерного та *UART* (АПІ). У регістрі *MBAR* є біт

достовірності та інші біти, які визначають права доступу до службових регістрів. До складу БКК входить також монітор шини та вартовий таймер. Монітор шини програмується на видавання запиту виключення “помилка звернення до шини” у разі, коли сигнал підтвердження обміну *DTACK#* = 0 не надходить від зовнішнього пристрою упродовж 128, 256, 512 або 1024 тактів.

Контролер переривань КПР забезпечує обробку чотирьох зовнішніх запитів переривань та чотирьох внутрішніх запитів від вартового таймера, таймерів 1 та 2, АПІ.

Порт *A* використовується задля паралельного двоспрямованого обміну даними.

Таймерний модуль складається з 16-розрядних таймерів 1 та 2. Таймер 1 функціонує як таймер загального призначення і може використовуватись у режимі вимірювання часових інтервалів або частоти сигналів, формування імпульсів заданої частоти та тривалості. Таймер 2 працює в режимі лічби внутрішніх імпульсів і може використовуватись задля генерування періодичних переривань.

Асинхронний послідовний інтерфейс АПІ (*UART*) використовується задля послідовного обміну МП із зовнішніми пристроями і в перебігу приймання контролює помилки парності, порушення кадру, переповнення, порушення зв’язку. Виявлення помилок призводить до формування запиту на переривання. Цей запит оброблюється КПР, так само як і запити після передавання чергового символу, за порожнього буфера передавача, після прийняття символу та заповнення буферної пам’яті приймача.

Основними областями використовування сімейства *ColdFire* є зв’язок, промислова автоматика, обчислювальна техніка, системи телекомунікацій, контрольно-вимірювальна апаратура тощо.

**Контрольні питання:**

1 Якими головними рисами можна схарактеризувати *RISC*-процесори?

2 Які удосконалення структури порівняно з процесором *MPC*604 мають *RISC*-процесори *MCF*5*XXX*?

3 З якою метою у процесорі *MCF*5*XXX* використовуються команди змінної довжини?

4 Яку структуру має інтегрований процесор *MCF*5204?

5 У яких пристроях телекомунікацій використовуються *RISC*-процесори фірми *Motorola*?

**Контрольні питання підвищеної складності:**

1 З якою метою в режимі супервізора можна задавати базову адресу та режим роботи внутрішнього ОЗП, регістрів різних модулів та блоків процесора *MCF*5*XXX*?

2 З якою метою у процесорі *MCF*5*XXX* використовуються 6 банків баних?

## 14.3 Система команд *RISC*-мікропроцесорів сімейства *PowerPC*

**Вхідний контроль:**

1 Які способи адресування використовують універсальні МП *MC*68*XXX*?

2 Які способи адресування використовуються при обробленні одно- та двовимірних масивів?

Задля забезпечення підвищення продуктивності мікропроцесорів PowerPC притаманні іншим мікропроцесорам способи адресування операндів застосовуються в обмеженому обсязі: регістровий, непрямий регістровий зі зміщенням, непрямий регістровий з індексуванням, відносний, абсолютний, безпосередній.

Всі команди арифметичних та логічних операцій, порівняння та зсувів виконуються лише з регістровим та безпосереднім адресуванням. Уникнення звертання до циклів звернення до шини з метою вибирання операндів підвищує продуктивність процесора. Команди мають триадресний формат, що дозволяє запобігти зайвим пересиланням поміж регістрами. Регістри операндів називаються *rA* та *rB*, регістр розміщення результату є *rD*. Звернення до регістрів здійснюється за їхніми номерами *GPR*31…*GPR*0. Безпосередній операнд зі знаком *SIm* або без знаку *UIm* задається 32-розрядним словом, яке слідує за кодом команди.

Команди завантажування та зберігання вмісту регістрів використовують при зверненні до пам’яті непряме регістрове адресування зі зміщенням або індексуванням. Ефективна адреса операнда *EA* визначається виразами:

*EA = (rA*!0) + *d*32 – при адресуванні зі зміщенням;

*EA = (rA*!0) + (*rB*) – при адресуванні з індексуванням;

де *rA, rB* – номери регістрів загального призначення *GPR*31...*GPR*1; *d*32 – 32-розрядне зміщення. Регістр *GPRO* не використовується за цих способів адресування: якщо в якості номера вказано 0, то формується адреса *EA = d*32 при адресуванні зі зміщенням, *EA = (rB)* – при адресуванні з індексуванням. Так зреалізовується пряме адресування зі зверненням за заданою адресою *d*32 та непряме регістрове адресування за адресою, заданою вмістом регістра *rB*.

У командах розгалужень використовують пряме та відносне адресування. За адресу переходу слугує вказана в команді адреса *t-adr* або сума (*PC* + *t-adr*).

Усі команди мають 32-розрядний формат коду операції. При використовуванні безпосереднього адресування після коду операції йде 32-розрядний операнд. У командах, які використовують непряме регістрове адресування зі зміщенням, після коду операції йде 32-розрядне зміщення, а в командах розгалужень – 32-розрядна адреса *t-adr*. Отже, всі команди мають 4 або 8 байт.

Команди арифметичних операцій (табл. 14.1) мають кілька модифікацій, які відрізнюються формуванням прапорців за результатами операцій. Якщо після мнемокоду операції йде символ “**.**”, то за результатами операції встановлюються відповідні значення бітів *SO, EQ, GT, LT* у полі *CR*0 регістра *CR*. Якщо мнемокод команди має суфікс “*O*”, то за результатами її виконання

встановлюються біти *OV, SO* у регістрі *XER*; біт *SO* дублюється у полі *CR*0. Команди додавання *addic, addc, adde, addme, addze* та віднімання *subfc, subfic, subfe, subfme, subfze* встановлюють прапорець перенесення *CA* у регістрі *XER* за результатом операції. Команда *addi* при значенні $rA = 0$ завантажує безпосередній операнд зі знаком *SIm* до регістра *rD*, а при $rA = /0$ – виконує додавання цього операнда з вмістом регістра *rB*. Команда безпосереднього додавання *addis* виконує при $rA = /0$ додавання вмісту *rA* з безпосереднім операндом *SIm*, зсунутим на чотири розряди ліворуч, а при $rA = 0$ – завантажує зсунутий операнд *SIm* до регістра *rD*.

Таблиця 14.1 – Команди арифметичних операцій

| Синтаксис Асемблера | Операція |
|---|---|
| *addi(addis) rDrA,SIm* | *rA\|O+Slm →rD* (зсув SIm) |
| *add(add., addo, addo.) rDrA,rB* | *rA+rB →rD* |
| *addic(addic.) rD,rA,SIm* | *rA+SIm →rD*, встановлення *CA* |
| *addc(addc., addco, addco.) rD,rA,rB* | *rA+rB →rD*, встановлення *CA* |
| *adde(adde., addeo, addeo.) rD,rA,rB* | *rA+rB+CA→rD*, встановлення *CA* |
| *addme(addme., addmeo, addmeo.) rD,rA* | *rA-1+CA → rD*, встановлення *CA* |
| *addze(addze., addzeo, addzeo.) rD,rA* | *rA+CA → rD*, встановлення *CA* |
| *subf(subf., subfo, subfo.) rD,rA,rB* | *-rA+rB →rD* |
| *subfic rD,rA,Slm* | *-rA+Slm → rD*, встановлення *CA* |
| *subfc(subfc., subfco, subfco.) rD,rA,rB* | *-rA+rB → rD*, встановлення *CA* |
| *subfe(subfe., subfeo, subfeo.) rD,rA,rB* | *-rA+rB+CA → rD*, встановлення *CA* |
| *subfme(subfme., subfmeo, subfmeo) rD,rA* | *-rA-1+CA → rD,* встановлення *CA* |
| *subfze(subfze., subfzeo, subfzeo) rD,rA* | *-rA+CA → rD*, встановлення *CA* |
| *mulli rD,rA,Slm* | *rA*SIm → rD*, знакове 16x16 |
| *mullw(mullw., mullwo, mullwo) rD,rA,rB* | *rA*rB → rD*, знакове 16x16 |
| *mulhw(mulhw.) rD,rA,rB* | *rA*rB → rD,* знакове 32x32 |
| *mulhwu(mulhwu.) rD,rA,rB* | *rA*rB → rD*, беззнакове 32x32 |
| *divw(divw., divwo, divwo.) rD,rA,rB* | *rA/rB → rD*, знакове 32:32 |
| *divwu(divwu., divwuo, divwuo.) rD,rA,rB* | *rA/rB → rD*, беззнакове 32:32 |
| *neg(neg., nego, nego$_k$) rD,rA* | *-rA → rD* |
| *extsb(exstb.) rA,rS* | rS(мол.байт) *→rA* ⎫ Розширення |
| *extsh(exsth) rA,rS* | *rS*(мол.п/слова) *→rA* ⎭ знаком |

Команди множення та ділення можуть оперувати з операндами зі знаком та без знаку. При знаковому множенні 16-розрядних операндів (команди *mulli, mullw*) у регістрі *rD* розміщується 32-розрядний результат. При множенні 32-розрядних чисел зі знаком (команда *mulhw*) або без знаку (команда *mulhwu*) до регістра *rD* записуються старші 32 розряди результату, а до наступного регістра з номером (*rB* + 1) – молодші 32 розряди. При діленні 32-розрядних чисел зі знаком (команда *divw*) або без знаку (команда *divwu*) до регістра *rD* заноситься 32-розрядний результат.

Команда *neg* змінює знак операнда. Команда *extsb* перетворює молодший байт вмісту регістра *rS* шляхом розширення знаку у 32-розрядне дане, яке

розміщується в *rD*. Команда *extsh* виконує аналогічне розширення знаку для 16-ти молодших розрядів (молодше півслово) вмісту регістра *rS*.

Команди логічних операцій (табл. 14.2) виконуються над операндами без знаку, які є записані до регістрів *rS, rB* або задані безпосередньо в команді (*UIm*). МП зреалізовує вісім логічних операцій: “ТА”, “АБО”, “виключне АБО”, “ТА-НІ”, “АБО-НІ”, інверсію, імплікацію, еквівалентність. Команди, які мають після мнемокоду символ “**.**”, виставляють відповідні прапорці в полі *CR*0 регістра *CR*. Команди логічних операцій зі зсувом *ands., oris., xoris*. перед виконуванням вказаних в них операцій виконують зсув безпосереднього операнда ліворуч на 16 розрядів. До даної групи входить також команда *cntizw*, яка визначає кількість послідовно розташованих нулів у регістрі *rS*, розпочинаючи зі старшого розряду. Це число заноситься до регістра *rA*.

Таблиця 14.2 – Команди логічних операцій

| Синтаксис Асемблера | Операція |
|---|---|
| *andi.(andis.) rA,rS,UIm* | *rS ^ UIm* →*rA, (*зсув *UIm)* |
| *ori.( oris.) rA,rS,UIm* | *rS v UIm* →*rA, (*зсув *UIm)* |
| *xori.( xoris) rA,rS,UIm* | *rS ⊕ UIm* →*rA, (*зсув *UIm)* |
| *and(and.) rA,rS,rB* | *rS ^ rB* →*rA* |
| *or(or.) rA,rS,rB* | *rS v rB* →*rA* |
| *xor(xor.) rA,rS,rB* | *rS ⊕ rB* →*rA* |
| *and(nand.) rA,rS,rB* | *rS ^ rB* →*rA* |
| *nor(nor.) rA,rS,rB* | *rS v rB* →*rA* |
| *andc(andc.) rA,rS,rB* | *rS ^ rB* →*rA* |
| *orc(orc.) rA,rS,rB* | *rS v rB* →*rA* |
| *egv(egv.) rA,rS,rB* | *rS ⊕ rB* →*rA* |
| *cntizw(cntizw.) rA,rS* | Число старших 0 в *rS*→*rA* |

Команди порівняння (табл. 14.3) здійснюють віднімання операндів, які зберігаються в регістрах *rA, rB,* або заданих безпосередньо (*SIm, UIm*), які можуть бути числами зі знаком або беззнаковими. За результатами віднімання встановлюються прапорці у полі *CRi* регістра *CR*. Номер поля *i* = 7...2 задається операндом *crdD* в команді Асемблера. Якщо цей операнд не є заданий або дорівнює 0, то прапорці встановлюються у полі *CR*0. Операнд *L* приймається таким, що дорівнює 0.

Команди зсувів зреалізовують багаторозрядні логічні (*sew, srw*), арифметичні (*sraw, srawi*) та циклічні (*rewinm, rewnm, rewimi*) зсуви вмісту регістра *rS* з розміщенням результату в *rA*. Кількість розрядів зсуву задається або безпосереднім операндом *Ns*, або вмістом шести молодших розрядів регістра *rB*. При логічних зсувах ліворуч (команда *sew*) або праворуч (команда *srw*) розряди, які звільнюються, заповнюються нулями. При арифметичному зсуві праворуч (команди *sraw, srawi*) в старші розряди дублюється знак зсуваного операнда.

Таблиця 14.3 – Команди порівняння та зсувів

| Синтаксис Асемблера | Операція |
|---|---|
| *cmpi crfD,L,rA,SIm* | *-rA + Sim (*знакове*)* |
| *cmp crfD,L,rA,rB* | *-rA + rB (*знакове*)* |
| *cmpli crfD,L,rA,UIm* | *-rA + Sim (*беззнакове*)* |
| *cmpl crfD,L,rA,rB* | *-rA + rB (*беззнакове*)* |
| *slw(slw.) rA,rS,rB* | *[rS←s(rB)]→rS,* заповнення нулями |
| *srw(srw.) rA,rS,rB* | *[rS→s(rB)]→rA,* заповнення нулями |
| *srawi(srwi.) rA,rS,Ns* | *[rS→s(Ns)]→rA,* заповнення нулями |
| *sraw(sraw.) rA,rS,rB* | *[rS→s(rB)]→rA,* заповнення нулями |
| *riwinm(riwinm.) rA,rS,Ns,Mb,Me* | *[rS←r(Ns)] ^ M→rA* |
| *riwnm(riwnm.) rA,rS,rB,Mb,Me* | *[rS←r(rB)] ^ M→rA* |
| *rlwimi(rlwimi.) rA,rS,Ns,Mb,Me* | *([rS←r(Ns)] ^ M) v (rA ^ M) →rA* |

Циклічні зсуви зреалізовуються лише ліворуч, а результат логічно множиться на маску *M*. Маска формується відповідно до задаваних у команді номерів початкового *Mb* та кінцевого *Mc* бітів. Біти маски з номерами від *Mc* до *Mb* мають значення 1, тому відповідні біти результату зберігаються. Решта бітів маски дорівнюють 0, і відповідні біти результату набирають такого самого значення. Здобутий внаслідок зсуву та маскування результат записується до регістра *rA*. При виконуванні команд *rewinm, rewnm* попередній вміст регістра *rA* не зберігається. Команда rewimi зберігає в регістрі *rA* значення бітів, для яких біти маски становили 0.

Команди зсувів можуть зумовлювати встановлення прапорців у полі *CR*0 регістра *CR* залежно від результату операції. Для цього після мнемокоду команди слід поставити символ “**.**”.

Команди завантаження та зберігання (табл. 14.4) завантажують до регістра *GPR* з номером *rD* байт, слово, довге слово (32 розряди) з пам’яті або записують до пам’яті байт, слово та довге слово з регістра *rS*. Задля звернення до пам’яті використовується непряме регістрове адресування зі зміщенням або індексуванням, у цьому разі в команді зазначаються адресний *rA* та індексний *rB* регістри, при адресуванні зі зміщенням – адресний регістр *rA* та зміщення *d*32. При завантаженні байта (команди *lbz* та *lbzx*) старші розряди регістра *rD* заповнюються нулями. При завантаженні слова старші розряди заповнюються або нулями (команди *lbhz, lnzx*) або значенням старшого знакового біта *b*15 завантажуваного довгого слова (команди *lha, lhax*).

Команди завантаження-зберігання мають модифікації з суфіксом u, який надає команді додаткові функції: до регістра *rA* після пересилання завантажується ефективна адреса *EA*, використовувана в даній команді.

Таблиця 14.4 – Команди завантаження та зберігання вмісту регістра *GPR*

| Синтаксис Асемблера | Операції |
|---|---|
| *lbz(lbzu) rD,d(rA)* | *(EA)→rD,* байт*, (EA→rA)* |
| *lbzx(lbzux) rD,rA,rB* | *(EA)→rD,* байт*, (EA→rA)* |
| *lhz(lhzu) rD,d(rA)* | *(EA)→rD,* півслово*, (EA→rA)* |
| *lhzx(lhzux) rD,rA,rB* | *(EA)→rD,* півслово*, (EA→rA)* |
| *lha(lhau) rD,d(rA)* | *(EA)→rD,* півслово*, (EA→rA)* |
| *lhax(lhaux) rD,rA,rB* | *(EA)→rD,* півслово*, (EA→rA)* |
| *lwz(lwzu) rD,d(rA)* | *(EA)→rD,* слово*, (EA→rA)* |
| *lwzx(lwzux) rD,rA,rB* | *(EA)→rD,* слово*, (EA→rA)* |
| *stb(stbu) rS,d(rA)* | *rS→(EA),* байт*, (EA→rA)* |
| *stbx(Stbux) rS,rA,rB* | *rS→(EA),* байт*, (EA→rA)* |
| *sth(sthu) rS,d(rA)* | *rS→(EA),* півслово*, (EA→rA)* |
| *sthx(sthux) rS,rA,rB* | *rS→(EA),* півслово*, (EA→rA)* |
| *stw(stwu) rS,d(rB)* | *rS→(EA),* слово*, (EA→rA)* |
| *stwx(stwux) rS,rA,rB* | *rS→(EA),* слово*, (EA→rA)* |
| *lhbrx(lwbrx) rD,rA,rB* | *(EA)→rD,* переставлення байтів |
| *sthbrx(stwbrx) rA,rB* | *(EA)→rD,* переставлення байтів |
| *lmw rD,d(rA)* | *(EA)→rD...GPR31,* групове пересилання |
| *stmw rS,d(rA)* | *Rs…GPR31→(EA),* групове пересилання |
| *Lswi(lswx) rD,rA,Nb(rS)* | *(EA)→rD,* пересилання рядка |
| *stswi(stswx) rS,rA,Nb(rB)* | *rS→(EA),* пересилання рядка |

Команди *lhbrz* та *lwbrz* переставляють байти при завантаженні до регістра *rD* слова або довгого слова відповідно. Команди *sthbrx, stwbrx* виконують аналогічне переставлення байтів у разі запису з регістра *rS* до пам’яті слова чи подвійного слова.

Команди групового пересилання *Imw, stmw* завантажують або записують до пам’яті вміст групи регістрів, розпочинаючи із заданого в команді номера *rD* або *rS* до останнього регістра з номером *GPR*31. Завантажувані дані розташовуються в комірках пам’яті, розпочинаючи з адреси *EA*, яка визначається непрямим регістровим адресуванням зі зміщенням. Власне регістр *rA* не повинен входити до групи регістрів, які адресуються.

Команди пересилання рядків символів *lswi, lswx, stswi, stswx* здійснюють завантаження до регістрів або запис до пам’яті n байтів. Кількість пересиланих байтів задається зазначеним у команді операндом *Nb* (команди *lswi, stswi*) або значенням восьми старших бітів поля *SS* регістра *XER* (команди *lswx, stswx*). Початкова адреса пересиланих байтів є *EA = (rA*) для команд *lswi, stswi* та *EA = (rA!0) + (rB*) – для команд *lswx, stwx*. Дані у регістрах розміщуються чи вибираються, починаючи з молодшого байта регістра *rD* або *rS*. Команди спричинюють виключення, якщо *rA* або *rB* входить до числа регістрів, в які завантажується або з яких вибирається рядок символів.

Керування програмою здійснюється командами безумовних та умовних переходів, програмних переривань *tw, twi,* системного виклику *sc* та повернення з підпрограм оброблення переривань *rfi* (табл. 14.5). Адреса команди переходу визначається за допомогою прямого (команди *b, bl, bc, bcl*) адресування.

Таблиця 14.5 – Команди керування програмою

| Синтаксис Асемблера | Операція |
|---|---|
| *b t-adr*<br>*ba t-adr*<br>*bl t-adr*<br>*bla t-adr* | *PC + (t-adr)→PC*<br>*(t-adr)→PC*<br>*PC→LR, PC + (t-adr)→PC*<br>*PC→LR, (t-adr)→PC* |
| *bc BO,BI,t-adr*<br>*bca BO,BI,t-adr*<br>*bcl BO,BI,t-adr*<br><br>*bcla BO,BI,t-adr* | *PC + (t-adr)→PC,* якщо *BO = CRi*<br>*(t-adr)→PC,* якщо *BC = CRi*<br>*PC→LR, PC + (t-adr)→PC,* якщо *BO = CRi*<br>*PC→LR, (t-adr)→PC,* якщо *BO = CRi* |
| *bclr BO,BI*<br>*bclrl BO,BI*<br>*bcctr BO,BI*<br>*bcctrl BO,BI* | *LR→PC,* якщо *BO = CRi*<br>*PC→LR,* якщо *BO = CRi*<br>*CTR→PC,* якщо *BO = CRi*<br>*PC→LR, CTR→PC,* якщо *BO = CRi* |
| *twi TO,rA,SIm*<br><br>*tw TO,rA,rB* | *-rA+Slm,* якщо *TO,* то *PC,*<br>*MSR→SRRO, SRR*1*, Ve→PC*<br>*-rA+rB,* якщо *TO,* то *PC, MSR→SRRO,*<br>*SRR*1*, Ve→PC* |
| *sc*<br>**rfi* | *PC, MSR→SSRO, SRR*1*, Ve→PC*<br>*SSRO, SRR*1*→PC, MSR* |

Задане в команді число *t-adr* є абсолютною адресою чи зміщенням, яке визначає значення адреси переходу. За наявності у мнемокоді команди переходу суфікса l (команди *bl, bla, bcl, bcla*), поточне значення програмного лічильника заноситься до регістра зв'язку *LR* для забезпечення можливості повернення до наступної команди програми, тобто ці команди можуть слугувати для виклику підпрограм. Команди *bc, bca, bcl, bcla* зреалізовують умовні переходи або виклики підпрограм. Умовою розгалуження є збіг вмісту поля *CRi* в регістрі умов *CR* з чотирма молодшими бітами заданого в команді операнда *BO*. Номер поля і, яке використовується задля порівняння, задається операндом *BI*. За збігу заданого значення *BO* і вмісту поля *CRi* здійснюється перехід до команди, адреса якої визначається за допомогою абсолютного (команди *bca, bcla*) або відносного (команди *bc, bcl*) адресування. За відсутності збігу виконується наступна команда програми. Команди *bcl, bcla*, які зберігають у регістрі *LR* адресу наступної команди, слугують для реалізації умовних викликів підпрограм. Команди *bclr, bclrl* зреалізують умовне повернення з підпрограм, якщо задане значення *BO* збігається зі містом поля *CRi*. У цьому разі до програмного лічильника завантажується вміст регістра

зв'язку *LR*. Команда *bcrfri* зберігає в регістрі зв'язку *LR* поточний вміст *PC* задля можливості повернення з підпрограми.

При виконуванні команд *bcctr, bcctri* до програмного лічильника завантажується вміст регістра *CTR*; якщо виконується умова *BO = Cri*, команда *bcctrd* забезпечує зберігання поточного вмісту програмного лічильника в регістрі зв'язку *LR*.

Програміст може вказувати у програмі на підвищену ймовірність розгалуження, якщо встановить у 1 п'ятий біт операнда *BO*. У цьому разі блок оброблення розгалужень *BPU* забезпечить вибирання до конвеєра наступних команд відповідно до вказаного завбачання. Такий спосіб називається статичним. За нульового значення п'ятого біта в операнді *BO* блок *BPU* здійснює динамічне завбачання за допомогою кеша адрес *BTAC* і таблиці історії *BHT*.

Команди програмних переривань tw, twi зреалізовують умовне звернення до підпрограми їхнього обслуговування. Поточний вміст програмного лічильника та регістр *MSR* зберігаються у регістрах *SRQO* та *SRR*1. Умовою звернення є збіг прапорців у полі *CRO*, які встановлюються при порівнянні вмісту регістра *rA* з безпосереднім операндом *SIm* (команда *twi*) або з вмістом регістра *rB* (команда *tw*), з бітами заданого в команді 4-розрядного операнда *TO*. Переривання зреалізовується, якщо хоча б один із встановлених у 1 прапорців у *CRO* збігається з відповідним бітом *BO*. Вміст *rA*, *rB* та операнд *SIm* подаються при порівнянні як число зі знаком. Команда *sc* здійснює виклик системного переривання з підпрограмою обслуговування. Команда *rfi* здійснює повернення з переривання, за якого з регістрів *SRR*0, *SRR*1 відновлюється вміст програмного лічильника та регістра керування *MSR*, забезпечуючи повернення до виконування перерваної програми. Ця команда виконується лише в режимі супервізора.

При виконуванні команд умовних переходів використовуються різні поля регістра *CR*. Операції з бітами цього регістра зреалізуються за допомогою команд, поданих у табл. 14.6. Команда mtcrd завантажує до *CR* вміст регістра *rS*, який логічно множиться на 32-бітову маску *Mcr*, задану в команді. Ця команда встановлює потрібні значення бітів *CR* або його немаскованих полів. Команда mfcr пересилає вміст *CR* до регістра *rD*. Команда mcrxr копіює чотири молодших біти з регістра виключень *XER* (прапорці *SO, OV, CA*) в полі *CRi* регістра *CR* , де номер поля і задається операндом *crft*. Після копіювання ознаки *SO, OV, CA* у регістрі *XER* обнулюються. Команда *mcrf* копіює вміст поля *CRi* у поле *CRj*, де значення *i, j* задаються операндами *crfS* та *crfD*. Команди логічних операцій над окремими бітами *bi, bj* вмісту регістра *CR* використовують номери *i, j* цих бітів, які задаються операндами *crbA* та *crbB*, а результат операції розміщується у біті *bk*, номер якого *k* визначається у біті *crbD*.

Команда crand виконує операцію "ТА", *cror* – "АБО", *crxor* – "виключне АБО", *crnand* – "ТА-НІ", *cregv* – еквівалентність , *crande* – заборона, *crorc* – імплікація.

Таблиця 14.6 – Команди, які змінюють вміст регістра *CR*

| Синтаксис | Операція |
|---|---|
| *mtcrf Mcr,rS* | *rS ^ Mcr→CR* |
| *mfcr rD* | *CR→rD* |
| *mcrxr crfD* | *XER(3-0) →CRi* |
| *mcrf crfD,crfS* | *Cri→CRj* |
| *crand crbD,crbA,crbB* | *bi ^ bj→bk* |
| *cror crbD,crbA,crbB* | *bi v bj→bk* |
| *crxor crbD,crbA,crbB* | *bi ⊕ bj→bk* |
| *cmand crbD,crbA,crbB* | *bi ^ bj→bk* |
| *crnor crbD,crbA,crbB* | *bi v bj→bk* |
| *cregv crbD,crbA,crbB* | *bi ⊕ bj→bk* |
| *crandc crbD,crbA,crbB* | *bi ^ bj→bk* |
| *crorc crbD,crbA,crbB* | *bi v bj→bk* |

Команди *sync, isync, cicio* використовуються для синхронізації виконування програм кількома процесорами у багатопроцесорній системі. Команда *sync* зумовлює завершення виконування усіх попередніх команд, а потім дозволяє виконування наступних, тобто затримує конвеєр до моменту звільнення усіх виконавчих пристроїв. Команда *isync* забезпечує завершення виконування попередніх команд, після чого звільнює конвеєр від наступних команд, які до нього надійшли, і вибирає з пам’яті нову пару команд, тобто очищує конвеєр. Команда *cicio* завершує виконування попередніх команд, які потребують звернення до зовнішньої пам’яті даних. Наступні звернення виконуються лише після виконання цієї команди, тобто зреалізовується затримка звернень, яка може використовуватись іншими пристроями. При виконуванні команди *cicio* на зовнішніх виводах, які визначають тип циклу звернення до шини, встановлюється комбінація сигналів, яка використовується для впорядковування звернення різних пристроїв до спільної оперативної пам’яті. Команди *cciwx* та *ccowx* забезпечують звернення мікропроцесора до спеціалізованих зовнішніх пристроїв чи розділів пам’яті при прийманні або передаванні керувальної інформації. Команди *isync, cicio* та наступні виконуються мікропроцесором у режимі супервізора.

З 2002 року фірма *Motorola* випускає нові моделі *RISC*-процесорів: *MPC*7455, які мають суперскалярну структуру з 11-ма виконавчими механізмами. Ці моделі здатні виконувати чотири команди за один такт. Тактова частота *MPC*7455 сягає 1000 МГц, споживана потужність – 21,3 Вт. Напруга живлення становить 1,6 В, а схем введення-виведення – 2,5 В. Мікропроцесор має внутрішній кеш 1-го рівня (по 32 кбайт команд та даних) та 2-го рівня обсягом 256 кбайт. Мікропроцесори *MPC*7455, *MPC*740, *MPC*750 призначено переважно для роботи в мультипроцесорних системах, відповідно до принципу *SIMD*, з метою здійснювання цифрового оброблення сигналів.

**Контрольні питання:**

1 Який формат команд мають *RISC*-процесори?

2 У який спосіб зреалізовуються алгоритми завбачання у блоці оброблення розгалужень МП *MPC*604?

3 Скільки команд водночас може виконувати МП *MPC*604?

4 Які способи адресування операндів переважно використовують RISC-процесори сімейства *PowerPC*?

**Контрольні питання підвищеної складності:**

1 У який спосіб організовано звернення виконавчих пристроїв МП *MPC*604 водночас до одних і тих самих регістрів?

2 Завантажте програмно регістри *GPR*1, *GPR*2 та *GPR*30 відповідно даними $12345678, $*ABCDH, $EF*.

3 Зреалізуйте підпрограму затримки на час, який залежить від числа *K*, яке зберігається в регістрі *GPR*10.

# 15 АРХІТЕКТУРА ТА ПРИНЦИПИ ПОБУДОВИ ПРОЦЕСОРІВ ЦИФРОВОГО ОБРОБЛЕННЯ СИГНАЛІВ

**Вхідний контроль:**

1 З якою метою використовується цифрове оброблення сигналів у телекомунікаціях?

2 Який математичний апарат покладено у підгрунтя цифрового оброблення сигналів?

3 Які цифрові пристрої виконують цифрове оброблення сигналів?

4 Які особливості має гарвардська архітектура побудови МП?

5 У який спосіб подаються дані в обчислювальних системах на мікропроцесорах?

## 15.1 Основні напрямки цифрового оброблення сигналів (ЦОС)

До основних напрямків ЦОС належать цифрова фільтрація та спектральний аналіз. Цифрова фільтрація зреалізовується засобами пропускання цифрового сигналу через цифрові фільтри зі скінченною та нескінченною імпульсними характеристиками – СІХ та НІХ фільтри. Обидва типи фільтрів є лінійні системи з постійними параметрами, у яких вхідна $x_n$ та вихідна $y_n$ послідовності сигналів пов'язані відношеннями типу згортки. Відгук системи на поодинокий імпульс (імпульсна характеристика) позначимо через $h_k$, тоді залежність поміж відліками вихідного $y_n$ та вхідного $x_n$ сигналів можна описати виразом

$$y_n = \sum_{k=0}^{n} h_k x_{n-k} \ , \ n = 0, 1, 2, ..., \tag{15.1}$$

де $x_n$, $y_n$ – відліки вхідного та вихідного сигналів; $x_{n-k}$ – вхідний відлік, затриманий на k інтервалів дискретизації.

У СІХ-фільтрі відлік вихідного сигналу визначається лише значеннями вхідного сигналу, а у НІХ-фільтрі – значеннями вхідного та вихідного сигналів. Вираз, який описує НІХ-фільтр, має вигляд

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k} - \sum_{k=1}^{M-1} a_k y_{n-k} \ , \tag{15.2}$$

де $N$, $M$ є цілі константи; $a_k$, $b_k$ – постійні коефіцієнти, які описують конкретний фільтр; $x_n$, $y_n$ – відліки вхідного та вихідного сигналів.

СІХ-фільтр описується виразом

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k} \tag{15.3}$$

або

$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} + ... + b_n x_{n-N+1} . \tag{15.4}$$

Ефективна система цифрової фільтрації потребує ефективної реалізації дискретної згортки, яку можна розкласти на операції множення, накопичуючого підсумовування та операції затримки.

В області спектрального аналізу використовується пряме та обернене дискретне перетворення Фур'є (ДПФ), швидке перетворення Фур'є (ШПФ) тощо. Спектральний аналіз базується на відомих методах подання заданої функції за допомогою інших, які називаються базовими і властивості яких є відомі. Періодичну вхідну послідовність можна розкласти у ряд Фур'є:

$$x_n^p = \sum_{k=-\infty}^{\infty} X_k^p e^{jw_k n}, \tag{15.5}$$

де $X_k^p$ – амплітуда гармоніки; $e^{jw_k n} = \cos w_k n + j \sin w_k n$ – комплексна змінна, $w_k = 2\pi \cdot k / N$ – частота спектральної складової (гармоніки).

Ряд Фур'є можна записати також виразами

$$x_n^p = \sum_{k=0}^{N-1} X_k^p e^{j(2\pi/N)k-n} \quad \text{або} \quad x_n^p = \frac{1}{N} \sum_{k=0}^{N-1} X_k^p e^{j(2\pi/N)k-n}. \tag{15.6}$$

Вираз (15.6) є обернене перетворення, яке описує Фур'є-образ функції.

Для обчислення коефіцієнтів ряду Фур'є використовується ДПФ:

$$X_k^p = \sum_{n=0}^{N-1} x_n^p e^{-j(2\pi/N)nk} = \sum_{n=0}^{N-1} x_n^p W^{nk}, \text{ де } W = e^{-j(2\pi/N)}. \tag{15.7}$$

Аналіз цього виразу дозволяє дійти висновку, що основними операціями при його обчислюванні є операції комплексного множення та підсумовування.

Обчислення коефіцієнта $W_N^k = \cos[(2\pi/n)k] - j \sin[(2\pi/N)k]$ можна здійснити:

– використовуючи підпрограми або таблиці синуса та косинуса;

– у прямий табличний спосіб (вибиранням готових значень з таблиці);

– за допомогою рекурентної формули

$$W_N^k = (W_N^{k-1}) W_N^L \text{ при } W_N^0 = 1;$$

– у таблично-алгоритмічний спосіб.

Трудомісткість прямого обчислення виразу (15.7) є велика і зростає зі зростанням *N*. Оцінити міру складності алгоритмів ШПФ, а також їхні особливості можна через аналіз обчислювальної системи.

Використовування алгоритму ШПФ з проріджуванням за частотою потребує переставлення елементів вихідної послідовності. Операція "метелик", яка визначає двоточкове ДПФ, потребує обчислення виразів

$$x = A + B \cdot W_N^k;$$

$$y = A - B \cdot W_N^k,$$

де *A, B* – вхідні значення; $W_N^k$ – коефіцієнт.

Задля віднаходження вихідної послідовності у правильному порядку слід переставити вхідну послідовність в такий спосіб, щоби двійкові номери

вихідної послідовності здобувались додаванням 1 до старшого розряду з поширенням перенесення у бік молодших розрядів (праворуч). Така операція адресування називається *біт-реверсивною*.

Задля здобуття амплітуд та фаз складових спектра (гармонік) треба виконати обчислення

$$\frac{\sqrt{X_{Rc}^2 + X_{im}^2}}{N}, \quad arctg\frac{X_{Rc}}{X_{im}},$$

де $X_{Rc}, X_{im}$ – дійсна та уявна частини комплексних коефіцієнтів.

Зазвичай додатково буває потрібне обчислення функцій $\log_2 x$ та $2^x$.

Потрібні для цифрового оброблення сигналів операції підтримуються апаратно у процесорах цифрового оброблення сигналів. Узагальнену архітектуру процесорів *DSP* подано на рис. 15.1.

**Контрольні питання:**

1 Які алгоритми цифрової обробки сигналів у часовій області Вам відомі?

2 Які алгоритми цифрової обробки сигналів у спектральній області Вам відомі?

3 Від яких чинників залежить трудомісткість алгоритмів перетворення Фур'є?

**Контрольні питання підвищеної складності:**

1 При виконуванні ШПФ відліки біт-інверсної послідовності розташовуються у порядку слідування двійково-інверсних номерів вхідних відліків у такий спосіб:

| Вхідна послідовність | | Біт-інверсна послідовність | |
|---|---|---|---|
| відлік | двійковий номер | двійковий номер | відлік |
| $x_0$ | 000 | 000 | $x_0$ |
| $x_1$ | 001 | 100 | $x_4$ |
| $x_2$ | 010 | – | $x_2$ |
| $x_3$ | – | 110 | $x_3$ |
| $x_4$ | – | 001 | $x_1$ |
| $x_5$ | 101 | 101 | $x_5$ |
| $x_6$ | 110 | – | $x_3$ |
| $x_7$ | 111 | 111 | $x_7$ |

Заповніть у наведеній таблиці пропущені значення.

2 Як називається процес проріджування цифрового сигналу за часом?

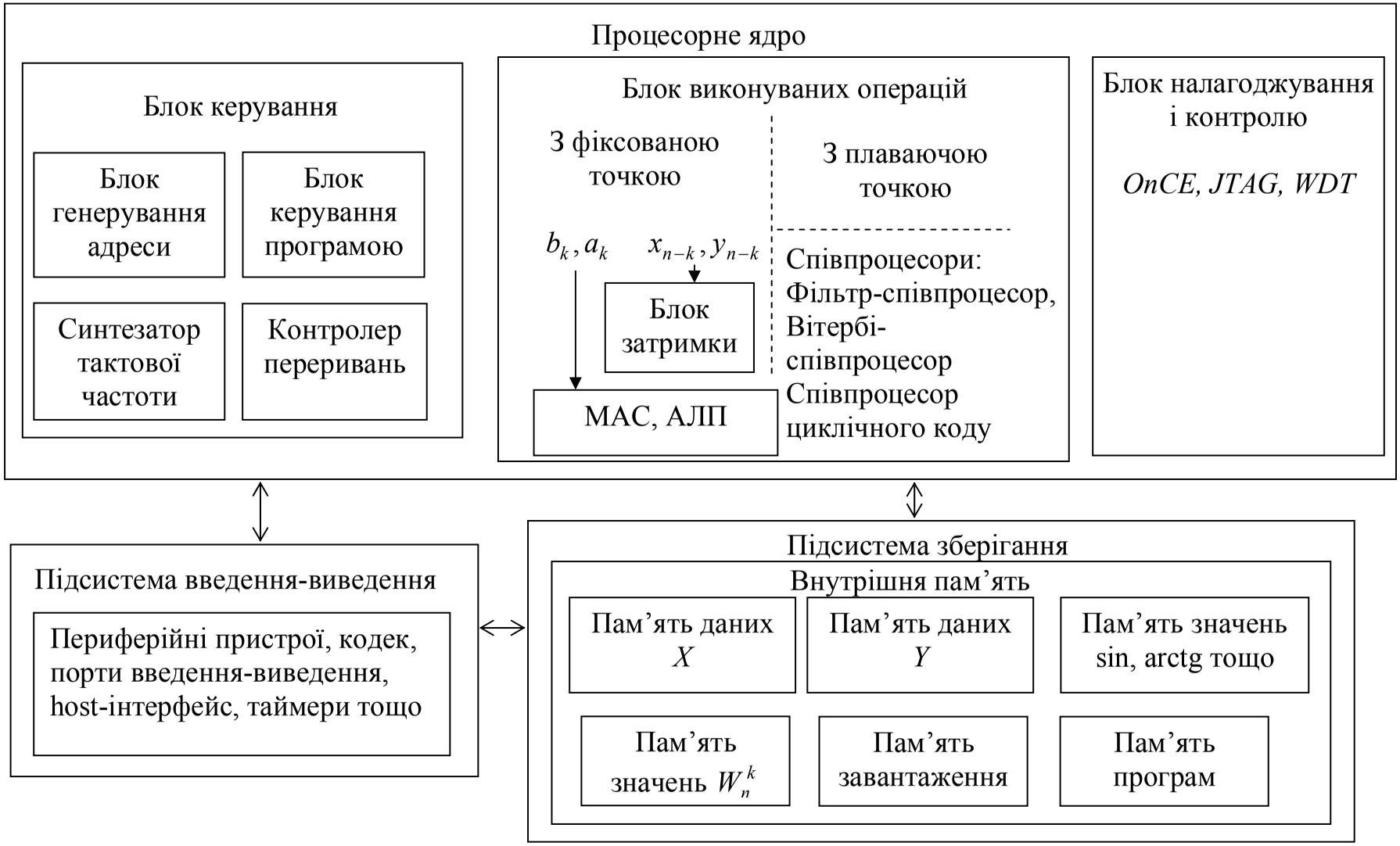



Рисунок 15.1 – Узагальнена архітектура *DSP*

## 15.2 Узагальнена архітектура процесорів сімейства *DSP*563*XX*

**Вхідний контроль:**

1 Які операції цифрової обробки сигналів є найпоширенішими?

2 В який спосіб сприяє гарвардська архітектура підвищенню продуктивності процесорів?

3 Поясніть, як сприяє режим ПДП підвищенню швидкості обміну даними поміж МПС та периферійними пристроями?

Процесори *DSP*563*XX* використовуються в мобільному зв'язку, цифрових системах комутації, пристроях оброблення мовного сигналу, тонального набору, цифрових телевізійних- та радіосистемах, диктофонах, локальних мережах, портативній техніці, відеотелефонах, цифрових фільтрах, аналізаторах спектра, криптографії, системах керування, навігаційному обладнанні, супутниковому зв'язку, розпізнаванні образів тощо. *DSP*563*XX* побудовані за гарвардською архітектурою і мають середню швидкодію 80 *MIPS*.

Периферія вміщує 8-бітний паралельний хост-інтерфейс, 32-бітний універсальний хост-інтерфейс, два розширені синхронні послідовні інтерфейси – *ESSI*1 та *ESSI*0, послідовний комунікаційний інтерфейс *SCI*, модуль таймера. На кристалі є вбудовані співпроцесори:

– фільтр-співпроцесор *FCCP* (*Filter Co-Processor*), який зреалізовує алгоритми фільтрації;

– Вітербі-співпроцесор *VCCP (Viterbi Co-Processor*), який зреалізовує алгоритм задля відновлення з максимальною вірогідністю сигналу зі спотвореннями, наприклад *GSM*;

– співпроцесор циклічного коду *CCOP (Cyclic-code Co-Processor*), який зреалізовує кодування та декодування даних, генерування коду парності та контролю.

Підсистема пам'яті – ОЗП програм, кеш інструкцій, ОЗП даних *X*, ОЗП даних *Y* – може бути сконфігурована в чотири способи і вміщує також ПЗП програм з організацією 6144x24, ПЗП даних *Y* – 3072x24, ПЗП, яке завантажує програми із зовнішньої пам'яті – 192x24.

Узагальнену архітектуру процесорів сімейства *DSP*563*XX* подано на рис. 15.2.

Сімейство DSP 563*XX* вміщує нове ядро, базоване на новітніх технологіях, які зумовили низьку вартість, низьке енергоспоживання, високу ефективність мікропроцесорів. За рахунок включення кеша команд обсягом 1Кx24 процесори припускають підключення повільної зовнішньої пам'яті при зберіганні такої самої ефективності. Продуктивність процесора лінійно залежить від частоти генератора. На кристалі ВІС розміщено синтезатор частоти, порт налагоджування *JTAG*, потрійний таймер, *host*-інтерфейс (*HI*) для

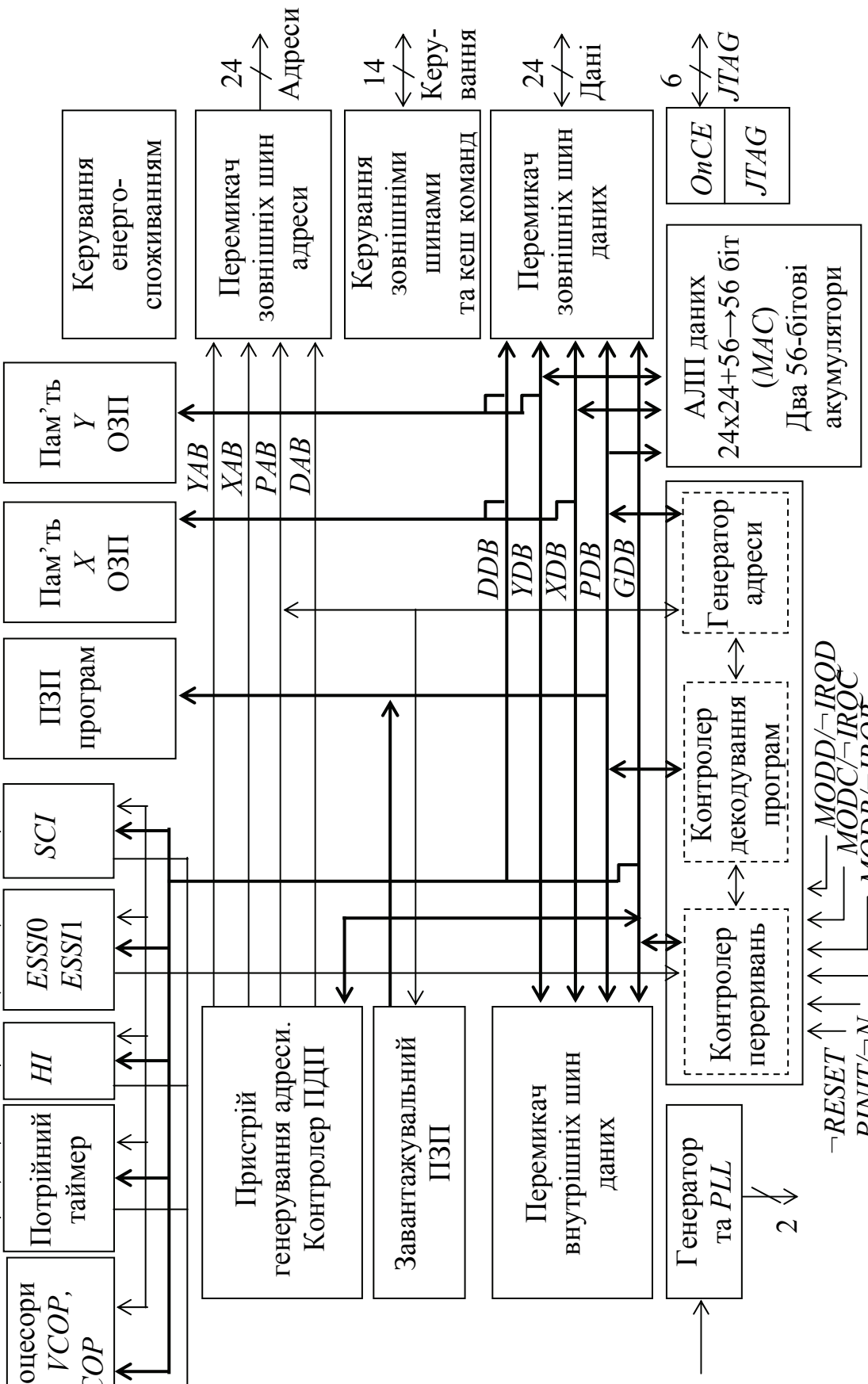



Рисунок 15.2 – Узагальнена архітектура процесорів сімейства *DSP563XX*

побудови багатопроцесорних систем, розширені синхронні послідовні інтерфейси *ESSI*0 та *ESSI*1, контролер кеша, послідовний комунікаційний інтерфейс SCI, пристрій керування потужністю, шестиканальний контролер ПДП контролер переривань задля оброблення переривань у режимах *MODA, MODB, MODC, MODD*.

До архітектури процесорів сімейства долучено додаткову шину даних *DDB (DMA Data Bus*), що дозволяє за допомогою контролера ПДП передавати блоки інформації, не уповільнюючи роботу процесора.

Сімейство *DSP*563*XX* підтримує промислові стандарти щодо комп'ютерної техніки, мікропроцесорів, *DSP* та контролерів ПДП.

32-розрядна шина хост-інтерфейса (*HI*) зреалізовує три класи інтерфейсів: шини *PCI*, універсальну шину, порт введення-виведення загального призначення.

Сімейство *DSP*563*XX* має продуктивність 66/80/100 *MIPS* на частотах відповідно 66/80/100 МГц.

**Контрольні питання:**

1 Які співпроцесори вбудовуються у кристали сигнальних процесорів сімейства *DSP*563*XX* фірми *Motorola***?**

2 У який спосіб пов'язані в архітектурі сигнальних процесорів наявність пам'яті програм, пам'яті *X* та *Y* ОЗП, окремих шин до пам'яті цих видів з можливістю виконувати в одній команді операції та кілька пересилань?

3 У яких областях телекомунікацій використовуються *DSP*?

**Контрольні питання підвищеної складності:**

1 У який спосіб здійснюється тестування сигнальних процесорів відповідно до стандарту *IEEEII*49.1 через *JTAG*-порт?

2 З якою метою на кристалі *DSP*563*XX* розташовується синтезатор частоти?

### 15.3 Організація циклічного буфера в *DSP*

**Вхідний контроль:**

1 Наведіть приклади використовування режиму обміну ПДП поміж МПС та зовнішніми пристроями.

2 Що таке модульна арифметика?

Завдання цифрової обробки сигналів потребують реалізації потокової обробки великих обсягів даних у реальному режимі часу. Це є можливе за високої швидкодії процесора та наявності апаратних засобів інтенсивного обміну із зовнішніми пристроями.

Циклічний буфер слугує за "вікно" заданого розміру, через яке "протягується" задана кількість відліків вхідного сигналу з метою їхнього оброблення в реальному часі.

Циклічний буфер у складі *DSP* – це обмежена за обсягом пам'ять *X* або *Y* з *M* комірок, які адресуються за типом непрямого регістрового адресування. Ефективна адреса комірки обчислюється як алгебраїчна сума вмісту регістра адреси *Rn*, регістра зміщення *Nn* та регістра модифікації *Mn* за правилами модульної арифметики. Модульна арифметика забезпечує звернення лише до тих комірок пам'яті, які належать конкретному буферові.

Нижня межа буфера визначається вмістом регістра *Rn*, а верхня дорівнює (*Rn* + *M* – 1), де значення (*M* – 1) зберігається в регістрі модифікації *Mn*. Значення нижньої межі, яка називається ще базою адреси (*Bs*), має мати нулі у *k* молодших розрядах, *k* вибирається з умови $2^k \geq M$; база адреси є кратна до $2^k$.

Для організації буфера обсягом 5 комірок пам'яті (*M* = 5) кількість *k* молодших нульових бітів у базі *Bs* може становити 3. База *Bs* може бути 101000 або 111000, тобто бути кратною до $2^k$. У загальному випадку база вибирається такою, що дорівнює

$$\underbrace{\text{XXXX...XX}}_{(16-k)\text{ біт}}\underbrace{\text{00…0}}_{k\text{ біт}}, \text{ де } X = 0 \text{ або } 1.$$

Верхня межа буфера дорівнює *Bs* + *M* – 1, а гранична межа – *Bs* + $2^k$ – 1.
На рис. 15.3 подано організацію циклічного буфера.

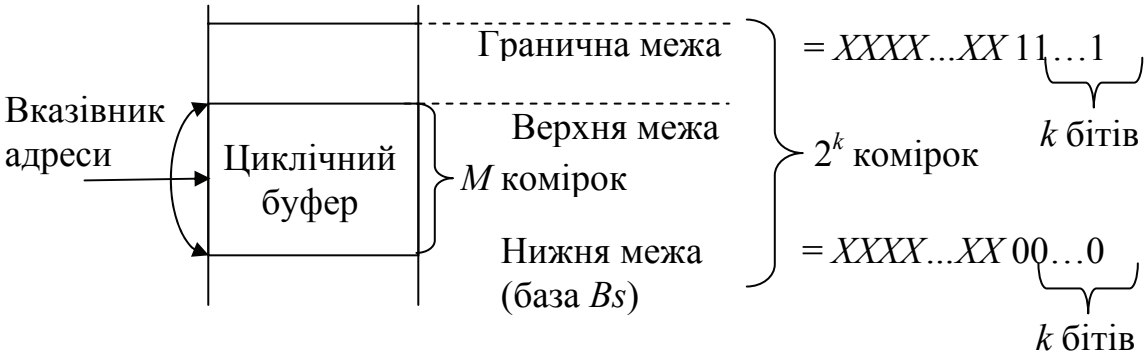



Рисунок 15.3 – Організація циклічного буфера

Початкова адреса, тобто вміст регістра *Rn* може встановлюватись довільно у зазначених межах.

При використовуванні регістра *Nn* адреса обчислюється за формулою *((Rn* ± *Nn)mod M*. На рис. 15.4 подано виконування команди

MOVE X0, X: (RY) + N0

при роботі з циклічним буфером. Згідно з командою, вміст вхідного регістра АЛП даних *X*0 пересилається до комірки *X*-пам'яті даних за адресою, зазначеною в *R*0, після чого здійснюється інкрементування адреси на вміст регістра *N*0 з використовуванням арифметики за *mod M*. Всі адреси зазначено в десятковій системі числення; циклічний буфер має обсяг 90 комірок з базою *Bs* = 128, верхньою межею 217 та граничною межею 255.

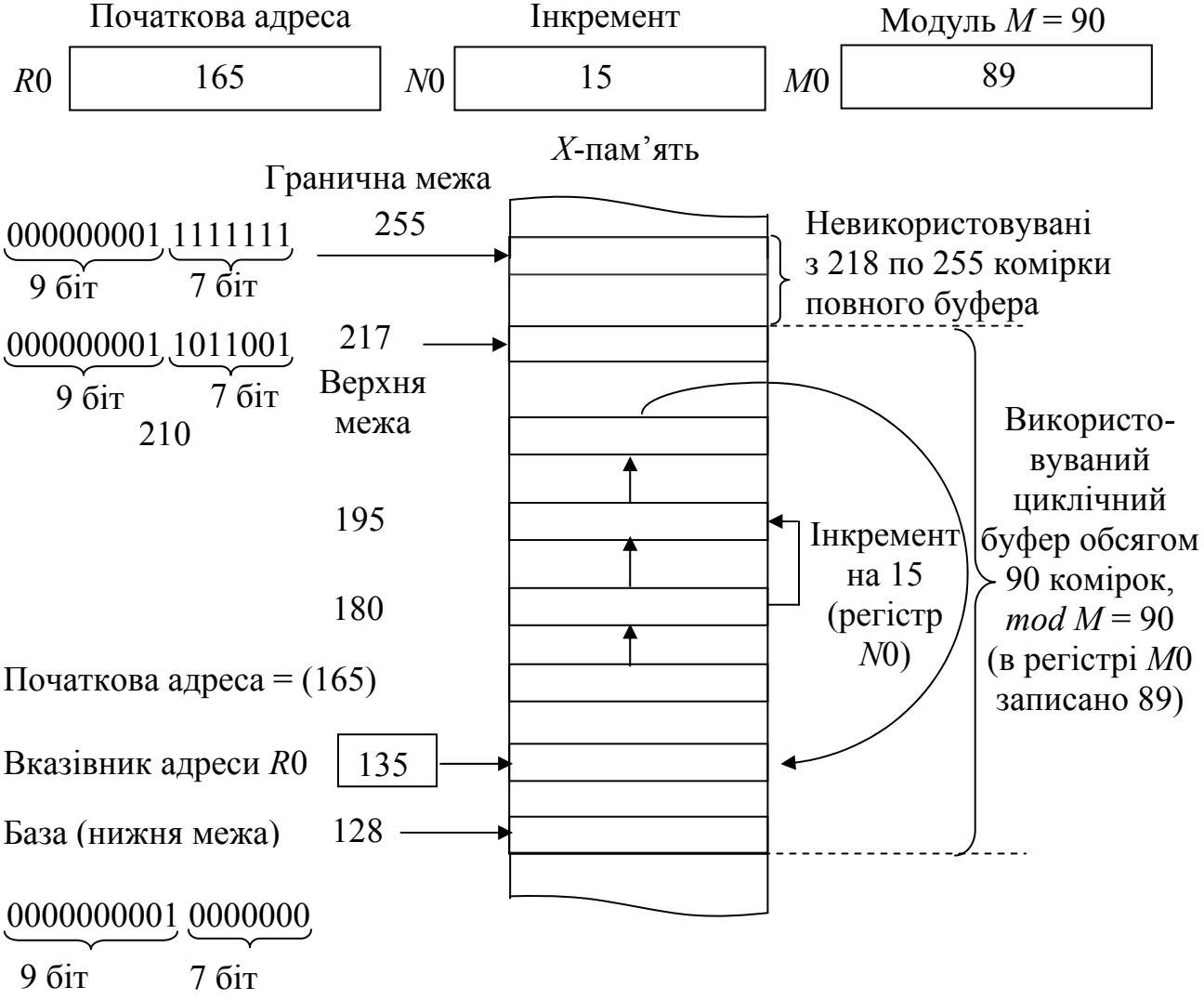



Рисунок 15.4 – Використовування арифметики за модулем $M$

Кількість молодших нульових бітів у базі вибрано з умови $2^k \geq 90$, тоді $k = 7$. $Bs = 128$ (крайнє $2^k$). Верхня межа становить $128 + 90 = 217$. Гранична межа становить $128 + 2^7 - 1 = 255$; комірки з 218 по 255 повного буфера не використовуються і можуть використовуватись для інших цілей. Після першого виконання команди початкова адреса інкрементується на 15 і вміст $R0$ дорівнює 180. При подальшому інкрементуванні адреси вміст $R0$ послідовно стає 195 та 210. При черговому інкрементуванні адреса циклічного буфера могла б стати 225 і вийти за верхню межу буфера. Модульна арифметика примушує вміст $R0$ залишатися усередині буфера, тобто $(R0) = 225 - 90 = 135$.

**Контрольні питання:**

1 Як вибирається початкова адреса в межах циклічного буфера?
2 У якій пам'яті, $X$ або $Y$, може бути зорганізовано циклічний буфер?
3 В який спосіб визначаються верхня та нижня межі циклічного буфера?

**Контрольні питання підвищеної складності:**

1 З якою метою у *DSP* використовується циклічний буфер?

2 Організуйте у пам'яті циклічний буфер обсягом сім комірок.

3 Вкажіть значення нижньої, верхньої та граничної межі в пам'яті циклічного буфера.

4 Покажіть, як спрацьовує модульна арифметика при обчисленні адреси пам'яті у межах циклічного буфера?

## 15.4 Програмна реалізація цифрового фільтра СІХ

**Вхідний контроль:**

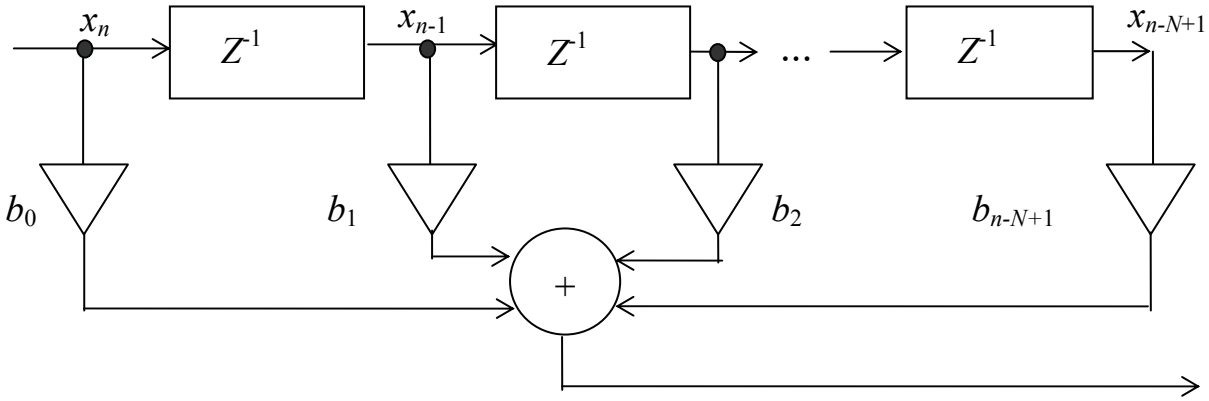
1 На якій операції цифрової обробки сигналів базується алгоритм реалізації цифрового фільтра СІХ?

2 Фільтри яких частот можна побудувати на базі цифрових СІХ-фільтрів?

Алгоритм реалізації фільтра має вигляд:

$$y_n = \sum_{k=0}^{N-1} b_k x_{n-k}\ , \quad \text{де } n = 0, 1, 2, \ldots$$

Цифровий фільтр має структуру:




Фрагмент програми, яка зреалізовує цифровий фільтр 4-го порядку:

```
1 ADDR_A EQU 5000          ; Задання адреси циклічного
                           ; буфера (ЦБ)
2 ADDR_B EQU 6000          ; Задання адреси b_k
3 ADDR_C EQU 4000          ; Задання адреси x_n
4 ADDR_D EQU 3000          ; Задання адреси y_n
5 ADDR_E EQU 2000          ; Задання адрес N та M
6 ORG Y:ADDR_B             ; В Y-пам'яті, розпочинаючи
7 DC 0.9,0.8               ; з адреси 6000, записано
8 DC 0.7,0.6               ; вектор коефіцієнтів b
```

```
9 ORG X:ADDR_C                          ; В X-пам'яті, розпочинаючи
                                        ; з адреси 4000, записано вектор
                                        ; відліків xn
10 ORG X:ADDR_E                         ; В X-пам'яті за адресою 2000
11 DC 4                                 ; занесено розмір буфера N
12 ORG Y:ADDR_E                         ; В Y-пам'яті за адресою 2000
13 DC 4                                 ; занесено кількість відліків M
14 ORG P:$100                           ; Об'єктний код завантаження до
                                        ; пам'яті команд,
                                        ; розпочинаючи з адреси $100
15 MOVE #ADDR_E,R6                      ; R6 – вказівник на N та M
16 MOVE #ADDR_D,R5                      ; R5 – вказівник на Yn
17 MOVE #ADDR_B,R4                      ; R4 – вказівник на bk
18 MOVE #ADDR_C,R1                      ; R1 – вказівник на вектор xn
19 MOVE #ADDR_A,R0                      ; R0 – вказівник на ЦБ
20 MOVE X:(RG),A                        ; Запис N до акумулятора
21 DEC A                                ; Організація ЦБ
22 MOVE A,M0                            ; довжиною N (M0 = N – 1)
23 CLR A                                ; Обнулення циклічного
24 REP X:(R6)                           ; буфера
25 MOVE A,X(R0)+
26 MOVE M0,M4
27 MOVE X:(R1)+,X0                      ; Завантаження x0→X0
28 DO Y:(R6),loop
29 CLR A X0,X(R0)- Y:(R4)+,Y0           ; xn→циклічний буфер
                                        ; bk→Y0
30 REP M0
31 MAC X0,Y0,A X: (R0)-, X0 Y: (R4)+, Y0 ; Команда MAC-операції; сума
                                        ; bk·xn-k від k = 0 до k = N – 2
32 DO Y: (R6), loop                     ; xn→ЦБ
33 CLR A X0, X: (R0)- Y: (R4)+, Y0      ; bk→Y0
34 REP M0
35 MAC X0, Y0, A X:(R0)-,X0 Y:(R4)+,Y0  ; Сума bk·xn-k від k = 0 до k = N – 2
36 MACR X0,Y0,A (R0)+                   ; A – yn, R0 вказує на xn-N+1
37 MOVE X:(R1)+,X0 A,Y:(R5)+            ; xn→X0, запис результату
                                        ; yn до Y-пам'яті за адресою
                                        ; від 3000
   loop                                 ;
```

Команда *MAC* (±)*S*1,*S*2, *D* виконує знакове множення з акумулюванням: множення двох знакових 24-бітних операндів джерела *S*1 та *S*2 та додавання або віднімання результату з/від акумулятора приймача *D*; результат зберігається в *D*.

Команда *MACR* (±)*S*1,*S*2, *D* виконує знакове множення з акумулюванням та округленням: множення двох знакових 24-бітних операндів джерела *S*1 та *S*2,

додавання або віднімання результату з/від акумулятора приймача *D* та подальше округлення результату з використовуванням конвергентного округлення; результат зберігається в *D*.

**Контрольні питання:**

1 Які операції виконує команда *MAC*?

2 З якою метою виконується обмеження результату за програмної реалізації СІХ-фільтра?

3 У який спосіб у процесорі *DSP*563*XX* апаратно підтримується команда *MAC X0, Y0, A X: (R0)$^{-}$, X0 Y: (R4)$^{+}$, Y0* ?

**Контрольні питання підвищеної складності:**

1 Наведіть структуру цифрового фільтра 5-го порядку.

2 У яких командах наведеного вище фрагмента програми треба зробити зміни, щоби зреалізувати цифровий фільтр 5-го порядку, і які саме?

# 16 МПС НА МІКРОКОНТРОЛЕРАХ, МІКРОПРОЦЕСОРАХ ТА *DSP*

**Вхідний контроль:**

1 Яку архітектуру можуть мати багатопроцесорні системи?

2 Які функції має виконувати інтерфейс у багатопроцесорній системі?

3 Яку розрядність шини адреси можуть мати мікроконтролери фірми *Motorola MC*68*HC*05*J* та *MC*68*HC*11?

4 З якою метою зреалізовується клямкування адреси у МПС, якщо шина адреси/даних мікропроцесора є мультиплексована?

5 З якою метою в МПС використовується пріоритетний шифратор?

6 З якою метою в МПС використовується декодер вектора переривань?

*DSP* працюють переважно у складі багатопроцесорних систем, принаймні двопроцесорних, де другим процесором, так званим *Host*-процесором, може бути мікропроцесор, мікроконтролер, інший *DSP* або апаратний ПДП. Зв'язок поміж *DSP* та *Host*-процесором зреалізовується через *Host*-інтерфейс (*HI*). На прикладі *Host*-інтерфейсу *DSP* сімейства *DSP*56000 фірми *Motorola* розглянемо його структуру. *HI* – 8-бітний повнодуплексний, з подвійною буферизацією паралельний порт, який долучується безпосередньо до шини даних Host-процесора. *HI* – асинхронний інтерфейс, який вміщує два банки регістрів: один, доступний *Host*-процесору, і другий банк, доступний процесору *DSP*. *HI* забезпечує швидкість передавання пакетів 8 Мбайт/с, максимальна швидкість передавання даних при обробленні переривань становить 1,71 мільйона 24-розрядних слів/с.

Структуру *HI* подано на рис. 16.1. Процесор *DSP* розглядає *HI* як периферійний пристрій, який займає три 24-бітові слова у просторі пам'яті даних. Регістри інтерфейсу є доступні за допомогою стандартних команд процесора і способів адресування. При програмному та апаратному скиданні *HI* використовується задля стандартного введення-виведення.

На рис. 16.2 наведено двопроцесорну систему на базі *DSP*56000 та мікроконтролера *MC*68*HC*11 фірми *Motorola*, який має мультиплексовані шини адреси та даних і тому потребує клямкування адреси. Усі невикористовувані входи може бути підімкнено до живлення через резистор, як, наприклад, ¬*HACK*, задля запобігання виникнення помилкових сигналів.

Двоспрямована шина даних *H*7...*H*0 використовується для передавання даних поміж *Host*-процесором та *DSP*. Виходи адреси *HA*2...*HA*0 забезпечують адресне вибирання регістрів *HI*, вони є стабільні, якщо вхід ¬*HEN* дозволу переривань має низький активний рівень. Вхід *HR*/¬*W* вибирає напрямок передавання даних при доступі до *Host*-процесора. Якщо *HR*/¬*W* = 1 та ¬*HEN* є активний, дані передаються з *DSP* до *Host*-процесора; якщо *HR*/¬*W* = 0, то за активного входу ¬*HEN* дані передаються з *Host*-процесора до *DSP*. Вхід *HR*/¬*W* є стабільний, якщо ¬*HEN* є активний. Вхід ¬*HEN* – дозвіл *HOST*, дозволяє передавати дані шиною даних *HOST*. Якщо ¬*HEN* не є активний, лінії шини даних перебувають у третьому стані. Вихід ¬*HREQ* забезпечує надходження від *DSP* до *Host*-процесора, контролера ПДП або іншого зовнішнього контролера

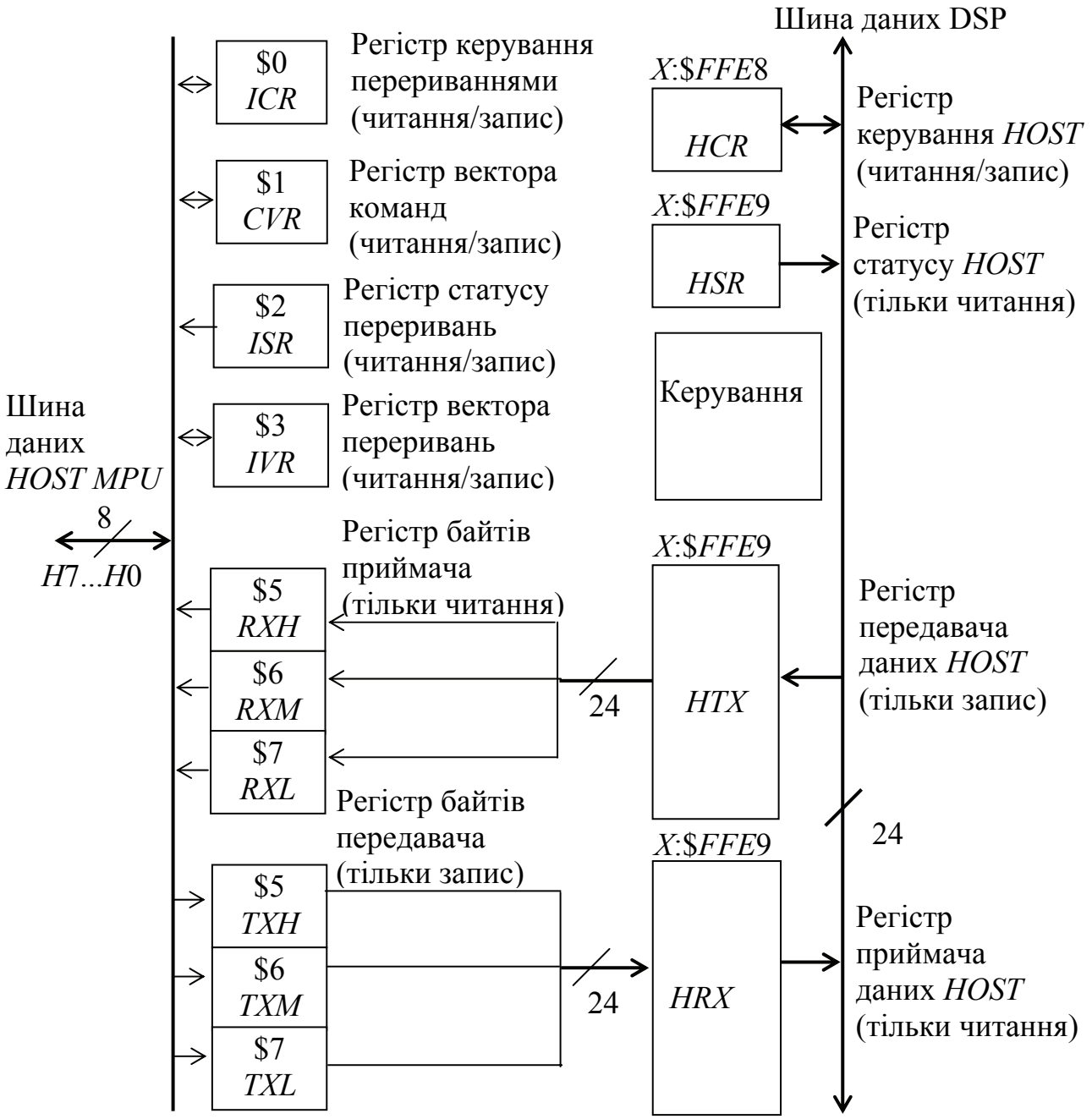



Рисунок 16.1 – Структура *HI*

сигналу запиту на переривання ¬*IRQ*. Вихід ¬*HREQ* може сполучуватись з контактом запиту на переривання *Host*-процесора, запитом передавання контролера або входом керування зовнішнього пристрою. Вихід ¬*HACK* – забезпечення сигналу відповіді за операцій ПДП та сигналу відповіді – за переривання для сумісності з процесорами сімейства *MC*68*XXX*. У першому випадку сигнал ¬*HACK* використовується для стробування даних за операцій ПДП, а у другому – для дозволу видавання на шину даних вектора переривань, якщо сигнал *HREQ* є активний.

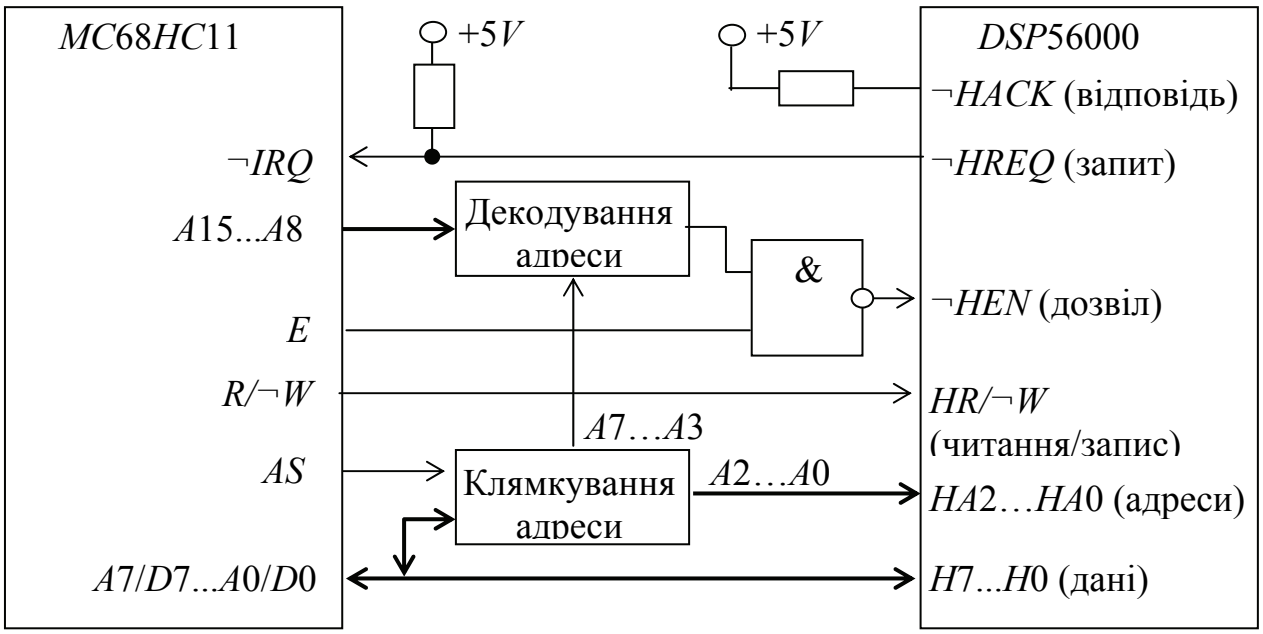



Рисунок 16.2 – Система на базі *DSP* та *MC*68*HC*11

На рис. 16.3 наведено двопроцесорну систему на базі *DSP*56000 та процесора *MC*68000 фірми *Motorola*. Процесор *MC*68000 може використовувати команду *MOVEP* зі словом, довгим словом задля передавання до *DSP* або читання з нього послідовності даних. При використовуванні в якості Host-процесора *MC*68020 або *MC*68030 у будь-якій команді може використовуватись динамічно змінюваний розмір шини.

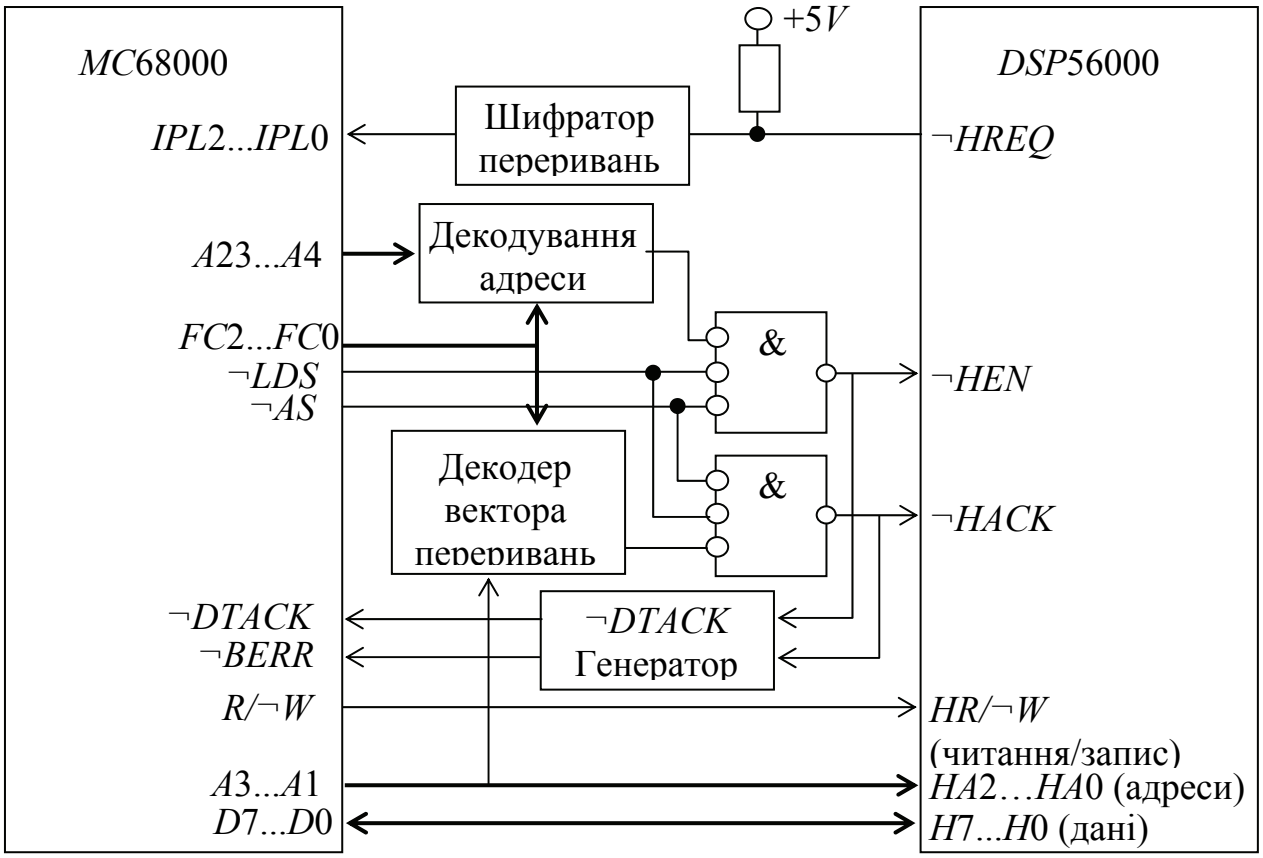



Рисунок 16.3 – Система на базі DSP та MC68000

На рис. 16.4 наведено мікропроцесорну систему з чотирьох *DSP*, які сполучуються за допомогою одного *Host*-інтерфейсу. Така МПС може виконувати до 41-го мільйона команд на секунду і може масштабуватись для підвищування продуктивності. *SSI* на рис. 16.4 – послідовний інтерфейс.

**Контрольні питання:**

1 Яку роль у багатопроцесорних системах відіграє мікроконтролер і яку – *DSP*?

2 Яку роль у багатопроцесорних системах відіграє мікропроцесор і яку – *DSP*?

3 За допомогою якого вузла *DSP* зреалізовується долучання його до *Host*-процесора?

4 Які пристрої можуть використовуватись в якості *Host*-процесора?

**Контрольні питання підвищеної складності:**

1 У який спосіб можуть обмінюватися інформацією процесори у багатопроцесорних системах?

2 У який спосіб за послідовного чи паралельного передавання даних зреалізовується обмін поміж *DSP* та *Host*-процесором?

3 Чи є передавання даних через *Host*-інтерфейс: синхронне? асинхронне?

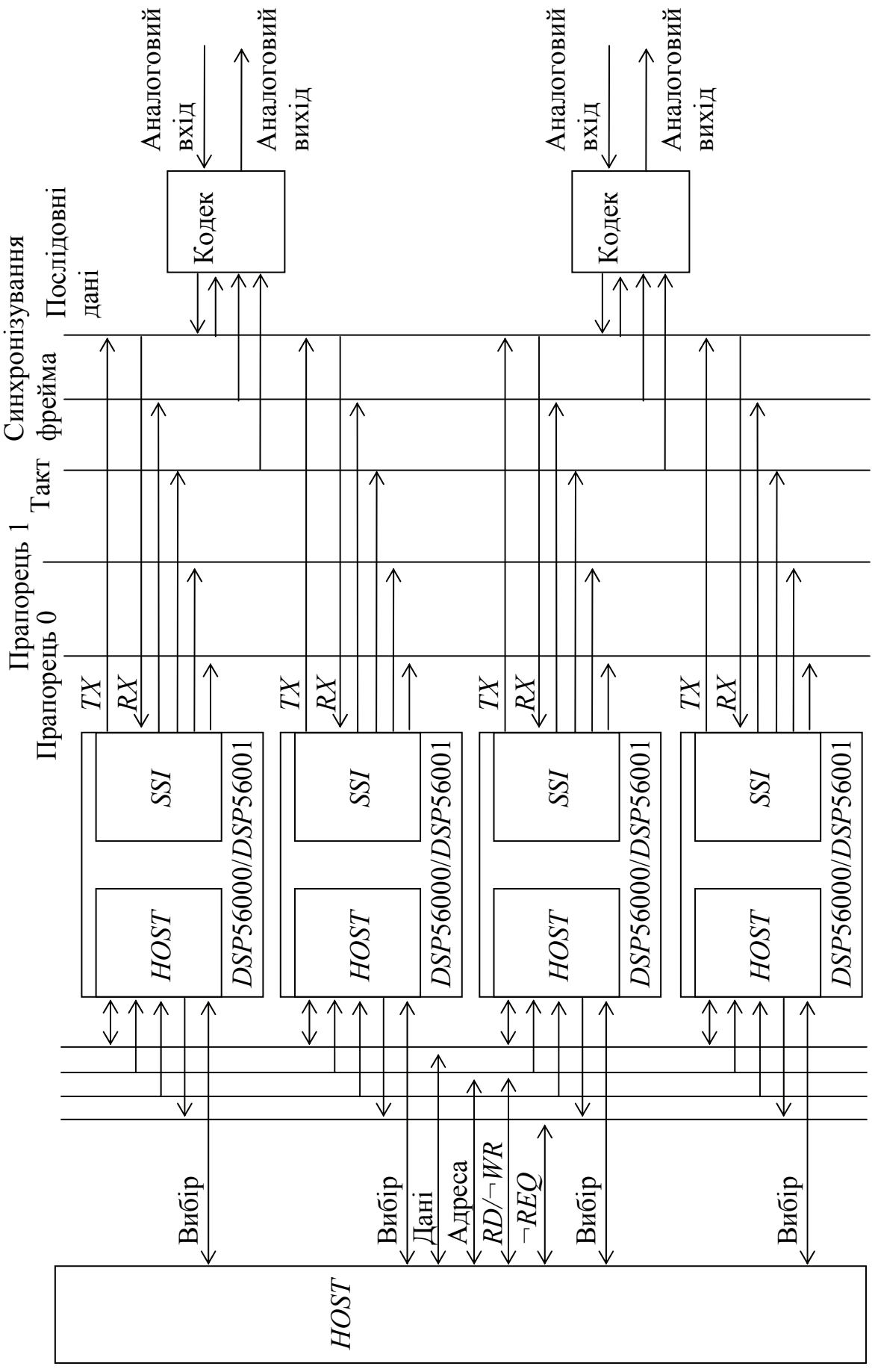



Рисунок 16.4 – Система на базі кількох *DSP*

## СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ДО Частини II 2-го МОДУЛЯ


1 Шагурин И. И. Микропроцессоры и микроконтроллеры фирмы *Motorola*: справ. пособие / Шагурин И. И. – М.: Радио и связь, 1998. – 560 с.: ил.

2 Шагурин И. И. Современные микроконтроллеры и микропроцессоры *Motorola*: справ. пособие / Шагурин И. И. – М.: Горячая линия – Телеком, 2004. – 952 с.: ил.

3 *MC*68*HC*05 *APPLICATIONS GUIDE – ©MOTOROLA INC*., 1989.

4 *MC*68*HC*705*J*1*A TECHNICAL DATE – ©MOTOROLA*, 1995.

5 *James M. Sibigtroth MC68HC705J1A Understanding Small Microcontrollers – ©MOTOROLA*, 1995.

6 Куприянов М. С. Цифровая обработка сигналов: процессоры, алгоритмы, средства проектирования / Куприянов М. С., Матюшин Б. Д. – [2-е изд., перераб. и доп.] – С.Пб.: Политехника, 1999. – 592 с.: ил.

7 Солонина А. И. Цифровые процессоры обработки сигналов фирмы *Motorola* / Солонина А. И., Уласович Д. А., Яковлев Л. А. – С.Пб.: БХВ-Петербург, 2000. – 512 с.: ил.

8 Белами Дж. Цифровая телефония / Белами Дж.; Пер. с англ.; под ред. А. Н. Берлина, Ю. Н. Чернышова. – М.: Око-Трендз, 2004. – 640 с.: ил.

# ПРЕДМЕТНИЙ ПОКАЖЧИК


***D***

*DSP*563*XX*, 424

***R***

*RISC*-процесори
- *ColdFire*, 407
- *PowerPC*, 401

**А**

Арифметично-логічний пристрій (АЛП), 15, 16, 27, 28, 52, 57, 58, 59, 60, 61, 62, 65, 96, 97, 98, 110, 113, 114, 115,135, 157, 246, 423, 425,427

Архітектура
- мікропроцесорів, 93, 94, 99, 100, 110, 111, 120, 130, 133, 136, 137, 138, 139, 141, 243, 285, 343, 380, 402, 408, 420, 422, 423, 424, 425, 426
- обчислювальних систем, 9, 10, 12, 145, 146, 232

Асинхронний послідовний адаптер *RS*-232-*C*, 86, 222, 233, 235

**З**

Запам’ятовувальний пристрій, 16, 17, 18, 25, 26, 27, 52, 58, 64, 65, 66, 68, 72, 73, 74, 77, 128, 259, 260, 345, 361
- класифікація, 64, 65
- постійний, 18, 25, 66, 68, 77, 89, 259, 261, 262, 345, 348, 361, 371, 375
- оперативний, 18, 66, 67, 68, 72, 73, 74, 77, 81, 120, 128, 259, 261, 262, 263, 264, 348, 361, 371, 375, 408

**І**

Інтерфейс, 11, 16, 17, 19, 21, 23, 24, 83, 84, 85, 86, 103, 127, 128, 135, 138, 139, 140, 146, 147, 149, 152, 158, 229, 259, 261, 266, 267, 268, 278, 279, 283, 284, 292, 301, 302, 303, 305, 361, 362, 368, 375, 376, 377, 378, 407, 408, 409, 410, 423, 424, 426, 432, 435

**К**

Комп’ютер, 9, 10, 12, 15, 16, 17, 18, 19, 20, 21, 22, 23, 30, 66, 69, 76, 86, 87, 109, 140, 141, 142, 144, 156, 212, 234, 243, 244, 333, 334, 401

Команди мови Асемблер-86
- зсувів, 201, 202
- логічних операцій, 201
- організації циклів, 114, 207
- пересилань, 156, 179, 213
- перетворення даних мови Асемблер-86, 189
- умовних та безумовних переходів, 200, 203, 204, 216, 217

Команди мови Асемблер МП *MC*68*XXX*
- арифметичних операцій, 313, 385, 387, 411, 412
- звернення до підпрограм, 325
- зсувів, 319, 388, 389, 412, 413
- логічних операцій, 319, 388, 412
- пересилань, 312, 315, 325, 385, 386
- умовних та безумовних переходів, 321, 322, 392, 406, 417

**М**

Мікроконтролери
- 8-розрядні, 333, 344
- 16-розрядні, 375
- 32-розрядні, 344, 380

Мікропроцесори *Pentium*, 133, 401

Мікропроцесорна система (МПС), 9, 10, 11, 22, 23, 24, 25, 26, 27, 28, 35, 43, 52, 57, 58, 62, 63, 64, 65, 66, 67, 71, 72, 74, 77, 78, 79, 80, 81, 83, 84, 85, 86, 93, 95, 98, 99, 100, 101, 102, 103, 104, 106, 108, 111, 141, 156, 160, 169, 170, 229, 231, 243, 244, 246, 248, 250, 251, 253, 256, 258, 259, 260, 261, 262, 267, 277, 279, 288, 289, 291, 292, 293, 295, 326, 327, 332, 333, 334, 337, 338, 339, 340, 341, 343, 408, 432, 435
- на мікроконтролерах, мікропроцесорах та *DSP*, 432

Мікросхеми пам’яті, 68, 77, 262, 263, 264, 295, 298

Мова Асемблер програмування МП фірми *Motorola*, 307

Мова програмування Асемблер-86, 169, 189, 190, 201, 209, 229, 234

МП фірми *Intel*, 95, 111, 120, 144, 156, 160, 161, 169, 177, 243, 246
МП фірми *Motorola*, 243, 307, 328, 332, 333

**Н**
Налаштовування вбудованих засобів мікроконтролерів, 394

**О**
Обчислювальна система, 9, 10, 12, 15, 19, 20, 21, 29, 52, 133, 169, 230, 231, 421
Організація
16-розрядних мікропроцесорів, 109
32-розрядних мікропроцесорів, 120
підсистеми введення-виведення для МПС на *MC*68000, 266
підсистеми введення-виведення для МПС на *MC*68020, 298
підсистеми пам'яті для МПС на *MC*68000, 262
підсистеми пам'яті для МПС на *MC*68020, 295
Основні напрямки цифрового оброблення сигналів (ЦОС), 420, 422

**П**
Побудова МПС
на 16-розрядних мікропроцесорах фірми *Motorola*, 256
на 32-розрядних мікропроцесорах фірми *Motorola*, 288
Подання даних
в обчислювальних системах, 29
у кодах, 36
Порозрядні операції над даними, 39
Програмна модель
МП I8086, 113
МП *MC*68000, 244
МП *MC*68020, 285, 286
Програмне забезпечення МПС на МП фірми *Motorola*, 332
Програмовані логічні інтегральні схеми (ПЛІС), 52, 62, 63, 230
Програми
лінійні, 209, 328
розгалужені, 215, 329
циклічні 209, 221, 329
Продуктивність мікропроцесорів, 141
Процесори фірми *AMD*, 139

**Р**
Режим переривань МП *I*8086, 116
Розподіл адресного простору МПС
на МП *MC*68000, 260, 261
на МП *MC*68020, 292, 293
Розробка апаратно-програмних комплексів, 230

**С**
Сегментування пам'яті, 110, 156, 160
Співпроцесор, 15, 19, 20, 112, 120, 127, 190, 243, 253, 285, 289, 304, 305, 315, 424
Способи адресування операндів МП фірми *Intel*, 161, 169
Способи реалізації алгоритмів, 229
Суматори, 25, 43, 52, 53, 54, 55, 57, 58, 97, 111, 334
Система команд мікроконтролерів фірми *Motorola*, 384
Система команд *RISC*-мікропроцесорів сімейства *PowerPC*, 411

**Ф**
Формат команди, 175, 177, 202
Фрагмент програми
передавання даних через асинхронний адаптер *RS*-232-*C*, 235
приймання даних через асинхронний адаптер *RS*-232-*C*, 235
Функціонування обчислювального пристрою, 25, 26
Функціонування обчислювальних систем, 12

**Ц**
Циклічний буфер, 426, 427, 428
Цифрові автомати (ЦА), 43, 44
Цифрові компаратори, 52, 55, 56
Цифровий фільтр СІХ, 420, 429

Навчальне видання

**Хіхловська Ірина Вільгельмівна**
**Антонов Олександр Сергійович**

# Обчислювальна техніка та мікропроцесори

**Підручник**

Редактор –
Верстка – Ж.А. Гардиман

Видавець ОНАЗ ім. О.С. Попова
(Свідоцтво ДК №3633 від 27.11.09 р.)
м. Одеса, вул. Ковальська, 1

Підписано до друку 1.05.2011 р.
Формат . Обсяг д.а.
Наклад 300 прим. Зам. № 871.
Віддруковано з готового оригінал-макету
у типографії «Абрикос-Компани»
м. Одеса, вул. Зоопаркова, 25