МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”
КАФЕДРА АВТОМАТИКИ ТА УПРАВЛІННЯ У ТЕХНІЧНИХ СИСТЕМАХ

Комп’ютерна електроніка: Мікропроцесорні системи: Програмування мікропроцесорних систем: Навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія»

Київ НТУУ “КПІ” 2014

Комп’ютерна електроніка: Мікропроцесорні системи: Програмування мікропроцесорних систем: Навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах /Автор.: А.О. Новацький– К: НТУУ „КПІ”, 2014– 307с.

Автор: А.О. Новацький

Відповідальний за випуск: Л.Ю. Юрчук

Рецензенти: А.Ю.Дорошенко, О.А.Чемерис

Посібнику надано гриф «Рекомендовано Методичною радою НТУУ «КПІ»
Протокол № 10 від 19 червня 2014 р

Посібник складається із 8 розділів, в яких розглядаються питання, пов`язані із програмуванням мікропроцесорнах систем: основні поняття мікропроцесорної техніки; схеми типових МПС, МП–рів та МК–рів; мова Асемблера; характеристика команд типового МП–ра та МК–ра та ін.

Робота може бути корисною студентам відповідних спеціальностей при вивченні дисциплін, пов’язаних із використанням та проектуванням мікропроцесорних систем, а також при виконанні бакалаврських робіт, курсових та дипломних проектів, в яких використовуються мікропроцесорні пристрої. Останнє було враховано при оформленні роботи, яке виконано згідно вимог до конструкторської документації.

ЗМІСТ

ВСТУП

Навчальний посібник орієнтовано на підготовку фахівців з напряму «Системна інженерія». Цей напрям сьогодні використовується в якості інтеграційної основи при організації та здійсненні діяльності по розробці інформаційних систем широкого класу. Для успішного оволодіння майбутніми спеціалістами цим напрямом необхідно засвоїти принципи побудови та використання мікропроцесорних пристроїв, на основі яких розробляються та функціонують мікропроцесорні та обчислювальні системи. Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для вивчення базової частини дисципліни «Комп`стерна електроніка». Навчальний посібник складається із восьми розділів, в яких розглядаються питання, пов`язані із програмуванням мікропроцесорних систем: основні поняття та особливості мікропроцесорної техніки; структурна та функціональна схеми мікропроцесорних систем; структура типового мікропроцесора та мікроконтролера; програмні моделі МП–ра та МК–ра; мова Асемблера та її використання у МПС; характеристика команд МП–рів та МК–рів; способи адресації операндів та команди типового МП–ра і8086 та МК–ра AT89C51. Тематика посібника відповідає робочій програмі з дисципліни «Комп`стерна електроніка», яка є нормативною дисципліною у навчальному плані підготовки бакалаврів з напряму 6.050201 «Системна інженерія». Робота може бути корисною студентам відповідних спеціальностей при вивченні дисциплін, пов’язаних із проектуванням та використанням мікропроцесорних систем, а також при виконанні бакалаврських робіт, курсових та дипломних проектів, в яких використовуються мікропроцесорні пристрої.

СПИСОК СКОРОЧЕНЬ

АВЕ – аналоговий виконавчий елемент

АК – аналоговий компаратор

АЛП – арифметико–логічний пристрій

АМПС – аналоговий мультиплексор

АЦП – аналого–цифровий перетворювач

БО – безпосередній операнд

БСШ – блок сполучення з шиною

ВБ – виконавчий блок

ВІ – вузол індикації

ВПДП – вимога прямого доступу до пам’яті

ДК – двійковий (машинний) код

ДНЛ – диференціальна нелінійність

ДОН – джерело опорної напруги

ДРВВ – додаткові регістри введення/виведення

ЕСППЗП – електронно стираємий програмований постійний запам’ятовуючий пристрій

ЗВПР – зовнішній пристрій

ЗПП – зовнішня пам'ять програм

ЗРС – зовнішній регістр стану

ІМС ОП – інтегральна мікросхема операційного підсилювача

ІНЛ – інтегральна нелінійність

КПДП – контролер прямого доступу до пам’яті

КЗ – канал зв’язку

КМОН – комплементарний метал–оксид–напівпровідник

КОП – код операції команди

КПЕР – контролер переривань.

ЛМПСК – локальна мікропроцесорна система керування

МА – мова асемблера

МЗР – молодший значущий розряд

МК – мікроконтролер

МПЛ – мультиплексор

МПС – мікропроцесорна система

МПСК – мікропроцесорна система керування

НПДП – надання прямого доступу до пам’яті

ОЗП – оперативний запам’ятовуючий пристрій

ОК – об'єкт керування

ОП – операційний підсилювач

ПВВ – порти введення/виведення

ПВЗ – пристрій вибірки–зберігання

ПДП – прямий доступ до пам'яті

ПЗП – постійний запам’ятовуючий пристрій

ПЛМ – програмована логічна матриця

ППЗП – перепрограмовуваний постійний запам'ятовуючий пристрій

ПП – пам’ять програм

ППІ – паралельний програмований інтерфейс

ППР – периферійний пристрій

РВВ – регістр введення/виведення

РЗП – регістр загального призначення

РПП – резидентна пам'ять програм

СЗР – старший значущий розряд

СМА – суматор адреси

СОЗП – статичний оперативний запам’ятовуючий пристрій

СПД – статична пам’ять даних

СУР – схема узгодження рівнів

ТТЛШ – транзисторно–транзисторна логіка з діодами Шоттки

УАПП – універсальний асинхронний програмований приймач/передавач

УСАПП – універсальний синхронно/асинхронний приймач–передавач

ФАШ – формувач адресної шини

ФКС – формувач керуючих сигналів

ФШД – формувач шини даних

ЦАП – цифро–аналоговий перетворювач

ЦПП – центральний процесорний пристрій

ША – шина адреси

ШД – шина даних

ШІМ – широтно–імпульсна модуляція

ШФ – шинний формувач

ACK (acknowledge) – підтвердження

CPHA (Clock Phase) – фаза тактового сигналу

CPOL (Clock Polarity) – полярність тактового сигналу

EEPROM–пам’ять (Electrically Erasable Programmable Read–Only Memory) – енергонезалежна пам'ять, що може бути електрично стерта та перепрограмована

ETIMSK (Extended Timer/Counter Interrupt MaSK Register) – розширений регістр маски переривань від таймерів/лічильників

FIFO (First In, First Out) – принцип «першим зайшов, першим вийшов», якому відповідає структура даних «черга»

GTCCR (General Timer/Counter Control Register) – загальний регістр керування таймером/лічильником

IEEE (Institute of Electrical and Electronics Engineers) – Інститут інженерів з електротехніки та електроніки

I2C (InterIC, або IIC) – двонаправлена двопровідна шина для так званого міжмікросхемного застосування

IR – регістр команд

JTAG (Joint Test Action Group) – інтерфейс, призначений для підключення складних цифрових мікросхем або пристроїв рівня друкованої плати до стандартної апаратури тестування і налагодження

MISO (Master In Slave Out) – сигнал входу для ведучого, виходу для веденого

MOSI (Master Out Slave In) – сигнал виходу для веденого, входу для ведучого

NACK (no acknowledge) – не підтвердження

OCR (Output Compare Register) – регістр порівняння

PC – лічильник команд

SCK (Serial Clock) – лінія тактового сигналу

SCL (Serial CLock) – послідовна лінія тактування

SDA (Serial DAta) – послідовна лінія даних

SFIOR (Special Function Input Output Register) – регістр спеціальних функцій введення/виведення

SP – покажчик стека

SPI (Serial Peripheral Interface) – послідовний периферійний інтерфейс

SS – стековий сегмент

TAP (Test Access Port) – порт тестового доступу

TCCR (Timer/Counter Clock Control Register) – регістр керування таймером/лічильником

TIFR (Timer/Counter Interrupt Flag Register) – регістр прапорців переривань від таймерів/лічильників

TIMSK (Timer/Counter Interrupt MaSK Register) – регістр маски переривань від таймерів/лічильників

TWI (Two Wire (Serial) Interface) – двопровідний інтерфейс

UDR (Universal Data Register) – буферні регістри приймача і передавача

USART (Universal Synchronous and Asynchronous Receiver and Transmitter) – універсальний синхронний/асинхронний приймач–передавач

WDTCR (Watch Dog Timer Control Register) – регістр керування вартовим таймером

1 ОСНОВНІ ПОНЯТТЯ ТА ОСОБЛИВОСТІ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

1.1 Деякі положення та визначення мікропроцесорної техніки

Мікропроцесор (МП) — пристрій, що обробляє інформацію у відповідності з програмою, що подається у вигляді команд на його входи, і реалізований в одній великій інтегральній схемі (ВІС). Очевидно, що МП не може функціонувати без інших інтегральних схем, які виконують функції пам'яті, введення/виведення, синхронізації, узгодження за навантаженням і т. ін.

МП виконує арифметичні і логічні операції, аналізує і приймає рішення, що змінюють процес обчислення, керує процесом введення і виведення інформації, тобто він реалізує функції, зазвичай покладені на центральний процесор ЕОМ.

Мікро–ЕОМ — універсальний блок обробки даних і формування керуючих сигналів, призначений для вбудовування в різні спеціалізовані системи контроля і керування в цілях розширення їх функціональних можливостей. Мікро–ЕОМ – включає в себе мікропроцесор, пам'ять (ОЗП, ПЗП), пристрій введення/виведення (ПВВ) і т. ін..

Сукупність інтегральних схем, сумісних за конструктивно–технологічним виконанням і призначених для спільного використання при побудові модуля МП, мікро–ЕОМ і мікропроцесорних систем, називається *мікропроцесорним комплектом.*

Поява МП, перш за все, пов'язана з успіхами інтегральної технології. З моменту «виходу в світ» ВІС, одразу став очевидним той факт, що тільки ті схеми цифрової обчислювальної техніки можуть бути ефективно реалізовані в них, які мають наступні властивості: регулярність структури; серійність, тобто будуть вироблятися великим тиражем; обмежене число

входів і виходів, відношення якого до загальної складності ВІС**,** тобто до числа транзисторів, реалізованих в ньому, повинно бути невеликим.

Серед стандартних пристроїв обчислювальної техніки ці властивості притаманні запам'ятовуючим пристроям, які першими в середині 60–х років почали випускатися у вигляді ВІС. Що стосується керуючих автоматів, то реалізація їх в ВІС пов’язана з великими витратами через те, що кожна керуюча схема — унікальна і, отже, вимагає створення спеціалізованої (замовленої) ВІС в єдиному екземплярі. Прагнення виробників ВІС уникнути великих витрат і разом з тим задовольнити попит на будь–які різноманітні пристрої, здатні виконувати**,** як обчислювальні, так і керуючі функції, привело до думки про створення програмно–керованих універсальних кристалів, тобто наділити логічний елемент властивістю програмуватися, яка залишається з ним в процесі його експлуатації. Виділивши в керуючому пристрої центральний процесор, пам'ять, схеми керування обміном з зовнішнім світом, різні фірми і промислові об'єднання почали виготовляти універсальні елементи, здатні виконувати різноманітні послідовності команд, тобто які мають властивість програмуватися. Ідея ЕОМ була реалізована на рівні кристала. Перші МП були призначені відповідно для застосування в калькуляторах і відеотерміналах, однак завдяки їх універсальності вони знайшли широке застосування і в інших пристроях.

Центральною ланкою будь–якого МП є арифметико–логічний пристрій. Крім того, до складу МП можуть входити: швидкодіюча пам'ять для збереження операндів і проміжних результатів — регістри загального призначення (дуже часто серед них виділяється акумулятор — регістр, в якому зберігається операнд і поміщається результат операції); регістри спеціального призначення — лічильник команд (PC), регістр команд (IR), прапорці і покажчик стека (SP); дешифратор команд і схеми організації

машинних циклів; схеми синхронізації; схема формування сигналів, необхідних для керування обміном з пам'яттю і зовнішніми пристроями, а саме – реалізації обміну за перериванням і в режимі прямого доступу до пам'яті мікропроцесорної системи; формувачі — схеми, необхідні для збільшення навантажувальної здібності виводів, та узгодження схем з різноманітним типом технологій. Прийнято ВІС, що виконують функції МП або деяку частину з них і що вміщують деяку підмножину перерахованих пристроїв, називати мікропроцесорними ВІС**.**

Крім пристроїв, які входять до складу МП (крім власне МП–ра) в ВІС можуть бути вміщені: пам'ять (ОЗП – оперативний запам‘ятовуючий пристрій та ПЗП – постійний запам‘ятовуючий пристрій) для збереження даних, програм і результатів обробки; порти введення/виведення зі схемами керування; таймери; АЦП, ЦАП і т. ін. На теперішній час такі ВІС відносять до класу мікроконтролерів *(однокристальних мікро–ЕОМ)* (всі основні пристрої мікро–ЕОМ розміщені на одному кристалі). Треба підкреслити, що не можна провести чіткої межі між можливостями мікропроцесорів та однокристальних мікро–ЕОМ. Намітилась певна тенденція збільшення можливостей мікроконтролерів і поступового стирання граней між ними і мікропроцесорами. В свою чергу, в недалекому майбутньому передбачається включати до складу однокристальних мікро–ЕОМ потужні контролери введення/виведення, призначені для реалізації обміну з спеціалізованими зовнішніми пристроями, такими як накопичувачі на магнітних дисках, відеотермінали, клавіатура і т. ін. Серед властивостей мікропроцесорних ВІС**,** за якими можна оцінювати їх можливості, найбільш вагомими є: функціональне призначення, технічні характеристики, архітектурні особливості.

До числа основних характеристик МП–в і МК–в можна віднести: довжину слова даних (розрядність); кількість комірок пам'яті, що адресуються, і швидкодію.

Довжина слова даних визначається максимальною кількістю розрядів оброблюваних даних, що розглядаються апаратною частиною МП–ра (МК–ра), як єдине ціле. У більшості вживаних на теперішній час МП–х і МК–х довжина слова даних становить 8, 16 та 32 біти.

Кількість адресованих комірок пам'яті залежить від числа адресних виводів МП–ра або МК–ра. Наприклад, 8–розрядний МП–р i8080 має 16 адресних ліній (виводів), що дозволяє адресувати 2^{16} = 65536 комірок пам'яті, довжиною 8 біт (1 байт), тобто 65536 байт. Частіше для вказівки об'єму пам'яті використовують скорочення 1К = 1024. Тоді процесор i8080 адресує 64 Кбайт пам'яті. МП–р i8086 має 20 адресних виводів, тобто здатен адресувати 2^{20} = 1 Мбайт пам'яті. МК типу МК51 містить 16 адресних виводів, що забезпечує адресацію 64Кбайт пам'яті.

Швидкодія (продуктивність) мікропроцесора (МК–ра) визначається часом виконання окремих команд і програм. Для оцінки швидкодії МП–в і МК–в можна використовувати:

а) максимальне значення тактової частоти ($f_{такт}$) або мінімальне значення періоду тактової частоти (тривалість такту машинного циклу);

б) час виконання найпростішої операції, наприклад, пересилання із регістра в регістр;

в) час виконання тестових програм.

Іноді як основну характеристику МП–в або МК–в називають потужність, яка в значній мірі визначається названими раніше параметрами: довжиною слова даних, кількістю адресуємих комірок пам'яті і швидкодією.

1.2 Системи числення, коди, двійкова і двійково–десяткова арифметика, що використовуються в МПС

1.2.1 Двійкові, десяткові, двійково–десяткові (в упакованому і неупакованном форматі), восьми і шістнадцяткові числа і коди, що використовуються в МПС. Коди ASCII і КОІ–7

1.2.1.1 Загальна характеристика

Система числення – представляє собою сукупність способів і засобів запису чисел. В обчислювальній і мікропроцесорній техніці використовуються позиційні системи числення – двійкова, як основна, десяткова, шістнадцяткова і вісімкова, як допоміжні.

Крім названих систем числення, у розглянутій області техніки застосовуються коди с тими ж назвами:

– двійковий;

– шістнадцятковий і т.ін.

Про коди говорять в тих випадках, коли в тій чи іншій системі числення представлені дискретні повідомлення (дискретна інформація), що називаються даними.

1.2.1.2 Десяткова система числення

Найбільш часто використовується десяткова система числення, в якій числа утворюються за допомогою цифр від 0 до 9. Слово «цифра» перекладається від латинського *digitus*, що означає «палець». Така кількість цифр – десять, пояснюється тим, що у нас десять пальців. Десяткова система є прикладом *позиційної системи числення*, коли

положення (позиція, розряд) кожної цифри в числі визначає її значення, або вагу. Наприклад, число 374.29 є скороченим записом виразу

$$(3\times10^{2})+(7\times10^{1})+(4\times10^{0})+(2\times10^{-1})+(9\times10^{-2}).$$

Положення будь–якої цифри визначає степінь числа з основою 10 (вага позиції (розряду) числа), на яке ця цифра помножується. В даному прикладі 3 знаходиться в розряді сотень (10^{2}), 7 – десятків (10^{1}), 4 – одиниць (10^{0}), 2 – десятих долей (10^{-1}) і 9 – сотих долей (10^{-2}).

Основою десяткової системи є число 10. Назвемо три важливих характеристики позиційних систем числення:

- кількість цифр системи дорівнює її основі;
- найбільша цифра на одиницю менше основи;
- кожна цифра помножується на вагу позиції (основа в степені, значення якої визначається положенням цифри).

Крапка в позиційному представленні чисел називається *десятковою крапкою* і використовується для відділення цілої частини числа від дробової частини. Тому можна записати, що

$$N = N_I + N_F,$$

де N_I и N_F – відповідно ціла і дробова частини числа. В нашому прикладі N_I=374, а N_F=0.29.

1.2.1.3 Двійкова система числення

Найбільш простою позиційною системою числення є двійкова. Як зрозуміло з назви, система має основу 2. Для представлення чисел використовуються дві десяткові цифри – 0 і 1. Наприклад, число 1011.1101_2 еквівалентно десятковому числу:

$$(1\times2^{3})+(0\times2^{2})+(1\times2^{1})+(1\times2^{0})+(1\times2^{-1})+(1\times2^{-2})+(0\times2^{-3})+(1\times2^{-4})=11.8125_{10}.$$

Записуючи числа в тій чи іншій системі числення, підрядкові символи числа визначають основу системи числення. Їх часто опускають, якщо відомо, про яку основу йде мова.

Крім підрядкових символів для вказівки системи числення, в якій записано число, можуть використовуватися латинські букви: B – для двійкової (бінарної); D – для десяткової; H – для шістнадцяткової. Наприклад: 0110B; 7548D; 7598H; ABBAH.

Необхідно відзначити, що в наведеному прикладі крапка в позиційному (двійковому) представленні числа відокремлює його цілу і дробову частину, як і в випадку десяткової крапки. При позиційному принципі запису можливе безпосереднє переведення чисел із двійкової системи числення в десяткову.

1.2.1.3.1 Переведення чисел із десяткової системи числення в двійкову

Припустимо, ми хочемо представити десяткове число в двійковій формі. Одним із методів виконання такого перетворення є метод *ділення–множення*. При використанні цього методу цілі числа переводяться в систему числення із заданою основою шляхом послідовного ділення на цю основу. В попередньому прикладі (1.2.1.3) ціла частина числа, яка дорівнює 11_{10}, перетвориться в двійкову форму наступним чином:

11 : 2 = 5 Залишок 1 (МЗР)
5 : 2 = 2 1 ↑
2 : 2 = 1 0 |
1 : 2 = 0 1 (СЗР).

Процес продовжується до тих пір, доки не вийде результат, що дорівнює 0. Залишки від ділення потім виписуються, починаючи з

останнього, або *старшого значущого розряду* (СЗР), і закінчуючи першим, або *молодшим значущим розрядом* (МЗР). У розглянутому випадку

$$N_I=11_{10}=1011_2=1011B.$$

Перетворення дробової частини числа в систему числення з заданою основою здійснюється шляхом виконання ряду послідовних операцій множення на цю основу. В розглянутому прикладі (1.2.1.3) дробова частина числа N_F , яка дорівнює 0.8125_{10}, переводиться в двійкову систему наступним чином:

0.8125 * 2 = 1.6250 = 0.6250	Перенесення 1	(СЗР)
0.6250 * 2 = 1.2500 = 0.2500	1	↓
0.2500 * 2 = 0.5000 = 0.5000	0	
0.5000 * 2 = 1.0000 = 0.0000	1	(МЗР).

Процес послідовного множення продовжується до одержання нульового результату (що не завжди можливо) або до заповнення двійкових розрядів, виділених для представлення числа. Цифри перенесення потім виписують, починаючи з (СЗР). В розглянутому прикладі $N_F = 0.8125_{10} = 0.1101_2$.

Тому маємо $N = 11.8125_{10} = 1011.1101_2$.

1.2.1.3.2 Двійкове лічення

Лічба в двійковій системі числення менш складна, ніж в інших системах, оскільки тут використовується усього дві цифри – 0 і 1. Якщо має місце деяка послідовність подій, то першій події відповідає цифра 1. Однак вже для другої події цифри два в двійковій системі не існує. В цьому випадку відбувається перенесення в наступний розряд, що дає дворозрядне двійкове число. Якщо перевести це число в десяткову систему числення, то отримаємо бажаний результат:

$$10_2 = (1\times 2^1) + (0\times 2^0) = 2_{10}.$$

В таблиці 1.1 представлено послідовність двійкових чисел, яка відповідає десятковим числам від 0 до 9. Відмітимо, що перенесення в старший розряд відбувається при відображенні усіх парних десяткових чисел. При відображенні десяткових чисел 2, 4, 8, 16 і т. ін. відбуваються відповідно перенесення із розряду одиниць (2^0), двійок (2^1), четвірок (2^2), вісімок (2^3) і т. ін.

Таблиця 1.1– Представлення чисел у двійковій та десятковій системах числення

Двійкове число	Десяткове число
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9

1.2.1.4 Вісімкова система числення

Одна з основних систем числення – вісімкова система, яка також інколи використовується при роботі з МП–м (МК–ром). Це система з основою 8 і набором цифр від 0 до 7. Щоб проілюструвати позиційну структуру даної системи, розглянемо вісімкове число 673.12_8. Еквівалентне йому десяткове число виходить наступним чином:

$$(6\times 8^2)+(7\times 8^1)+(3\times 8^0)+(1\times 8^{-1})+(2\times 8^{-2})=443.15625_{10}.$$

Як і в випадку двійкової і десяткової системи числення, крапка при позиційному вісімковому представленні числа відокремлює його цілу частину від дробової.

1.2.1.4.1 Переведення чисел із десяткової системи числення у вісімкову

Переведення чисел із десяткової системи у вісімкову можна виконати за допомогою методу ділення–множення, описаного вище. Над цілою частиною числа виконується ряд послідовних операцій ділення на основу 8. В наведеному вище прикладі цілу частину числа N_I, яка дорівнює 443_{10}, перетворимо у вісімкове число за допомогою наступної процедури:

443 : 8 = 55 Залишок 3 (МЗР)
55 : 8 = 6 7 ↑
6 : 8 = 0 6 (СЗР).

Записавши залишки від найбільшої значущої цифри (СЗР) до найменшої (МЗР), одержимо

$$N_I = 443_{10} = 673_8.$$

Для переведення дробової частини числа необхідно виконати ряд послідовних операцій множення. В нашому прикладі дробова частина числа N_F, що дорівнює 0.15625_{10}, перетвориться наступним чином:

0.15625 x 8 = 1.25000 = 0.25000 Перенесення 1 (СЗР)
↓
0.25000 x 8 = 2.00000 = 0.00000. 2 (МЗР)

Виписавши цифри перенесення від СЗР до МЗР, отримаємо

$$N_F = 0.15625_{10} = 0.12_8.$$

Таким чином, отримаємо $N = 443.15625_{10} = 673.12_8.$

1.2.1.4.2 Переведення із двійкової системи числення у вісімкову

Мікропроцесор оперує двійковими числами. Однак при двійковому записі великих чисел доводиться заповнювати багато розрядів, що вкрай стомлює і вимагає багато часу. Тому для більш короткого запису

двійкових чисел застосовують інші системи числення, наприклад, вісімкова система. В зв'язку з цим часто виникає необхідність переведення чисел із двійкової системи числення у вісімкову.

Три розряди двійкових чисел дозволяють представити вісім комбінацій (від 000 до 111). Тому 3–розрядні двійкові числа можна безпосередньо заміняти вісімковими. Для перетворення двійкового числа у вісімкове, цілу частину двійкового числа розбивають на групи по три розряди, рахуючи вліво від двійкової крапки. Далі кожна 3–х розрядна група замінюється вісімковим еквівалентом. Описаний процес наведений у прикладі перетворення двійкового числа 11101110100.00111101:

	11	101	110	100	.	001	111	01,	
(011)									(010)
	3	5	6	4	.	1	7	2 .	
(СЗР)									(МЗР)

Таким чином, $11101110100.00111101_2 = 3564.172_8$.

При формуванні 3–х розрядних груп може виникнути необхідність доповнити нулями першу (СЗР) і останню (МЗР) групи, як показано вище. В останній групі, наприклад, 01 не є вісімковою одиницею. В даному випадку 01 відповідає 010, або вісімковій цифрі 2.

Процедура переведення із вісімкової системи числення у двійкову є зворотною і легко реалізується шляхом заміни кожної вісімкової цифри її 3–х розрядним двійковим еквівалентом.

1.2.1.5 Представлення чисел в шістнадцятковій системі числення

Двійковими числами великої розрядності важко оперувати, тому використовують їх більш компактне представлення, тісно пов'язане з двійковим. В шістнадцятковій системі числення числа представляються за степенями основи 16 ($16=2^4$), що еквівалентно кількості комбінацій

чотирьохрозрядного двійкового числа. В шістнадцятковій системі числення використовуються наступні символи: 0,1,2,3,4,5,6,7,8,9,A, B,C,D,E,F (таблиця 1.2).

Таблиця 1.2 – Представлення чисел у десятковій, двійковій та шістнадцятковій системах числення

Десяткове число	Двійкове число	Шістнадцяткове число
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Переведення із шістнадцяткової системи в інші системи числення здійснюється аналогічно двійковій і десятковій. Також аналогічно виконуються арифметичні дії над шістнадцятковими числами.

1.2.1.6 Двійково–десяткові числа (коди)

Існує протиріччя між машинним представленням чисел (двійкова система числення) і представленням чисел у нашому повсякденному житті (десяткові числа). Перетворення між ними, у випадку великого об'єму вхідних даних і вихідних результатів, веде до помітних втрат процесорного часу. Разом з тим, є велике коло задач, що характеризуються значними об'ємами числових даних і порівняно простою їх обробкою (економічні задачі, статистичні та бухгалтерські розрахунки і т.ін.). Тому потрібно було розробити такі форми представлення чисел, в яких якимсь чином поєднувалися б двійкова і десяткова системи числення. Такі форми отримали загальну назву *двійково–десяткового кодування* (Binary – Coded Decimal) або BCD – кодування. Загальна властивість цих форм виявляється в тому, що за основу береться десяткове число і кожна його цифра зображується тим чи іншим двійковим еквівалентом. До теперішнього часу з багаточисельних систем двійково–десяткового кодування практичне застосування знаходять тільки дві: двійково–десяткові числа в упакованому і не упакованому форматі.

1.2.1.6.1 Двійково–десяткові числа в упакованому форматі

В упакованому форматі, байт містить дві десяткові цифри. Менша цифра займає праву тетраду (біти 3…0), старша – ліву тетраду (біти 7…4). Обидві цифри представлені своїми двійковими еквівалентами та називаються також кодом 8421 (за двійковими вагами). Розміщення десяткових цифр у байті показано на рисунку 1.1, а.

Багаторозрядні упаковані десяткові числа займають декілька суміжних байтів. При необхідності роботи зі знаковими числами старша

тетрада (іноді – менша) старшого байта призначається для знака числа (рисунок 1.1, б).

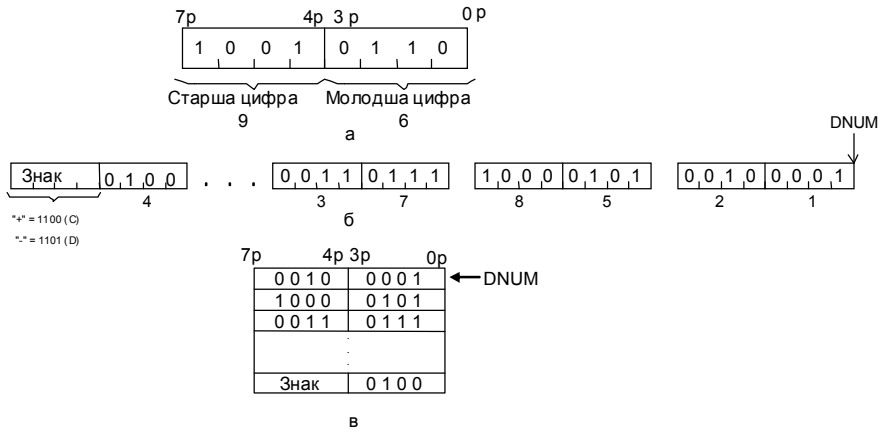


Рисунок 1.1 – Формат упакованих десяткових чисел: а – кодування байта, б– багаторозрядне число, в – розміщення багаторозрядного числа в пам'яті

Для кодування знака можна використовувати шість заборонених тетрад 1010…1111 (або у шістнадцятковій системі від A до F), які не являють собою десяткові цифри. Зазвичай, для зображення знака “плюс” застосовується код 1100 (C), а знака “мінус” – код 1101 (D). Через необхідність зображення знака багатобайтові упаковані десяткові числа мають непарне число розрядів (байт).

Під час програмування упаковані десяткові числа зазвичай визначаються початковою адресою – DNUM (це адреса молодшого байта) і числом байтів – N. Розміщення упакованого десяткового числа в пам'яті показано на рисунку 1.1, в.

1.2.1.6.2 Двійково–десяткові числа в не упакованому форматі

Не упакований двійково–десятковий формат наведений на рисунку 1.2. В цьому коді десятковим числам відповідають двійкові набори від 0000 0000 (цифра 0) до 0000 1001 (цифра 9), або в шістнадцятковій системі

від 00H до 09H. Отже, можна вважати, що власне значення десяткової цифри займає молодшу тетраду і представлено її двійковим еквівалентом, а в старшій тетраді знаходиться комбінація 0000.

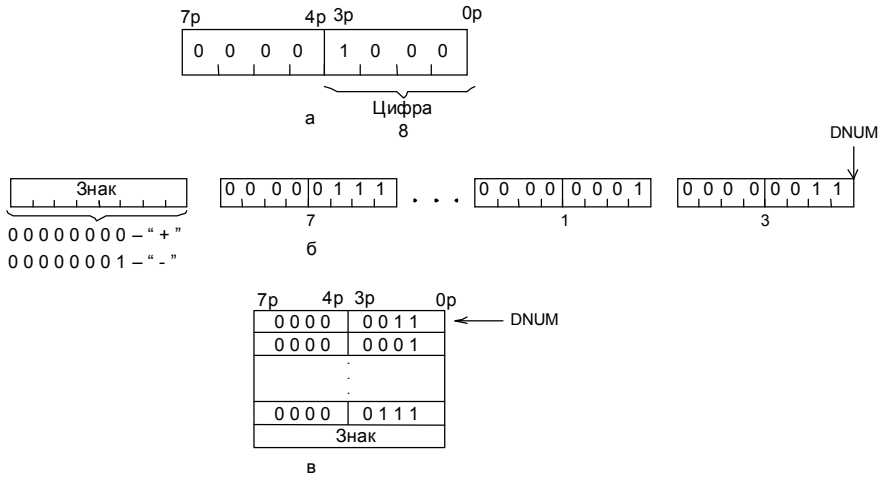


Рисунок 1.2 – Формат не упакованих двійково–десяткових чисел: а – кодування байта, б – багаторозрядне число, в – розміщення багаторозрядного числа в пам'яті

Багаторозрядні не упаковані двійково–десяткові числа займають суміжні байти (рисунок 1.2, б). Для знака числа призначається старший байт, в якому комбінація 0000 0000 (00H) означає знак плюс, а комбінація 0000 0001 (01H) – знак мінус. Такі числа визначаються в програмах їх початковою адресою – DNUM і числом розрядів (байт) або довжиною – N. Розміщення не упакованого двійково–десяткового числа в пам'яті показано на рисунку 1.2, в.

1.2.1.6.3 Представлення десяткових чисел в коді ASCII

Для кодування алфавітно–цифрової інформації або символів використовується 7–розрядний код ASCII (американський стандартний код для обміну інформацією), який в російськомовній літературі називають кодом КОІ–7 (таблиця 1.3)

В цьому коді десятковим цифрам відповідають двійкові набори від 00110000 (цифра 0) до 00111001 (цифра 9), або в шістнадцятковій системі від 30h до 39h.

Дана форма відрізняється від не упакованого формату тим, що старша тетрада байта містить 0011B=3D замість 0000B=0D.

Більш того, на території однієї держави може бути декілька кодувань. Наприклад, на території СНД найбільше поширення одержали три види кодувань російських букв (таблиці 1.4…1.6):

– CP–866 (при роботі в DOS);
– CP–1251 (при роботі в Windows);
– KOI–8 (при роботі в UNIX).

Таблиця 1.3 – Код ASCII

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
0	0		53	35	5	106	6A	j	159	9F	Я
1	1		54	36	6	107	6B	k	160	A0	а
2	2		55	37	7	108	6C	l	161	A1	б
3	3		56	38	8	109	6D	m	162	A2	в
4	4		57	39	9	110	6E	n	163	A3	г
5	5		58	3A	:	111	6F	o	164	A4	д
6	6		59	3B	;	112	70	p	165	A5	е
7	7		60	3C	<	113	71	q	166	A6	ж
8	8		61	3D	=	114	72	r	167	A7	з
9	9		62	3E	>	115	73	s	168	A8	и
10	A		63	3F	?	116	74	t	169	A9	й
11	B		64	40	@	117	75	u	170	AA	к
12	C		65	41	A	118	76	v	171	AB	л
13	D		66	42	B	119	77	w	172	AC	м
14	E		67	43	C	120	78	x	173	AD	н

Продовження таблиці 1.3

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
15	F		68	44	D	121	79	y	174	AE	о
16	10		69	45	E	122	7A	z	175	AF	п
17	11		70	46	F	123	7B	{	176	B0	░
18	12		71	47	G	124	7C	|	177	B1	▒
19	13		72	48	H	125	7D	}	178	B2	▓
20	14		73	49	I	126	7E	~	179	B3	│
21	15		74	4A	J	127	7F		180	B4	┤
22	16		75	4B	K	128	80	А	181	B5	╡
23	17		76	4C	L	129	81	Б	182	B6	╢
24	18		77	4D	M	130	82	В	183	B7	╖
25	19		78	4E	N	131	83	Г	184	B8	╕
26	1A		79	4F	O	132	84	Д	185	B9	╣
27	1B		80	50	P	133	85	Е	186	BA	║
28	1C		81	51	Q	134	86	Ж	187	BB	╗
29	1D		82	52	R	135	87	З	188	BC	╝
30	1E		83	53	S	136	88	И	189	BD	╜
31	1F		84	54	T	137	89	Й	190	BE	╛
32	20		85	55	U	138	8A	К	191	BF	┐
33	21	!	86	56	V	139	8B	Л	192	C0	└
34	22	“	87	57	W	140	8C	М	193	C1	┴
35	23	#	88	58	X	141	8D	Н	194	C2	┬
36	24	$	89	59	Y	142	8E	О	195	C3	├
37	25	%	90	5A	Z	143	8F	П	196	C4	─
38	26	&	91	5B	[	144	90	Р	197	C5	┼
39	27	′	92	5C	\	145	91	С	198	C6	╞
40	28	(	93	5D	]	146	92	Т	199	C7	╟
41	29	)	94	5E	^	147	93	У	200	C8	╚
42	2A	*	95	5F	–	148	94	Ф	201	C9	╔

Продовження таблиці 1.3

DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR	DEC	HEX	CHAR
43	2B	+	96	60	'	149	95	Х	202	CA	╩
44	2C	′	97	61	a	150	96	Ц	203	CB	╦
45	2D	D	98	62	b	151	97	Ч	204	CC	╠
46	2E	.	99	63	c	152	98	Ш	205	CD	═
47	2F	/	100	64	d	153	99	Щ	206	CE	╬
48	30	0	101	65	e	154	9A	Ъ	207	CF	╧
49	31	1	102	66	f	155	9B	Ы	208	D0	╨
50	32	2	103	67	g	156	9C	Ь	209	D1	╤
51	33	3	104	68	h	157	9D	Э	210	D2	╥
52	34	4	105	69	i	158	9E	Ю	211	D3	╙
212	D4	╙	224	E0	p	236	EC	ь	248	F8	Ї
213	D5	╒	225	E1	c	237	ED	э	249	F9	ї
214	D6	╓	226	E2	т	238	EE	ю	250	FA	Ў
215	D7	╫	227	E3	у	239	EF	я	251	FB	ў
216	D8	╪	228	E4	ф	240	F0	Ё	252	FC	ⁿ
217	D9	┘	229	E5	х	241	F1	ё	253	FD	²
218	DA	┌	230	E6	ц	242	F2	Ґ	254	FE	■
219	DB	▉	231	E7	ч	243	F3	ґ	255	FF	
220	DC	▄	232	E8	Ш	244	F4	Є			
221	DD	▌	233	E9	Щ	245	F5	є			
222	DE	▐	234	EA	Ъ	246	F6	І			
223	DF	▀	235	EB	Ы	247	F7	і			

В наведених вище та нижче таблицях (таблиці 1.3…1.6) починаючи з №128 по 255 коди символів залежать від національного кодування (власна для кожної країни).

Таблиця 1.4 – Код CP–866 (Dos)

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
10	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	⌂
80	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
90	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
a0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
b0	░	▒	▓	│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
c0	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
d0	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	█	▄	▌	▐	▀
e0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
f0	Ё	ё	Є	є	Ї	ї	Ў	ў	°	∙	·	√	№	¤	▪	

Таблиця 1.5 – Код CP–1251 (Windows)

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
10	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	⌂
80	?	?	'	?	"	:	†	‡	?	‰	?	<	?	?	?	?
90	?	'	'	"	"	•	-	-	?	T	?	>	?	?	?	?
a0		ў	ў	?	¤	?	¦	§	Ё	c	Є	«	¬	-	R	Ї
b0	°	±	I	i	?	ч	¶	·	ё	№	є	»	?	?	?	ї
c0	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
d0	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
e0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
f0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	

Таблиця 1.6 – KOI–8 (UNIX)

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
10	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	⌂
80	─	│	┌	┐	└	┘	├	┤	┬	┴	┼	▀	▄	█	▌	▐
90	░	▒	▓	⌠	▪	•	√	⌡	≈	≤		≥	⁰	∙	·	÷
a0	═	║	╒	ё	╓	╔	╕	╖	╗	╘	╙	╚	╛	╜	╝	╞
b0	╟	╠	╡	Ё	╢	╣	╤	╥	╦	╧	╨	╩	╪	╫	╬	¤
c0	ю	а	б	ц	д	е	ф	г	х	и	й	к	л	м	н	о
d0	п	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ъ
e0	Ю	А	Б	Ц	Д	Е	Ф	Г	Х	И	Й	К	Л	М	Н	О
f0	П	Я	Р	С	Т	У	Ж	В	Ь	Ы	З	Ш	Э	Щ	Ч	

1.2.2 Формати подання чисел: цілі числа зі знаком і без знака

1.2.2.1 Подання чисел зі знаком

Всі системи числення допускають використання як додатних, так і від’ємних чисел, для позначення яких, як відомо, застосовують знаки плюс (+) або мінус (–). Число в такому поданні називається числом зі знаком. Оскільки ЕОМ побудовані на схемах двійкової логіки і для подання двійкових станів застосовуються символи 0 і 1, тому ці ж символи необхідно використовувати і для позначень знака числа. Як правило, додатковий розряд для подання знака числа, називається знаковим розрядом і розміщується з лівого боку найбільшого значущого розряду числа. Для позначення додатних і від’ємних значень використовують відповідно 0 і 1.

Опишемо три важливих засоби подання двійкових чисел зі знаком, відомих як прямий, зворотний і додатковий код числа.

1.2.2.1.1 Прямий код числа

Найбільш простим способом подання числа зі знаком є прямий двійковий код числа. Число у прямому двійковому коді називають також двійковим числом зі знаком. Для числа, що займає n двійкових розрядів, розряд СЗР використовується для подання знака, а в n–1 розрядах, що залишилися, записується абсолютне значення числа в двійковій системі. Розглянемо, наприклад, запис числа -13_{10} за допомогою п'яти двійкових розрядів. Маємо

$$-13_{10} = 1,1101,$$

знаковий розряд (–).

В даному записі для відділення знакового розряду від цифрових розрядів використовувалась кома, яка, зазвичай, опускається. Приведемо ще один приклад:

$$+27_{10} = 0,11011,$$

знаковий розряд (+).

Недолік цього способу полягає в тому, що процедури додавання чисел з різними знаками ускладнені.

1.2.2.1.2 Зворотний код числа

Іншим способом подання від'ємних чисел є подання числа за допомогою зворотного коду. Для одержання зворотного коду числа береться його додатне двійкове подання і зліва доповнюється нулями до

числа розрядів, що відповідають одному машинному слову. Після цього кожний нуль замінюється на одиницю, а кожна одиниця – на нуль:

$$(0\ 0\ 0\ 0\ 0\ 0\ 1\ 1)_2 \Rightarrow (1\ 1\ 1\ 1\ 1\ 1\ 0\ 0)_2.$$

прямий код зворотний код

Зворотний код від'ємного числа називають також доповненням до одиниці. Перевагами такого кодування є легкість одержання зворотного коду і можливість не враховувати знаки при додаванні та відніманні.

Недоліки кодування полягають в наступному: утворюються два різних подання нуля, не еквівалентних за записом; необхідне циклічне перенесення при додаванні для одержання правильного результату.

Перетворення від'ємного числа в зворотному коді в десяткову систему числення можна здійснити двома способами. Один з них є зворотним процедурі порозрядного доповнення числа до одиниці. Розглянемо, наприклад, 4–розрядне число 1,011. Утворюючи порозрядне доповнення до одиниці, одержуємо число 0,100, або +4. Таким чином, 1,011 = – 4.

В іншому способі використовується той факт, що знаковий розряд n–розрядного двійкового числа має від'ємну вагу, яка дорівнює ($2^{n-1}-1$). В нашому випадку n = 4, 2^{4-1} –1= 7 і вага знакового розряду є (–7). Таким чином, маємо

$$1,011 = -7 + 3 = -4.$$

Даний спосіб дуже зручний, оскільки тут ми просто переводимо в десяткову систему вміст цифрових розрядів і до отриманого результату додаємо від'ємну вагу знакового розряду.

1.2.2.1.3 Додатковий код числа

Ще одним способом подання від'ємного числа є додатковий код, або доповнення до двох. Доповнення до двох утворюється відніманням числа

фіксованої довжини (тобто доповненого нулями зліва) з цілого числа, на одиницю більшого, ніж максимальне число, яке може оброблятися в даному процесорі. Наприклад, число –3 можна подати в додатковому коді наступним чином:

$$\underbrace{(1\,0\,0\,0\,0\,0\,0\,0\,0)_2}_{(+256D)} - \underbrace{(0\,0\,0\,0\,0\,0\,1\,1)_2}_{(+3D)} = \underbrace{(1\,1\,1\,1\,1\,1\,0\,1)_2}_{(+253D \text{ або } -3D)}.$$

Існує, однак, і більш компактний метод одержання додаткового коду. Для цього необхідно:

- замінити всі нулі на одиниці, а одиниці на нулі в двійковому поданні модуля числа;
- додати одиницю до результату.

Подання чисел у вигляді додаткового коду має наступні переваги порівняно з іншими формами подання:

- додавання і віднімання на практиці виконуються просто;
- існує єдине подання нуля.

Як видно з таблиці 1.7 за допомогою 8 біт можна подати в двійковій формі десяткові числа зі знаком від – 128 до + 127, включаючи 0. В цій таблиці показано два типові для мікропроцесорів способи використання двійкових кодів: як двійкових чисел зі знаком, так і без знака. Лівий стовпчик містить двійкові числа (двійкові коди) від 0000 0000 до 1111 1111, правий стовпчик – їх десяткові еквіваленти від 0 до 255, отримані з припущення, що розглядаються числа без знаку. У центральному стовпчику знаходяться десяткові еквіваленти двійкових чисел лівого стовпчика, отримані з припущення, що від’ємні числа записувалися в додатковому коді.

Таблиця 1.7 – Десяткові еквіваленти двійкових чисел

	Десятковий еквівалент	
8–розрядне двійкове число 7р 0р	Двійкового числа зі знаком (від'ємне число в додатковому коді)	Двійкового числа без знака
0000 0000	+0	0
0000 0001	+1	1
0000 0010	+2	2
0000 0011	+3	3
.	.	.
.	.	.
.	.	.
0111 1100	+124	124
0111 1101	+125	125
0111 1110	+126	126
0111 1111	+127	127
1000 0000	–128	128
1000 0001	–127	129
1000 0010	–126	130
1000 0011	–125	131
.	.	.
.	.	.
.	.	.
1111 1100	–4	252
1111 1101	–3	253
1111 1110	–2	254
1111 1111	–1	255

Тут додатним двійковим числам (від 0000 0000 до 0111 1111) відповідають десяткові числа від 0 до + 127, а тим, що вміщують 1 в

восьмому розряді (сьомому, якщо рахувати від нуля) від'ємним двійковим числам (від 1000 0000 до 1111 1111) – десяткові від: – 128 до – 1.

Використовуючи вісім двійкових розрядів і подаючи від'ємні числа в додатковому коді, можна записати 256 різних чисел: 127 додатних, нуль і 128 від'ємних. Згадану вище процедуру формування доповнення – подання від'ємного числа в зворотному коді і додавання одиниці продемонструємо на наступному прикладі:

Число 4_{10} в двійковій формі 0000 100;

Зворотний код числа 4_{10} 1111 011;

Додавання до зворотного коду 1 1 ;

Число -4_{10} в додатковому коді 111 1100.

Порівняйте отриманий результат з відповідним кодом в таблиці 1.7.

Для подання двійкового числа в додатковому коді можна користуватися іншим способом, відмінним від описаного вище і більш коротким за числом операцій. В пошуках першого біта, що дорівнює одиниці, переглядають справа наліво розряди двійкового подання модуля числа, починаючи з найменшого за значенням. До тих пір, доки зустрічаються нулі, їх копіюють в однойменні розряди результату. Перша одиниця, що зустрілася, також копіюється у відповідний розряд результату, але кожний наступний біт модуля вихідного числа замінюється на зворотний.

Відповідно до таблиці 1.7 арифметичні операції над двійковими числами без знака нічим не відрізняються від подібних операцій над двійковими числами зі знаком, від'ємні з яких подані своїми доповненнями. Це істотно спрощує апаратну реалізацію подібних операцій в мікропроцесорі. Однак слід не втрачати з виду, з якими числами ви маєте справу в даний момент: без знака чи зі знаком. Наприклад, при додаванні двох чисел без знака результат – число без знака подається у вигляді деякої послідовності бітів, котру можна інтерпретувати і як від'ємне число в

додатковому коді. В загальному випадку при додаванні або відніманні чисел зі знаком результат є число зі знаком. Якщо при цьому біт старшого розряду дорівнює одиниці, то результат – від'ємне число в додатковому коді. Якщо потрібно визначити абсолютне значення (величину) від'ємного результату, останній необхідно представити в зворотному коді, а потім додати одиницю. Існує більш простий спосіб визначення абсолютного значення від'ємного двійкового числа: необхідно додати ваги двійкових розрядів, які вміщують нулі, і до суми додати одиницю.

В простоті операцій над від′ємними числами у вигляді доповнень до двох можна переконатися на прикладі операції віднімання, яка виконується наступним чином: визначається додатковий код від'ємника і виконується додавання цього коду зі зменшуваним. Якщо різниця – число додатне (біт старшого розряду дорівнює 0), то біт перенесення необхідно відкинути: отримана послідовність бітів і є двійковий код. Якщо різниця – число від'ємне (біт старшого розряду дорівнює 1), то вона представлена в додатковому коді. Вище вказувалося, що необхідно зробити для визначення абсолютної величини від'ємного числа, поданого в такому вигляді.

Приклад 1. Обчислити різницю чисел 58 – 23:

а) Визначення додаткового коду числа 23

0 0 0 1 0 1 1 1	Число 23_{10}
1 1 1 0 1 0 0 0	Зворотний код числа 23_{10}
0 0 0 0 0 0 0 1	Одиниця, що додається до зворотного коду
_____	
1 1 1 0 1 0 0 1	Додатковий код числа 23_{10} (-23_{10})

б) Обчислення різниці

Десяткова арифметика		Двійкова арифметика	
58		0 0 1 1 1 0 1 0	Число 58_{10}
–		+	
23		1 1 1 0 1 0 0 1	Додатковий код числа 23_{10}
35		1 0 0 1 0 0 0 1 1	Різниця 35_{10}.

Одиниця перенесення, що відкидається у випадку додатного результату.

Приклад 2. Обчислити різницю чисел 26 – 34:

а) Визначення додаткового коду числа 34

0 0 1 0 0 0 1 0	Число 34_{10}
1 1 0 1 1 1 0 1	Зворотний код числа 34_{10}
0 0 0 0 0 0 0 1	Одиниця, що додається до зворотного коду
1 1 0 1 1 1 1 0	Додатковий код числа 34_{10}

б) Обчислення різниці

Десяткова арифметика		Двійкова арифметика	
26		0 0 0 1 1 0 1 0	Число 26_{10}
–		+	
34		1 1 0 1 1 1 1 0	Додатковий код числа 34_{10} (-34_{10})
– 08		1 1 1 1 1 0 0 0	Різниця в формі доповнення до двох (оскільки в старшому розряді 1)

в) Визначення абсолютного значення різниці

1 1 1 1 1 0 0 0	Різниця у додатковому коді
0 0 0 0 0 1 1 1	Зворотний код різниці
0 0 0 0 0 0 0 1	Одиниця, що додається до зворотного коду
0 0 0 0 1 0 0 0	Абсолютне значення різниці (8_{10}).

Другий спосіб визначення абсолютного значення від’ємного числа 11111000:

$$2^0+2^1+2^2+1=1+2+4+1=8.$$

1.2.3 Формати чисел з нефіксованою крапкою

Крім розглянутих вище форматів чисел з фіксованою крапкою, в МП–й техніці застосовуються також формати з нефіксованою крапкою, розглянуті в [9, 10].

1.2.4 Угруповання біт

Окремі двійкові біти (розряди) можуть об'єднуватися у групи, що мають назви:

- тетрада (група з 4–х біт);
- байт (група з 8–ми біт);
- слово (група з 16–ти біт);
- подвійне слово (група з 32–х біт).

1.2.5 Двійкова арифметика: додавання, віднімання, множення, ділення

1.2.5.1 Двійкове додавання

Двійкове додавання виконується за наступними правилами:

а) 0 + 0 = 0,

б) 0 + 1 = 1,

в) 1 + 1 = 0 + перенесення у наступний розряд,

г) 1 + 1 + 1 = 1 + перенесення у наступний розряд.

Для ілюстрації цих правил розглянемо додавання чисел 1110 і 1100:

1 1 розряди перенесення,

1 1 1 0 перший доданок,

+ 1 1 0 0 другий доданок,

1 1 0 1 0 сума.

1.2.5.2 Двійкове віднімання

Двійкове віднімання виконується за наступними правилами:

а) 0 – 0 = 0,

б) 1 – 0 = 1,

в) 1 – 1 = 0,

г) 0 – 1 = 1 – позика одиниці зі старшого розряду.

Використання цих правил ілюструється на прикладі віднімання числа 1100 із 10110.

(0) ⟵ Позика одиниці з старшого розряду,

1 0 1 1 0 зменшуване,

– 1 1 0 0 від'ємник,

0 1 0 1 0 різниця.

В МП–рі операція віднімання замінюється додаванням зменшуваного з додатковим кодом від'ємника. Тому операція 10110В – 1100В виконується наступним чином:

00010110 зменшуване

+

11110100 додатковий код від'ємника

00001010 різниця.

1.2.5.3 Двійкове множення

При двійковому множенні частковий добуток зсувається на один розряд вліво для обробки кожного наступного розряду множника. Множення чисел здійснюється за наступними правилами [10]:

а) 0 х 0 = 0,

б) 0 х 1 = 0,

в) 1 х 0 = 0,

г) 1 х 1 = 1.

Правила множення ілюструються у наступному прикладі:

5 х 3 = 15 в десятковому представленні чисел

$0\ 1\ 0\ 1_2 - 5_{10}$

$\underline{x\ 0\ 0\ 1\ 1_2} - 3_{10}$

$0\ 1\ 0\ 1$

$\underline{+0\ 1\ 0\ 1}$

$0\ 1\ 1\ 1\ 1_2 - 15_{10}.$

1.2.5.4 Двійкове ділення

Операція ділення є зворотною по відношенню до множення і реалізується схожими циклічними діями. Алгоритм ділення приведений у [10].

1.2.6 Двійково–десяткова арифметика

Вище були розглянуті два формати представлення двійково–десяткових чисел: упакований і неупакований. В сучасних МП–х немає команд, що дійсно оперують названими числами. Виконання операцій над двійково–десятковими числами здійснюється за два етапи:

– на першому етапі операнди обробляються як цілі двійкові числа командами двійкової арифметики;
– на другому – спеціальні команди корекції перетворюють проміжний результат в двійково–десятковий формат.

Найбільш повно команди корекції для обох форматів двійково–десяткових чисел представлені в процесорі i8086 [4, 9, 11, 28].

1.3 Приклад гіпотетичної МПС керування, та проблеми, що пов‘язані з її проектуванням

Роботу і застосування будь–якої мікропроцесорної системи (МПС) необхідно розглядати як на апаратному, так і на програмному рівні, оскільки ці дві властивості МПС невід′ємні одна від одної.

Аналіз МПС на апаратному рівні починають з вивчення структурної або функціональної схеми системи.

Розглянемо функціональну схему гіпотетичної мікропроцесорної системи керування (МПСК), яку наведено на рисунку 1.3.

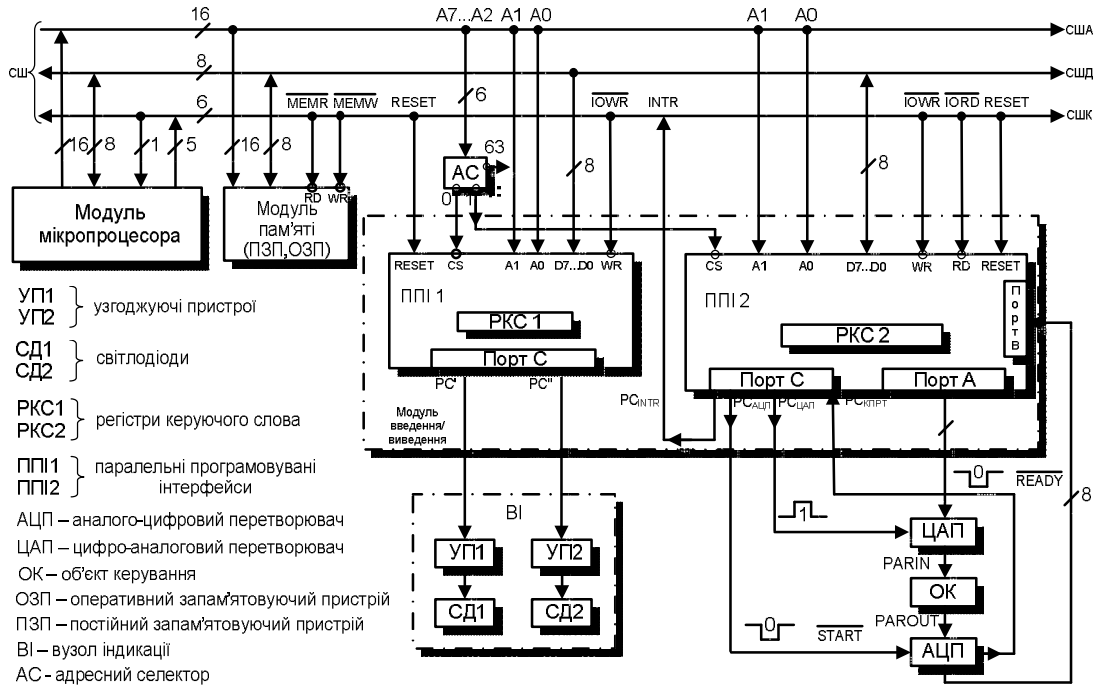


Рисунок 1.3 – Функціональна схема гіпотетичної МПСК

Дана мікропроцесорна система керування орієнтована на 8–ми розрядний МП–р, наприклад і8080, і містить модулі:

– мікропроцесора (МП);
– пам'яті: ПЗП, ОЗП ;
– введення/виведення;
– ЦАП;
– АЦП;
– об'єкта керування (ОК);
– вузла індикації (ВІ).

В постійному запам'ятовуючому пристрої (ПЗП) зберігається робоча програма, під керуванням якої буде працювати проектована МПСК, а у оперативному запам’ятовуючому пристрої зберігаються дані, а також знаходиться стек, куди записується адреса повернення з підпрограми. Модуль введення/виведення, що складається з двох паралельних програмованих інтерфейсів (ППІ), забезпечує обмін інформацією між мікропроцесором, об'єктом керування і індикацією. Аналого–цифровий перетворювач (АЦП) і цифро–аналоговий перетворювач (ЦАП) призначені для поєднання аналогового об'єкта керування з цифровою МПСК.

Названі вузли зв'язані між собою системною шиною (СШ), що складається із трьох окремих системних шин:

- адреси (США);
- даних (СШД);
- керування (СШК).

По США передається адреса від мікропроцесора до пристрою, з яким в даний момент часу буде здійснюватися обмін інформацією: пам'яттю або пристроєм введення/виведення (ПВВ).

По СШД мікропроцесор обмінюється інформацією (даними) між пам'яттю і ПВВ. В системі, що розглядається, із пам'яті в МП по СШД передаються команди робочої програми, яка керує роботою МПСК.

Через перший інтерфейс ППІ1 від мікропроцесора до вузла індикації (ВІ) по СШД передається інформація, що відображає стан об'єкта керування (ОК).

Від ППІ2 в МП по СШД передається інформація про поточне значення контрольованого параметра PAROUT.

Від МП до ППІ2 по СШД пересилається керуюча інформація, підтримуючи контрольований параметр ОК на заданому рівні.

По СШК передаються керуючі сигнали, які виробляються модулем мікропроцесора.

В розглянутому прикладі таких сигналів п’ять:

- RESET – системне скидання (здійснює початкове налагодження (ініціалізацію) ППІ1 і ППІ2);
- $\overline{\text{MEMR}}$ – читання пам'яті (керує читанням програми з ПЗП);
- $\overline{\text{MEMW}}$ – запис у пам'ять (керує записом до ОЗП);
- $\overline{\text{IORD}}$ – читання введення/виведення (керує введенням інформації від зовнішнього пристрою (ЗВПР) до МП);
- $\overline{\text{IOWR}}$ – запис в пристрій введення/виведення (керує виведенням інформації з МП в зовнішній пристрій).

Один керуючий сигнал (INTR) передається по СШК в МП–р і дозволяє організувати обмін даними через ППІ2 за перериванням.

В реальних системах число керуючих сигналів, що передаються по СШК, може бути більше названих п'яти.

В конкретний момент часу МП виконує передачу інформації в одному з двох напрямів:

- із МП в пам'ять (для МПСК, що мають ОЗП) або ПВВ;

– в МП із пам'яті або ПВВ.

В першому випадку говорять, що відбувається запис (виведення, передавання), а в другому – читання (введення, прийом). В обох випадках інформація передається по СШД.

Крім того, необхідно підкреслити, що МП не може одночасно працювати з пам'яттю і ПВВ, а взаємодіє тільки з одним із них, який обраний адресним селектором і керуючим сигналом $\overline{\text{MEMRD}}/\overline{\text{MEMW}}$ або $\overline{\text{IORD}}/\overline{\text{IOWR}}$.

Оскільки передбачається, що в прикладі (рисунок 1.3) об'єкт керування – аналоговий, він містить аналогові давач і виконавчий елемент. Тоді для зв'язку такого ОК з цифровою МПС використовуються: АЦП при введенні в МПСК значення контрольованого параметра від давача, і ЦАП при виведенні керуючого впливу на виконавчий елемент.

До складу ЦАП умовно включено:

– власне ЦАП, що містить резисторну матрицю R–2R та операційний підсилювач;
– буферний регістр, в який у двійковому коді переписується керуючий вплив із порту А ППІ2 в момент надходження синхроімпульсу, що програмно сформований на лінії порту С ППІ2 ($РС_{ЦАП}$).

В реальній системі буферний регістр може не використовуватися, а його функцію виконує буфер (регістр) порту А ППІ2.

Для запуску АЦП на лінії $РС_{АЦП}$ порту С ППІ2 програмно формується нульовий керуючий імпульс ($\overline{\text{START}}$). Після завершення аналого–цифрового перетворення АЦП формує сигнал “Готовність” ($\overline{\text{READY}}$), який має низький рівень та поступає в систему по лінії $РС_{КПРТ}$ порту С ППІ2.

Можливі два способи введення даних у МП через порт В ППІ2:

– програмно – керований;

– за перериванням.

У першому випадку, програма постійно перевіряє значення сигналу $РС_{КПРТ}$ і, при виявленні логічного 0, керує введенням даних у МП. Очевидно, що при цьому МП використовується не дуже ефективно, тому що витрачає свій машинний час на постійне опитування біта $РС_{КПРТ}$.

В другому випадку, МП може виконувати необхідну роботу по обробці даних до надходження сигналу “INTR – запит переривання”, що формується в момент закінчення перетворення АЦП і по СШК передається на однойменний вивід мікропроцесора.

При цьому відбувається переривання основної програми, і черговий байт вводиться в МП (у нашому прикладі через порт В ППІ2).

Після введення чергового байта формується сигнал “RESET” для АЦП, що передається по тій же лінії, що і сигнал “$\overline{START}$”, але має інверсне до нього (одиничне) значення.

Робота МПСК (рисунок 1.4) зводиться до підтримування аналогового вихідного сигналу об'єкта керування, позначеного PAROUT, у межах заданого допуску:

PARMIN $\leq$ PAROUT < PARMAX, де PARMIN і PARMAX – відповідно мінімальне і максимальне значення регульованого параметра.

На початку роботи для виходу ОК на робочий режим передається початкове значення регульованого параметра $PARIN_{поч}$.

Якщо в процесі роботи системи значення PAROUT знаходиться в межах зазначеного вище допуску, то значення $PARIN_{поч.}$ не змінюється. Якщо PAROUT вийшов за межі допуску з боку мінімуму чи максимуму, то поточне значення PARIN змінюється з кроком $\pm\Delta$, розмір якого може програмно змінюватися.

Ця зміна буде вироблятися доти, доки PAROUT знову не опиниться в межах допуску.

Порушення чи не порушення границь допуску в розглянутій системі відображаються простою індикацією, що складається з двох світлодіодів СД1 і СД2. При цьому відображаються три стани регульованого об'єкта:

- норма (PAROUT знаходиться в межах допуску) – обидва світлодіоди вимкнено;
- PAROUT ≥ PARMAX (вихід за межі допуску з боку максимуму) – вмикається СД1 і вимикається СД2;
- PAROUT < PARMIN (вихід за межі допуску з боку мінімуму) – вимикається СД1 і вмикається СД2.

Зображений на рисунку 1.3 адресний селектор може бути реалізовано за допомогою двійкового дешифратора, що перетворює вхідний паралельний 6–розрядний ДК (розряди А7…А2 США) у багатопозиційний унітарний код (число позицій дорівнює 2^6=64) з активними нульовими вихідними сигналами. Логічний нуль буде з'являтися на тому виході дешифратора, номер якого є десятковим еквівалентом вхідного ДК. У розглянутому прикладі використовуються тільки два виходи з 64–х (наприклад, 0 і 1), що у потрібний момент обирають один із двох ППІ. Інші виходи в реальній системі можуть застосовуватися для адресації інших зовнішніх пристроїв.

Варто відмітити, що в реальній системі розглянуті вище функції можуть бути виконані за допомогою тільки одного ППІ.

Розробку програмного забезпечення звичайно починають зі схеми алгоритму роботи системи, яку для нашого прикладу приведено на рисунку 1.4

Після цього розробляється і налагоджується керуюча програма мовою Асемблер або СІ обраного мікропроцесора або мікроконтролера.

Після налагодження ця програма компілюється – транслюється в машинні коди процесора і створюється об'єктний модуль, що на спеціальному пристрої – програматорі завантажується в ПЗП МПСК.

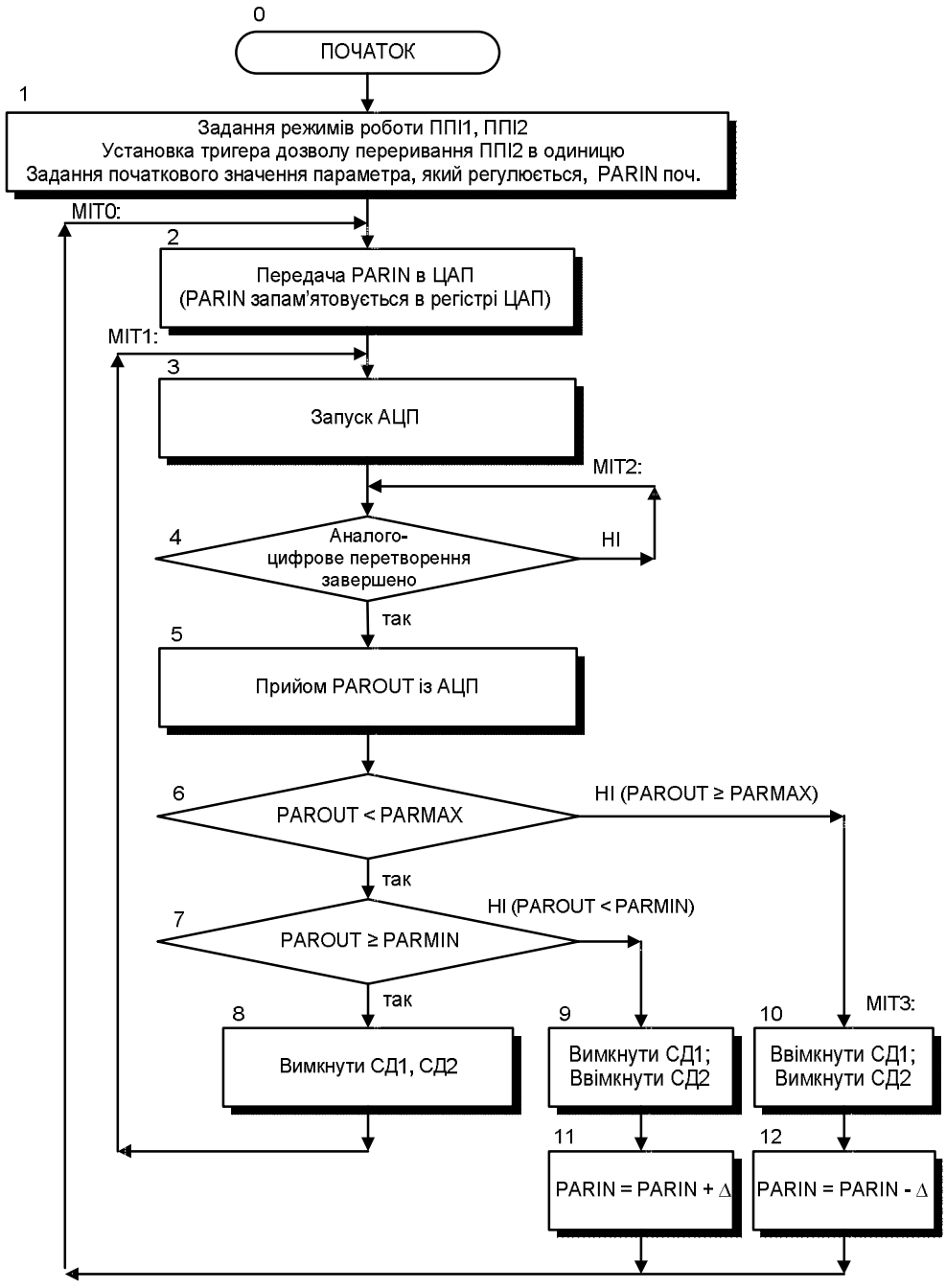


Рисунок 1.4 – Схема алгоритму роботи МПСК, яка проектується

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Що таке мікропроцесор? Скільки біт даних за раз обробляв перший мікропроцесор?
2. Що таке мікро–ЕОМ? Що називається мікропроцесорним комплектом?
3. Що таке системна шина керування? Наведіть приклад керуючих сигналів.
4. Опишіть обмін даними між МП та об’єктом керування через ППІ2.
5. Від чого залежить кількість адресованих комірок пам'яті МП–ра або МК–ра?
6. Скільки адресних виводів містить МК типу МК51? Який об’єм пам’яті вони дозволяють адресувати?
7. За значеннями яких параметрів можна оцінити швидкодію мікропроцесора?
8. Що таке система числення? Назвіть відомі вам системи числення.
9. Що таке позиційна система числення? Наведіть приклад позиційної системи числення.
10. Запишіть беззнакове число:
   - 01011101_2 у десятковій системі числення;
   - 00101011_2 у вісімковій системі числення;
   - 11001110_2 у шістнадцятковій системі числення;
   - 723_{10} у двійковій системі числення;
   - 241_{10} у вісімковій системі числення;
   - 596_{10} у шістнадцятковій системі числення;
   - $8C_{16}$ у двійковій системі числення;
   - $E5_{16}$ у вісімковій системі числення;
   - $F5A_{16}$ у десятковій системі числення;
   - 705_8 у двійковій системі числення;

– 132_8 у шістнадцятковій системі числення;

– 208_8 у десятковій системі числення;

– 735_{10} в упакованому BCD ;

– 571_{10} в не упакованому BCD.

11. Запишіть в прямому, зворотному та додатковому кодах знакове число -11_{10}.

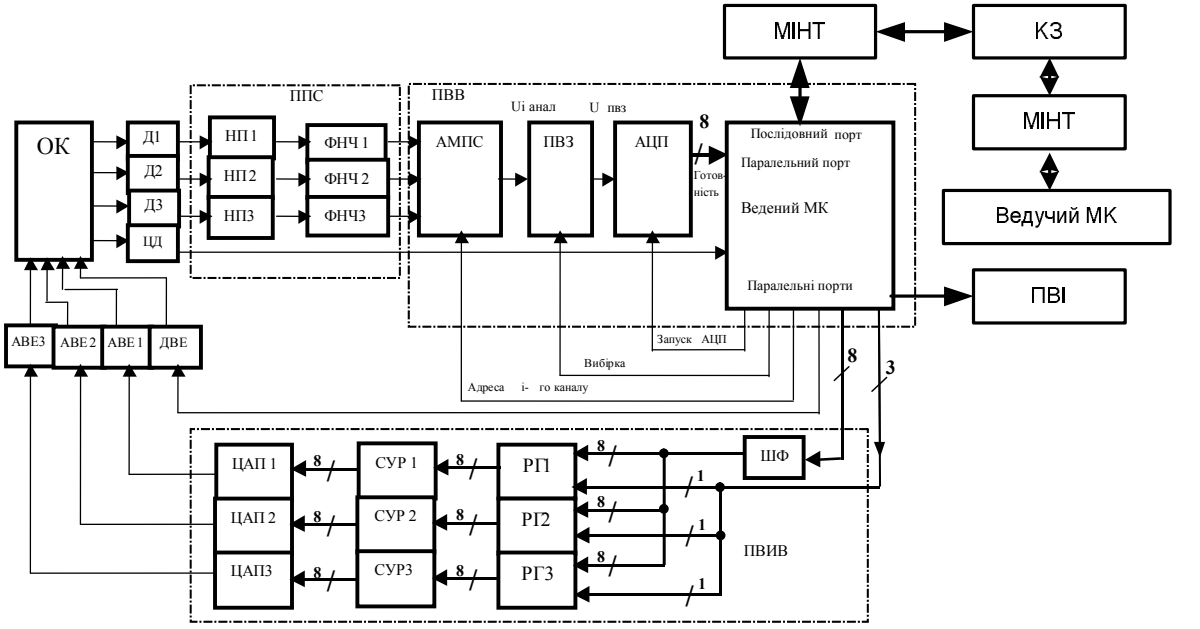
12. Знайдіть суму, різницю, добуток і частку беззнакових чисел 01001101_2 і 00000111_2.

ЛІТЕРАТУРА [1…5, 8…15, 18]

2 СТРУКТУРНА ТА ФУНКЦІОНАЛЬНА СХЕМИ ТИПОВОЇ МПС

2.1 Структура типової локальної мікропроцесорної системи керування (ЛМПСК)

Розглянемо приклад типової локальної мікропроцесорної системи керування (ЛМПСК), структурну схему якої приведено на рисунку 2.1.



ППС – пристрій перетворення сигналів
ПВВ – пристрій введення
ПВИВ – пристрій виведення
ОК-об'єкт керування
Д1…Д3-аналогові давачі
ЦД-цифрові давачі

ФНЧ1…ФНЧ3 – фільтри нижніх частот
НП1…НП3-нормуючі перетворювачі
АМПС- аналоговий мультиплексор
ПВЗ – пристрій вибірки-зберігання
АЦП – аналого-цифровий перетворювач
МК – мікроконтролер

ШФ – шинний формувач
РГ1…РГ3- регістри
ЦАП1…ЦАП3 – цифро-аналогові перетворювачі
АВЕ1…АВЕ3 – аналогові виконавчі елементи
ДВЕ – дискретні виконавчі елементи
СУР1…СУР3 – схеми узгодження рівнів

МІНТ - мережевий інтерфейс
ПВІ - пристрій відображення інформації
КЗ- канал зв'язку

Рисунок 2.1 – Структурна схема локальної мікропроцесорної системи керування

ЛМПСК керує визначеним об'єктом керування (агрегатом) за декількома параметрами, наприклад, температурою, тиском, кутом повороту, переміщенням і т. ін. Система названа локальною, тому що керування виробляється і здійснюється на нижньому (локальному) рівні складної ієрархічної системи керування, що включає безліч різних

агрегатів (об'єктів керування). Основним елементом ЛМПСК є однокристальний мікроконтролер (МК) який називається веденим, тому що передбачається, що в складній системі мається декілька подібних ведених МК, які керують окремими агрегатами на локальному рівні.

Зараз замість терміна однокристальна мікро–ЕОМ (ОМЕОМ) використовується термін мікроконтролер (МК), тому цей термін також застосовується у даній роботі. На більш високому рівні ієрархії системи керування може знаходитися ведучий МК, який на основі інформації про стан окремих агрегатів виробляє необхідні значення заданих керуючих впливів для ведених МК. Ведучий і ведений МК можуть бути зв'язані між собою, наприклад, спільним моноканалом [15].

ЛМПСК підтримує кожний з конкретних параметрів на заданому рівні. Інформація про поточне значення параметрів контролю знімається з аналогових давачів (Д1...Д3) і проходить через нормуючі перетворювачі (НП1...НП3), які перетворюють діапазон зміни електричних сигналів, що знімаються з давачів, до діапазону, що відповідає обраному аналого–цифровому перетворювачу (АЦП). Оскільки інформаційні сигнали в більшості систем керування – низькочастотні, то для придушення високочастотних завад використовуються фільтри нижніх частот (ФНЧ). Аналоговий мультиплексор по черзі підключає до АЦП один з декількох аналогових електричних сигналів, які відображають поточні значення контрольованих параметрів. У випадку, якщо за час перетворення АЦП, зміна вхідного сигналу відповідає зміні вихідного двійкового коду більше, ніж на одиницю молодшого значущого розряду (МЗР), то для зменшення так званої „апертурної” похибки, яка виникає при цьому, у систему включають пристрій вибірки–зберігання (ПВЗ) [6, 34].

ПВЗ запам'ятовує миттєві значення вхідних аналогових сигналів у момент вибірки і підтримує їх постійними на вході АЦП протягом часу

перетворення останнього. З виходу АЦП інформація в паралельному двійковому коді надходить у ведений МК, який порівнює поточне значення контрольованого параметру з заданим значенням і виробляє керуючий вплив відповідно до сигналу розузгодження та обраним законом керування (П, ПІ, ПІД і т. ін.). Сигнали керування, що знімаються з виходу одного з паралельних портів МК, запам'ятовуються в зовнішніх регістрах РГ1...РГ3. Для підвищення навантажувальної здатності виходів МК, у системі використано шинний формувач (ШФ). Виходи РГ1…РГ3 через схеми узгодження рівнів СУР1…СУР3 зв'язано з входами цифро–аналогових перетворювачів ЦАП1…ЦАП3, що формують аналогові керуючі впливи, спрямовані на усунення сигналу розузгодження і призначені для відпрацьовування аналоговими виконавчими елементами (АВЕ1…АВЕ3). СУР1…СУР3 необхідні в тих випадках, коли рівні одиничних логічних сигналів, що знімаються з виходів регістрів, не відповідають необхідним рівням одиничних сигналів на входах ЦАП [35].

У якості СУР, як правило, використовують логічні елементи з відкритим колектором [34].

У загальному випадку, ЛМПСК крім аналогових давачів і виконавчих елементів можуть містити цифрові давачі і дискретні виконавчі елементи, які через паралельні порти поєднуються з мікроконтролером.

2.2 Призначення і схемна реалізація окремих вузлів ЛМПСК

2.2.1 Аналоговий мультиплексор (АМПС)

АМПС використовується для почергової передачі поточного значення одного з трьох аналогових контрольованих параметрів на вхід

ПВЗ і АЦП. Для цього може бути, наприклад, використано мікросхему К561КП1.

На рисунку 2.2 зображено позначення цієї мікросхеми на електричних схемах і пояснюється яким чином АМПС зв'язано з іншими частинами ЛМПСК.

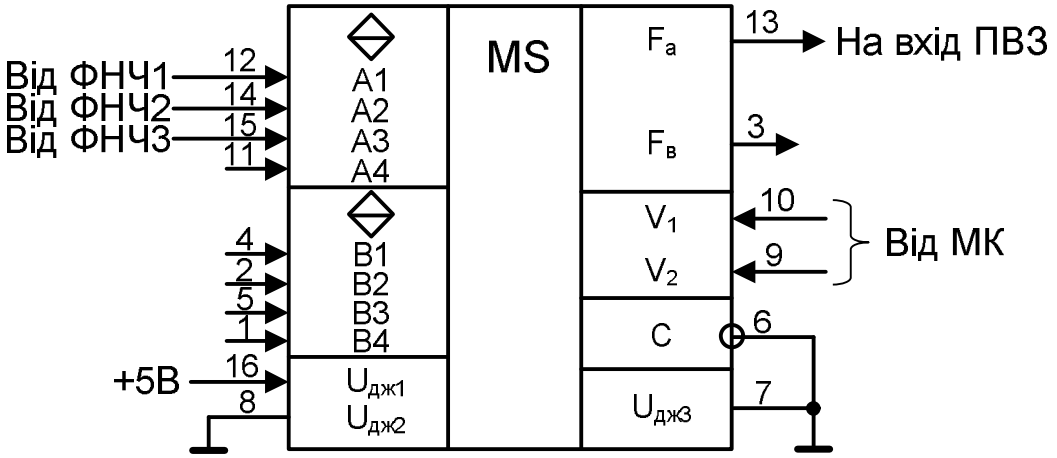


Рисунок 2.2 – Схема включення АМПС

Розглянутий пристрій відноситься до класу мультиплексорів–селекторів (мультиплексорів–демультиплексорів). Мікросхема містить два мультиплексори–селектори. У нашому прикладі використано половину мікросхеми в якості мультиплексора. В залежності від значень адресних сигналів, що надходять від МК–ра на входи V1, V2, у мультиплексорі утворюється наскрізний низькоомний канал між виходом F_a і одним із входів A1, A2, A3, на які подаються інформаційні сигнали від ФНЧ. З виходу F_a обраний сигнал надходить на вхід ПВЗ.

2.2.2 Пристрій вибірки–зберігання (ПВЗ)

ПВЗ призначено для запам'ятовування миттєвого значення вхідного аналогового сигналу в момент вибірки і підтримки цього значення на постійному рівні під час перетворення інформації в АЦП. Подібний

пристрій необхідно застосовувати в тих випадках, коли за час перетворення інформації в АЦП зміна його вхідного аналогового сигналу еквівалентна дискретній зміні вихідного сигналу більш ніж на одиницю молодшого значущого розряду (МЗР). У якості ПВЗ може бути, наприклад, використано мікросхему К1100СК2. На рисунку 2.3 зображено позначення цієї мікросхеми на електричних схемах і пояснюється, яким чином ПВЗ пов'язано з іншими частинами ЛМПСК. Тривалість імпульсу запису інформації у ПВЗ (імпульсу вибірки) $t_{зап}$ ($t_в$) при значенні ємності збереження $C_{ЗБ}$=1нф дорівнює 5 мкс.

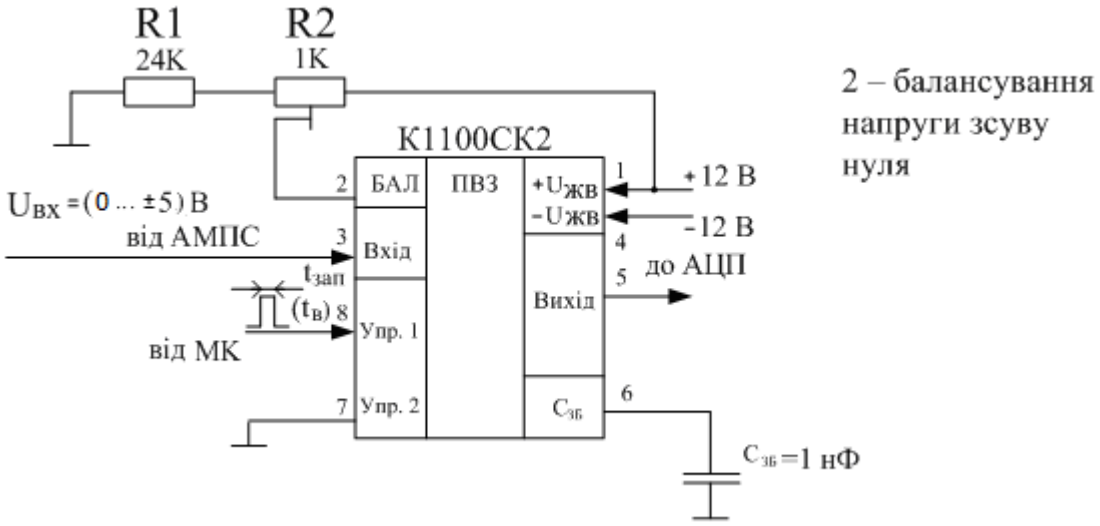


Рисунок 2.3 – Схема включення ПВЗ

2.2.3 Аналого–цифровий перетворювач (АЦП)

АЦП виконує перетворення аналогової напруги в 8–розрядний паралельний двійковий код, що вводиться в МК.

У якості АЦП може бути використано, наприклад, мікросхему К1113ПВ1. На рисунку 2.4 наведено позначення цієї мікросхеми на електричних схемах і показується яким чином АЦП пов'язано з іншими частинами ЛМПСК. Особливості взаємодії АЦП і МК пояснюють часові діаграми роботи АЦП (рисунок 2.5).

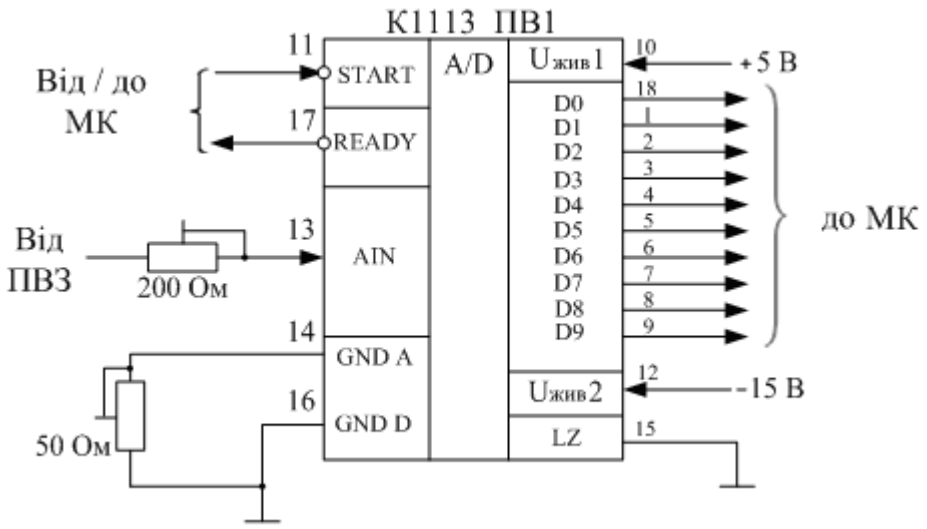


Рисунок 2.4 – Схема включення АЦП

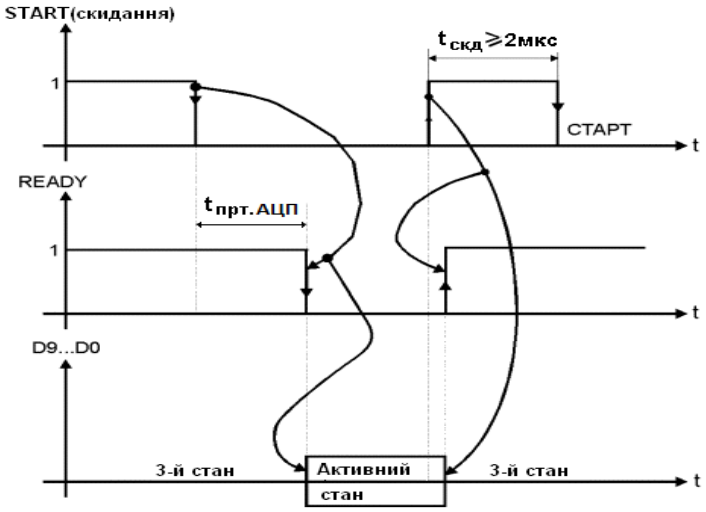


Рисунок 2.5 – Часові діаграми роботи АЦП

Запуск АЦП відбувається при переключенні сигналу на вході START (СТАРТ) з логічної одиниці в нуль. Під час перетворення на виході READY (ГОТОВНІСТЬ) присутня логічна одиниця, а шина даних знаходиться в третьому (високоімпедансному) стані. По закінченню перетворення вихідні сигнали на виводах даних D0…D9 переходять в активний стан, а сигнал на виході READY переключається з 1 в 0.

Одержавши сигнал готовності, МК зчитує (вводить) дані від АЦП і переводить сигнал на вході START у стан 1 на час, не менший 2 мкс. Цим здійснюється „скидання” АЦП, після якого може вироблятися наступний „запуск” АЦП і т.ін.

2.2.4 Ведений мікроконтролер

Ведений мікроконтролер отримує інформацію про поточний стан об'єкта керування, робить порівняння цього стану з заданим, виробляє сигнали розузгодження, реалізує необхідні закони керування і видає керуючі впливи на виконавчі елементи. У якості веденого МК може бути використано, наприклад, мікросхему AT89C51. На рисунку 2.6 наведено позначення цієї мікросхеми на електричних схемах і пояснюється яким чином вона пов'язана з іншими частинами ЛМПСК. За допомогою ланцюжка C1, R1 виробляється сигнал автоматичного „скидання” МК при включенні напруги живлення.

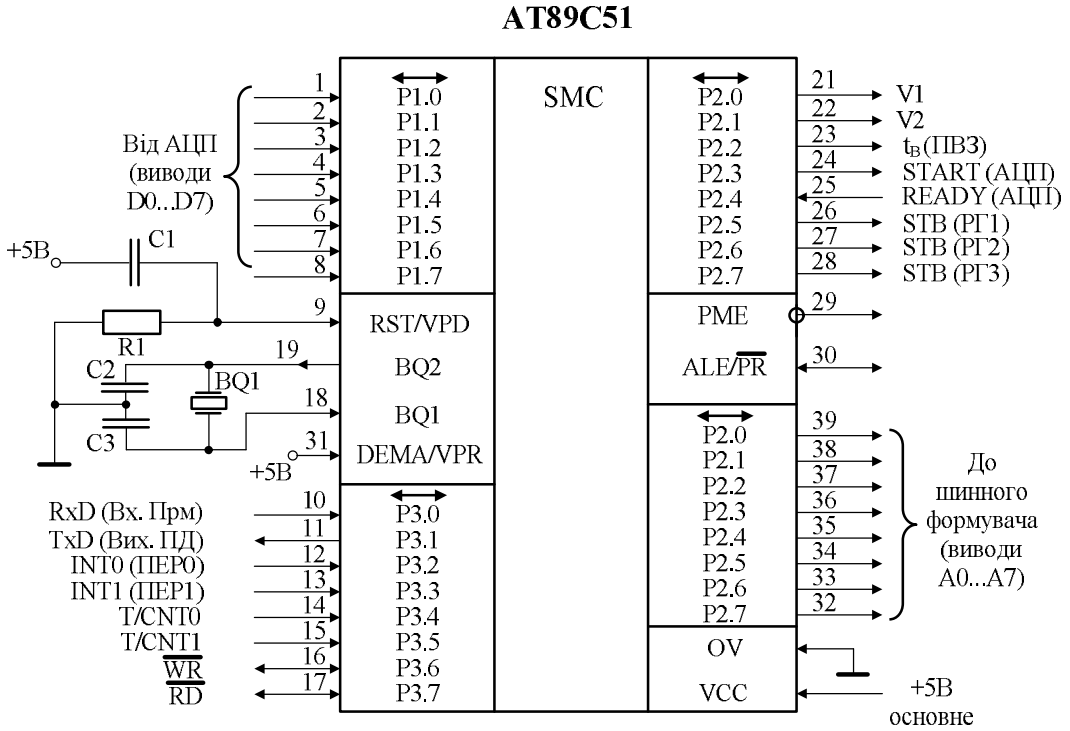


Рисунок 2.6 – Схема включення МК–ра

2.2.5 Шинний формувач (ШФ)

ШФ застосовується для підвищення навантажувальної здатності виводів МК, що для порту Р0 дорівнює двом входам логічного елемента типу ТТЛШ/КМОН (рисунок 2.7).

Оскільки виводи порту Р0 повинні підключатися до інформаційних входів трьох регістрів (рисунок 2.1), то для підсилення сигналів використовується шинний формувач. У якості ШФ може бути, наприклад, обрано мікросхему КР1533АП26. На рисунку 2.7 наведено позначення цієї мікросхеми на електричних схемах і пояснюється, яким чином ШФ пов'язано з іншими частинами ЛМПСК.

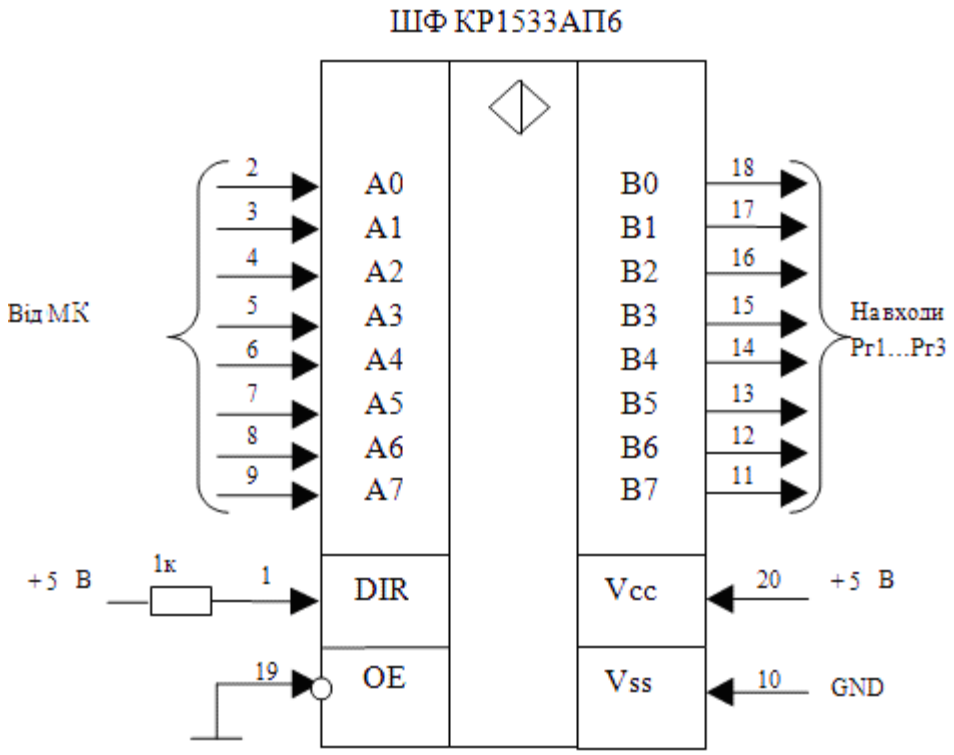


Рисунок 2.7 – Схема включення шинного формувача

2.2.6 Регістри (РГ1…РГ3)

Паралельні регістри РГ1...РГ3 призначено для запам'ятовування значень керуючих впливів по кожному з трьох каналів. Ці впливи видаються з МК–ра у паралельному двійковому коді і супроводжуються стробуючим сигналом, що записує сформований керуючий вплив у необхідний регістр. Вміст регістрів залишається незмінним до нового запису, що ініціюється подачею на відповідний вхід регістра стробуючого імпульсу. Регістри РГ1...РГ3 виконують функцію демультиплексора.

У якості регістрів може бути використано, наприклад, мікросхему КР1533ИР23. На рисунку 2.8 наведено позначення цієї мікросхеми на електричних схемах і пояснюється, яким чином регістри пов'язані з іншими частинами ЛМПСК.

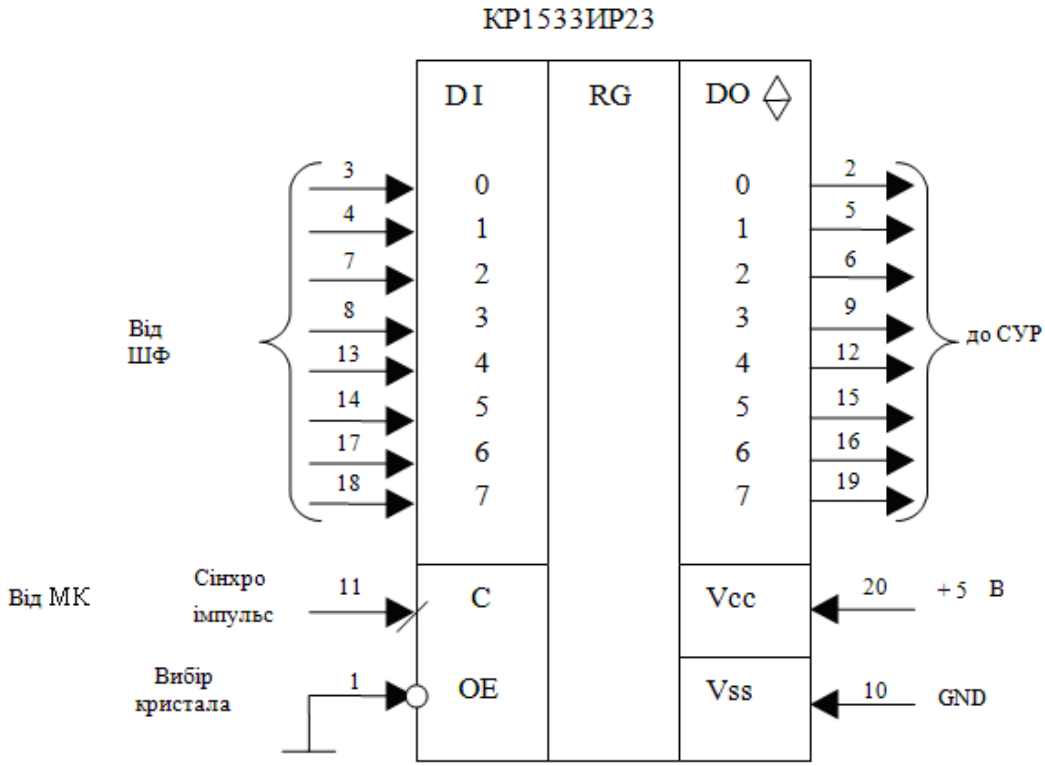


Рисунок 2.8 – Схема включення регістра

2.2.7 Схеми узгодження рівнів (СУР1…СУР3)

СУР1...СУР3 необхідно застосовувати в тих випадках, коли мінімальні рівні напруг логічної одиниці, що з'являються на виходах регістрів, не відповідають діапазону вхідних напруг логічної одиниці ЦАП, який виконано, наприклад, на мікросхемі К572ПА1, та якщо останній живиться напругою +15В. СУР не здійснюють ніяких логічних перетворень і містять виходи з відкритим колектором, які через зовнішні колекторні резистори підключаються до напруги живлення, значення якої визначається необхідними величинами рівнів вхідних напруг логічної одиниці ЦАП (у нашому прикладі від 4,5 до 15 В).

У якості СУР може бути використано, наприклад, мікросхему К555ЛН4. На рисунку 2.9 наведено позначення цієї мікросхеми на електричних схемах і пояснюється, яким чином СУР пов'язано з іншими частинами ЛМПСК.

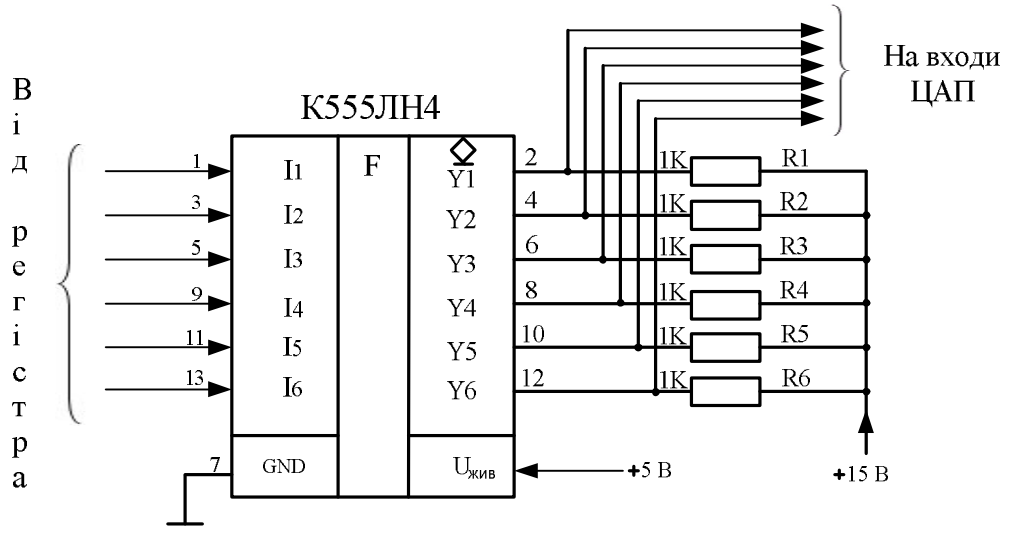


Рисунок 2.9 – Включення схеми узгодження рівнів

Подібних мікросхем у розглянутому прикладі (рисунок 2.1) потрібно чотири, тому що одна мікросхема включає шість повторювачів з відкритим

колектором, а загальна кількість логічних сигналів, що вимагають перетворення рівнів, дорівнює 3 х 8 = 24.

2.2.8 Цифро–аналогові перетворювачі (ЦАП1…ЦАП3)

ЦАП1...ЦАП3 здійснюють перетворення цифрових керуючих сигналів, які формує МК, в аналогові керуючі впливи, що відпрацьовуються аналоговими виконавчими елементами (АВЕ1...АВЕ3).

У якості ЦАП може бути використано, наприклад, мікросхему К572ПА1, схема включення якої показано на рисунку 2.10. Коефіцієнт передачі цього ЦАП $К_{пер}$ = 10мВ/мзр, діапазон зміни вихідної аналогової напруги при 8–розрядному вхідному двійковому сигналі, який подається на входи D0...D7 ЦАП, складає $U_{вих.ан}$ = 0...2,55 В.

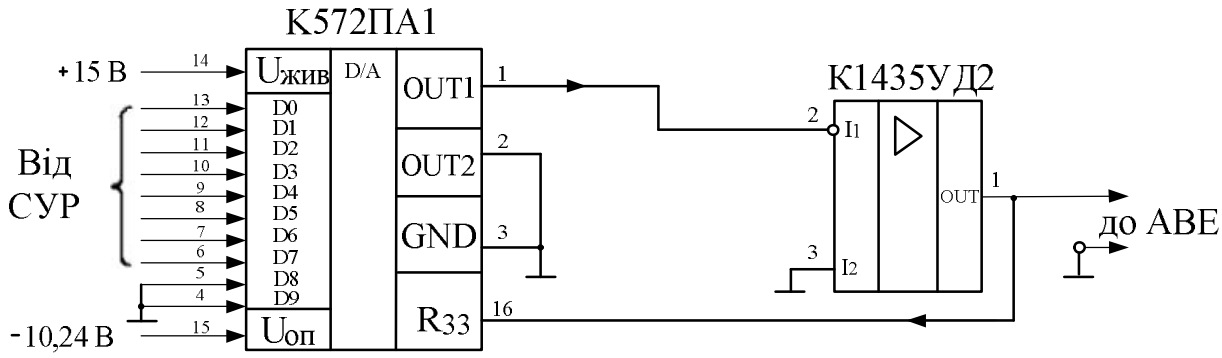


Рисунок 2.10 – Включення схеми ЦАП

2.3 Модульна структура МПС

На рисунку 2.11 приведено модульну структуру найпростішої МПС, яку побудовано на основі 8–розрядного МП–ра, наприклад, і8080 (КР580ВМ80А) [1…4, 6, 31].

До складу найпростішої МПС входять наступні модулі: модуль МП–ра, пам'яті і введення/виведення.

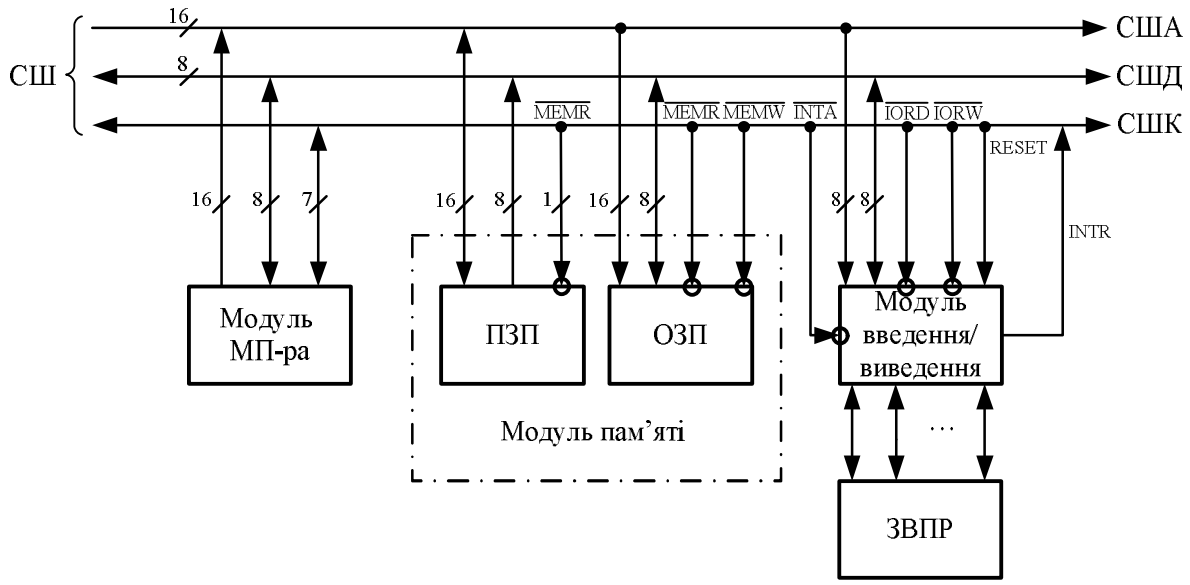


Рисунок 2.11 – Модульна структура МПС

2.3.1 Модуль мікропроцесора (МП)

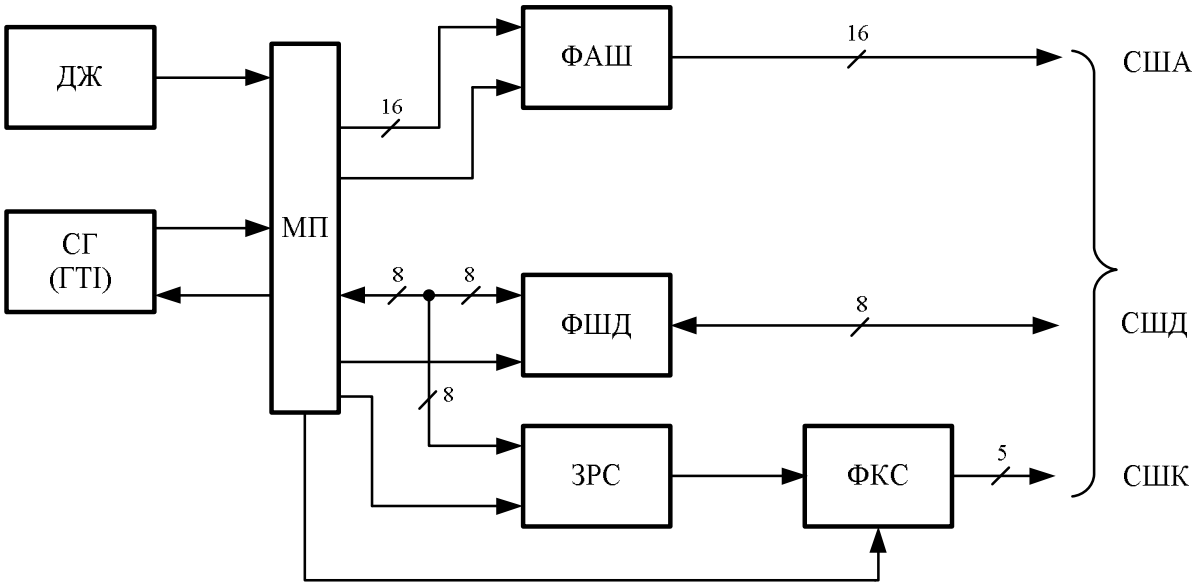
Модуль МП – керує запам'ятовуючими пристроями (ЗП) і зовнішніми пристроями (ЗВПР), а також здійснює обробку інформації.

До складу модуля мікропроцесора, наприклад, входять (рисунок 2.12):

- СГ – системний генератор тактових імпульсів (ГТІ) високої частоти;
- МП – мікропроцесор;
- ФАШ – формувач адресної шини;
- ФШД – формувач шини даних;
- ЗРС – зовнішній регістр стану;
- ФКС – формувач керуючих сигналів.

ФАШ і ФШД – це шинні формувачі. Вони необхідні для збільшення навантажувальної здатності виводів МП (підсилення сигналів, що знімаються з його виходів). Крім того, вони призначені для забезпечення

двостороннього обміну інформацією між МП і пам'яттю або ЗВПР, а також для формування трьох станів вихідних сигналів.



МП – мікропроцесор
ДЖ – джерело живлення
ГТІ – генератор тактових імпульсів
ФАШ – формувач адресної шини
ФШД – формувач шини даних
ЗРС - зовнішній регістр стану
ФКС – формувач керуючих сигналів

Рисунок 2.12 – Структура модуля мікропроцесора

Зовнішній регістр стану (ЗРС) призначений для запам'ятовування слова стану мікропроцесора, яке, наприклад, у 8–розрядного МП–ра типу i8080 з'являється на його шині даних на початку кожного машинного циклу виконання поточної команди.

У 8–розрядному МП–рі (наприклад, i8080) є два слова стану:

– перше – це слово стану програми, позначається PSW і вміщує вміст акумулятора (старший байт PSW) і регістра прапорців (ознак) (молодший байт PSW);

– друге – відображає стан МП–ра, тобто ідентифікує тип машинного циклу (МЦ), який виконується в даний момент часу. В цьому випадку слово стану – це байт даних, кожний біт якого несе певну інформацію.

Для названого процесора існують 10 різних типів МЦ, наприклад, вибір коду операції команди, читання пам'яті, запис в пам'ять, введення, виведення і т. ін. Відповідно з модульною структурою МПС (рисунок 2.11) мікропроцесор може обмінюватися інформацією між пам'яттю і пристроєм введення/виведення (ПВВ). Обмін з кожним із названих пристроїв може вестися в одному із двох напрямів: із МП–ра (запис, виведення, передача) і до МП–ра (читання, введення, прийом). В конкретний момент часу МП–р може здійснювати обмін тільки з одним із зазначених пристроїв (пам'ять або ПВВ) і тільки в одному із двох напрямів (запис або читання). Для керування обміном використовуються чотири керуючих сигнали, які формуються формувачем керуючих сигналів (ФКС) і видаються на системну шину із модуля МП:

– $\overline{MEMR}$ ($\overline{ЧТП}$) – читання з пам'яті;
– $\overline{MEMW}$ ($\overline{ЗПП}$) – запис у пам'ять;
– $\overline{IORD}$ ($\overline{ЧТВВ}$) – читання введення/виведення;
– $\overline{IOWR}$ ($\overline{ЗПВВ}$) – запис введення/виведення.

Крім названих чотирьох основних сигналів керування, ФКС може формувати сигнал $\overline{INTA}$ ($\overline{ППЕР}$) – підтвердження переривання і т. ін. Активним значенням усіх названих сигналів є логічний нуль.

При розробці модуля мікропроцесора на основі МП і8086 (рисунок 2.13) виникають наступні задачі: розподілу (демультиплексування) шини адреси/даних (ШАД), буферування шини адреси (ША) і шини даних (ШД), а також формування системних керуючих сигналів для блоків пам'яті і зовнішніх пристроїв.

Перша задача вирішується, наприклад, за допомогою інтегральних мікросхем К1810ІР82/83, що виконують функції адресної защіпки. Оскільки сигнал $\overline{BHE}$ формується в тому ж інтервалі часу, що і адресні сигнали, то його також необхідно зафіксувати у защіпці.

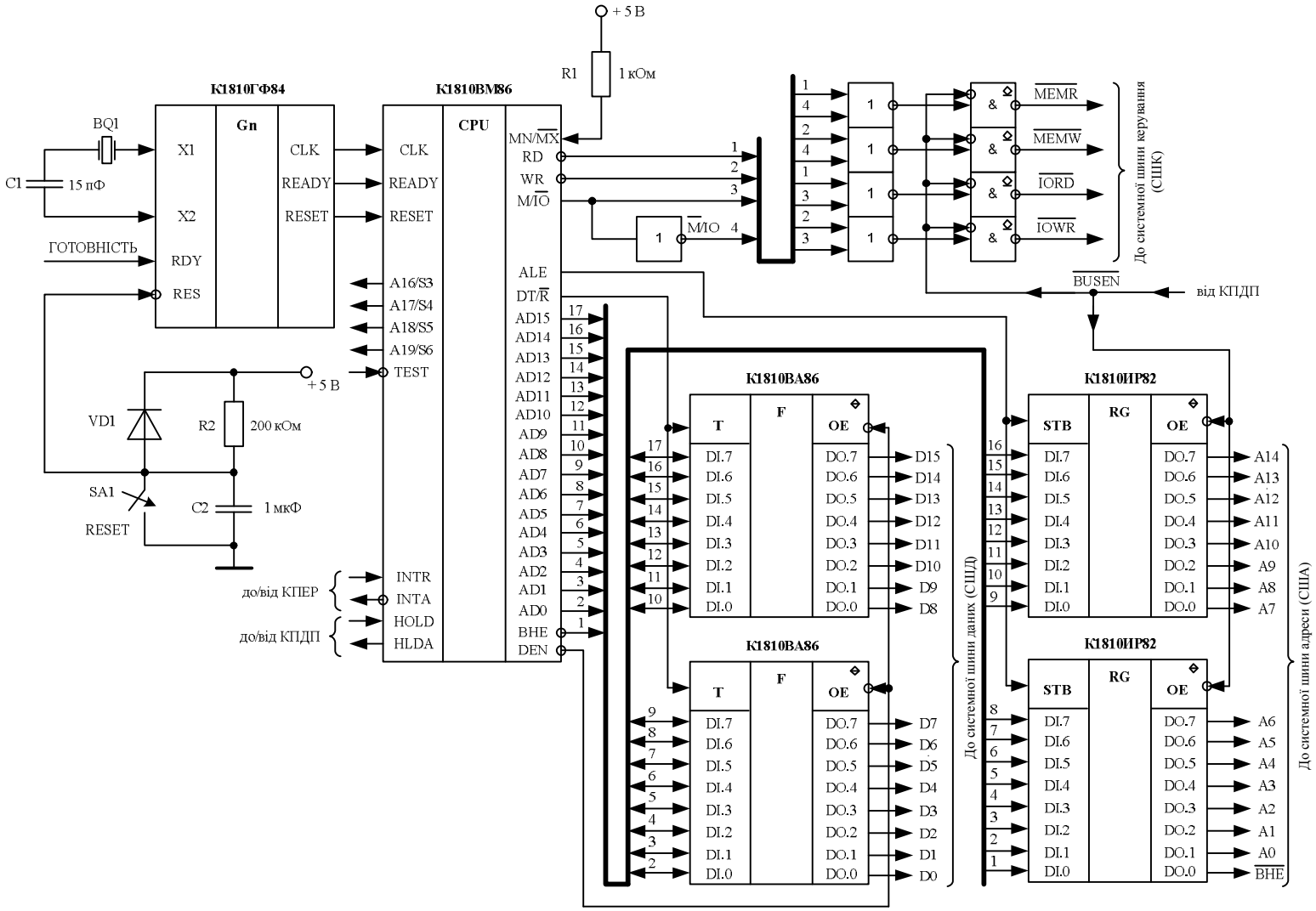


Рисунок 2.13 – Функціональна схема модуля мікропроцесора

Тому зображені на рисунку 2.13 два 8–бітових регістра К1810ИР82 забезпечують запам'ятовування 15 розрядів адреси (A0…A14) та сигналу $\overline{BHE}$.

Для доступу до пам'яті максимальної ємності 1 Мбайт необхідно підключити ще один регістр, на який подаються старші розряди, що залишилися, AD15, A16/S3…A19/S6.

Друга задача вирішується за допомогою двоспрямованих 8–бітових шинних формувачів К1810ВА86/87, які підсилюють сигнали для передачі на системну шину даних.

Третя задача може бути вирішена за допомогою комбінаційних логічних схем, які формують необхідні керуючі сигнали на основі сигналів $\overline{RD}$, $\overline{WR}$ і $M/\overline{IO}$, що виробляються МП. Якщо в системі використовується адресний простір введення/виведення, ізольований від простору пам'яті, то доцільно сформувати сигнали: $\overline{MEMW}$, $\overline{MEMR}$, $\overline{IORD}$, $\overline{IOWR}$. Ці сигнали керують запам'ятовуючими і зовнішніми пристроями подібно тому, як це робиться в системах, побудованих на основі МП і8080 [2…4, 12, 32]. Роль формувачів сигналів можуть виконувати логічні елементи (рисунок 2.13) або дешифратор на три входи (наприклад, К155ИД7).

Якщо ж в МПС введення/виведення організоване з відображенням на пам’ять, то сигнал $M/\overline{IO}$ не використовується і на ЗВПР подаються сигнали $\overline{RD}$ і $\overline{WR}$ після підсилення.

Підсилювачі і формувачі повинні забезпечувати три стани вихідних сигналів, щоб можна було організувати прямий доступ до пам'яті. Це стосується виходів шинних формувачів та регістрів. У цьому випадку після переведення МП в стан захоплення ці пристрої переходять в третій стан по зняттю сигналу $\overline{\text{ДОЗВІЛ ШИН}}$ ($\overline{BUSEN}$), що поступає від контролера ПДП. Керуючі сигнали для системної шини керування знімаються з логічних

елементів з відкритим колектором. Це зроблено для того, щоб поєднати ці сигнали із аналогічними сигналами, які формуються контролером ПДП.

При цьому кожен з аналогічних керуючих сигналів від модуля МП–ра і КПДП поєднуються та за допомогою зовнішнього резистора підключаються до напруги живлення +5В. Таке з’єднання у літературі зветься монтажне «АБО» для нульових вихідних сигналів, або монтажне «І» для одиничних вихідних сигналів. Якщо захоплення шин і обмін даними за ПДП не передбачено, то необхідність в такому перемиканні відпадає.

2.3.2 Модуль пам’яті

Модуль пам'яті – призначено для збереження програм і даних. До його складу входять ПЗП і ОЗП. ПЗП – постійний запам'ятовуючий пристрій, призначений для збереження програм і даних, що беруть участь в операціях. Інформація до ПЗП заноситься на етапі виготовлення або перепрограмування системи. Це енергонезалежна пам'ять. ОЗП – оперативний запам'ятовуючий пристрій, призначений для збереження даних, що оброблюються, і для збереження програм, які часто змінюються.

Більш детально види ОЗП та ПЗП розглядаються в окремій лекції.

2.3.3 Модуль введення/виведення

Модуль введення/виведення – забезпечує взаємозв'язок МП–ра із зовнішніми пристроями. До його складу можуть входити: паралельний програмований інтерфейс (ППІ) і програмований послідовний інтерфейс, наприклад, УАПП – універсальний асинхронний програмований приймач/передавач, який виконує послідовний обмін інформацією між ЗВПР та МП.

Додатково до складу МПС можуть входити також наступні модулі: контролер переривань (КПЕР), контролер прямого доступу до пам'яті (КПДП), таймер, АЦП–ЦАП і т.ін.

2.3.4 Контролер прямого доступу до пам’яті (КПДП)

Необхідність використання режиму ПДП в МПС викликано декількома факторами. По–перше, при використанні даного режиму з’являється можливість початкового завантаження програми в основну пам’ять МПС із пристроїв введення. По–друге, що є більш важливим, режим ПДП забезпечує можливість обміну даними між пам’яттю та ЗВПР блоками фіксованого розміру, починаючи з 128 байт. Причому цей обмін може проводитися з високою швидкістю, яка обмежена часом звернення до пам’яті, і в заздалегідь визначеній послідовності.

Забезпечення необхідної швидкості обміну блоків даних між пам’яттю МПС та ЗВПР за допомогою програмно–керованого обміну не можливе, оскільки на обмін кожним байтом між пам’яттю та ЗВПР через МП витрачається, як правило, декілька команд процесора, сумарний час виконання яких часто перевищує максимально допустимий час на обмін одним байтом із зовнішнім пристроєм. Необхідна швидкість обміну забезпечується режимом ПДП, при якому час на обмін одним байтом обмежується швидкодією пам’яті.

Для реалізації режиму ПДП необхідно забезпечити безпосередній зв’язок контролера ПДП і пам’яті МПС. Для цього можна було б використовувати спеціально виділені шини адреси, керування та даних, які пов’язують контролер ПДП з пам’яттю. Проте це приведе до значного ускладнення системи, особливо при підключенні декількох зовнішніх пристроїв. Контролер ПДП підключається до пам’яті за допомогою

системної шини МПС. Проблема сумісного використання системної шини МП та контролером ПДП найчастіше розв'язується таким чином.

По системній шині керування МПС від контролера ПДП мікропроцесору передається керуючий сигнал «Вимога прямого доступу до пам'яті» (ВПДП). Процесор після отримання сигналу ВПДП призупиняє виконання програми, видає на системну шину керування керуючий сигнал контролеру ПДП «Надання прямого доступу до пам'яті» (НПДП) і відключається від шини адреси (ША), шини керування (ШК) та шини даних (ШД). Для відключення модуля МП від ША та ШД використовується можливість переведення його виходів, які підключені до цих шин, у третій (високоімпедансний) стан. Керуючі сигнали від КПДП до СШК підключаються за допомогою елементів з відкритим колектором (стоком), як це описано в 2.3.1. З цього моменту системна шина МПС передається у розпорядження контролера ПДП. Контролер ПДП, використовуючи системну шину, обмінюється блоками даних з пам'яттю, а потім, знявши сигнал ВПДП, повертає керування системною шиною МП.

Застосування в МПС режиму ПДП потребує попередньої підготовки. Для кожного зовнішнього пристрою необхідно виділити область пам'яті, яка буде використана при обміні, та вказати її розмір, тобто кількість байтів (слів) інформації, що записуються в пам'ять або зчитуються з пам'яті. Тому контролер ПДП повинен мати у своєму складі регістр адреси та лічильник байтів (слів). Перед початком обміну із ЗВПР у режимі ПДП процесор повинен виконати програму початкової ініціалізації КПДП. Ця програма забезпечує запис у вищезгадані регістри контролера початкової адреси виділеної ЗВПР пам'яті та її розміру в байтах або словах, в залежності від того, якими порціями інформації буде здійснюватися обмін.

Згадане не відноситься до початкового завантаження програм в пам'ять МПС в режимі ПДП. У цьому випадку вміст регістра адреси та

лічильника байтів встановлюється перемикачем або перемичками безпосередньо на платі контролера.

Типовий представник сучасного контролера ПДП (КПДП) – мікросхема КР580ВТ57, яка входить до складу мікропроцесорної серії КР580 [6, 12, 33].

КР580ВТ57 – це чотириканальний програмований контролер прямого доступу до пам’яті, призначений для організації безпосереднього зв’язку між зовнішніми пристроями та пам’яттю в МПС на основі комплекту КР580 та К1810.

Основна функція контролера – формування адреси пам’яті та керуючих сигналів зчитування (запису) пам’яті від зовнішнього пристрою. Обмін даними між пам’яттю та зовнішнім пристроєм здійснюється без участі контролера, але під його безпосереднім керуванням. Контролер приймає запити ПДП від зовнішніх пристроїв, здійснює їх пріоритетну обробку, формує мікропроцесору сигнал захоплення шин, в результаті чого системна шина МПС відключається від МП і формується послідовність адрес пам’яті та керуючих сигналів зчитування (запису). Після закінчення передачі даних контролер видає керуючий сигнал мікропроцесору про закінчення обміну. Кожен з чотирьох каналів КПДП містить 16–розрядний регістр, який дає змогу адресувати пам’ять об’ємом 64 Кбайт, і 14–розрядний регістр числа циклів обміну (лічильник циклів), який дає змогу здійснювати пересилку масивів слів даних об’ємом до 16 Кбайт.

КПДП дає змогу здійснювати операції зчитування даних з пам’яті та запису даних в пам’ять. Існує можливість контролю мікросхеми у режимі ПДП шляхом виконання його циклів без пересилання даних. Режими роботи та функції, що виконуються, задаються програмно від МПС. Контролер має два режими пріоритетної обробки запитів ПДП – з фіксованим та з циклічним пріоритетами; два режими синхронізації обмінів – звичайного та

подовженого запису; режим автоматичної зупинки при закінченні відліку – режим ЗВ–стоп; режим регенерації та стикування масивів даних – режим автозавантаження.

Конструктивно мікросхема виконана у пластмасовому корпусі з 40 виводами.

2.3.5 Контролер переривань (КПЕР)

На технічні характеристики мікропроцесорних систем керування вагомо впливають засоби обміну інформацією між обчислювальним ядром і різноманітними периферійними пристроями. Ці засоби створюють підсистему введення/виведення інформації, яка включає в себе програмні та апаратні засоби інтерфейсу.

Суттєво підвищити ефективність роботи засобів підтримки обміну інформацією в мультимодульних системах можна, у випадку надання права ініціації обміну кожному модулю системи. Для цього обчислювальне ядро системи повинно мати можливість приймати запити на переривання від інших модулів, переривати обробку поточного процесу і переходити на обслуговування запитів по обміну інформацією.

Засоби, які реалізують обмін за перериванням в мікропроцесорних системах, називаються підсистемою переривання і реалізуються у вигляді окремих ВІС або входять в склад МК–ра. В мікропроцесорному комплекті серії КР580, наприклад, використовується спеціальна мікросхема К580ВН59, за допомогою якої можна реалізувати багаторівневу систему переривань з можливістю врахування пріоритетів і маскування входів [1, 6, 33].

Зовнішні пристрої (ЗВПР) можуть видавати сигнал на обслуговування переривань в будь–який момент часу, подаючи сигнал рівня «лог. 1» на вхід INT мікропроцесора (апаратний запит переривань), або за допомогою

встановлення ознаки (прапорця) в одному із розрядів слова стану ЗВПР, яке зчитується мікропроцесором (програмний запит переривання).

При роботі згідно з останнім типом переривань мікропроцесорна система (МПС) в процесі виконання програми звертається до ЗВПР і перевіряє логічний рівень ознаки запиту на переривання. При підтвердженні запиту МПС організовує виконання відповідної програми обслуговування переривання. Підключення ЗВПР в такому режимі нічим не відрізняється від стандартного варіанту підключення ЗВПР до МПС (наприклад, за допомогою паралельного інтерфейсу КР580ВВ55А).

Якщо в системі використовується апаратний запит на обслуговування переривання, то МП, отримавши сигнал запиту INT від ЗВПР, формує відповідний сигнал $\overline{INTA}$, що інформує зовнішній пристрій про те, що мікропроцесор (МП) припинив виконання основної програми і перейшов до режиму обслуговування переривання.

Після завершення виконання поточної команди основної програми, МП приймає від ЗВПР команду переходу на виконання підпрограми його обслуговування. За цією командою МП зберігає в стеку стан поточної програми, що виконується, та переходить до підпрограми обслуговування переривання. Завершується підпрограма обслуговування переривання командою RETI, що відновлює із стека стан перерваної основної програми та продовжує її виконання.

У МК–х запити на переривання можуть надходити ззовні, або від окремих периферійних модулів.

2.3.6 Модуль таймера

Модуль таймера використовується в МПС для формування інтервалів часу та підрахунку зовнішніх подій.

У мікропроцесорних системах існують наступні способи формування часових інтервалів (часових затримок, одиночних імпульсів, послідовностей імпульсів і т. ін.):

– апаратний;
– програмний;
– апаратно–програмний.

Перший спосіб реалізується використанням спеціалізованих електронних пристроїв, наприклад, ліній затримки, одновібраторів, мультивібраторів і т. ін.

В другому способі, використовуючи систему команд конкретного МП чи МК і з огляду на те, що виконання кожної команди займає деякий час, можна розробити підпрограми, що формують часові затримки, одиночні імпульси, послідовності імпульсів і т. ін.

Апаратно–програмний спосіб реалізується застосуванням програмованих таймерів.

Восьми– і шістнадцятирозрядні МП не містять вбудованих таймерів і використовують зовнішні мікросхеми, наприклад, КР580ВИ53 [6, 12, 32].

Однокристальний МК–р містить декілька внутрішніх 8–ми та 16–розрядних програмованих таймерів.

Крім формування часових інтервалів, таймери можуть виконувати підрахунок імпульсів, що ідентифікують настання зовнішніх подій, тобто працювати як лічильники зовнішніх подій з апаратним чи програмним запуском.

2.3.7 Системна шина

Окремі модулі МПС об'єднані між собою внутрішньо системним інтерфейсом і взаємодіють за адресним принципом – усі підлеглі пристрої і

їх складові частини мають адреси, що не повторюються, за якими до них звертаються пристрої, які виконують функції керування.

Внутрішньо системний інтерфейс реалізується частіше за все на основі єдиної системної шини (СШ), по якій передаються адреси, дані, команди і сигнали керування. При цьому для передачі даних і команд використовується шина даних (СШД). Адреси передаються по окремій шині адреси (США), яка керується процесором.

Системна шина керування (СШК) призначена для обміну керуючими сигналами між МП–м і пам'яттю або ЗВПР.

Основні сигнали СШК:

- $\overline{MEMR}$ ($\overline{ЧТП}$) – читання з пам'яті;
- $\overline{MEMW}$ ($\overline{ЗПП}$) – запис у пам'ять;
- $\overline{IORD}$ ($\overline{ЧТВВ}$) – читання введення/виведення;
- $\overline{IOWR}$ ($\overline{ЗПВВ}$) – запис введення/виведення;
- INT – сигнал переривання до МП;
- $\overline{INTA}$ ($\overline{ППЕР}$) – підтвердження переривання (із МП–ра до КПЕР).

Процесор обробляє інформацію трьох типів: дані, адреси і команди програми. Над даними виконуються арифметичні та логічні операції, реалізовані процесором. Обробка адреси визначається способом збереження і доступу до даних і команд, і також базується на виконанні обчислювальних операцій. Обробка команд складається з перетворення коду команди в послідовність керуючих впливів (мікрооперацій) відповідно до алгоритму виконання команди. Кожна мікрооперація (або їх сукупність – мікрокоманда) виконується у фіксовані інтервали часу (такти), а вся команда – за термін повного циклу (командного циклу) і утворює мікропрограму. Таким чином, обробка команд складається в їх представленні (інтерпретації) у формі мікропрограм. Під керуванням

мікропрограми виконується обробка даних і адрес, а також керування іншими пристроями МПС через внутрішньо системний інтерфейс.

2.4 Функціональна схема типової МПС керування

Роботу і застосування будь–якої мікропроцесорної системи (МПС) необхідно розглядати як на апаратному, так і на програмному рівні, оскільки ці дві властивості МПС є невід′ємними одна від одної.

Аналіз МПС на апаратному рівні розпочинають з вивчення структурної або функціональної схеми системи.

Розглянемо функціональну схему гіпотетичної МПСК, яку наведено на рисунку 2.14.

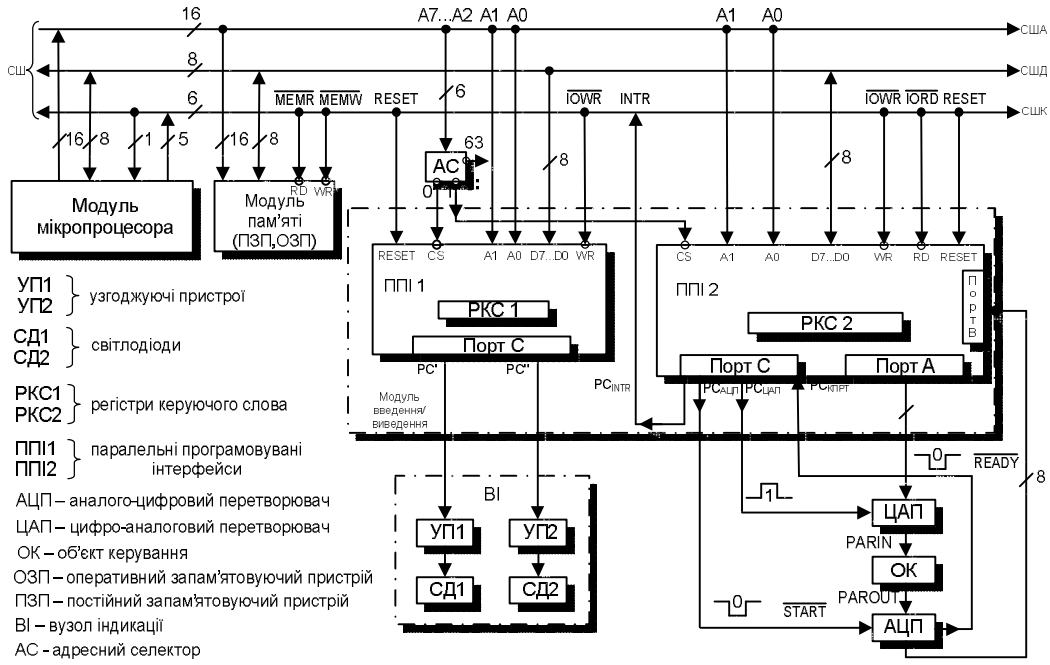


Рисунок 2.14 – Функціональна схема структури МСК

Дана мікропроцесорна система керування орієнтована на 8–ми розрядний МП і містить модулі:

– мікропроцесора (МП);

– пам'яті – ПЗП та ОЗП;

– введення/виведення;

– ЦАП;

– АЦП;

– об'єкт керування (ОК);

– вузол індикації (ВІ).

У постійному запам'ятовуючому пристрої (ПЗП) зберігається робоча програма, під керуванням якої буде працювати МПСК, яка проектується. Модуль введення/виведення, що складається з двох паралельних програмованих інтерфейсів (ППІ), забезпечує обмін інформацією між мікропроцесором, об'єктом керування та індикацією.

Аналого–цифровий перетворювач (АЦП) і цифро–аналоговий перетворювач (ЦАП) призначені для узгодження аналогового об'єкта керування з цифровою МПСК. Названі вузли зв'язані між собою системною шиною (СШ), що складається із трьох окремих системних шин:

– адреси (США);

– даних (СШД);

– керування (СШК).

Мікропроцесорна система (МПС), на відміну від універсальної обчислювальної системи (ОС) на основі персонального комп'ютера (ПК) призначена для виконання конкретної задачі в системі керування, контролю, вимірювання і т. ін. МПС є спеціалізованою системою. Основою кожної МПС є МП. Проте лише одного мікропроцесора недостатньо для побудови, як мікроконтролера, так і різних пристроїв керування на їх основі. Кожна мікропроцесорна система повинна включати засоби для збереження програм, даних і результатів обробки даних, засоби введення/виведення інформації і засоби відображення результатів обробки.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Що зберігається в постійному запам’ятовуючому пристрої?
2. Назвіть основні задачі, які виникають при розробці модуля мікропроцесора на основі МП і8086.
3. Для чого призначений оперативний запам’ятовуючий пристрій?
4. Обґрунтуйте необхідність використання режиму прямого доступу до пам’яті в МПС.
5. Назвіть основну функцію контролера ПДП (КПДП).
6. Який об’єм пам’яті дозволяє адресувати регістр адреси КПДП?
7. Опишіть особливості взаємодії МП–ра та КПДП.
8. Що називають підсистемою переривання?
9. Що виконує команда RETI?
10. Назвіть способи формування часових інтервалів у мікропроцесорних системах.
11. Назвіть основні сигнали системної шини керування.
12. Назвіть типи інформації, що обробляє процесор.
13. Для чого використовується аналоговий мультиплексор?
14. Що таке аналого–цифровий перетворювач? Для чого призначені АЦП?
15. Для чого призначені паралельні регістри РГ1...РГ3 в структурі локальної МПС керування?
16. Для чого використовуються схеми узгодження рівнів?
17. Що являє собою цифро–аналоговий перетворювач?

ЛІТЕРАТУРА [1…6, 12…15, 25, 32, 39]

3 СТРУКТУРА ТИПОВОГО МП–РА ТА МК–РА

3.1 Основні вузли МП–ра та МК–ра

3.1.1 Структура мікропроцесора i8080

Розглянемо структурну схему мікропроцесора (МП) (i8080), яку зображено на рисунку 3.1. МП i8080 є 8–розрядним монолітним МП, виготовленим за n – канальною МОП–технологією, виконаний в корпусі з 40 виводами, має 78 основних команд, частоту синхронізації – 0,5–4 МГц.

Призначення виводів і керуючих сигналів МП i8080 наведено у таблиці 3.1.

Нижче наведено опис основних вузлів МП–ра.

АЛП(Арифметико–логічний пристрій) – є центральним вузлом кожного МП і призначений для виконання арифметичних та логічних операцій, а також операцій зсуву. Вихідні дані для виконання операцій можуть передаватися з акумулятора і регістрів загального призначення або через буфер шини даних з пам'яті, причому один з операндів завжди міститься в акумуляторі. Результат виконаної операції повертається в акумулятор. Вбудований в МП–р блок десяткової корекції дозволяє виконувати операцію додавання чисел в упакованому BCD – форматі.

Регістри:

Акумулятор (А) – регістр, який є найуніверсальнішим і таким, що найбільш часто використовується у командах. В ньому завжди міститься один з операндів, що приймають участь в арифметичних, логічних операціях або операціях зсуву, а також результат операцій що виконується в АЛП. Обмін інформацією із зовнішніми пристроями в цьому МП також виконується тільки через акумулятор.

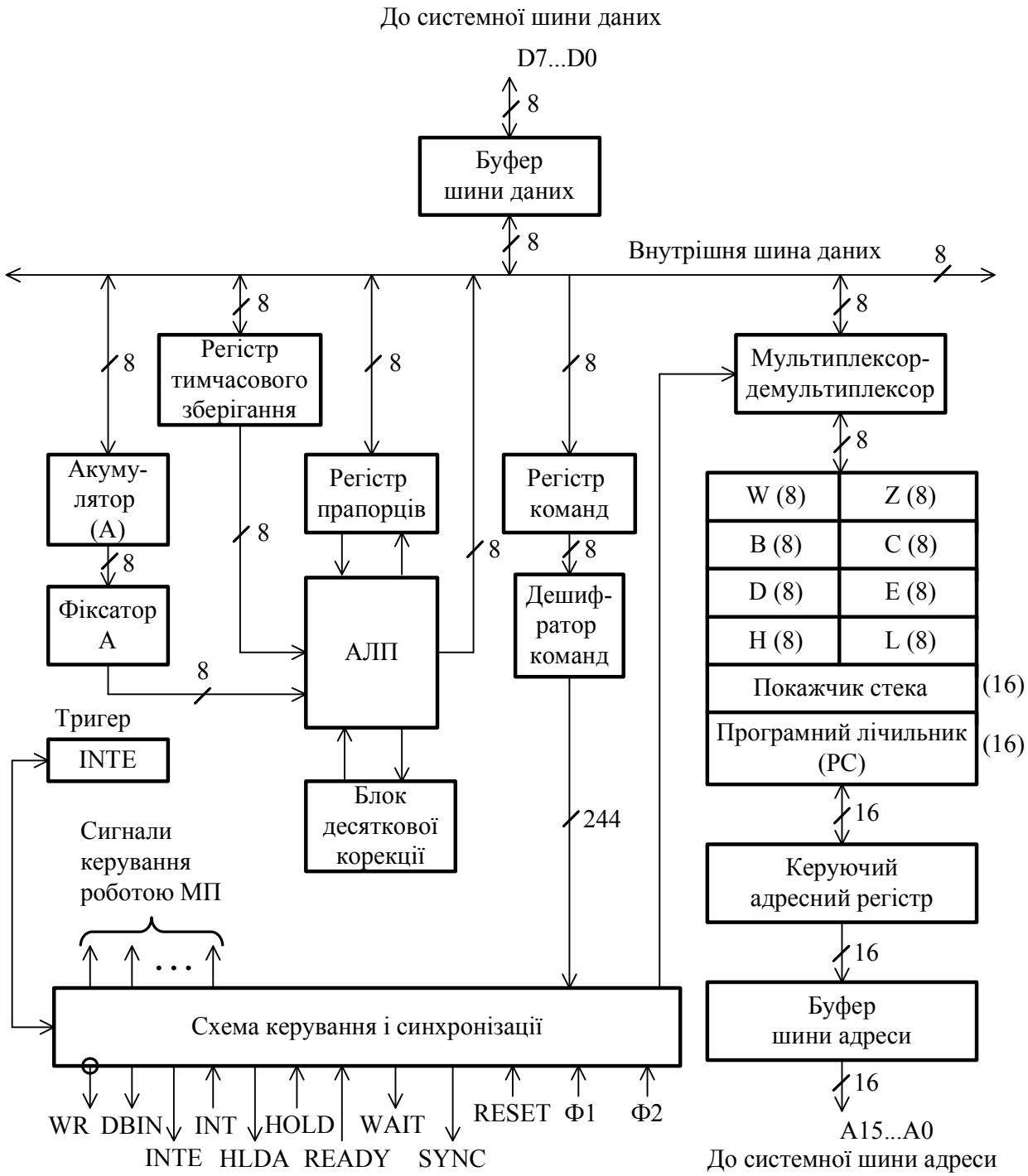


Рисунок 3.1 – Структурна схема МП i8080

Таблиця 3.1 – Призначення виводів і керуючих сигналів МП і8080

Позначення виводу	Призначення
D7…D0	Виводи двонаправленої 8–розрядної шини даних
A15…A0	Виводи 16–розрядної шини адреси
Ф1, Ф2	Тактові входи, на які подаються сигнали тактування (синхронізації), що генеруються зовнішнім генератором синхроімпульсів
SYNC	Вихід синхронізації, оповіщає ПВВ і пам'ять про початок нового машинного циклу, чим забезпечується узгодження роботи цих пристроїв з МП
+12В, –12В, +5В	Виводи для підключення напруги живлення
GND	«Земля»
RESET	Скидання, при подачі 1 на цей вхід скидається лічильник команд, а також тригери стану виходів INTE і HLDA
WR	Вихід сигналу запис, для інформування пам'яті або портів зовнішнього пристрою під час виконання МП операцій запису (низький рівень сигналу) або читання (високий рівень сигналу)
READY	Вхід сигналу готовності, який дає інформацію МП про готовність пам'яті або порту ПВВ до обміну даними
DBIN	Вихід сигналу «ШД в режимі введення», генерується під час прийому даних в МП (при зчитуванні їх з пам'яті або при виконанні операції введення від ПВВ)
WAIT	Вихід сигналу чекання, для вказівки того, що ЦП знаходиться в режимі чекання
HOLD	Вхід сигналу захоплення шин, служить для відключення ЦП від шин даних, керування і адреси шляхом переведення цих виводів в високоімпедансний стан. Цей сигнал використовується для організації режиму ПДП
HLDA	Вихід сигналу підтвердження захоплення шин, служить для інформування других блоків про захоплення шин МП іншим пристроєм
INT	Вхід сигналу «запит переривання» від ЗВПР
INTE	Вихід сигналу, що відображає стан тригера дозволу переривання

Регістри загального призначення (РЗП) – призначені для забезпечення швидкого доступу до операндів, що зберігається в них, оскільки знаходяться всередині МП і, по суті, є понадшвидкодіючими (ПОЗП). Позначаються вони буквами B, C, D, E, H, L і, як видно з рисунка 3.1, об'єднані попарно BC, DE, HL, що дозволяє обробляти операнди довжиною як 8 біт, так і 16 біт. Остання можливість використовується, якщо необхідно організувати збереження в регістрі адреси пам'яті або виконати обчислення над 2–х байтовими операндами.

Лічильник команд (програмний лічильник – PC) – 16–розрядний регістр, призначений для збереження адреси наступної за програмою команди.

Регістр команд (РК) – регістр, призначений для зберігання коду операції поточної виконуваної команди. Код операції завжди міститься в першому байті машинного коду команди. Програмно недоступний.

Регістр стану (прапорців) – регістр, що об'єднує ряд прапорців, значення яких залежить від виконання деяких команд в АЛП. В наведеному МП використовується 5 прапорців, а інші 3 біти 8–бітного регістра прапорців не використовуються. Формат регістра стану приведено на рисунку 3.2.

$7_р$	6	5	4	3	2	1	$0_р$
S	Z	0	AC	0	P	1	C

Рисунок 3.2 – Регістр стану МП i8080 (регістр прапорців)

Опис прапорців:

– S – прапорець знака, встановлюється в 1, якщо результат має від′ємне значення, і дорівнює старшому розряду коду результату;

– Z – прапорець нуля, який установлюється в 1, якщо після виконання операції усі розряди акумулятора мають нульове значення;

– C – прапорець перенесення/запозичення, встановлюється в 1, якщо при виконанні операції додавання відбулося перенесення з 7–го розряду в неіснуючий 8–ий або при відніманні відбулася позика одиниці з неіснуючого 8–го розряду;

– AC – прапорець допоміжного перенесення, встановлюється в 1, якщо відбулося перенесення з 3–го в 4–й розряд або позика з 4–го в 3–й розряд. Цей прапорець використовується спеціальною командою під час корекції результату додавання чисел в упакованому BCD – форматі;

– P – прапорець парності, встановлюється в 1, якщо результат містить парну кількість одиниць.

Покажчик стека (SP) – 16–розрядний регістр, який призначений для адресації комірок стекової пам'яті. В ньому завжди зберігається адреса верхівки стека, останньої комірки стекової пам'яті, куди записана корисна інформація. Запис чергового байта в стек ведеться з декрементом регістра–покажчика стека в МП і8080 та з інкрементом в мікроконтролері МК51.

Регістри тимчасового зберігання: фіксатор A, регістр тимчасового зберігання, W і Z, програмно – недоступні, використовуються для проміжного збереження операндів і адрес під час вибірки команд з пам'яті та їх виконання.

Буферні регістри – використовуються для узгодження МП–ра з СШ, програмно недоступні. Буфер шини даних – двонаправлений, а буфер шини адреси – однонаправлений.

Керуючий адресний регістр складається з:

- регістра адреси пам'яті (РА);
- схеми інкремента–декремента адреси (ІДА).

Регістр адреси пам'яті – 16–розрядний, приймає і зберігає адресу від кожного 16–розрядного регістра. Вихід РА через схему ІДА зв'язано з буфером шини адреси і з входом кожного 16–розрядного регістра, який містить адресу.

Схема інкремента–декремента адреси зв'язана з виходом регістра адреси пам'яті і призначена для зміни вмісту регістра адреси пам'яті на ±1 (для формування поточних адрес пам'яті програм і стека). В процесі вибірки коду операції команди з пам'яті вміст регістрів РС і РА збігається. Після декодування коду операції поточної команди у програмний лічильник записується адреса коду операції наступної команди, а вміст РА змінюється у відповідності з виконуваною командою.

Блок десяткової корекції призначено для корекції результату після додавання чисел в упакованому BCD – форматі. Для корекції використовується спеціальна команда (DAA).

Мультиплексор забезпечує введення інформації з декількох паралельних каналів в один послідовний під час передачі даних, наприклад, з блоку РЗП на внутрішню шину даних.

Демультиплексор служить для прийому інформації з одного послідовного каналу, наприклад з внутрішньої шини даних, і виведення її на один з паралельних каналів, наприклад в один з РЗП.

Дешифратор команд призначено для декодування коду операції команди, який знаходиться в першому байті команди. За результатом декодування виробляється потрібна послідовність сигналів керування, що призводить до зчитування наступних байтів команди і/або власне до її виконання.

Схема керування і синхронізації – є однією з найважливіших вузлів МП, що забезпечує правильну послідовність подій в МП. Після зчитування команди з пам‘яті та її декодування пристрій керування генерує послідовність сигналів, необхідну для виконання команди. Робота схеми мікропрограмується під час виготовлення мікропроцесора. Схема керування містить маленький мікропроцесор всередині мікропроцесора. Одна з основних функцій схеми керування – організація зв'язку МП–ра з зовнішніми пристроями і системною шиною.

3.1.2 Структура мікропроцесора i8086

При розробці МП–ра i8086 (рисунок 3.3) застосовано нові в порівнянні з МП i8080 архітектурні рішення, до яких, зокрема, відноситься розділення функцій сполучення з шиною та виконання команд. Відповідно до цього структуру МП i8086 можна умовно розділити на дві частини: виконавчий блок (ВБ) і блок сполучення з шиною (БСШ).

Виконавчий блок призначено для зберігання даних, що обробляються і виконання операцій над ними. Він включає вісім 16–розрядних регістрів загального призначення (РЗП), 16–розрядний арифметико–логічний пристрій (АЛП), регістр прапорців (ознак) – RF, регістр команд і пристрій керування і синхронізації. Регістри загального призначення (РЗП) є надоперативним запам‘ятовуючим пристроєм (НОЗП). Вони повністю доступні програмісту, можуть використовуватися довільно, але в багатьох командах вони мають специфічне призначення. У відповідності з цим РЗП розбиваються на дві групи.

Перша група регістрів: AX, BX, CX, DX в арифметичних і логічних командах може використовуватися для зберігання операндів і результатів виконання операцій.

Переважніше для цієї мети повинен застосовуватися регістр AX – акумулятор.

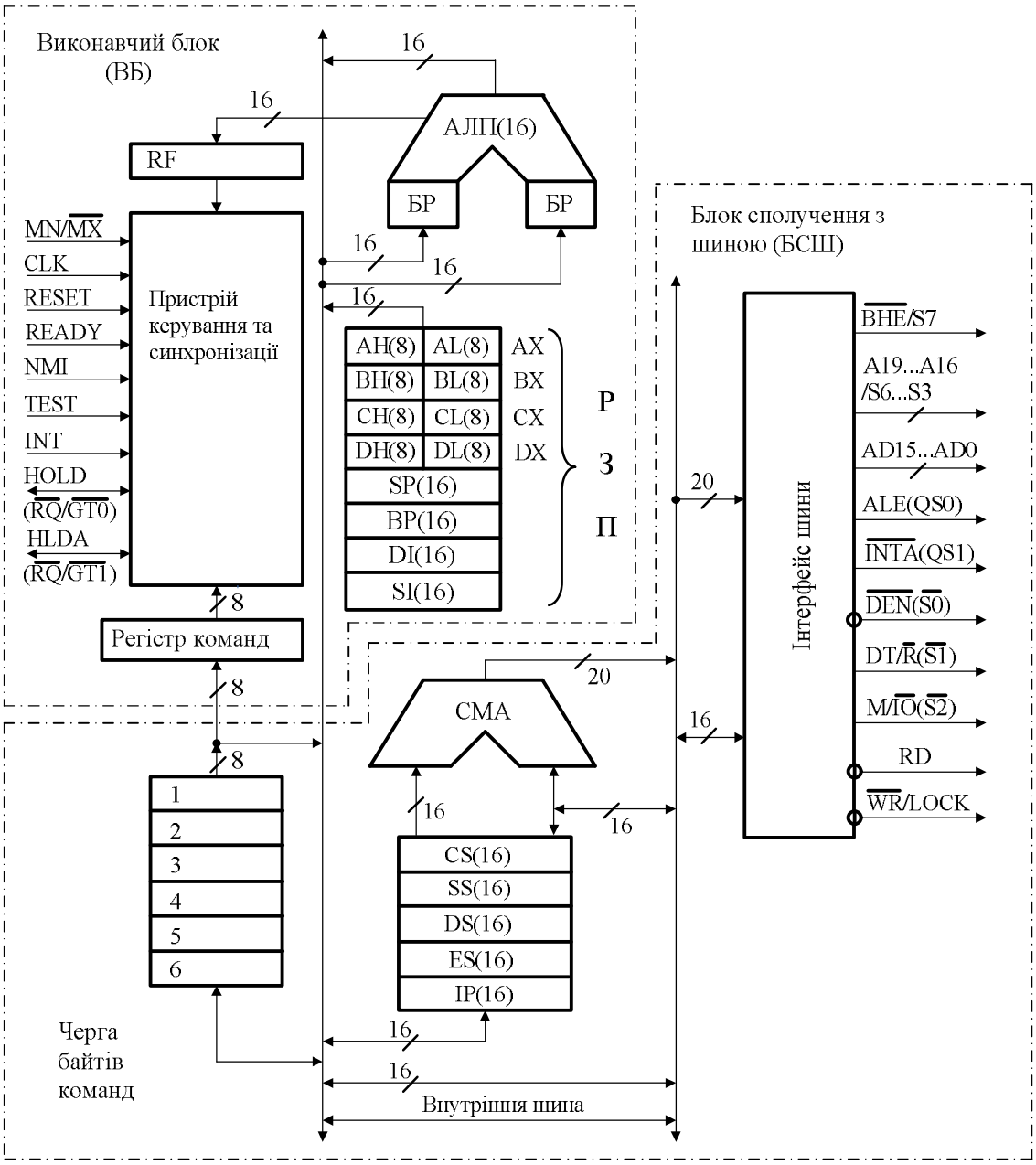


Рисунок 3.3 – Структура МП i8086

У регістрі BX – регістрі бази, як правило, зберігається початкова адреса структури даних, що обробляється, яка розташована в сегменті даних DS. Регістр CX в ряді команд використовується як лічильник при організації циклічних обчислень. Регістр даних DX в командах множення і

ділення зберігає половину 32–розрядних операндів, а в командах введення/виведення може містити адресу зовнішніх пристроїв. У процесорі передбачена можливість роботи з 8–розрядними "половинками" названих регістрів. Регістр AX розпадається на два: регістр молодшого байта акумулятора AL і регістр старшого байта акумулятора AH.

Аналогічно регістр BX– розпадається на два регістри BL і BH, регістр CX – на два регістри CL і CH, а регістр DX – на регістри DL і DH.

Друга група регістрів складається з двох 16–розрядних регістрів–покажчиків SP і BP та двох 16–розрядних індексних регістрів SI і DI. Регістри–покажчики використовуються для роботи зі стековою пам'яттю (стековим сегментом пам'яті): покажчик стека SP містить адресу вершини стека, а покажчик бази BP зберігає адресу початкового елемента структури даних у стековому сегменті (SS).

Індексні регістри використовуються для доступу до елементів рядків даних: регістр індексу джерела SI зберігає зсув адреси для читання елемента рядка з сегмента даних DS, регістр індексу приймача DI – зсув адреси для запису елемента рядка в екстра–сегмент ES.

Арифметико–логічний пристрій призначено для виконання арифметичних, логічних операцій та операцій зсуву над 8– і 16–розрядними операндами – даними, що беруть участь в операціях. Операнди надходять в АЛП з регістрів загального призначення і/або пам'яті. Результат операції повертається в регістр загального призначення чи в пам'ять. АЛП крім схем для виконання операцій містить два програмно недоступних регістри для тимчасового збереження операндів (БР).

Регістр прапорців RF (рисунок 3.4) містить 9 ознак (прапорців), що характеризують стан програми, виконуваної мікропроцесором, та керують мікропроцесором.

Шість прапорців встановлюються відповідно до результату виконаної в АЛП чергової операції .

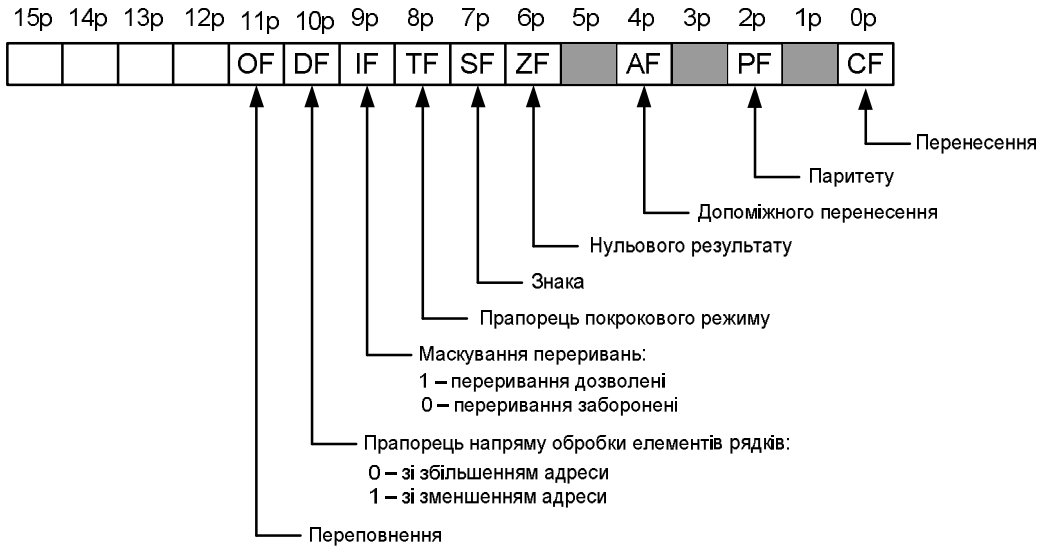


Рисунок 3.4 – Формат регістра прапорців мікропроцесора i8086

Якщо розрядність оброблюваних кодів позначити n, а нумерацію розрядів здійснити справа наліво від 0 до (n–1), то:

- *прапорець перенесення CF* встановлюється в 1, якщо в результаті виконання операції виникло перенесення з (n–1)–го розряду результату в неіснуючий більш старший n–й розряд або була потрібна позика з неіснуючого n–го розряду;
- *прапорець паритету PF* доповнює код результату до непарного числа одиниць, тобто встановлюється в 1, якщо кількість одиниць у коді результату парна;
- *прапорець допоміжного перенесення AF* встановлюється в 1, якщо при виконанні операції виникло перенесення з 3–го розряду результату в 4–й або виконалася позика з 4–го розряду в 3–й. Прапорець AF найбільш часто використовується для виконання команд десяткової корекції результату при виконанні операцій над BCD–числами;

– *прапорець нульового результату ZF* приймає одиничне значення, якщо всі розряди результату нульові. Якщо хоча б один розряд результату відмінний від нуля, то прапор ZF встановлюється в нуль;
– *прапорець знака SF* встановлюється рівним старшому (n–1)–му розряду кода результату. При виконанні операцій над числами зі знаком прапорець SF відповідає знаку результату. Якщо результат більший за нуль або дорівнює нулю (додатний), то SF встановлюється в 0, якщо результат менший за нуль (від'ємний), те SF буде встановлено у 1;
– *прапорець переповнення OF* встановлюється в 1, якщо в результаті виконання арифметичних операцій, над числами зі знаком відбувається переповнення розрядної сітки, тобто результат не укладається в діапазон, виділений n – розрядному числу зі знаком (перенесення з (n – 2) – го розряду в (n – 1) – й не збігається в перенесенням з (n – 1) – го розряду в неіснуючий n – й).

Три прапорці, що залишилися, характеризують стан пристрою керування мікропроцесора:

– *прапорець напряму передачі DF* визначає спосіб зміни адрес джерел – приймачів операндів у командах роботи з рядками даних. Якщо DF = 0*,* то адреси збільшуються (автоінкремент адреси), якщо DF = 1, то адреси зменшуються (декремент адреси);
– *прапорець дозволу переривання IF*, що скидається в 0, забороняє (маскує) обробку зовнішнього переривання, а встановлений в 1, дозволяє її;
– *прапорець покрокового режиму TF*, що знаходиться в одиничному стані, задає спеціальний режим виконання програми, у якому після обробки кожної команди генерується програмне переривання.

В регістр команд поміщається код операції команди, що підлягає виконанню (програмно недоступний). Цей код декодується дешифратором, що дозволяє ідентифікувати тип виконуваної команди.

Пристрій керування і синхронізації забезпечує функціонування окремих вузлів і мікропроцесора в цілому. Він аналізує вхідні і генерує вихідні керуючі сигнали, керує обміном інформацією в самому МП та виконанням команд. Після декодування коду операції чергової виконуваної команди пристрій керування формує послідовність керуючих сигналів, що забезпечують виконання цієї команди.

Блок сполучення із шиною (БСШ) забезпечує обмін інформацією між МП і зовнішніми по відношенню до нього приймачами або джерелами даних (пам'ять, ПВВ і т.ін.).

До складу БСШ входять: група сегментних регістрів, регістр адреси команди, черга байтів команд, суматор адреси та інтерфейс шини (рисунок 3.3).

Сегментні регістри призначено для збереження початкових (базових) логічних адрес сегментів пам'яті. Процесор містить чотири 16–розрядних сегментних регістри, що відповідає чотирьом поточним сегментам пам'яті:

– *регістр сегмента команд CS:* визначає сегмент пам'яті, у якому зберігається виконувана програма (містить логічну адресу початку поточного сегмента CS);

– *регістр сегмента даних DS:* задає сегмент пам'яті, що містить оброблювані дані (містить логічну адресу початку поточного сегмента даних DS);

– *регістр сегмента стека SS:* визначає сегмент пам'яті, доступ до елементів якого можливий за допомогою стекової адресації (містить логічну адресу початку поточного сегмента стека SS);

– *регістр екстра–сегмента ES:* задає додатковий сегмент, у якому також можуть зберігатися оброблювані дані (містить логічну адресу початку поточного додаткового сегмента ES).

Регістр адреси команди IP (покажчик команд) зберігає 16–розрядну адресу (зміщення) чергової команди, що повинна витягатися із сегмента команд пам'яті, тобто є зсувом відносно початку поточного сегмента команд.

Шість 8–розрядних регістрів черги команд складають внутрішню проміжну пам'ять програми зі швидким доступом, яку призначено для збереження послідовності команд, що виконуються.

Суматор адреси (СМА) призначено для формування 20–розрядної фізичної адреси зовнішньої пам'яті. Для цього використовуються дві 16–розрядні логічні адреси: початку сегмента та зміщення в сегменті (зсуву відносно початку сегмента).

Інтерфейс шини забезпечує обмін інформацією із системною шиною і формує керуючі сигнали.

У МП і8086 передбачена асинхронна паралельна робота ВБ та БСШ. У ході виконання програми БСШ вибирає з пам'яті послідовність команд і розміщує їх у черзі байтів команд (до 6 байт). Після завершення виконання попередньої операції в ВБ чергова команда вибирається з молодших регістрів черги і направляється на обробку в ВБ. Виконавчий блок виконує запропоновану операцію, а блок сполучення із шиною завантажує чергу новою командою чи командами, у залежності від кількості регістрів черги, що звільнилися. Якщо ВБ потрібно прочитати дані з пам'яті або записати їх у пам'ять, то процес вибірки команд переривається і БСШ виконує цикл читання чи запису даних. Після цього відновляється вибірка команд, що продовжується до заповнення черги. Таке перекриття етапів вибірки і виконання команд практично цілком виключає час читання команд із

сумарного часу виконання програми, підвищуючи тим самим продуктивність МП. Якщо ВБ виконує команду, що передає керування іншій комірці пам'яті команд (наприклад, команди CALL, JMP), то стара черга команд стирається та БСШ знову встановлює чергу команд, починаючи заповнювати її з нового значення. У більшості випадків ВБ може відразу одержувати команду з черги, а не чекати, доки вона буде вибрана з пам'яті.

3.1.3 Структура мікроконтролера AT89C51

3.1.3.1 Загальні відомості

МК (рисунок 3.5) складається з таких основних функціональних вузлів: блока керування і синхронізації; блока арифметико–логічного пристрою АЛП (англ. **Arifmetic–Logic Unit**); резидентної пам'яті даних РПД (англ. **Resident Data Memory**) об'ємом 128 байт; резидентної пам'яті програм РПП (англ. **Resident Program Memory**) об'ємом 4 Кбайт; блока переривань (англ. **Interrupt System**), таймерів (англ. **Timer**); послідовного порту (англ. **Serial Port**); чотирьох паралельних портів введення/виведення (англ. **Parallel Port**), які програмуються; схеми десяткової корекції вмісту акумулятора (СДКА); внутрішнього генератора тактових імпульсів ГТІ (англ. **Oscillator**); резидентної шини даних РШД (англ. **Resident Data Buss**) і групи регістрів:

- **А** – акумулятор;
- **В** – регістр розширення акумулятора;
- **Т1, Т2** – регістри тимчасового збереження операндів;
- **РСП (PSW)** – регістр стану програми (прапорців);
- **РК (IR)** – регістр команд;
- **ЛК (PC)** – лічильник команд (програмний лічильник);

- **РПД (DPTR)** – регістр–покажчик даних, що складається з 2–х частин: молодшої – **DPL** і старшої – **DPH**;
- **РПС (SP)** – регістр–покажчик стека;
- **РА (RAR)** – регістр адреси;
- **РРТЛ (TMOD)** – регістр режимів таймерів/лічильників;
- **РКСТ (TCON)** – регістр керування–статусу таймерів/лічильників;
- **РКПП (SCON)** – регістр керування приймачем–передавачем послідовного порту;
- **SBUF (буфер ПРМ і буфер ПД)** – буфери приймача і передавача послідовного порту;
- **РМП (IE)** – регістр масок переривань;
- **РП (IP)** – регістр пріоритетів переривань;
- **РКП (PCON)** – регістр керування потужністю споживання енергії від джерела живлення.

Структуру МК наведено на рисунку 3.5 [15,16].

3.1.3.2 Блок керування та синхронізації

Блок керування та синхронізації призначено для формування синхронізуючих і керуючих сигналів, що забезпечують координацію спільної роботи блоків МК у всіх припустимих режимах його роботи.

До складу блока керування входять: пристрій формування часових інтервалів, логіка введення/виведення, регістр команд, дешифратор команд, програмована логічна матриця (ПЛМ) і логіка керування мікроконтролером.

Пристрій формування часових інтервалів призначено для формування і видачі внутрішніх синхросигналів станів, фаз і циклів.

Кількість машинних циклів (англ. **machine cycle**) визначає тривалість виконання команд.

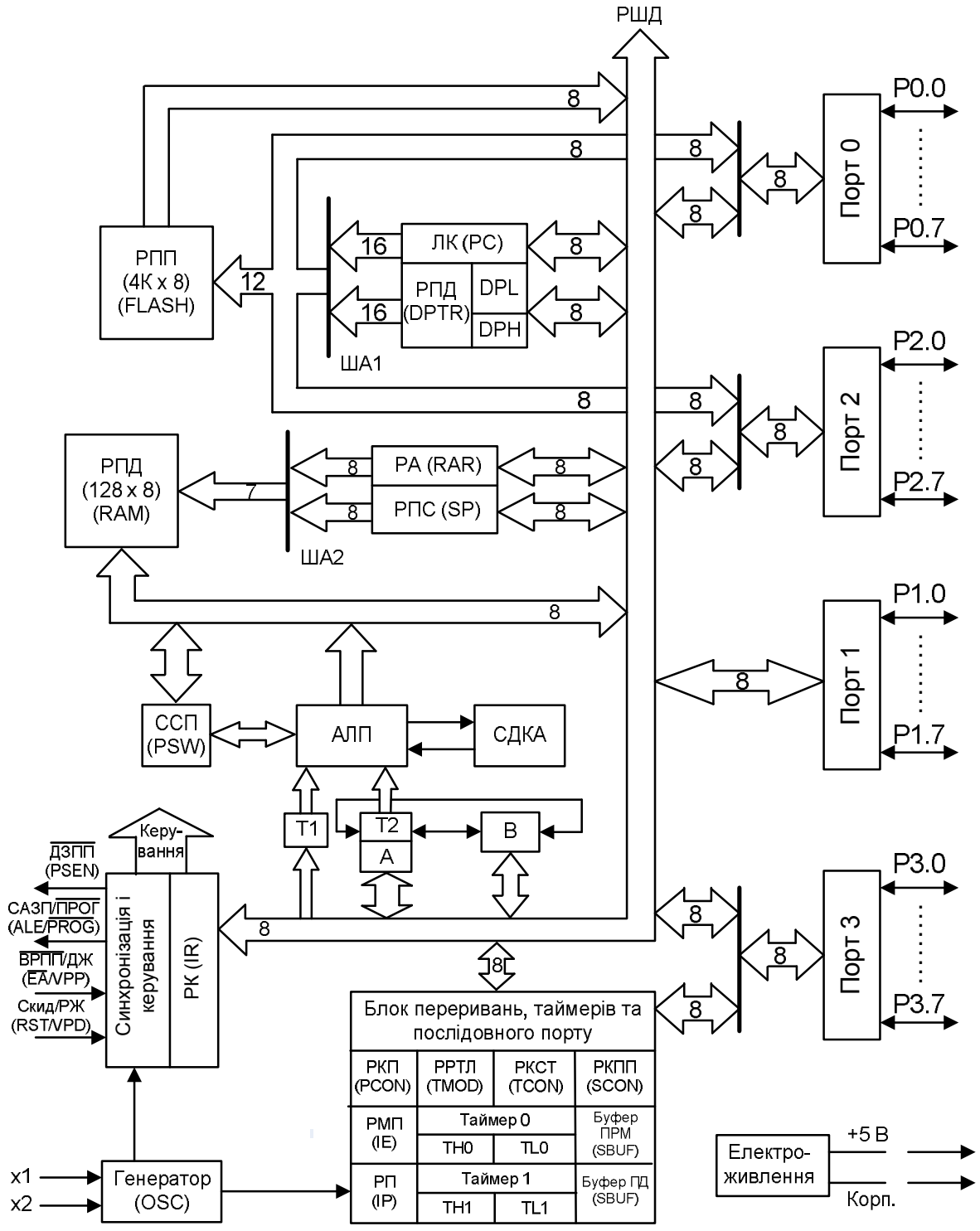


Рисунок 3.5 – Структурна схема МК–ра AT89C51

Практично всі команди МК виконуються за один або два машинних цикли, крім команд множення MUL A, B і ділення DIV A, B, тривалість

виконання яких складає чотири машинних цикли. Машинний цикл має фіксовану тривалість і містить шість станів (англ. **state**) S1…S6, кожен з яких складається з двох часових інтервалів, які називаються фазами (англ. **phase**) P1 і P2.

Тривалість фази дорівнює періоду тактового генератора, що є первинним сигналом синхронізації МК. Сигнал синхронізації формується або вмонтованим тактовим генератором МК (при підключенні до його виводів 18 (BQ2) і 19 (BQ1) кварцового резонатора або LC–ланцюжка), або зовнішнім джерелом тактових сигналів.

Схему підключення кварцового резонатора до виводів BQ2 і BQ1 показано на рисунку 3.6.

Рисунок 3.7 ілюструє формування машинних циклів в МК. Всі машинні цикли однакові, складаються з 12 періодів тактового сигналу, починаються фазою S1 P1 і закінчуються фазою S6 P2.

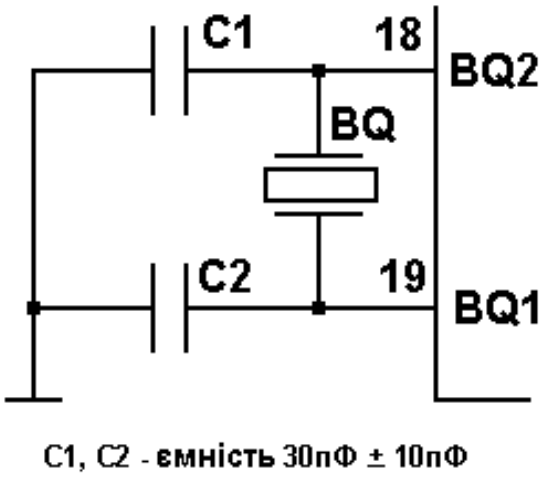


Рисунок 3.6 – Схема підключення кварцового резонатора

Двічі за один машинний цикл формується сигнал дозволу фіксації адреси ALE (англ. **Address Latch Enable),** що видається на однойменний вивід. Якщо, наприклад, тактова частота f_{BQ} = 12 МГц, то тривалість машинного циклу $T_{МЦ}$ = 1 мкс.

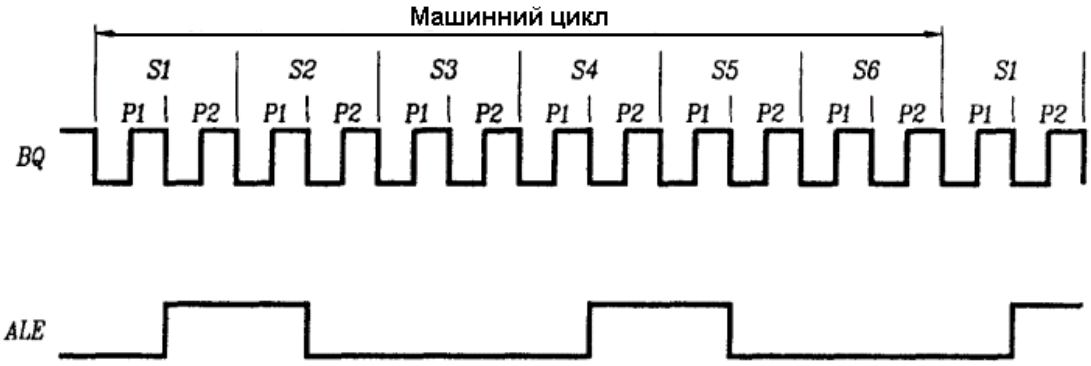


Рисунок 3.7 – Діаграма формування машинних циклів МК

В **регістр команд (РК)** пересилається з пам'яті програм код операції чергової команди, що виконується. Дешифратор команд декодує код операції та ідентифікує тип команди, що підлягає виконанню.

Після цього з програмованої логічної матриці (ПЛМ) викликається послідовність керуючих сигналів для виконання команди.

3.1.3.3 Блок арифметико–логічного пристрою

Арифметико–логічний пристрій АЛП являє собою паралельний 8–розрядний пристрій, що забезпечує виконання арифметичних і логічних операцій, а також операцій логічного зсуву, скидання, встановлення і т.ін.

Блок АЛП складається з регістрів тимчасового збереження операндів Т1, Т2, ПЗП констант, суматора, додаткового регістра (регістра В), акумулятора, регістра стану програми.

Регістри тимчасового збереження операндів Т1, Т2 – 8–розрядні регістри, призначені для прийому і збереження операндів на час виконання операцій над ними. Програмно не доступні.

ПЗП констант забезпечує формування коду коригування при двійково–десятковому представленні даних, коду маски при бітових операціях і коду констант.

Паралельний 8–розрядний суматор являє собою схему комбінаційного типу з послідовним перенесенням, призначену для виконання арифметичних операцій додавання, віднімання і логічних операцій додавання, множення, нерівнозначності тощо.

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

Акумулятор являє собою 8–розрядний регістр, призначений для прийому і збереження результату, отриманого при виконанні арифметично–логічних операцій або операцій пересилання.

Регістр стану програми (PSW) призначено для збереження інформації про стан АЛП при виконанні програми. Позначення розрядів регістра PSW і призначення його розрядів наведено відповідно в таблицях 3.2 і 3.3.

Таблиця 3.2 – Позначення розрядів регістра PSW

Біти	7	6	5	4	3	2	1	0
Позначення	CY	AC	F0	RS1	RS0	0V	–	P

3.1.3.4 Резидентна пам'ять даних

Резидентна пам'ять даних РПД призначена для прийому, збереження і видачі інформації, яка використовується в процесі виконання програми.

Таблиця 3.3 – Призначення окремих розрядів регістра PSW

Біти	Найменув.	Призначення бітів	Доступ до біта
7	CY	Прапорець перенесення. Змінюється під час виконання деяких арифметичних і логічних інструкцій.	апаратно або програмно
6	AC	Прапорець додаткового перенесення. Апаратно встановлюється/скидається під час виконання інструкцій додавання або віднімання для вказівки перенесення або позики в біті 3 при утворенні молодшої тетради результату (D0…D3).	апаратно або програмно
5	F0	Прапорець 0. Прапорець стану, обумовлений користувачем.	програмно
4	RS1	Покажчик банку робочих регістрів РПД	програмно
3	RS0	Покажчик банку робочих регістрів РПД	програмно
	RS1 RS0		
	0 0	Банк 0 з адресами (00H … 07H)	
	0 1	Банк 1 з адресами (08H … 0FH)	
	1 0	Банк 2 з адресами (10H … 17H)	
	1 1	Банк 3 з адресами (18H … 1FH)	
2	0V	Прапорець переповнення. Апаратно встановлюється/скидається під час виконання арифметичних інструкцій над числами зі знаком для вказівки стану переповнення	апаратно або програмно
1	–	Резервний. Містить тригер, доступний за записом ("0" і "1") і читанням, який можна використовувати	
0	P	Біт парності. Апаратно скидається/ встановлюється в кожному циклі інструкцій для вказівки парної/непарної кількості розрядів акумулятора, що знаходяться в стані "1".	апаратно або програмно

Пам'ять даних ділиться на внутрішню (резидентну) – РПД і зовнішню – ЗПД. До складу вузла, що називається РПД на рисунку 3.5, входить ОЗП (RAM) ємністю 128 байт і дешифратор адреси. Керують роботою РПД два

регістри: РА (RAR) – регістр адреси; РПС (SP) – покажчик стека.

Регістр адреси ОЗП (РА) призначено для прийому і збереження адреси комірки пам'яті, що вибирається за допомогою дешифратора і може містити як біт, так і байт інформації.

ОЗП являє собою 128 8–розрядних регістрів, призначених для прийому, збереження і видачі різної інформації. 16 із цих регістрів припускають побітову адресацію.

Покажчик стека являє собою 8–розрядний регістр, призначений для прийому і збереження адреси комірки стека. При виконанні команд LCALL, ACALL вміст покажчика стека збільшується на 2. При виконанні команд RET, RETI вміст покажчика стека зменшується на 2. При виконанні команди PUSH direct вміст покажчика стека збільшується на 1. При виконанні команди POP direct вміст покажчика стека зменшується на 1. Після скидання в покажчику стека встановлюється адреса 07H, що відповідає початку стека з адресою 08H.

Більш докладно організація пам'яті даних мікропроцесорних систем, що використовують дану МК, розглянута в окремому розділі.

На рисунку 3.8 наведено розподіл адресного простору РПД і область бітів, що адресуються прямо.

3.1.3.5 Резидентна пам'ять програм

Резидентну пам'ять програм РПП призначено для збереження програм. Вона має окремий від пам'яті даних адресний простір об'ємом до 64 Кбайт, причому, для мікросхеми AT89C51 частина пам'яті програм з адресами 0000H – 0FFFH розташована на кристалі МК.

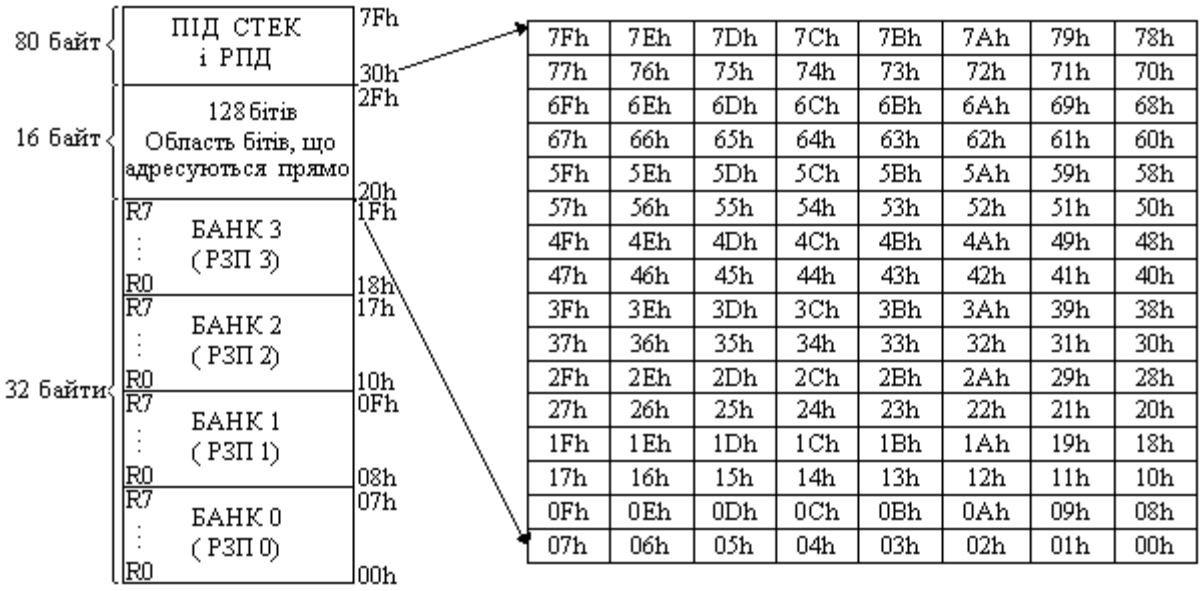


Рисунок 3.8 – Розподіл адресного простору РПД і область бітів, які адресуються прямо

Пам'ять програм, що розташована на кристалі (РПП), для мікросхеми AT89C51 складається з 12–розрядного дешифратора та FLASH–ПЗП ємністю 4Кх8 біт. Запис програм у ПЗП відбувається під час розробки системи.

Якщо на вивід МК ВРПП (DEMA) подається напруга живлення U_{CC} (логічна 1), то звертання до зовнішньої пам'яті програм відбувається автоматично при видачі лічильником команд адреси, що перевищує 0FFFH. Якщо адреса знаходиться в межах 0000H–0FFFH, звертання відбувається до пам'яті програм, розташованої на кристалі (резидентної пам'яті програм).

Якщо на вивід МК ВРПП подається сигнал "лог. 0", то внутрішня пам'ять програм відключається, і, починаючи з адреси 0000H, всі звертання виконуються до зовнішньої пам'яті програм.

Для формування поточної 16–розрядної адреси пам'яті програм служить лічильник команд (програмний лічильник) – ЛК (PC). 12 молодших розрядів цього регістра використовуються при адресації комірок РПП об'ємом 2^{12} = 4 Кбайт.

Більш докладно організація пам'яті програм мікропроцесорних

систем, що використовують даний МК, розглядається в окремому розділі.

3.1.3.6 Блок переривань

МК має систему переривань із п'ятьма векторами (адресами підпрограм обробки переривань) і двома рівнями пріоритетів. Джерелами переривань являються: два зовнішніх переривання, що надходять через порт 3; два переривання від переповнення таймерів/лічильників T/CNT0 і T/CNT1 та переривання при завершенні передавання або прийому даних при обміні через послідовний порт.

Для програмування і керування роботою системи переривань служать два регістри: РМП (IE) – регістр масок переривань і РП (IP) – регістр пріоритетів переривань, а також чотири молодших біти регістра РКСТ (таблиці 3.4, 3.5).

Регістр пріоритетів переривань (IP) призначено для встановлення рівня пріоритету переривання для кожного з п'ятьох джерел переривань. Позначення розрядів регістра IP показано в таблиці 3.4, а їхнє призначення зазначено нижче.

Таблиця 3.4 – Позначення розрядів регістра IP

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

PX0 (IP0) – встановлення рівня пріоритету переривання від зовнішнього джерела $\overline{\text{INT0}}$.

PT0 (IP1) – встановлення рівня пріоритету переривання від T/C0.

PX1 (IP2) – встановлення рівня пріоритету переривання від зовнішнього джерела $\overline{\text{INT1}}$.

PT1 (IP3) – встановлення рівня пріоритету переривання від T/C1.

PS (IP4) – встановлення рівня пріоритету переривання від послідовного порту.

X (IP7,IP6,IP5) – резервний розряд.

Наявність у розряді IP сигналу "лог. 1" встановлює для відповідного джерела високий рівень пріоритету, а наявність у розряді IP сигналу "лог. 0" – низький рівень пріоритету. При читанні резервних розрядів відповідні лінії магістралі даних не визначені. Користувач не повинен записувати "1" у резервні розряди, тому що вони зарезервовані для подальшого розширення сімейства МК51.

Регістр дозволу переривань (IE) призначено для дозволу або заборони переривань від відповідних джерел. Позначення розрядів регістра IE показано в таблиці 3.5, а їхнє призначення зазначено нижче.

Таблиця 3.5 – Позначення розрядів регістра IE

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

EA (IE7) – керування всіма джерелами переривань одночасно. Якщо EA = 0, то всі переривання заборонено. Якщо EA = 1, то переривання можуть бути дозволені індивідуальними дозволами EX0, ET0, EX1, ET1, ES.

X (IE6, IE5) – резервний розряд.

ES (IE4) – керування перериванням від послідовного порту. ES = 1 – дозвіл. ES = 0 – заборона.

ET1 (IE3) – керування перериванням від T/C1. ET1 = 1 – дозвіл. ET1 = 0 – заборона.

EX1 (IE2) – керування перериванням від зовнішнього джерела $\overline{\text{INT1}}$.

EX1 = 1 – дозвіл.

EX1 = 0 – заборона.

ET0 (IE1) – керування перериванням від T/C0. ET0 = 1 – дозвіл. ET0 = 0 – заборона.

EX0 (IE0) – керування перериванням від зовнішнього джерела $\overline{\text{INT0}}$.

EX0 = 1 – дозвіл.

EX0 = 0 – заборона.

При зчитуванні резервних розрядів відповідні лінії магістралі не визначені. Користувач не повинний записувати сигнал "лог. 1" у резервні розряди, тому що вони зарезервовані для подальшого розширення сімейства МК51.

Блок переривань містить також **схему логічної обробки прапорців переривань**, яка здійснює пріоритетний вибір запиту переривання, скидання його прапорця й ініціює формування апаратно реалізованої команди переходу на підпрограму обслуговування переривання LCALL.

Схема формування вектора переривання формує двобайтні адреси підпрограм обслуговування переривання в залежності від джерела переривання, що наведені в таблиці 3.6.

Таблиця 3.6 – Джерела переривань і адреси підпрограм, що їх обслуговують

Джерело переривання	Вектор переривання
Зовнішнє переривання $\overline{INT0}$	0003H
Таймер/лічильник Т/С0	000BH
Зовнішнє переривання $\overline{INT1}$	0013H
Таймер/лічильник Т/С1	001BH
Послідовний порт	0023H

3.1.3.7 Блок таймерів/лічильників

Таймери/лічильники (Т/Л) призначено для підрахунку зовнішніх подій, для одержання програмно керованих часових затримок і виконання функцій, які задають часові інтервали МК.

До складу блока Т/Л входять:

- два 16–розрядних регістри Т/Л0 і Т/Л1;
- 8–розрядний регістр режимів Т/Л (TMOD);
- 8–розрядний регістр керування (TCON);

– схема інкремента;
– схема фіксації $\overline{INT0}$, $\overline{INT1}$, T0, T1;
– схема керування прапорцями;
– логіка керування Т/Л.

Два 16–розрядних регістри Т/С0 і Т/С1 виконують функцію зберігання вмісту лічби. Кожний із них складається з пари восьмирозрядних регістрів, відповідно TH0, TL0 і TH1, TL1. Причому регістри TH0, TH1 – старші, а регістри TL0, TL1 – молодші 8 розрядів. Кожний із восьмирозрядних регістрів має свою адресу і може бути використаний як РЗП, якщо Т/Л не використовуються (біт TR0 для Т/Л0 і біт TR1 для Т/Л1 у регістрі керування TCON дорівнюють "0").

Код величини початкової лічби заноситься в регістри Т/Л програмно. В процесі підрахунку вміст регістрів Т/Л інкрементується. Ознакою закінчення лічби, як правило, являється переповнення регістра Т/Л, тобто перехід його вмісту зі стану "всі одиниці" у стан "усі нулі". Всі регістри TH0, TH1, TL0, TL1 доступні за читанням, і, при необхідності, контроль досягнення необхідної величини підрахунку може виконуватися програмно.

Регістр режимів Т/Л (TMOD) призначений для прийому і збереження коду, що визначає:

– один із 4–х можливих режимів роботи кожного Т/Л;
– роботу в якості таймерів або лічильників зовнішніх подій;
– керування Т/Л від зовнішнього виводу.

Позначення розрядів регістра TMOD наведено в таблиці 3.7, а призначення розрядів – у таблиці 3.8

Таблиця 3.7 – Позначення розрядів регістра TMOD

Біти	7	6	5	4	3	2	1	0
Позн.	GATE1	$C/\overline{T}1$	Ml.1	M0.1	GATE0	$C/\overline{T}0$	M1.0	M0.0

Таблиця 3.8 – Призначення розрядів регістра TMOD

<table>
<tr><th>Біти</th><th>Найменування</th><th>Призначення бітів</th><th>Примітка</th></tr>
<tr><td>0–1
4–5</td><td>M0 – M1</td><td>Визначають один із 4–х режимів роботи, окремо для Т/Л1 і Т/Л0
<table><tr><th>M1</th><th>M0</th><th>Режим</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>2</td></tr><tr><td>1</td><td>1</td><td>3</td></tr></table></td><td rowspan="3">Усі біти встановлюються програмно; біти 0–3 визначають режим роботи Т/Л0, біти 4–7 визначають режим роботи Т/Л1.</td></tr>
<tr><td>2, 6</td><td>C/$\overline{T}$ 0
C/$\overline{T}$ 1</td><td>Визначають роботу в якості:
C/T0, C/T1 = 0 – таймера
C/T0, C/T1 = 1 – лічильника</td></tr>
<tr><td>3, 7</td><td>GATE</td><td>Дозволяє керувати таймером від зовнішнього виводу ($\overline{INT0}$ – для Т/Л0, $\overline{INT1}$ – для Т/Л1).
GATE = 0 – керування заборонено
GATE = 1 – керування дозволено</td></tr>
</table>

При роботі в якості таймера вміст регістра Т/Л інкрементується в кожному машинному циклі, тобто Т/Л являється лічильником машинних циклів МК. Оскільки машинний цикл складається з 12 періодів частоти синхронізації МК f_{BQ}, то частота підрахунку в даному випадку дорівнює f_{BQ} /12.

При роботі Т/Л в якості лічильника зовнішніх подій вміст регістра Т/Л інкрементується у відповідь на перехід із стану "лог. 1" у "лог. 0" сигналу на рахунковому вході МК (вивід Т0 для Т/Л 0 і вивід Т1 для Т/Л 1). Рахункові входи апаратно перевіряються у фазі S5 P2 кожного машинного циклу.

Коли перевірки показують високий рівень на рахунковому вході в одному машинному циклі і низький рівень в іншому машинному циклі, регістр Т/Л інкрементується. Нове (інкрементоване) значення заноситься в регістр Т/Л у фазі S3 P1 машинного циклу, що безпосередньо іде за тим, у

якому був виявлений перехід із стану "лог. 1" у "лог. 0" на рахунковому вході МК. Оскільки для розпізнавання такого переходу потрібно два машинних цикли (24 періоди частоти синхронізації МК f_{BQ}), то максимальна частота підрахунку Т/Л в режимі лічильника дорівнює
f_{BQ} / 24.

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

Регістр керування (TCON) призначено для прийому і збереження коду керуючого слова. Позначення розрядів регістра TCON наведено в таблиці 3.9, а призначення розрядів – у таблиці 3.10.

Таблиця 3.9 – Позначення розрядів регістра TCON

Біти	7	6	5	4	3	2	1	0
Позначення	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

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

Прапорці TF0 і TF1 програмно доступні і можуть бути встановлені/скинуті програмою. Використовуючи цей механізм, переривання по TF0 і TF1 можуть бути викликані (встановлення TF) і скасовані (скидання TF) програмою.

Таблиця 3.10 – Призначення розрядів регістра TCON

Біти	Наймен ування	Призначення бітів	Примітка
6 4	TR1 TR0	Біти включення Т/Л, окремо для Т/Л0 і Т/Л1. TR = 0 – відключений, TR = 1 – включений.	Біти встановлюються і скидаються програмно. Доступні за читанням.
7 5	TF1 TF0	Прапорці переповнення Т/Л.	Біти скидаються і встановлюються апаратно і програмно. Доступні за читанням.
2 0	IT1 IT0	Біти, що визначають вид переривання за входами INT1, INT0. IT = 0 – переривання за рівнем (низьким), IT = 1 – переривання за фронтом (перехід із "1" у "0")	Біти встановлюються і скидаються програмно. Доступні за читанням
3 1	IE1 IE0	Прапорці запиту зовнішніх переривань за входами INT1, INT0.	Біти скидаються і встановлюються апаратно і програмно. Доступні за читанням
			Біти 4, 5 відносяться до Т/Л0; біти 6, 7 – до Т/Л1.Біти 0, 1 визначають зовнішні переривання за входом INT0, біти 2, 3 – за входом INT1.

Прапорці IE0 і IE1 встановлюються апаратно від зовнішніх переривань (відповідно входи INT0 і INT1) або програмно й ініціюють виклик програми обробки відповідного переривання. Скидання цих прапорців виконується апаратно при обслуговуванні переривання тільки в тому випадку, коли переривання було викликано за фронтом сигналу. Якщо переривання було викликано рівнем сигналу на вході INT0 (INT1), то скидання прапорця IE повинна виконувати програма обслуговування переривання, впливаючи на джерело переривання для зняття ним запиту.

Схема інкремента призначена:

— для збільшення на 1 у кожному машинному циклі вмісту регістрів Т/Л0, Т/Л1, для яких встановлений режим таймера і дозволений підрахунок;

— для збільшення на 1 вмісту регістрів Т/Л0, Т/Л1, для яких встановлений режим лічильника, дозволений підрахунок і на відповідному вході МК (Т0 для Т/Л0 і Т1 для Т/Л1) зафіксований рахунковий імпульс.

Схема фіксації INT0, INT1, T0, T1 являє собою чотири тригери. У кожному машинному циклі в момент S5 P2 (рисунок 3.7) у них запам'ятовується інформація з виводів INT0, INT1, T0, T1.

Схема керування прапорцями виробляє і знімає прапорці переповнення Т/Л і прапорці запитів зовнішніх переривань.

Логіка керування Т/Л синхронізує роботу регістрів Т/Л0 і Т/Л1 відповідно до запрограмованих режимів роботи і синхронізує роботу блока Т/Л з роботою МК.

Більш докладно режими роботи й особливості застосування таймерів/лічильників розглянуто в окремому розділі.

3.1.3.8 Блок послідовного порту (інтерфейсу)

Блок послідовного інтерфейсу призначено для організації введення/виведення послідовних даних.

До складу блока входять: буфер інтерфейсу, логіка керування інтерфейсом, регістр керування, буфер передавача, буфер приймача, приймач – передавач послідовного порту.

Буфер інтерфейсу забезпечує побайтовий обмін інформацією між внутрішньою (резидентною) магістраллю даних і шиною інтерфейсу.

Логіка керування інтерфейсом призначена для формування сигналів керування, що забезпечують чотири режими роботи послідовного інтерфейсу.

Регістр керування (SCON) призначено для прийому і зберігання коду восьмибітового слова, що керує послідовним інтерфейсом. Позначення розрядів регістра SCON наведено в таблиці 3.11.

Таблиця 3.11 – Позначення розрядів регістра SCON

Біти	7	6	5	4	3	2	1	0
Позначення	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Всі розряди регістра SCON програмно доступні по запису ("лог. 0" і "лог. 1") і зчитуванню.

Розряди SM0, SM1 визначають режим роботи інтерфейсу, як зазначено в таблиці 3.12.

Таблиця 3.12 – Вплив розрядів SM0, SM1 SCON на режим роботи інтерфейсу

SM0	SM1	Режим	Найменування	Швидкість передачі
0	0	0	Регістр зсуву	f_{BQ} / 12
0	1	1	8–бітовий універсальний асинхронний приймач–передавач (УАПП)	змінна, задається Т/Л1
1	0	2	9–бітовий УАПП	f_{BQ} / 64 або f_{BQ} / 32
1	1	3	9–бітовий УАПП	Змінна, задається Т/Л1

Інші біти регістра мають таке призначення:

– **SM2** – дозвіл багатопроцесорної роботи. У режимах 2 і 3 при SM2 = 1 прапорець RI не активізується, якщо дев'ятий прийнятий біт даних дорівнює "0". У режимі 1 при SM2 = 1 прапорець RI не активізується, якщо не прийнятий стоп–біт, рівний "1". В режимі 0 біт SM2 повинний бути встановлений у "0".

– **REN** – дозвіл прийому послідовних даних. Встановлюється і скидається програмою відповідно для дозволу і заборони прийому.
– **TB8** – дев'ятий біт даних, що передаються, у режимах 2 і 3. Встановлюється і скидається програмою.
– **RB8** – дев'ятий біт прийнятих даних у режимах 2 і 3. У режимі 1, якщо SM2 = 0, RB8 є прийнятим стоп–бітом. У режимі 0 біт RB8 не використовується.
– **TI** – прапорець переривання передавача. Встановлюється апаратно наприкінці часу видачі 8–го біта в режимі 0 або на початку стоп–біта в інших режимах. Скидається програмно.
– **RI** – прапорець переривання приймача. Встановлюється апаратно наприкінці часу прийому 8–го біта в режимі 0 або через половину інтервалу стоп–біта в режимах 1, 2, 3 при SM2 = 0. При SM2 = 1 див. опис для біта SM2.

Буфер передавача призначено для прийому з шини інтерфейсу паралельних даних і видачі їх на передавач послідовного порту.

Буфер приймача служить для прийому даних у паралельній формі від приймача послідовного інтерфейсу.

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

В усіх режимах роботи послідовного порту передача ініціюється будь–якою командою, що використовує SBUF як регістр призначення.

Приймач/передавач послідовного порту призначено для прийому послідовного потоку символів зі входу послідовного порту, виділення даних

і видачі їх у буфер приймача, а також для прийому паралельних даних із буфера передавача, перетворення їх у послідовний потік символів і видачі його на вихід послідовного порту.

Більш докладно режими роботи й особливості застосування послідовного інтерфейсу розглянуто в окремому розділі.

3.1.3.9 Паралельні порти введення/виведення

МК, що розглядається, містить 4 паралельних 8–розрядних порти введення/виведення дискретної інформації, що програмуються: P0, P1, P2, P3. Порти P0, P1, P2, P3 є двонаправленими портами введення/виведення і призначені для забезпечення обміну інформацією МК із зовнішніми пристроями, створюючи 32 лінії введення/виведення. Кожний з портів містить фіксатор–защіпку, що являє собою восьмирозрядний регістр, що має байтову і бітову адресацію для встановлення/скидання його розрядів за допомогою відповідних команд.

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

- P0 – 80H, при бітовій адресації 80H … 87H;
- P1 – 90H, при бітовій адресації 90H … 97H;
- P2 – A0H, при бітовій адресації A0H … A7H;
- P3 – B0H, при бітовій адресації B0H … B7H.

Крім роботи в якості звичайних портів введення/виведення лінії портів P0 – P3 можуть виконувати ряд додаткових (альтернативних) функцій, описаних нижче.

Через порт P0:

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

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

— задаються дані при програмуванні внутрішнього ПЗП, і читається вміст внутрішньої пам'яті програм.

Через порт P1:

— задається молодший байт адреси при програмуванні внутрішнього ПЗП і при читанні внутрішньої пам'яті програм.

Через порт P2:

— виводиться старший байт адреси A8 – A15 при роботі з зовнішньою пам'яттю програм і зовнішньою пам'яттю даних (для зовнішньої пам'яті даних – тільки при використанні команд MOVX A, @DPTR і MOVX @DPTR, A, що формують 16–розрядну адресу);

— задаються старші розряди (A8 – A11) адреси при програмуванні внутрішнього ПЗП і при читанні внутрішньої пам'яті програм.

Кожна лінія порту P3 має індивідуальну альтернативну функцію:

- P3.0 – RxD, вхід послідовного порту, призначений для введення послідовних даних у приймач послідовного порту;
- P3.1 – TxD, вихід послідовного порту, призначений для виведення послідовних даних із передавача послідовного порту;
- P3.2 – $\overline{\text{INT0}}$, використовується як вхід 0 зовнішнього запиту переривання;
- P3.3 – $\overline{\text{INT1}}$, використовується як вхід 1 зовнішнього запиту переривання;
- P3.4 – T0, використовується як вхід лічильника зовнішніх подій Т/Л 0;
- P3.5 – T1, використовується як вхід лічильника зовнішніх подій Т/Л 1;

– P3.6 – $\overline{WR}$, строб запису в зовнішню пам'ять даних, вихідний сигнал, що супроводжує виведення даних через порт P0 при використанні команд MOVX @Ri, A і MOVX @DPTR, A;
– P3.7 – $\overline{RD}$, строб читання із зовнішньої пам'яті даних, вихідний сигнал, що супроводжує введення даних через порт P0 при використанні команд MOVX A, @Ri і MOVX A, @DPTR.

Альтернативна функція будь–якої з ліній порту P3 реалізується тільки в тому випадку, якщо у відповідному до цієї лінії фіксаторі–защіпці міститься сигнал "лог. 1". Інакше на лінії порту P3 буде присутній сигнал "лог. 0".

3.1.3.10 Схема десяткової корекції акумулятора

АЛП МК дозволяє виконувати додавання двійково–десяткових даних в упакованому форматі (в 1 байт “упаковано” 2 десяткові цифри). При виконанні операції додавання таких чисел використовується звичайна команда “Додавання”, що підсумовує операнди за правилами двійкової арифметики і вміщує результат в акумулятор. Для виправлення можливої помилки (корекції вмісту акумулятора) застосовують команду десяткової корекції DA A, яка апаратно реалізується схемою десяткової корекції акумулятора.

3.1.3.11 Внутрішній тактовий генератор (OSC)

МК містить внутрішній тактовий генератор (рисунок 3.9), в якому BQ1 і BQ2 є відповідно входом і виходом підсилювача–інвертора, і який може бути включений у режим генератора при підключенні до виводів BQ1 і BQ2 резонатора або LC–ланцюжка (рисунок 3.10).

3.1.3.12 Резидентна шина даних

Мікроконтролер містить 8–розрядну внутрішню (резидентну) шину даних (РШД), через яку здійснюється обмін інформацією між різними частинами МК.

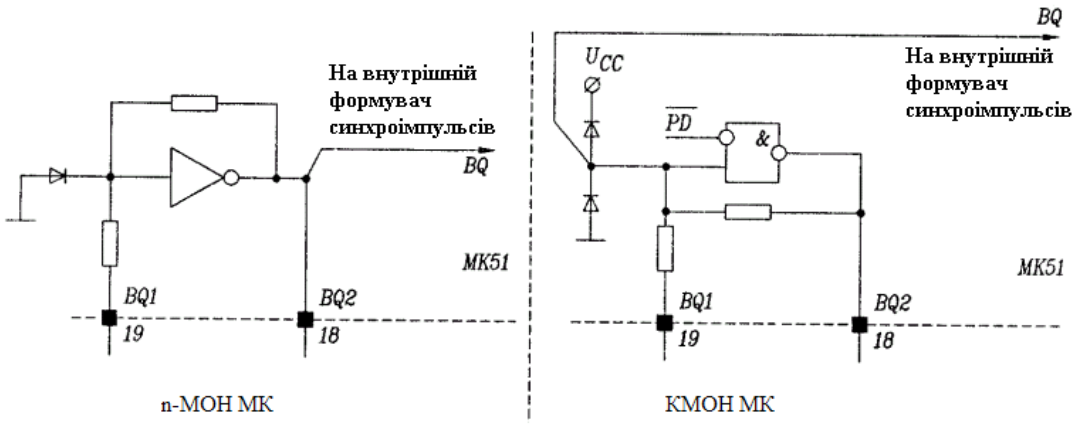


Рисунок 3.9 – Внутрішній тактовий генератор

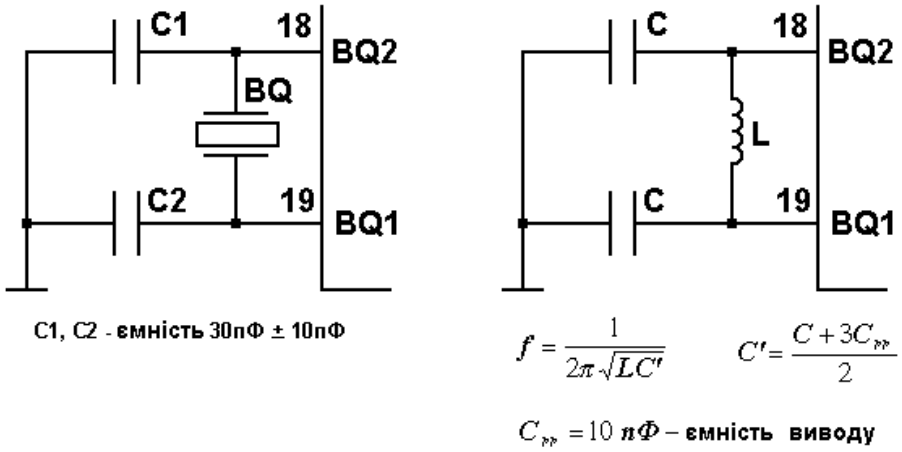


Рисунок 3.10 – Підключення до виводів BQ1 і BQ2 резонатора і LC–ланцюжка

3.1.3.13 Регістри

Акумулятор – 8–розрядний регістр, призначений для прийому і зберігання результату, отриманого при виконанні арифметичних і логічних операцій або операцій пересилання.

Регістр B – 8–розрядний регістр, який використовується при виконанні операцій множення і ділення. В інших випадках він може розглядатися як додатковий регістр загального призначення (частина НОЗП).

Регістри T1, T2 – 8–розрядні регістри тимчасового зберігання, призначені для прийому і зберігання операндів на час виконання операцій над ними в АЛП. Програмно недоступні.

Регістр стану програми (PSW) – служить для зберігання інформації про результат виконання операції в АЛП. Його називають також регістром прапорців (ознак). Позначення окремих розрядів регістра PSW і призначення розрядів приведено відповідно в таблицях 3.2, 3.3.

Прапорець перенесення CY може встановлюватися і скидатися як апаратно, так і програмно. Апаратно він встановлюється, якщо при виконанні арифметичних або логічних операцій формується перенесення/позика у старшому (сьомому) розряді 8–бітних операндів. При виконанні операцій множення і ділення прапорець CY скидається. Крім того, прапорець CY виконує функції "булевого акумулятора" у командах, що працюють з бітами.

Прапорець допоміжного перенесення AC встановлюється/скидається апаратно або програмно. Апаратно встановлюється при виконанні операцій додавання і віднімання при виникненні перенесення/позики в 3–му розряді при утворенні молодшої тетради результату. Частіше всього використовується в СДКА при виконанні команди DA A.

Прапорець користувача F0 – встановлюється/скидається програмно і може використовуватися програмістом за своїм розсудом.

Прапорці–покажчики поточного банку РЗП встановлюються/скидаються програмно і вказують, який із 4–х банків РЗП РПД (таблиця 3.3) в даний момент часу є робочим (поточним).

Прапорець переповнення OV встановлюється/скидається програмно або апаратно. Апаратно встановлюється тоді, коли при виконанні операції додавання/віднімання над числами зі знаком результат не вкладається в діапазон –128 .. +127 і старший, знаковий біт спотворюється. При виконанні операції ділення прапорець OV апаратно скидається, а у випадку ділення на нуль встановлюється. При множенні прапорець OV апаратно встановлюється, якщо результат більше 255.

Прапорець парності (паритету) P встановлюється/скидається апаратно. Він доповнює вміст акумулятора до парного числа одиниць. У 9–розрядному слові, що складається з 8 розрядів акумулятора і біта P, завжди міститься парне число одиничних бітів.

Всі сім названих прапорців програмно доступні за читанням.

Регістр команд РК (IR) призначено для зберігання коду операції (КОП) поточної команди, що виконується.

Лічильник команд ЛК (програмний лічильник (PC)) містить 16–розрядну адресу комірки пам'яті програм. До складу лічильника команд входять 16–розрядні буфер PC, регістр PC, схема інкремента, регістр адреси пам'яті.

Буфер PC здійснює зв'язок між 8–розрядною РШД і 16–розрядним **регістром PC**, у якому зберігається поточна 16–розрядна адреса пам'яті програм.

Схема інкремента збільшує поточне значення 16–розрядної адреси пам'яті програм на одиницю.

Регістр адреси пам'яті призначено для запису і зберігання виконавчої 16–розрядної адреси пам'яті програм або 8/16–розрядної адреси зовнішньої пам'яті даних.

Регістр покажчик даних РГПД (DPTR) призначено для зберігання 16–розрядної адреси зовнішньої пам'яті даних. Складається з двох 8–розрядних регістрів DPH і DPL, що входять у блок регістрів спеціальних функцій [15,16]. Вони програмно доступні і можуть використовуватися в якості двох незалежних РЗП, якщо немає необхідності у зберіганні 16–розрядної адреси зовнішньої пам'яті даних.

Покажчик стека (SP) адресує комірки спеціальної області пам'яті даних (РПД), яка називається стеком. SP адресує “верхівку” стека – останню комірку стекової пам'яті, у якій записана інформація. Покажчик стека являє собою 8–розрядний регістр, вміст якого при виконанні команд LCALL, ACALL збільшується на 2. При виконанні команд RET, RETI вміст покажчика стека зменшується на 2. При виконанні команди PUSH direct вміст SP збільшується на 1, а при виконанні команди POP direct – зменшується на 1.

Регістр адреси РА (RAR) – регістр комірки РШД, що адресується. Програмно не доступний.

Регістри РРТЛ (TMOD) і РКСТ (TCON) служать для програмування і керування роботою таймерів/лічильників і системи переривань. Формати, позначення і призначення їхніх окремих розрядів приведено в таблицях 3.7…3.10.

Регістр РКПП (SCON), буфери ПД і ПРМ (SBUF) призначено для програмування і керування роботою послідовного інтерфейсу. Формати, позначення і призначення їхніх окремих розрядів приведено в таблицях 3.11, 3.12.

Регістри РМП (IE) і РП (IR) програмують і керують роботою системи переривань МК. Формати, позначення і призначення РМП і РП наведено в таблицях 3.4, 3.5.

Регістр керування потужністю РКП (PCON) служить для програмного керування споживанням енергії від джерела живлення, а також швидкістю передачі по послідовному каналу МК. Формат, позначення і призначення його окремих розрядів наведено в таблицях 3.13, 3.14.

Більш докладно застосування цього регістра буде розглянуто в окремих розділах.

Таблиця 3.13 – Позначення розрядів регістра PCON

Біти	7	6	5	4	3	2	1	0
Позначення	SMOD	–	–	–	GF1	GF0	PD	IDL

Таблиця 3.14 – Призначення розрядів регістра PCON

Біти	Найменув.	Призначення бітів	Примітка
7	SMOD	Біт подвоєння швидкості передачі: при встановленні в "1" – швидкість передачі подвоюється.	При роботі послідовного порту
6	–	Резервний	
5	–	Резервний	
4	–	Резервний	
3	GF1	Прапорець загального призначення	
2	GF0	Прапорець загального призначення	
1	PD	Біт вмикання режиму мікроспоживання "1" – режим мікроспоживання	Якщо в PD і IDL одночасно записано "1", перевагу має PD
0	IDL	Біт холостого ходу "1" – режим холостого ходу	

3.2 Порівняльна характеристика МП–РА та МК–РА

3.2.1 Загальні риси

Структури обох пристроїв включають (рисунки 3.1, 3.3, 3.5):

– арифметико–логічний пристрій (АЛП);
– регістри тимчасового збереження операндів (програмно недоступні, на структурі МК позначені Т1, Т2);
– один з основних регістрів – акумулятор (А);
– регістр прапорців (ознак), на структурі МК позначений ССП – слово–стану програми (PSW);
– схему десяткової корекції акумулятора після додавання двійково–десяткових кодів (BCD–кодів) в упакованому форматі;
– лічильник команд – ЛК (програмний лічильник РС);
– регістр–покажчик стека (SP);
– схему керування та синхронізації;
– внутрішню 8–ми розрядну шину даних (ВШД);
– регістр команд – РК (IP) – програмно недоступний.

3.2.2 Відмінні риси

Структура МК (рисунок 3.5) додатково містить:

– резидентну пам'ять даних – РПД. Для МК типу AT89C51 її об'єм становить 128 комірок пам'яті, довжиною 8 біт, тобто 128 байт;
– резидентну пам'ять програм – РПП. Для МК типу AT89C51 об'ємом 4 КБайт (FLASH);
– чотири паралельних 8–ми розрядних програмованих порти введення/виведення (P0, P1, P2, P3);
– регістр розширення акумулятора В (використовується при виконанні команд множення і ділення, а також як 8–ми розрядний регістр загального призначення).
– послідовний програмований інтерфейс введення/виведення (для цього в режимі «альтернативних функцій» використовується дві лінії порту 3: P3.0 і P3.1);

– два 16–ти розрядних багаторежимних таймери/лічильники – T/CNT0; T/CNT1, які можуть виконувати функції програмованих таймерів або лічильників зовнішніх подій. При цьому імпульси, що ідентифікують зовнішні події, надходять через дві лінії порту 3 (P3.4; P3.5), який використовується в режимі «альтернативних функцій»;
– 16–ти розрядний регістр – покажчик даних – DPTR. Складається з старшого байта – DPH і молодшого – DPL. Може використовуватися як 16–ти розрядний або як два незалежних 8–ми розрядних регістри. Може зберігати 16–ти бітову адресу під час звертання до пам'яті програм або зовнішньої пам'яті даних;
– внутрішній тактовий генератор (OSC), частота якого задається за допомогою зовнішнього кварцового резонатора;
– систему переривань з п'ятьма векторами (адресами підпрограм) і двома рівнями пріоритетів (для AT89C51).
– Два переривання можуть викликатися зовнішніми джерелами і надходять по лініям порту 3 (P3.2, P3.3) в режимі «альтернативних функцій». Інші три переривання є внутрішніми: два від переповнення таймерів/лічильників і одне від завершення передачі або прийому по послідовному каналу;
– регістри керування – програмуючі регістри, що керують роботою:
– таймерів/лічильників – РРТЛ (TMOD) – регістр режимів таймерів/лічильників; РКСТ (TCON) – регістр керування–статусу таймерів/лічильників;
– послідовного каналу – РКПП (SCON) – регістр керування приймачами–передавачами; буфер ПЕР (SBUF) – буфер передавача; буфер ПРМ (SBUF) – буфер приймача. Обидва

буфери мають однакове ім'я під час програмного звертання до них – SBUF. Якщо команда записує дані в буфер, то звертання йде до буфера ПЕР, а якщо – читає, то до буфера ПРМ;

– системи переривань – РМП (IE) – регістр масок переривань; РП (IP) – регістр пріоритетів переривань.

Один з регістрів керування – РКП (PCON) – регістр керування потужністю, керує процесом споживання енергії (потужності) від джерела живлення і дозволяє переводити МК в режими: «холостого ходу» і «зниженого споживання енергії». Один з бітів цього регістра – PCON.7 (SMOD) дозволяє програмно змінювати швидкість передачі по послідовному каналу.

3.2.3 Підключення МП–ра до системної шини МПС

При використанні, наприклад МП i8080, використовується архітектура з 3 системними шинами: системної шини адреси (США), системної шини даних (СШД) і системної шини керування (СШК). На рисунку 3.11 приведено структурну схеми системи з 3 шинами для МП i8080, i8086.

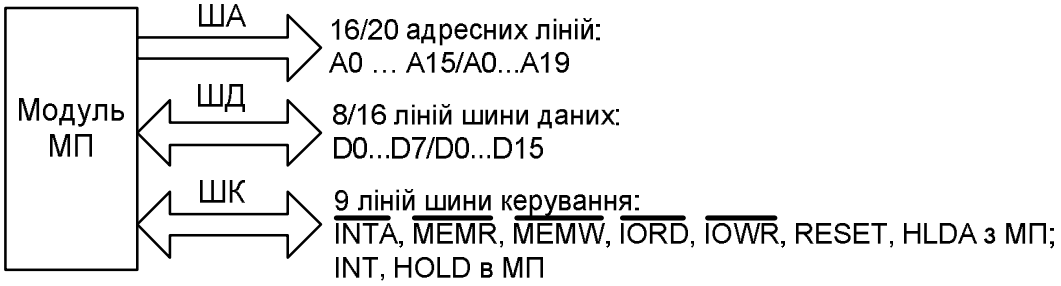


Рисунок 3.11– Структурна схема системи з 3 шинами

США є однонаправленою, тобто сигнали передаються лише від зовнішніх виводів МП. США в МПС пов'язана з безліччю адресних входів, що підключені паралельно, тому сумарний струм для адресних ліній МПС може за величиною перевершувати припустиме значення струму вихідних ліній МП. Отже, для збільшення навантажувальної здатності вихідних адресних ліній МП їх необхідно буферирувати. Як буфер США з МП i8080

може використовуватися формувач шини адреси – мікросхема КР580ВА86, виходи якої мають 3 стани, що дозволяє відключати ША від МП в режимі прямого доступу до пам'яті (ПДП) [5, 33].

При підключенні СШД до МП необхідно враховувати і можливість перевантаження виводів МП і те, що СШД є двунаправленою, тобто передача даних ведеться в обох напрямах (однак в кожний момент часу дані передаються лише в одному напрямі). Тому СШД підключається до МП через двонаправлений буфер шини даних, функції якого також може виконувати мікросхема КР580ВА86.

СШК є по суті однонаправленою, тобто по одним лініям сигнали надходять в МП, а по другим від нього, тобто, необхідності організації двостороннього обміну по одним і тим самим лініям немає. СШК підключається до МП через формувач керуючих сигналів (ФКС), який формує керуючі сигнали в залежності від слова–стану МП, що зберігається в зовнішньому регістрі стану (ЗРС), і вихідних сигналів МП : DBIN та $\overline{WR}$. ФКС виробляє 5 основних сигналів: $\overline{INTA}$, $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IORD}$, $\overline{IOWR}$. ФКС може бути виконано на твердій логіці або з використанням спеціалізованого системного контролера типу КР580ВК28, що виконує також роль шинного формувача.

Підводячи підсумок, зауважимо, що буферування системної шини використовується для збільшення навантажувальної здатності виводів МП, при цьому зберігається і можливість відключення системної шини від МП в режимі ПДП, а також забезпечення двостороннього обміну для ШД.

3.2.4 Часові співвідношення у МП–рі та МК–рі

Командним циклом (КЦ) називається час, необхідний для зчитування команди з пам'яті і її виконання.

КЦ складається з декількох машинних циклів (МЦ), точна кількість яких залежить від складності виконуваної команди і дорівнює числу звертань МП до пам'яті або одного з ПВВ (рисунок 3.12).

МЦ складається з деякої кількості елементарних дій, що називаються станами (тактами), і виконуються за один період системного сигналу тактування.

. Наприклад, в МП і8080 передбачено 10 типів МЦ. На рисунку 3.13 показано МЦ “читання коду операції”.

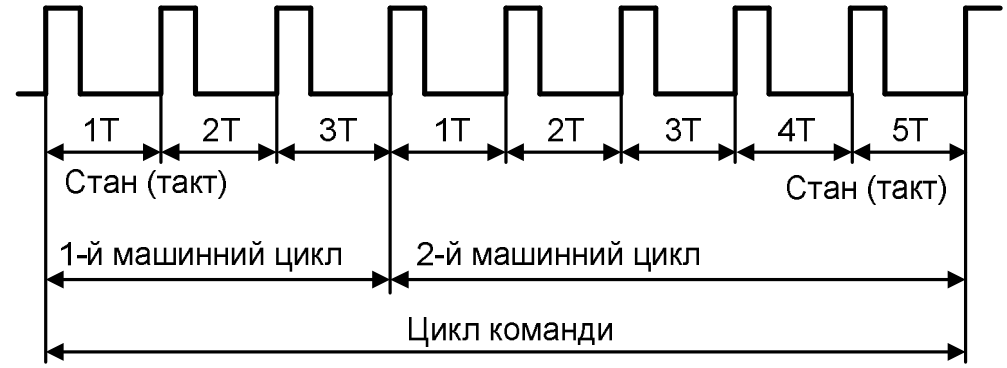


Рисунок 3.12 – Приклад часової структури командного циклу

Які саме МЦ будуть виконуватися, залежить від конкретної команди, однак, першим МЦ завжди є «Читання КОП».

В МП–рі і8080 виконуються наступні часові співвідношення:

– 1КЦ=(1…5) МЦ;

– 1МЦ=(3…5) тактів.

Якщо $f_т$=2 МГц,

то $t_т$=1/ $f_т$=0,5 мкс=500 нс.

В МК–рі типу МК51 виконуються такі часові співвідношення:

– 1КЦ=1, 2 або 4 МЦ (для операцій множення/ділення);

– 1МЦ=6 станів по 2 фази в кожному (12 фаз (тактів)).

Якщо $f_т$=12 МГц, то $t_т$=1/ $f_т$=1/12 мкс; $t_{МЦ}$=2*6* $t_т$= 1 мкс.

Тобто для визначення повного часу виконання команди потрібно знати, яке число тактів міститься в КЦ і чому дорівнює період сигналу тактування, а потім їх перемножити. Часові співвідношення у МК–рі описано у 3.1.3.2.

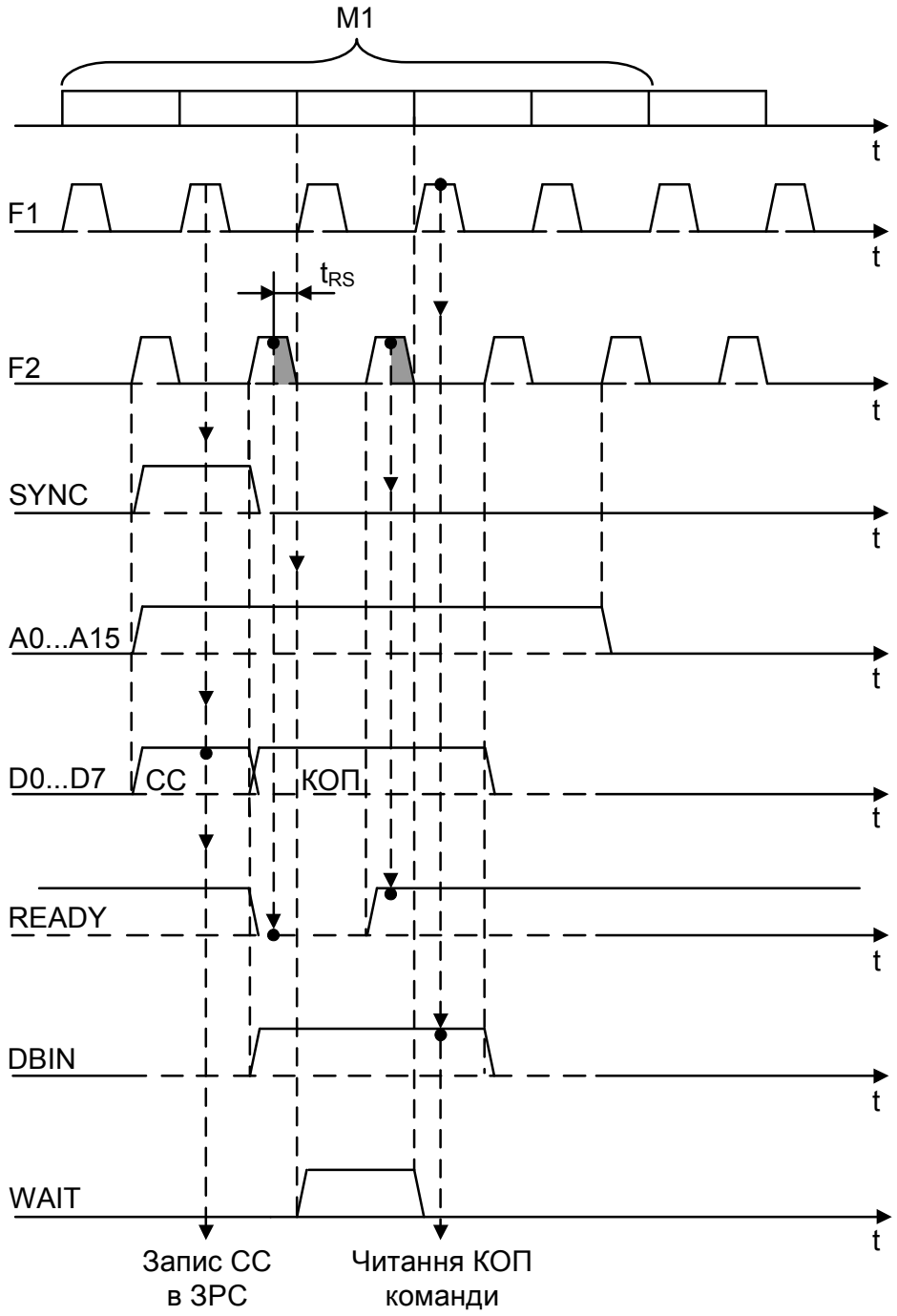


Рисунок 3.13 – Часові діаграми роботи МП–ра і8080 для МЦ «читання коду операції команди»

В 16–ти бітному МП–рі і8086 час виконання кожної команди вказується у вигляді

$$n_T = n + T_{BÀ}, \qquad (3.1)$$

де n_T – кількість тактів, що необхідні для виконання наведеної команди;

n – постійна величина;

$T_{BÀ}$ – додаткове число тактів, яке залежить від способу адресації операндів, що використовуються в наведеній команді.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Яку роль виконує регістр–акумулятор МП–ра i8080?
2. Що являє собою лічильник команд?
3. Назвіть елементи, що входять до складу керуючого адресного регістра.
4. Для чого призначені мультиплескор та демультиплексор?
5. Назвіть компоненти, що входять до складу виконавчого блоку МП i8086.
6. Яку розрядність має регістр BH?
7. Що зберігається в регістрі SI?
8. Назвіть умови встановлення прапорця CF в 1.
9. Для чого призначені сегментні регістри?
10. Для чого призначений суматор адреси?
11. Дайте визначення командного циклу.
12. Що потрібно знати для визначення повного часу виконання команди?
13. Скільки машинних циклів в МК–рі AT89C51 складає виконання команди MUL A, B?
14. Що являє собою арифметико–логічний пристрій?
15. Скільки регістрів ОЗП МК–ра МК51 допускають побітову адресацію?
16. Як змінюється покажчик стека при виконанні команд RET, RETI?
17. Для чого служить лічильник команд PC?
18. Скільки паралельних портів введення/виведення дискретної інформації має МК AT89C51?

ЛІТЕРАТУРА [1…5, 8…16, 29…34]

4 МІСЦЕ КЕРУЮЧОЇ ПРОГРАМИ У РОБОТІ МПС ТА ПРОГРАМНА МОДЕЛЬ МП–РА/МК–РА

4.1 Послідовність розробки робочої керуючої програми

Якщо МПС виконана на основі МП–ра, то керуюча програма знаходиться у зовнішній пам'яті програм (комірках ПЗП), а якщо основним вузлом МПС є МК, то програма може знаходитися в резидентній (РПП) і/або зовнішній пам'яті програм (ЗПП). Без керуючої програми жодна МПС працювати не буде.

Основні етапи створення керуючої програми:

- розробка схеми алгоритму, який повинна реалізувати конкретна керуюча програма;
- якщо задача складна, то програма може бути реалізована і відлагоджена на мові високого рівня або мові асемблера;
- компіляція — переведення програми з мови високого рівня або асемблера в машинні коди конкретного МП–ра, або МК–ра (створення об'єктного модуля);
- введення керуючої програми в ПЗП за допомогою програматора;
- в процесі початкової ініціалізації системи у програмний лічильник завантажується початкова адреса програми.

Керуюча програма може включати підпрограми.

Підпрограма — це частина основної програми, яка може бути викликана з основної програми або за перериванням.

4.2 Програмна модель МП–ра та МК–ра та її відмінність від структурної схеми

4.2.1 Програмна модель МП–ра i8080

Програмна (програмістська) модель включає ті частини архітектури МП або МК, до яких програміст може отримати доступ за допомогою тієї або іншої команди. На рисунку 4.1 приведена програмна (програмістська) модель МП–ра i8080.

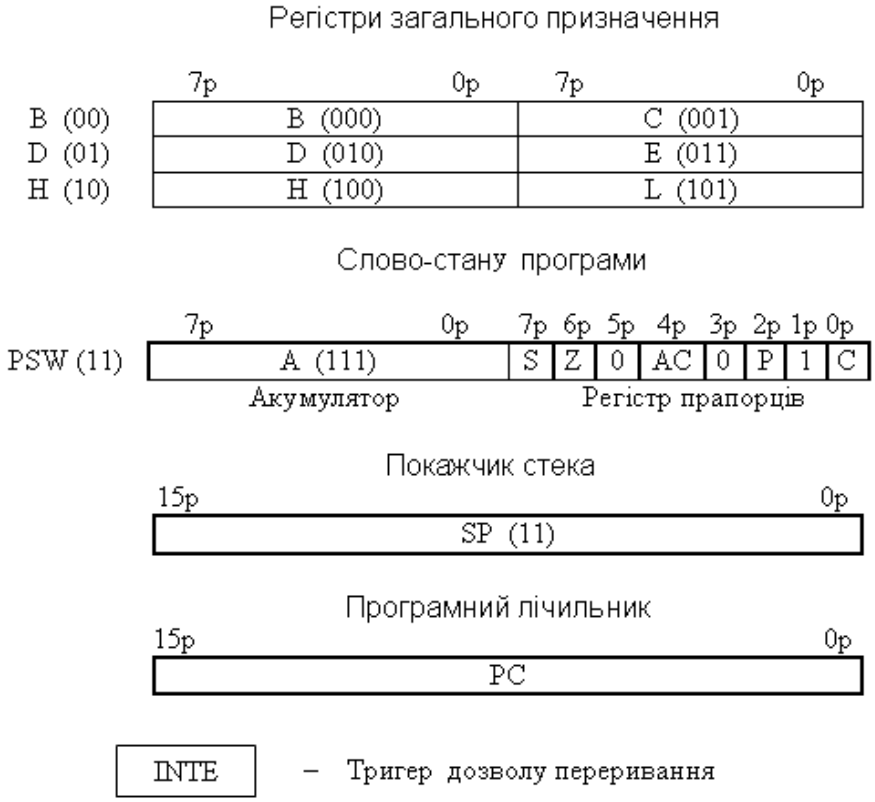


Рисунок 4.1 – Програмна (програмістська) модель МП i8080

До програмістської моделі МП входять:

– регістри B, C, D, E, H, L – регістри загального призначення (РЗП) – надоперативний запам'ятовуючий пристрій (НОЗП);

– слово–стану програми PSW (ССП) – Program Status Word (рисунок 4.2), яке складається з акумулятора (A) і регістра прапорців (RF): A — старший байт, RF — молодший байт.

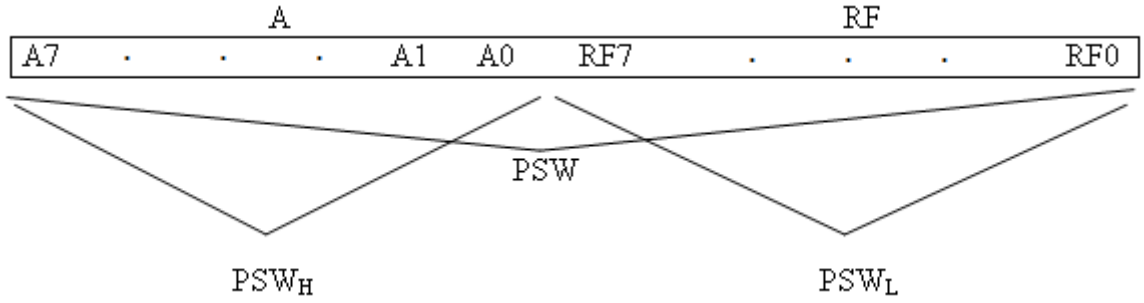


Рисунок 4.2 – Слово–стану програми (PSW) МП–ра i8080

Регістр прапорців являє собою набір тригерів, які показують стан програми після виконання команд, що впливають на прапорці (рисунок 4.3).

7p	6	5	4	3	2	1	0p
S	Z	0	AC	0	P	1	C

Рисунок 4.3 – Позначення розрядів регістра прапорців (регістра ознак)

До регістра прапорців входять:

– S – знаковий прапорець – дублює старший біт результату під час виконання операцій над числами зі знаком;

– Z – прапорець нуля – встановлюється в одиницю, коли в результаті операції всі біти нульові;

– AC – прапорець допоміжного перенесення – встановлюється під час перенесення одиниці з молодшої тетради в старшу або з 3–го біта в 4–й, якщо молодший біт нульовий. Наведений прапорець використовується блоком десяткової корекції, що виконує корекцію результату після виконання операції додавання BCD–чисел в упакованому форматі;

– P – прапорець парності (паритету) – визначає число одиниць результату.

P=1, якщо сума одиниць результату за модулем два дорівнює нулю (результат містить парне число одиниць);

– C – прапорець перенесення/запозичення – встановлюється в 1, якщо відбулося перенесення з 7 розряду в неіснуючий 8–й при виконанні додавання або під час віднімання зменшуване було менше від'ємника і відбулося запозичення одиниці з неіснуючого восьмого розряду. Даний прапорець часто використовується під час виконання багатобайтного додавання/віднімання.

4.2.2 Програмна модель мікропроцесора i8086

Однією з основних характеристик архітектури будь–якого МП є програмна (програмістська) модель, що включає частину структури МП, що доступна програмісту за допомогою системи команд.

У програмній моделі МП i8086, показаної на рисунку 4.4, є кілька груп регістрів з різним функціональним призначенням.

Група регістрів загального призначення (РЗП) включає регістри AX, BX, CX і DX. Їхня особливість полягає в тому, що в командах допускається вказувати старшу (H–Hight) і молодшу (L–Low) половини цих регістрів. Такий подвійний характер регістрів AX…DX дозволяє багатьом командам оперувати байтами і словами. Інші регістри можна використовувати тільки словами. Регістри AX…DX знаходяться в повному розпорядженні програміста й однаково беруть участь в арифметичних і логічних операціях. Але в деяких командах вони спеціалізовані, що відбито в їхніх назвах.

Регістр AX використовується в операціях множення, ділення, введення/виведення слів і в деяких операціях з рядками. Регістр AL бере участь в аналогічних операціях з байтами, в операціях перетворення і десяткової арифметики.

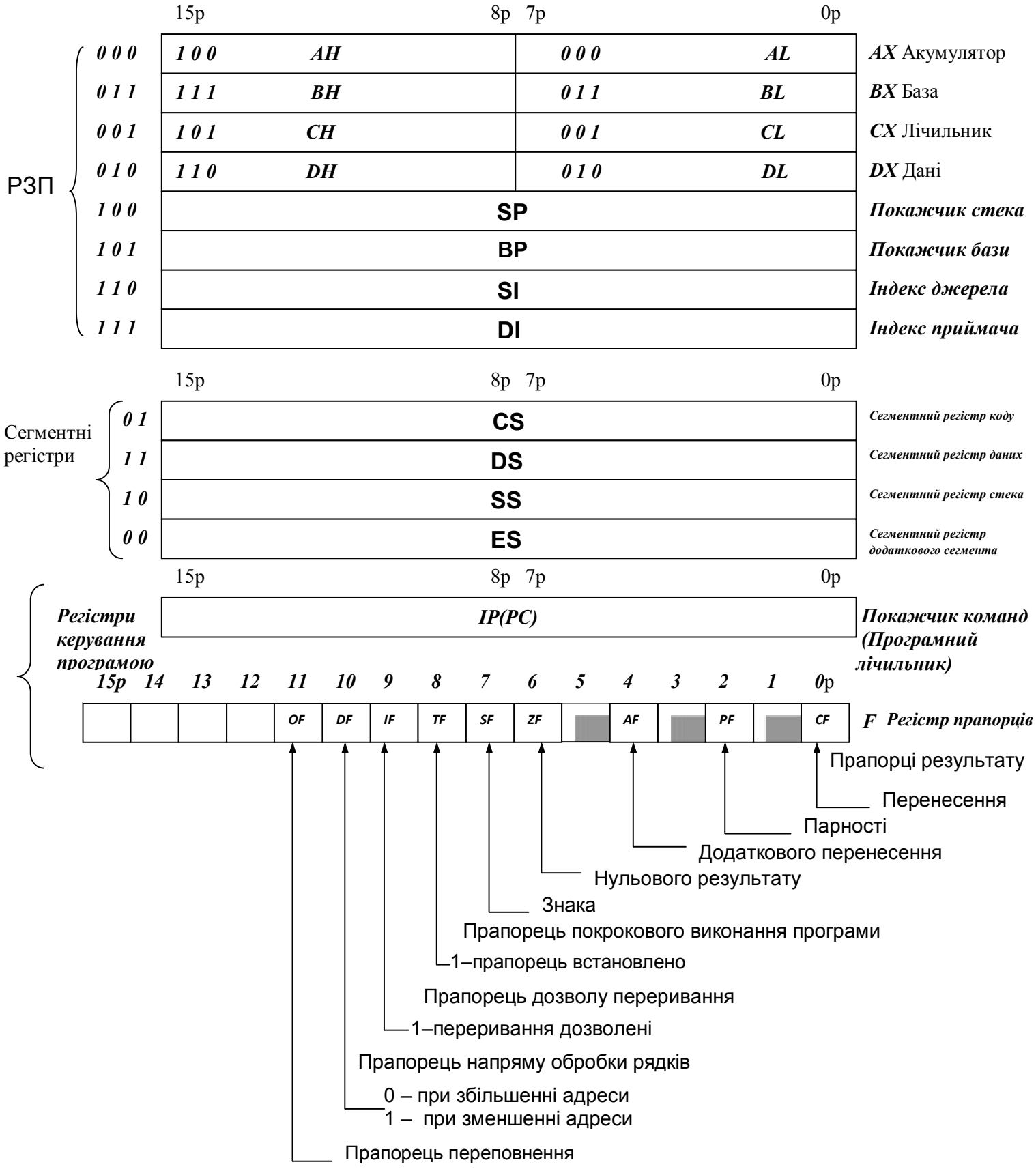


Рисунок 4.4 – Програмна модель МП i8086

Регістр AX використовується в множенні і діленні байтів. Програми стають компактнішими, якщо регістри AX і AL максимально залучаються для арифметичних і логічних операцій і пересилань даних.

Регістр BX широко застосовується для адресації структур даних у сегменті даних.

Регістр CX виконує функції лічильника повторень у програмних циклах і в операціях з рядками. Регістр CL служить лічильником зсувів.

Регістр DX бере участь в операціях множення і ділення слів. Крім того, він зберігає адресу порту в командах введення/виведення з непрямою адресацією.

Регістри BP, SP, SI і DI утворюють групу покажчиків і індексних регістрів. Вони можуть використовуватися для збереження адрес, забезпечуючи непряму адресацію пам'яті і беручи участь в обчисленнях ефективної адреси [2, 9, 11, 28]. Але ці регістри можуть залучатися для арифметичних і логічних операцій так само, як і регістри першої групи. Тому регістри AX…DX і SP…DI називаються *загальними регістрами* (регістрами загального призначення – РЗП). Усього є вісім 16–ти розрядних регістрів загального призначення і для вибору кожного з них у командах досить трьох бітів. Покажчик стека SP використовується для адресації верхівки стека в пам'яті. Обидві стекові операції (PUSH і POP) автоматично модифікують SP. За допомогою покажчика бази BP забезпечується простий доступ до даних, що знаходяться в стеку (не тільки у верхівці стека). Найбільш часто регістр BP залучається для адресації параметрів, що передаються через стек підпрограмам. Індексні регістри SI і DI застосовуються для адресації даних, а також елементів рядків у спеціальних командах.

Досить нерегулярна структура загальних регістрів вимагає, щоб програміст (чи транслятор з мови високого рівня) ретельно розподіляв

регістри і стежив за їх використанням. У той же час неявна вказівка деяких регістрів в операціях і режимах адресації дозволяє компактніше закодувати команди.

У нижній частині рисунка 4.4 поміщено регістр прапорців (ознак). Шість прапорців CF, PF, AF, ZF, SF і OF фіксують визначені ознаки результату арифметичної чи логічної операції. Команди МП впливають на них по–різному, але, у загальному, ці прапорці показують наступні особливості результату виконання окремих операцій (перші 5 прапорців аналогічні прапорцям МП i8080):

– прапорець перенесення CF фіксує значення біта перенесення , що виникає при додаванні (відніманні) байтів чи слів, а також значення біта, що висувається у всіх операціях зсуву. Він також показує особливість результату операцій множення та ділення. Прапорець перенесення грає винятково важливу роль при написанні програм;

– прапорець паритету (парності) PF переходом в 1 реєструє наявність парного числа одиниць у 8 молодших бітах результату операції. Він застосовується для контролю вірності передач даних;

– прапорець допоміжного перенесення AF аналогічний прапорцю CF, але фіксує перенесення з молодшої тетради результату (чи позики). Цей прапорець використовується в операціях десяткової арифметики;

– прапорець нуля ZF сигналізує про одержання нульового результату операції;

– прапорець знака SF повторює значення старшого біта результату, що у додатковому коді відповідає знаку числа;

– прапорець переповнення OF відзначає втрату старшого біта результату операції додавання чи віднімання над знаковими числами. Переповнення виникає, коли значення перенесень у старший біт і зі

старшого біта не збігаються. Прапорець OF показує також зміну старшого (знакового) біта при арифметичних зсувах вліво.

Три останніх прапорці керують деякими діями МП. Програміст може однією чи декількома командами задати стан кожного з них:

– прапорець напряму DF визначає сканування (перегляд) елементів рядків від менших адрес до більших (DF=0) чи навпаки (DF=1);

– прапорець переривання IF задає реакцію МП на запит переривання на вході INT . Якщо IF=0, запит переривання ігнорується, а якщо IF=1, МП розпізнає і відповідно реагує на нього. Прапорець IF не впливає на сприйняття переривань на вході NMI, які не маскуються, і внутрішніх переривань;

– прапорець трасування (покрокового виконання програми) TF при встановленні в 1 переводить МП у покроковий режим роботи, у якому МП автоматично генерує внутрішнє переривання після виконання кожної команди.

Покажчик команд IP виконує функції програмного лічильника PC (далі використовується остання назва). У процесі вибірки команд із програмної пам'яті відбувається відповідна модифікація IP для того, щоб він адресував наступну команду, яка повинна виконуватись.

Наявність у програмній моделі чотирьох сегментних регістрів: коду CS, даних DS, стека SS і додаткових даних ES пояснюється їхньою участю в адресації пам'яті. МП має 20–ти бітну шину зовнішньої фізичної адреси пам'яті, але в програмній моделі немає жодного регістра довжиною 20 біт. Усередині МП фізична адреса пам'яті представлена двома 16–бітними словами, одне з яких називається *логічною базовою (початковою*) *адресою сегмента*, а друге – *внутрішньосегментним зсувом*. Ці два слова являють собою 32–бітну логічну адресу комірки пам'яті. Пристрій перетворення адрес у складі шинного інтерфейса мікропроцесора при кожному звертанні

до пам'яті перетворює логічну адресу у фізичну. Для програми адресний простір пам'яті складається з чотирьох сегментів, що є логічними одиницями пам'яті з максимальним розміром 64Кбайт.

Реальний розмір сегмента необов'язково повинний бути максимальним. Мінімальний розмір сегмента дорівнює 16 байтам. На розміщення сегментів у просторі 1 Мбайт накладається тільки одне обмеження: базова 20–ти бітна фізична адреса сегмента повинна бути кратна 16, тобто його 4 молодших біти повинні бути нульовими. Іншими словами, фізична базова адреса сегмента має вигляд: XXXX0H. Нульові біти можна не зберігати, а мати на увазі, і тоді для логічної базової адреси сегмента досить 16 біт. Саме такі "урізані" логічні базові адреси сегментів знаходяться в регістрах CS, DS, SS і ES. Отже, при фіксованому вмісті сегментних регістрів максимальний робочий простір, до якого МП має доступ у будь–який момент часу, складається з 64Кбайт для коду (власне програми), 64Кбайт для стека і 128Кбайт для даних. Якщо програмі потрібний більший робочий простір, вона повинна модифікувати вміст сегментних регістрів. Для адресації конкретного байта в сегменті служить другий компонент логічної адреси – зсув. Він, є 16–ти бітним цілим беззнаковим числом та показує відстань цього байта від початку сегмента. Отже, для утворення фізичної адреси з пари: початок сегмента – зсув необхідно зсунути логічну 16–бітну базову адресу сегмента вліво на 4 біти і додати зсув.

4.2.3 Програмна модель мікроконтролера

4.2.3.1 Загальна характеристика

Програмна модель мікроконтролера, наприклад, сімейства МК51 AT89C51 наведена на рисунку 4.5.

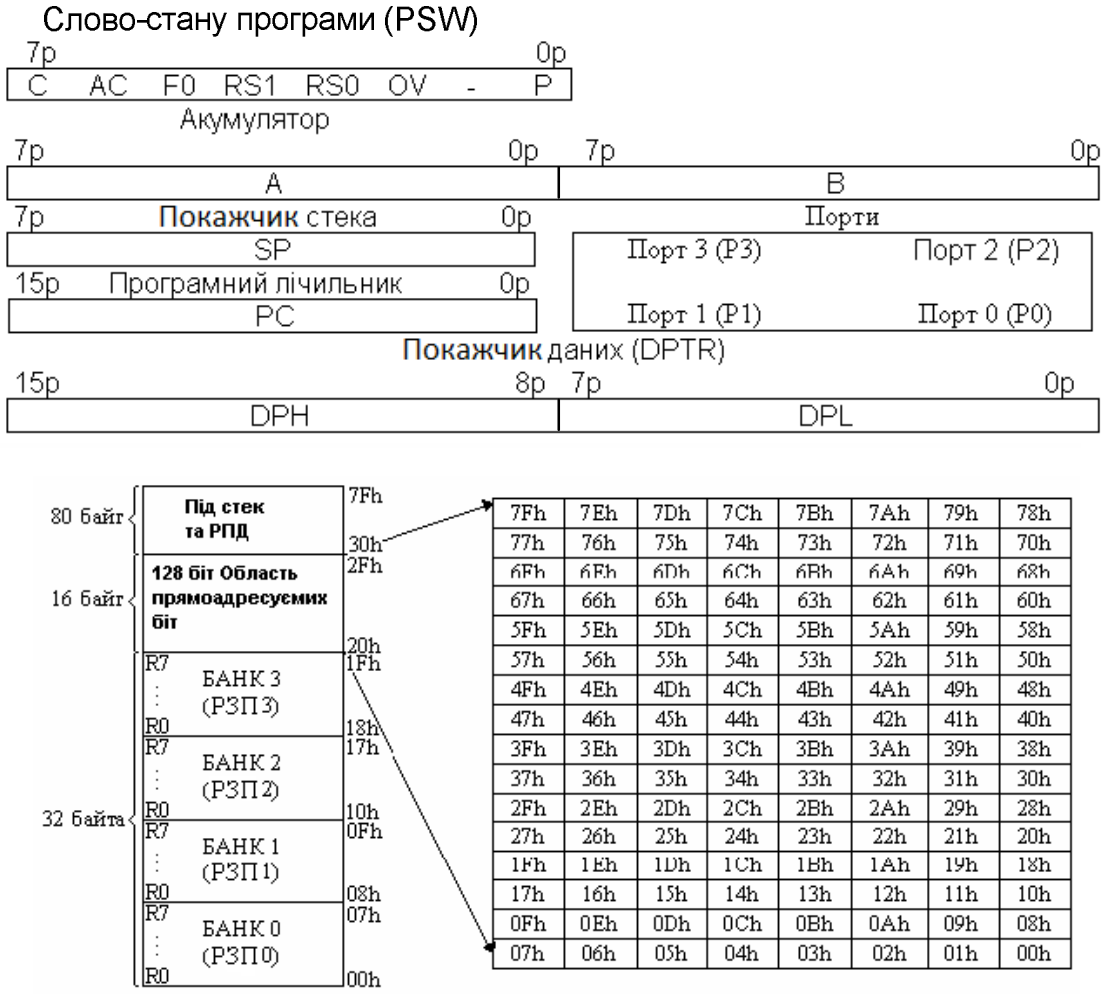
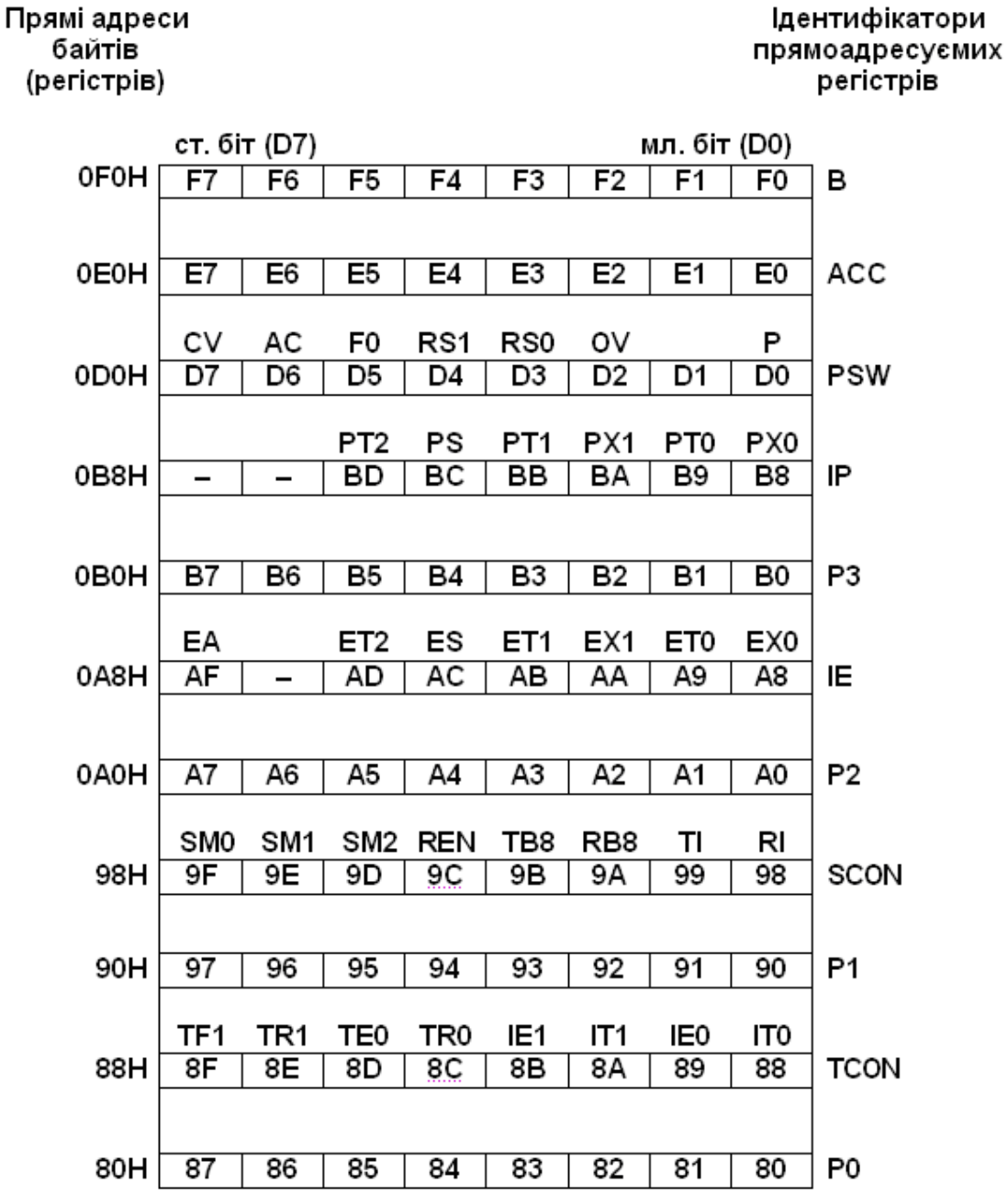


Рисунок 4.5 – Програмна модель МК51



Продовження рисунку 4.5

4.2.3.2 Резидентна пам’ять даних

Резидентна пам’ять даних РПД призначена для прийому, збереження та видачі інформації, яка використовується в процесі виконання програми.

Пам’ять даних ділиться на внутрішню (резидентну) пам’ять даних РПД і зовнішню пам’ять даних ЗПД.

РПД являє собою 128 восьмирозрядних регістрів, які призначені для прийому, збереження і видачі різноманітної інформації. Шістнадцять із цих регістрів допускають побітову адресацію.

На рисунку 4.6 наведено розподіл адресного простору РПД і області біт, що адресуються прямо.

В області молодших адрес РПД знаходяться 4 банки регістрів загального призначення (РЗП), кожен з яких має об’єм 8 регістрів (байт): R0, R1, …, R7.

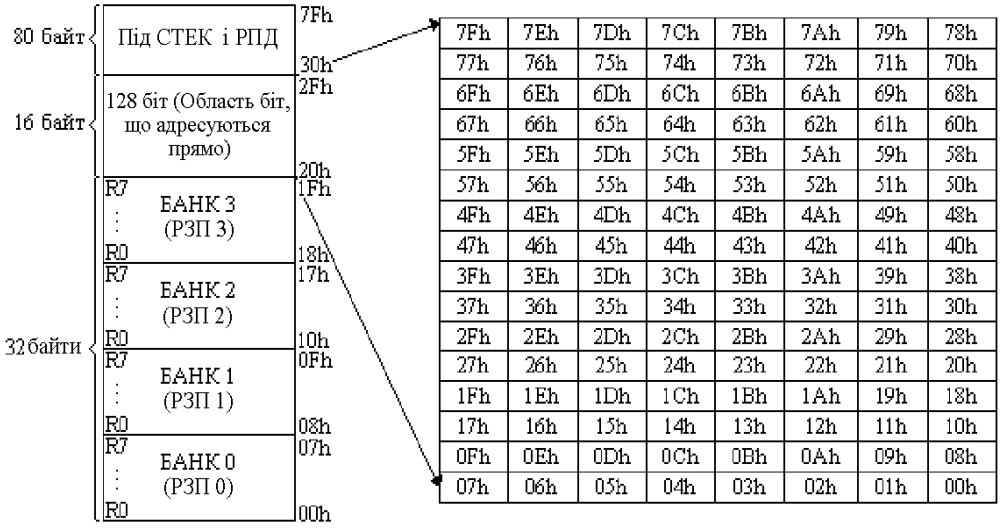


Рисунок 4.6 – Розподіл адресного простору РПД і області біт, які адресуються прямо

4.2.3.3 Регістри спеціальних функцій

Частину програмістської моделі складають регістри спеціальних функцій (таблиця 4.1). Нижче описано призначення цих регістрів.

Акумулятор А (англ. **Accumulator**) – один з найважливіших регістрів мікроконтролера. Команди, які призначені для роботи з акумулятором,

використовують його ім'я «А», наприклад, MOV A, P2. Ім'я «ACC» використовується, у більшості, при побітовій адресації акумулятора. Тому, наприклад, символічне ім’я п’ятого біта акумулятора при використанні мови асемблера ASM51 буде таким: ACC.5.

Таблиця 4.1 – Регістри спеціальних функцій

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

Прим: * – регістри, які допускають побітову адресацію.

Регістр B. Використовується під час операцій множення та ділення. Для інших команд регістр B може розглядатись, як додатковий регістр внутрішньої надоперативної пам’яті (НОЗП).

Регістр стану програми РСП (англ. **Program Status Word PSW**). Регістр PSW містить інформацію про стан програми. Формат регістра PSW (слово стану програми ССП) наведено у таблиці 4.2.

Таблиця 4.2 – Формат регістра ССП

Позиція	Символ	Ім'я та призначення
1	2	3
PSW.7	CY	**Прапорець перенесення** (англ. **Carry flag**). Встановлюється та скидається апаратними засобами при виконанні арифметичних і логічних операцій. Програмно доступний.
PSW.6	AC	**Прапорець допоміжного перенесення** (англ. **Auxiliary carry flag**). Встановлюється та скидається апаратними засобами при виконанні команд додавання та віднімання і сигналізує про перенесення (переповнення) або позику у біті 3 (рахуючи молодший біт нульовим). Програмно доступний.
PSW.5	F0	**Прапорець F0** (англ. **User controlled flag**). Може бути встановлений в 0 чи 1 або перевірений програмою як прапорець, специфікований користувачем.
PSW.4 PSW.3	RS1 RS0	**Вибір банку регістрів** (англ. **Register select bank switch flag**). Встановлюється та скидається програмою для вибору робочого банку регістрів.
PSW.2	OV	**Прапорець переповнення** (англ. **Overflow flag**). Встановлюється та скидається апаратно при виконанні арифметичних операцій над числами зі знаком. Програмно доступний.
PSW.1	–	Не використовується.
PSW.0	P	**Прапорець паритету** (англ. **Parity flag**). Встановлюється та скидається апаратно в кожному циклі команди і фіксує непарне-парне (1/0) число одиничних бітів в акумуляторі, тобто виконує контроль за парністю.

Регістр–покажчик стека РПС (англ. **Stac Pointer – SP**). 8–бітовий регістр, вміст якого інкрементується перед записом даних у стек при виконанні команд PUSH і CALL. При початковому скиданні в покажчик стека заноситься значення 07H, а область стека в ОЗП даних починається з адреси 08H. При необхідності, шляхом перевизначення покажчика стека, область стека може бути розташована в будь–якому місці внутрішнього ОЗП даних мікроконтролера.

Регістр–покажчик даних (англ. **Data Pointer (DPTR)**). Складається з старшого байта (DPH) та молодшого байта (DPL). Містить 16–бітову адресу при зверненні до зовнішньої пам'яті. Може використовуватися як 16–бітовий регістр або як два незалежних 8–бітових регістри.

Порт 0 … Порт 3 (англ. **Port P0, Port P1, Port P2, Port P3**). Виконують функцію 32–х ліній введення/виведення, згрупованих в чотири 8–бітові порти.

Буфер послідовного порту (англ. **Serial Bufer (SBUF)).** Являє собою два окремих регістри: буфер передавача та буфер приймача. Коли дані записуються в SBUF, вони надходять у буфер передавача, причому запис байта в SBUF автоматично ініціює його передачу через послідовний порт. Коли дані зчитуються з SBUF, вони вибираються з буфера приймача.

Регістри таймерів (англ**. Timer T**). Регістрові пари (TH0, TL0) та (TH1, TL1) утворюють 16–бітові регістри–лічильники відповідно таймера/лічильника 0 і таймера/лічильника 1.

Регістри керування. Регістри спеціальних функцій **IP, IE, TMOD, TCON, SCON і PCON** містять біти керування та біти стану системи переривань, таймерів/лічильників, послідовного порту, схеми керування споживанням енергії від джерела живлення:

- регістр пріоритетів переривань РП (англ. **Interrupt Pointer IP**);
- регістр дозволу переривань (англ. **Interrupt Enable IE**);
- регістр режимів таймерів/лічильників (англ. **Timer–Counter Mode TMOD**);
- регістр керування таймерами/лічильниками (англ. **Timer–Counter Control TCON**);
- регістр керування послідовним портом (англ. **Serial Control SCON**);
- регістр керування енергоспоживанням (англ. **Pover Control –PCON**).

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Назвіть основні етапи створення керуючої програми.
2. Дайте визначення підпрограми.
3. Назвіть регістри загального призначення МП–ра і8080.
4. Що являє собою регістр прапорців?
5. Назвіть регістри загального призначення мікропроцесора і8086.
6. В яких операціях використовується регістр АХ?
7. Який регістр зберігає адресу порту в командах введення/виведення з непрямою адресацією?
8. Назвіть індексні регістри мікропроцесора і8086.
9. Назвіть сегментні регістри мікропроцесора і8086.
10. Для чого використовується регістр SP?
11. Що показує прапорець OF?
12. Який прапорець потрібно встановити в 1, щоб перевести МП у покроковий режим роботи?
13. Яку функцію виконує покажчик команд IP?
14. Що називається логічною базовою адресою сегмента?
15. Як із пари: початок сегмента – зміщення в сегменті утворити фізичну адресу комірки пам’яті?
16. Що являє собою резидентна пам’ять даних?
17. Для чого призначена резидентна пам’ять даних?
18. Назвіть регістри спеціальних функцій МК51 AT89C51, які допускають побітову адресацію.

ЛІТЕРАТУРА [1…5, 8… 15, 16]

5 МОВА АССЕМБЛЕР ТА ЇЇ ВИКОРИСТАННЯ У МПС

5.1 Мова ассемблер (МА)

Мова Ассемблер (МА) – мова програмування низького рівня (машинно–орієнтована мова).

Існують мови програмування високого рівня і низького рівня (машинно–орієнтовані, асемблер).

Основу Асемблера складають команди МП–ра або МК–ра, які не можна змінити, а можна вивчити і використовувати під час програмування. Тому говорять, що Асемблер орієнтований на конкретний МП–р або МК–р (на його систему команд).

Окрім цих команд, які при компіляції перетворюються у машинний код, програма включає псевдокоманди та директиви, які вказують програмі компілятора, яка також зветься асемблером, як виконати відповідні дії по створенню виконуваного файлу для МК–ра.

Нагадаємо, що при створенні робочих програм на МА використовують такі керуючі програми:

- **транслятори** – перетворюють програму, що написана на одній мові програмування, в програму, що написана на іншій мові програмування;
- **компілятори** – є різновидом транслятора; переводять програму з мови високого рівня в машинні коди;
- **асемблери** – це окремий випадок компілятора; перетворює програму з мови асемблера в двійкові (машинні) коди (ДК);
- **відлагоджувачі** –керуюча програма для відлагодження програми у відповідній системі.

Це питання розглянемо на прикладі мови TASM мікропроцесора i8086 [10,35…38]. Існує два типи файлів (програм), що виконуються, один із яких

закінчується розширенням COM, інший – EXE. За допомогою асемблера можна створювати програми обох типів, проте ми зупинимося на розгляді EXE – програм, як таких, що мають більші перспективи в порівнянні з COM – програмами [39].

5.2 Послідовність створення EXE – програми

5.2.1 Створення вихідної програми

Користуючись одним з екранних редакторів персонального комп'ютера (ПК) згідно з вимогами до написання асемблерних програм у EXE–форматі, створюється вихідна програма Prog.asm (рисунок 5.1).

5.2.2 Асемблерування вихідної програми

За допомогою системної програми TASM.EXE (турбоасемблер) вихідна програма Prog.asm перетворюється (асемблерується, компілюється, транслюється) у машинний код, відомий як об'єктна програма Prog.obj (рисунок 5.1).

Для цього в командному рядку диска користувача, наприклад C, набирається і виконується команда: Tasm.exe /l /zi Prog.asm.

5.2.3 Компонування об'єктної програми

Після асемблерування вихідної програми за допомогою системної програми TLINK.EXE виконується компонування об'єктної програми (рисунок 5.1).

Файл Prog.obj містить тільки машинний код у шістнадцятковій системі числення. Тому що програма може завантажуватися майже в будь–яке місце пам'яті для виконання, то асемблер може не визначити всі машинні адреси. Крім того, можуть використовуватися інші програми (підпрограми) для

об'єднання з основною. Тому призначенням програми TLINK є завершення визначення адресних посилань і об'єднання (якщо потрібно) декількох програм і створення програми, що виконується, Prog.exe.

Для цього в командному рядку диска користувача, наприклад C, набирається і виконується команда: Tlink.exe /v Prog.obj (рисунок 5.1).

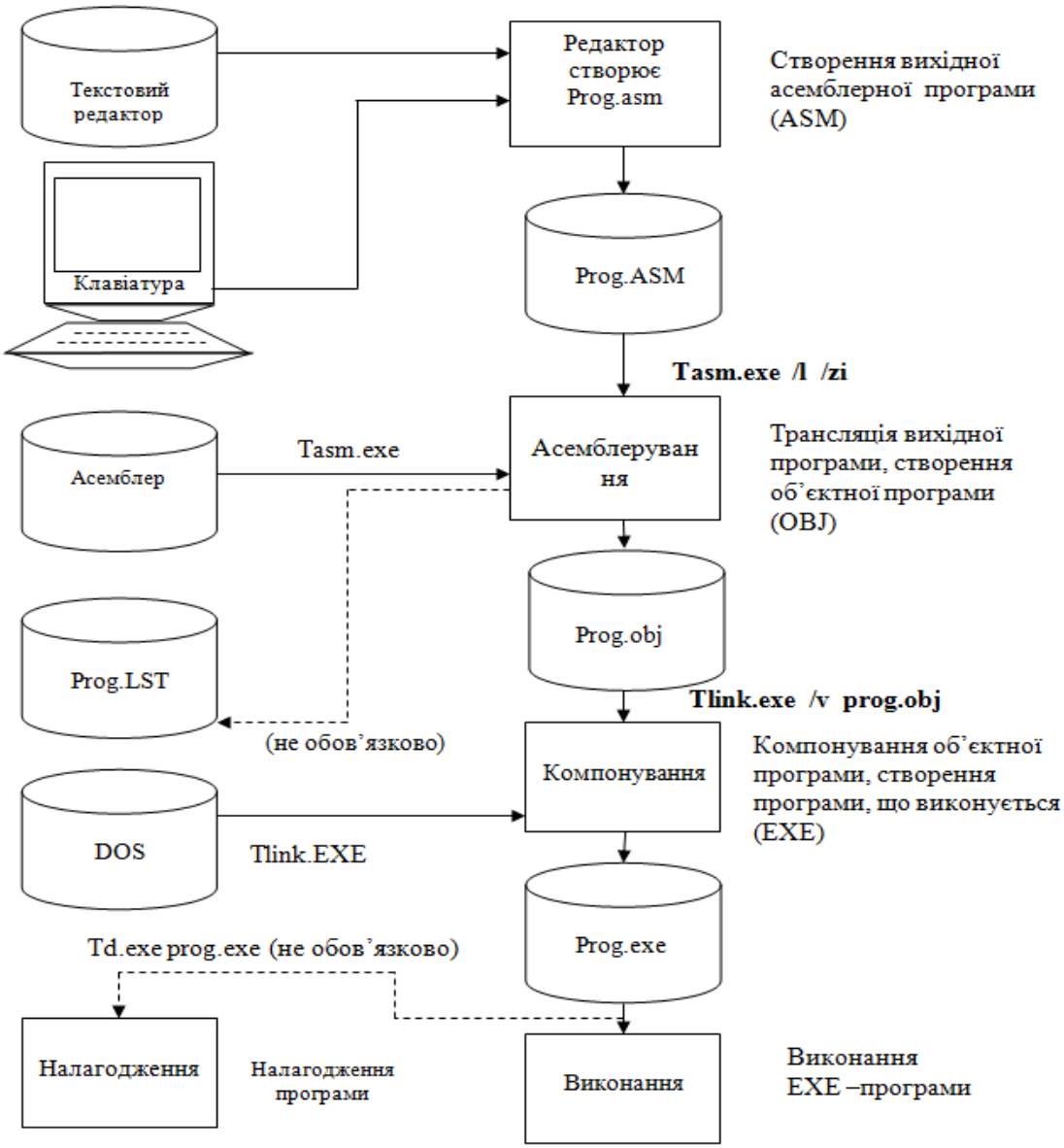


Рисунок 5.1 – Схема асемблерування, компонування, налагодження і виконання програми

5.2.4 Налагодження програми

Після одержання файлу, що виконується, Prog.exe можливе його налагодження (трасування) з метою пошуку і виправлення некоректно працюючих ділянок програми – пошуку логічних помилок.

Для входу в налагоджувач асемблера необхідно в командному рядку диска користувача, наприклад C, набрати і виконати команду: Td.exe Prog.exe (рисунок 5.1).

5.2.5 Запуск EXE – програми

Після завершення зазначених вище дій здійснюється виконання файлу Prog.exe.

5.2.6 Одержання лістингів програм

Після налагодження програми, що розроблюється, можна одержати і роздрукувати її лістинг. Для цього в команду, що виконує асемблерування вихідної програми, включають ключ: l, що викличе формування файлу Prog.lst.

Варто звернути увагу, що ніякі директиви (псевдокоманди) асемблера, а також коментарі, написані після крапки з комою, не генерують машинних кодів.

Лістинг містить не тільки вихідний текст програми, але також у лівій частині трансльовані машинні коди в шістнадцятковому форматі. Ліворуч від машинних кодів розташовані шістнадцяткові адреси команд і даних.

Описану послідовність створення і виконання EXE – програми відображає рисунок 5.1. На рисунках 5.2 і 5.3 наведено приклади відповідно вихідної асемблерної програми і її лістинг.

```
%TITLE "Приклад використання команд пересилань"
        IDEAL
        MODEL   small
        STACK   256

        DATASEG

 exCode   DB     0                   ;ім'я змінної exCode резервується для
                                     ;запису коду помилки, якщо відбудеться помилка
                                     ;і виконання програми буде перервано.
                                     ;У цьому випадку код помилки записується у
                                     ;комірці exCode і виконується команда
                                     ;JMP Exit.
 dataByte DB     99                  ;ініціалізована змінна, довжиною в
                                     ;один байт і значенням 99D
 dataWord DW     0FACEh              ;ініціалізована змінна довжиною в
                                     ;два байти(слово) і значенням FACEh
 array1   DB     0,1,2,3,4,5,6,7,8,9 ;ініціалізований масив
 array2   DB     10 DUP ('*')        ;ініціалізований масив
 storeAdr DD     ?                   ;неініціалізована змінна
                                     ;для збереження повної адреси
        CODESEG

 codeByte DB     11
 codeWord DW     0D57Fh

 Start:              ; Точка входу в програму. Визначається наприкінці
                     ; програми рядком: END Start
 ;============================= mov ==================================
        mov     ax, @data  ;ax<-@data
; @data- 16-и розрядна логічна адреса сегмента DS ;\
        mov     ds, ax     ;ds<-ax                  ; >  Ініціалізація ds i es
        mov     es, ax     ;es<-ax                  ;/

        mov     ah, 1                ;ah <- 1
        mov     bx, 0FFh             ;bx <- 0FFh

        mov     al, [dataByte]       ;al <- M(DS*16+offset dataByte )
        mov     [dataByte], ah       ;M(DS*16+offset dataByte ) <- ah
        mov     bx, offset dataWord  ;bx <- offset dataWord
        mov     si, offset dataByte  ;si <- offset dataByte

        mov     cx, ax               ;cx <- ax
        mov     dx, ds               ;dx <- ds

        mov     ax, [bx]             ;al <- M(DS*16+bx)
                                     ;ah <- M(DS*16+bx+1)
        mov     [si], cl             ;M(DS*16+si) <- cl
                                    ;пересилання у комірку пам'яті з адресою,
     ;що знаходиться в si, із регістра cl

        mov     si, 2                ;si <- 2(індекс масиву)
        mov     di, 6                ;di <- 6(індекс масиву)
        mov     [array1 + si], 16    ;M(DS*16+offset array1+si) <- 16
                                     ;array1(2)<-16
        mov     [array2 + di], 0EDh  ;M(DS*16+offset array2+di) <-0EDh
                                     ;array2(6) <- 0EDh
```

Рисунок 5.2 – Вихідна програма Prog.asm

```
        mov     ah, [array2 + di]     ;ah <- M(DS*16+offset array2 +di)
        mov     al, [byte di - 2]     ;al <- M(DS*16+di-2)
        mov     bl, [cs:codeByte]     ;bl <- M(CS*16+offset codeByte)
        mov     [cs:codeByte], bh     ;M(CS*16+offset codeByte)  <- bh

;========================== push & pop ================================

        pushf                         ; стек <- регістр прапорів
                                      ;M(SS*16 + SPпоч - 1) <- СБ RF
                                      ;M(SS*16 + SPпоч - 2) <- МБ RF
                                      ;SP<-SPпоч-2
        push    [word dataWord]       ; стек <- dataWord
                                      ;M(SS*16 + SPпоч - 1) <- СБ dataWord
                                      ;M(SS*16 + SPпоч - 2) <- МБ dataWord;
                                      ; SP<-SPпоч-2
        push    ax                    ; M(SS*16+SPпоч-1)<-ah
                                      ; M(SS*16+SPпоч-2)<-al) ;SP<-SPпоч-2
        push    bx                    ; M(SS*16+SPпоч-1)<-bh
                                      ; M(SS*16+SPпоч-2)<-bl ;SP<-SPпоч-2

        pop     ax                    ; al<-M(SS*16+SPпоч) = bl
                                      ; ah<-M(SS*16+SPпоч+1) = bh ;SP<-(SPпоч+2)
        pop     bx                    ; bl<-M(SS*16+SPпоч) = al
                                      ; bh<-M(SS*16+SPпоч+1) = ah  ;SP<-(SP+2)
                              ;відбувся обмін регістрів ax(акумулятор) і bx(база)
        push    ds                    ; стек <- ds
                                      ; M(SS*16+SPпоч-1)<- СБ DS
                                      ; M(SS*16+SPпоч-2)<- МБ DS: SP<-SPпоч-2
        pop     es                    ; ES <- стек
                                      ; ES(l) <- M(SS*16+SPпоч)
                                      ; ES(h) <- M(SS*16+SPпоч+1)
                                      ; SP<-SPпоч+2
     ;у такий спосіб додатковий сегмент(es) цілком повторив сегмент даних(ds)
        pop     [word cs:codeWord]; M(CS*16+offset codeWord )<-M(SS*16+SPпоч)
                                      ; M(CS*16+offset codeWord+1 )<-M(SS*16+SPпоч+1)
                                      ; SP <- SPпоч + 2
        popf                          ;регістр прапорів <- стек


;============================== xchg ==================================

        xchg    ax, ax                ;ax<->ax ;порожня операція(теж саме, що і nop)

        xchg    bx, ax                ;bx <-> ax ;обмін вмісту bx і ax
        xchg    cl, dh                ;cl <-> dh ;обмін вмісту cl і dh

        xchg    [byte dataByte], ah    ;M(DS*16+offset dataByte )<->ah
        xchg    bx, [word cs:codeWord] ;bl <->M(CX*16+offset codeWord )
                                       ;bh <->M(CX*16+offset codeWord+1 )
        xchg    ch, [array1 + 4]       ;ch <-> M(DS*16+offset array1 +4)

;========================== xlat & xlatb ==============================

        mov     al, 2                 ; al<-2
        mov     bx, offset array1     ;bx <- offset array1
        xlatb                         ;al <- M(DS*16 + bx + al), al <- array1(2
```

Продовження рисунка 5.2

```
        mov     al, 6                   ;al <- 6
        mov     bx, offset array2       ;bx <- offset array2
        xlat    [es:bx]                 ;al <- M(ES*16 + bx + al)

 ;=============================== lea ==================================

        lea     bx, [array1]          ;bx <- EA=offset array1
        lea     bx, [array2 + 2]      ;bx <- EA=offset array2+2
        mov     si, 4                 ;si <- 4
        lea     bx, [array1 + si - 1] ;bx <- EA=offset array1+si-1

;=============================== lds ==================================
 ;Приклад передачі параметрів через стек ;Безпосередня адресація не допускається
         push   cs                      ; Записуємо в стек адресу сегмента
                                        ; змінної codeByte (сегмент CS)
         mov    ax, offset codeByte     ; Записуємо в стек зсув
         push   ax                      ; змінної codeByte
 ;        ...                            ;
 ;        ...                            ;   Тут може йти інший код
          mov    bp, sp                  ; bp<-sp
         lds    si, [bp]                ;  SI l<- M(SS*16+bp)
                                        ;  SI h<- M(SS*16+bp+1)
                                        ;  DS l<- M(SS*16+bp+2)
                                        ;  DS h<- M(SS*16+bp+3)
                                        ; Відновлюємо в:
                                        ; DS - вміст CS;
                                        ; SI - зсув codeByte.
                                        ; Тепер у ds:si знаходиться адреса
                                        ; змінної codeByte
 ;=============================== les ==================================
 ;Приклад збереження і  відновлення повної адреси змінної codeByte
 ;Безпосередня адресація не допускається
      mov    [word storeAdr + 2],@code ;Зберігаємо адресу сегмента CS (@code)
                                  ; M(DS*16+offset storeAdr  + 2) <- CS l
                                  ; M(DS*16+offset storeAdr  + 2 +1) <- CS h
      mov    [word storeAdr], offset codeByte ;Зберігаємо зсув codeByte
                             ; M(DS*16+offset storeAdr) <-offset CodeByte l
                             ; M(DS*16+offset storeAdr+1) <-offset CodeByte h

 ;        ...                            ;
 ;        ...                            ;    Інший код
 ;        ...                            ;
         mov    bx, offset storeAdr      ;bx<-offset storeAdr
         les    si, [bx]            ; SI l<-M(DS*16+bx)= МБ offset CodeByte
                                    ; SI h<-M(DS*16+bx+1)= СБ offset CodeByte
                                         ; ES l<-M(DS*16+bx+2)= МБ CS
                                         ; ES h<-M(DS*16+bx+3)= СБ CS
                                         ; відновлюємо в:
                                         ; es - вміст cs;
                                         ; si - зсув CodeByte
;========================== lahf & sahf ===================================
 ;
        lahf                    ;ah <- F(7...0)
        xor     ah, 0FFh        ;ah <- ah xor 0FFh ;Інвертуємо всі прапорці
        sahf                    ;F(7...0) <- ah

 ;=========================================================================
```

Продовження рисунка 5.2

```
Exit:                       ;стандартний вихід у DOS із програми, що запускається
        mov     ah,04Ch         ;ah <-04Ch
                                ;4Ch – функція DOS (завершення підпроцесу
                                ;з поверненням керування програмі command.com)

        mov     al,[exCode]     ;al <- M(DS*16+offset exCode)
                                ;запис у al коду помилки

        int     21h             ;переривання DOS, що викликає функцію
                                ;DOS, обумовлену кодом у регістрі AH
        END Start                ;кінець програми / точка входу
```

Продовження рисунка 5.2

```
 1                                IDEAL
 2   0000                         MODEL   small
 3   0000                         STACK   256
 4
 5   0100                         DATASEG
 6
 7   0000  00   exCode  DB  0        ;ім'я змінної exCode резервується для
 8                                   ;запису коду помилки, якщо відбудеться помилка
 9                                   ;і виконання    програми буде     перервано.
10                                   ;У цьому випадку      код помилки  записується в
11                                   ;комірці exCode і виконується команда
12                                   ;JMP Exit.
13   0001  63   dataByte DB    99    ;ініціалізована змінна, довжиною в
14                                   ;один     байт і значенням 99D
15   0002  FACE  dataWord DW  0FACEh       ;ініціалізована змінна довжиною  в
16                                         ;два байти(слово) і значенням    FACEh
17 0004 00 01 02 03 04 05      06+  array1 DB 0,1,2,3,4,5,6,7,8,9 ;ініціалізований масив
18       07 08 09
19   000E  0A*(2A)    array2   DB     10 DUP ('*')    ; ініціалізований масив
20   0018  ????????  storeAdr DD     ?          ;неініціалізована змінна
21                                              ;для збереження повної      адреси
22   001C                         CODESEG
23
24   0000  0B               codeByte DB     11
25   0001  D57F             codeWord DW     0D57Fh
26
27   0003       Start:         ; Точка входу   в програму. Визначається наприкінці
28                                       ; програми рядком: END Start
29     ;============================== mov ==================================
30   0003  B8 0000s                      mov     ax, @data  ;ax<-@data
31                          ; @data- 16-и розрядна логічна адреса сегмента DS ;\
32   0006  8E D8    mov ds, ax     ;ds<-ax       ; >  Ініціалізація ds і es
33   0008  8E C0    mov     es, ax ;es<-ax       ;/
34
35   000A  B4 01    mov     ah, 1          ;ah <- 1
36   000C  BB 00FF  mov     bx, 0FFh             ;bx <- 0FFh
37
38   000F  A0 0001r  mov     al, [dataByte]       ;al <- M(DS*16+offset dataByte )
39   0012  88 26 0001r  mov     [dataByte], ah    ;M(DS*16+offset dataByte ) <- ah
40   0016  BB 0002r     mov     bx, offset     dataWord   ;bx <- offset dataWord
41   0019  BE 0001r     mov     si, offset     dataByte   ;si <- offset dataByte
42
43   001C  8B C8        mov     cx, ax           ;cx <- ax
44   001E  8C DA        mov     dx, ds           ;dx <- ds
45
```

Рисунок 5.3 – Лістинг програми Prog.asm

```
 46   0020  8B 07        mov     ax, [bx]          ;al <- M(DS*16+bx)
 47                                                ;ah <- M(DS*16+bx+1)
 48 0022  88 0C          mov     [si], cl          ;M(DS*16+si)   <- cl
                                                   ;пересилка у комірку пам'яті з
                                                   ;адресою, що знаходиться в si,із
                                                   ;регістра cl
 52   0024  BE 0002      mov     si, 2             ;si <- 2(індекс масиву)
 53   0027  BF 0006      mov     di, 6             ;di <- 6(індекс масиву)
 54   002A  C6 84 0004r 10  mov[array1 + si], 16  ;M(DS*16+offset array1+si) <- 16
 55                                                      ;array1(2)<-16
 56   002F  C6 85 000Er ED  mov[array2 + di], 0EDh  ;M(DS*16+offset array2+di) <-0EDh
 57                                                      ;array2(6) <- 0EDh
 58   0034  8A 45 FE        mov al,[byte di - 2]   ;al <- M(DS*16+di-2)
 59   0037  8A A5 000Er     mov ah,[array2 + di]   ;ah <- M(DS*16+offset array2+di)
 60
 61   003B  2E: 8A 1E   0000r mov bl,[cs:codeByte]   ;bl <- M(CS*16+offset codeByte )
 62   0040  2E: 88 3E   0000r mov [cs:codeByte],bh    ;M(CS*16+offset codeByte)<- bh
 63
 64    ;========================== push & pop ==================================
 65
 66   0045  9C         pushf   ; стек <- регістр прапорів
 67                             ;M(SS*16   + SPпоч     - 1) <-     СБ RF
 68                             ;M(SS*16   + SPпоч     - 2) <-     МБ RF
 69                             ;SP<-Spпоч-2
 70   0046  FF 36 0002r  push    [word dataWord]   ; стек <- dataWord
 71                             ;M(SS*16   + SPпоч     - 1) <-     СБ dataWord
 72                             ;M(SS*16   + SPпоч     - 2) <-     МБ  dataWord;
 73                             ;SP<-Spпоч-2
 74   004A  50         push    ax ; M(SS*16+SPпоч-1)<-ah
 75                                ; M(SS*16+SPпоч-2)<-al ;SP<-SPпоч-2
 76   004B  53         push    bx ; M(SS*16+SPпоч-1)<-bh
 77                                ; M(SS*16+SPпоч-2)<-bl ;SP<-SPпоч-2
 78
 79   004C  58         pop     ax    ; al<-M(SS*16+SPпоч)= bl
 80                                   ; ah<-M(SS*16+SPпоч+1) = bh ;SP<-(SPпоч+2)
 81   004D  5B         pop     bx    ; bl<-M(SS*16+SPпоч) = al
 82                                   ; bh<-M(SS*16+SPпоч+1) = ah ;SP<-(SP+2)
 83                    ;відбувся    обмін регістрів   ax(акумулятор)    і bx(база)
 84
 85    004E  1E          push    ds  ; стек <- ds
 86                                  ; M(SS*16+SPпоч-1)<- СБ DS
                                     ; M(SS*16+SPпоч-2)<- МБ DS
                                     ;SP<-Spпоч-2
 88    004F  07          pop     es  ; ES <- стек
 89                                  ; ES(l) <- M(SS*16+SPпоч)
 90                                  ; ES(h) <- M(SS*16+SPпоч+1); SP<-SPпоч+2
 91   ;у такий спосіб додатковий сегмент(es) цілком повторив сегмент даних(ds)
 92
 93 0050 2E:8F 06 0001r  pop[word cs:codeWord];M(CS*16+offset codeWord )<-M(SS*16+SPпоч)
 94                                 ; M(CS*16+offset codeWord+1 )<-M(SS*16+SPпоч+1)
 95                                 ; SP <- SPпоч + 2
 96    0055  9D          popf      ;регістр      прапорів <- стек
 97
 98
 99   ;============================== xchg ====================================
100
101    0056  90  xchg  ax, ax         ;ax<->ax   ;порожня операція(теж, що і nop)
```

Продовження рисунка 5.3

```
103    0057  93  xchg    bx, ax       ;bx <-> ax ;обмін вмісту bx  i ax
104    0058  86 CE   xchg    cl, dh   ;cl <-> dh ;обмін вмісту cl i dh
105
106    005A  86 26 0001r xchg    [byte dataByte], ah   ;M(DS*16+offset dataByte )<->ah
107    005E  2E:87 1E 0001r xchg bx,[word cs:codeWord] ;bl<->M(CS*16+offset codeWord)
108                                          ;bh     <->M(CS*16+offset codeWord+1 )
109    0063  86 2E 0008r xchg ch,[array1 + 4]     ;ch  <-> M(DS*16+offset array1 +4)
110
111  ;============================ xlat & xlatb  ================================
112
113    0067  B0 02     mov     al, 2          ; al<-2
114    0069  BB 0004r  mov     bx, offset  array1 ;bx  <- offset array1
115    006C  D7        xlatb                  ;al     <- M(DS*16 + bx   + al)
116                                           ;al     <- array1(2)
117    006D  B0 06     mov     al, 6          ;al     <- 6
118    006F  BB 000Er  mov     bx, offset  array2 ;bx  <- offset array2
119    0072  26: D7    xlat    [es:bx]           ;al  <- M(ES*16 + bx   + al)
120
121  ;================================    lea ===================================
122
123    0074  BB 0004r    lea     bx, [array1]      ;bx <- EA=offset array1
124    0077  BB 0010r    lea     bx, [array2 + 2]   ;bx <- EA=offset array2+2
125    007A  BE 0004       mov     si, 4           ;si <- 4
126    007D  8D 9C 0003r    lea     bx, [array1 + si – 1] ;bx <- EA=offset array1+si-1
127
128  ;================================    lds ===================================
129       ;Приклад передачі параметрів через    стек
130       ;Безпосередня  адресація не допускається
131    0081  0E     push   cs                  ; Записуємо в стек адресу сегмента
132                                      ; змінної codeByte (сегмент CS)
133    0082  B8 0000r  mov    ax, offset  codeByte     ; Записуємо в стек зсув
134    0085  50                  push   ax                 ; змінної codeByte
135                             ;            ...                  ;
136                             ;            ...        ;      Тут може йти інший
                 ;             ...                    ;          код
138    0086  8B EC                   mov    bp, sp               ; bp<-sp
139    0088  C5 76 00     lds    si, [bp]            ;      SI l <-     M(SS*16+bp)
140                                                  ;      SI h<- M(SS*16+bp+1)
141                                                  ;      DS l<- M(SS*16+bp+2)
142                                                  ;      DS h<- M(SS*16+bp+3)
143                                                  ; Відновлюємо в:
144                                                  ; DS – вміст CS;
145                                                  ; SI – зсув codeByte.
146                                          ; Тепер у    ds:si знаходиться адреса
147                                                  ; змінної codeByte
148  ;============================ les ===================================
149    ;Приклад збереження і  відновлення повної адреси змінної codeByte
150                             ; Безпосередня    адресація не допускається
  151  008B  C7 06 001Ar 0000s mov[word storeAdr + 2],@code ;Зберігаємо адресу сегмента
                                                          ;CS(@code)
152                                  ; M(DS*16+offset storeAdr  + 2) <- CS l
153                 ; M(DS*16+offset storeAdr  + 2    +1) <- CS h
154 0091 C7 06 0018r 0000r  mov[word storeAdr], offset codeByte
                              ;Зберігаємо зсув
                              ;codeByte
155                           ; M(DS*16+offset storeAdr) <-offset CodeByte l
156                           ; M(DS*16+offset storeAdr+1) <-offset CodeByteh
```

Продовження рисунка 5.3

```
158                ;            ...                    ;
159                ;            ...                    ;        Інший код
160                            ;            ...                  ;
161    0097  BB 0018r    mov    bx, offset storeAdr      ;bx<-offset storeAdr
162    009A  C4 37       les    si, [bx]; SI l<-M(DS*16+bx)= МБ offset  CodeByte
163                                 ; SI h<-M(DS*16+bx+1)= СБ offset CodeByte
164                                      ;     ES l<-M(DS*16+bx+2)= МБ CS
165                                      ;     ES h<-M(DS*16+bx+3)= СБ CS
166                                      ;     відновлюємо в:
167                                      ;     es – вміст  cs;
168                                      ;     si – зсув CodeByte
169  ;======================== lahf & sahf ===================================
170                            ;
171    009C  9F               lahf               ;ah <- F(7...0)
172    009D  80 F4 FF  xor ah, 0FFh    ;ah <- ah xor 0FFh ;Інвертуємо всі прапорці
173    00A0  9E                     sahf              ;F(7...0) <- ah
174
175 ;=============================================================================
176    00A1 Exit:        ;стандартний вихід у DOS із  програми, що запускається
177    00A1  B4 4C                  mov     ah,04Ch ;ah <-04Ch
178                                 ;4Ch – функція DOS (завершення підпроцесу
179                               ;з поверненням керування програмі command.com)
180
181    00A3  A0 0000r     mov     al,[exCode]  ;al <- M(DS*16+offset exCode)
182                                            ;запис у al коду помилки
183
184    00A6  CD 21     int     21h        ;переривання DOS, що викликає функцію
185                                       ;DOS, обумовлену  кодом у     регістрі AH
186                               END Start ; кінець програми / точка входу
```

Продовження рисунка 5.3

5.3 Шаблон для створення вихідних exe – програм

Замість того, щоб розробляти нову програму з нуля, корисно використовувати шаблон, що містить найнеобхідніше для написання EXE – програм (рисунок 5.4).

```
%TITLE “Оболонка для EXE – файлів”

        IDEAL
        MODEL  small
        STACK  256

;————————————— Вставте тут директиви INCLUDE “filename”

;————————————— Вставте тут макровизначення EQU та/або =
```

Рисунок 5.4 – Оболонка (шаблон) для EXE – файлів

```
DATASEG

;————————————— Якщо виникне помилка і програма вимушена буде перерватися,
;               запишіть відповідний код помилки в exCode і виконайте команду JMP Exit
;               Щоб це можливо було зробити з підмодуля, опишіть позначку Exit
;               за допомогою директиви EXTRN

;
exCode     DB    0

;————————————— Тут опишіть інші змінні за допомогою DB, DW тощо

;————————————— Тут опишіть всі змінні типу EXTRN

          CODESEG
;————————————— Тут визначіть всі підпрограми типу EXTRN

Start:
          mov ax, @data          ;Пересилання  в DS адреси
          mov ds, ax             ;сегмента даних(@data)
          mov es, ax             ;Встановлення es=ds
;————————————— Тут розташовується програма, виклик підпрограм тощо

Exit:
          mov ah, 04ch           ;Функція DOS (04Ch) : вихід з програми
          mov al, [exCode]       ;Повернення значення коду виходу (коду помилки)
          int 21h                ;Виклик DOS. Зупинка програми

          END Start              ;Кінець програми / точка входу
```

Продовження рисунка 5.4

Більшість програм мовою асемблера можна розділити на п'ять основних частин [36…39]:

- заголовок;
- макровизначення;
- сегмент даних;
- тіло програми;
- заключення.

5.3.1 Заголовок

Програма на мові асемблера починається з заголовка. У ньому містяться команди і директиви, що не приводять до створення машинного

коду при трансляції. Вони вказують асемблеру, як виконати визначені дії, генеруючи файл, що виконується.

На рисунку 5.5 представлено простий заголовок, характерний для більшості EXE – програм. Це фрагмент програми, так що не намагайтеся його транслювати.

Не обов'язковий рядок %TITLE описує призначення програми, і при роздрукуванні її в Turbo Assembler текст, що знаходиться в лапках, буде розташовуватися на початку кожної сторінки вихідного тексту. Директива IDEAL переводить Turbo Assembler у режим Ideal.

Якщо програма написана на Microsoft Macro Assembler (MASM), директиву Ideal потрібно опустити.

```
%TITLE “Текстовий заголовок – Не асемблеруйте ”

IDEAL
MODEL small

STACK 256
```

Рисунок 5.5 – Типовий заголовок асемблерної програми

Далі йде директива MODEL, що вибирає одну з декількох моделей пам'яті (таблиця 5.1), багато з яких використовуються при написанні комбінованих асемблерних програм з мовами C або Pascal. Найбільш вдалим вибором при програмуванні мовою асемблера є small (мала) модель пам'яті. Мала модель пам'яті виділяє програмі 64 Кбайт – для коду, що виконується, і 64 Кбайт – для даних, і дозволяє створювати програми розміром до 128 Кбайт, що практично є межею ефективності у світі машинних кодів.

Директива STACK на рисунку 5.5 резервує простір для стека програми – області пам'яті, у якій містяться два типи даних: проміжні значення, що записуються підпрограмами чи передаються їм, а також

адреси, на які підпрограми повертають керування. Робота зі стеком є важливою частиною технології програмування на асемблері. Значення, що йде за директивою STACK на рисунку 5.5, визначає кількість байтів (256), яке буде зарезервована Turbo Assembler під сегмент стека. В основному програми використовують невеликий стек, і навіть найбільші з них рідко вимагають більш 8 Кбайт.

Таблиця 5.1 – Моделі пам'яті

Назва	**Опис**
Tiny	Код, дані і стек містяться в одному сегменті 64 Кбайт. Використовується тільки для COM – програм
Small	Код і дані містяться в різних сегментах, розміром до 64 Кбайт. Використовується для невеликих і середніх EXE – програм. Щонайкраще підходить для більшості чисто асемблерних програм.
Medium	Необмежений розмір коду (більше одного сегмента, у межах можливостей ОС). Під дані виділяється один сегмент 64 Кбайт. Використовується для написання великих програм з невеликим обсягом даних
Compact	Розмір коду обмежений одним сегментом 64 Кбайт. Розмір даних необмежений (більше одного сегмента, у межах можливостей ОС). Використовується при написанні малих і середніх за розміром програм з великою кількістю змінних
Large	Розмір коду і даних необмежений (більше одного сегмента, у межах можливостей ОС). Використовується у великих програмах. Розмір змінної не може перевищувати 64 Кбайт
Huge	Розмір коду і даних необмежений (більше одного сегмента, у межах можливостей ОС). Аналогічна великій моделі пам'яті. (Введена для сумісності з мовами високого рівня)
Tchuge	Збігається з великою моделлю пам'яті, проте по іншому визначає регістри. В основному використовується при програмуванні на мовах Turbo C і Borland C++
Tpascal	Забезпечує підтримку ранніх версій Turbo Pascal. У Borland Pascal не застосовується
Flat	Використовується тільки в середовищі OS/2; в іншому аналогічна малій моделі пам'яті

5.3.2 Макровизначення

Після заголовка програми йдуть різні описи констант і змінних. У мові асемблера константи часто називають макровизначеннями, що використовують директиву EQU, яка зв'язує значення з ідентифікатором (визначеним ім'ям, замість якого в програмі Turbo Assembler підставить відповідне значення). Винятково для числових значень, крім директиви EQU, можна застосовувати знак рівності (=).

Макровизначення можуть розташовуватися в будь–якому місці програми, але щоб ваша програма легко читалася, розміщуйте макровизначення відразу після заголовка програми.

Використання ідентифікаторів макровизначень замість “магічних” чисел 0100h або 0B00h дозволяє вам звертатися за ім'ям до виразів, рядків та інших величин, що спрощує програму для читання і налагодження.

Нижче подано декілька прикладів макровизначень, які можуть йти за заголовком, наведеним на рисунку 5.6:

```
Count       EQU     10,
Element     EQU     5,
Size        =       Count * Element,
MyBoat      EQU     “Gypsy Venus”,
Size        =       0.
```

Рисунок 5.6 – Приклади макровизначень

Незважаючи на те, що більшість імен макровизначень просто замінюється відповідними значеннями і виразами – аналогічно використанню констант у мовах Pascal і C, існує кілька суворих правил, які треба пам'ятати при створенні і використанні макровизначень у мові асемблера:

– після опису імені константи за допомогою директиви EQU не можна змінювати його значення. Перевизначення цієї константи в рівності не допускається (наприклад, зміна значення Count на 11);
– це обмеження не поширюється на імена−ідентифікатори, описані за допомогою знака рівності (=),– ви можете вільно змінювати їхні значення. Зверніть увагу, як у розглянутому вище прикладі значення Size змінюється з 50 на 0. Це можна здійснювати в будь−якім місці програми, а не тільки в секції рівностей;
– EQU може описувати всі типи рівностей, включаючи числа, вирази і символьні рядки. Знак рівності може описувати тільки числові рівності, що складаються з чисел 10 чи 0Fh або виразів: Count*Size чи Address+2;
– ідентифікатори макровизначень рівностей не є змінними – ні вони, ні їхні значення не містяться в сегменті даних програми. Команди асемблера не можуть змінити значення ідентифікатора в макровизначеннях, незалежно від того, чи були вони описані за допомогою директиви EQU чи знака: = ;
– хоча ви можете розташовувати макровизначення в будь−якім місці програми, але краще віддавати перевагу їх розміщенню на початку тексту. Макровизначення, розташовані глибоко в тексті програми, можуть стать джерелом помилок, що важко виявляються;
– вирази, описані за допомогою EQU, обчислюються, коли відповідне ім'я−ідентифікатор використовується програмою. Вирази, описані через знак рівності (=), обчислюються безпосередньо в місці визначення. Асемблер зберігає *текст* EQU−виразу, а для виразу, описаного за допомогою знака рівності, – тільки його *значення*.

Для кращого розуміння останнього правила розглянемо кілька прикладів. Припустимо, що є три макровизначення:

```
LinesPerPage    =    66,
NumPages        =    100,
TotalLines      =    LinesPerPage * NumPages.
```

Очевидно, что TotalLines дорівнює добутку LinePerPage і NumPages, чи 6600. У більшості мов програмування зірочка “*” означає множення. Оскільки TotalLines описано за допомогою знака рівності (=) – мається на увазі числове значення, вираз обчислюється відразу і результат ставиться у відповідність TotalLines.

Якщо де–небудь у програмі ви зміните значення NumPages, значення TotalLines залишиться тим же. Ситуація зміниться, якщо ви опишете TotalLines за допомогою EQU:

```
TotalLines      EQU     LinePerPage*NumPages.
```

У цьому випадку Turbo Assembler збереже не значення, що обчислюється, а дійсний текст виразу, що йде за директивою EQU (у нашому випадку – текст виразу LinePerPage * NumPages). Пізніше в програмі, коли ви будете використовувати TotalLines, асемблер вставить цей текст так, начебто ви набирали його в цьому місці вихідного тексту, потім вираз буде обчислено і замінено кінцевим значенням. Якщо перед цим ви змінили значення одного чи обох ідентифікаторів, що використовуються у виразі (NumPages чи LinePerPage), отриманий результат також відповідно зміниться.

Цей вплив на значення виразу макровизначення може в ряді випадків виявитися корисним. Ви можете створити один програмний модуль, на значення виразів макровизначень якого будуть впливати макровизначення іншого модуля. Потрібно розуміти тонку різницю між макровизначеннями,

визначеними за допомогою знака рівності і директиви EQU. Їхнє недбале використання також може приводити до виникнення помилок.

5.3.3 Сегмент даних

Сегмент даних звичайно розташовується між визначеннями і командами програми. Можна, хоча це рідко буває корисним, визначати сегмент даних у довільному місці програми чи мати кілька сегментів даних у різних частинах програмного тексту. Незважаючи на ці можливості, ваша асемблерна програма буде простіша для розуміння і модифікації, якщо ви наслідуєте простим рекомендаціям, запропонованим нижче, визначаючи усі свої змінні між макровизначеннями і кодом, що виконується.

Сегмент даних вашої програми повинний починатися з директиви DATASEG. Вона дає вказівку асемблеру розмістити в пам'яті змінні, зазначені в сегменті даних програми, що може досягати 64 Кбайт для малої моделі пам'яті. Сегмент даних може містити два типи змінних: *ініціалізовані* і *неініціалізовані*. При виконанні програми ініціалізовані змінні мають визначені значення, які ви визначили в тексті програми, і містяться у файлі програмного коду на диску. Ці змінні автоматично завантажуються в пам'ять і доступні для читання при виконанні програми. Неініціалізовані змінні аналогічні ініціалізованим, за винятком того, що вони не займають простору у файлі, що виконується, і, отже, мають невизначені значення при виконанні програми. Внаслідок цього визначення великої неініціалізованої змінної для резервування в пам'яті великого буфера, заповнюваного з дискового файлу, не приводить до збільшення розміру файлу програми, що виконується.

Для того, щоб неініціалізовані змінні не містилися в об'єктному коді, вони повинні описуватися після останньої ініціалізованої змінної у вихідному тексті програми. Неініціалізовані змінні, описані між іншими

ініціалізованими змінними, будуть займати місце в об'єктному коді, тим самим збільшуючи на диску розмір файлу, що виконується.

5.3.4 Резервування простору під змінні

Правила опису змінних у сегменті даних програми докладно розглядається в [39]. Нижче розглянуто декілька простих прикладів.

Розглянемо типовий сегмент даних, що може йти після заголовка програми і макровизначень:

```
DATASEG
numRows         DB      25,
numColumns      DB      80,
videoBase       DW      0B00h.
```

Першою йде директива DATASEG, що інформує асемблер про необхідність виділення простору в пам'яті під сегмент даних програми. Потім визначено три змінні: numRows, numColumns і videoBase. Як правило, рекомендується записувати константи (Count, NumPages тощо) з великої літери, а змінні – з малої літери. Це необов'язкова угода, ви можете набирати символи на власний розсуд або великими, або малими літерами. Деякі програмісти використовують символ підкреслення для поліпшення прочитання назв змінних, що складаються з декількох слів, наприклад, записуючи num_rows і video_base замість злитого написання, що використано тут.

DB (визначити байт) і DW (визначити слово) – дві найчастіше використовувані директиви, що застосовуються для виділення простору під відповідні змінні програми. На відміну від мов високого рівня, у яких розташування змінних у пам'яті звичайно не є важливим, у мові асемблера необхідно виділяти простір у пам'яті під змінні і, у випадку ініціалізованих змінних, приписувати значення цьому простору. Відчуйте, наскільки це

відрізняється від ідентифікаторів макровизначень, що асоціюються із змінними виключно у вихідному тексті програми. Для них, на відміну від змінних, не виділяється простір у сегменті даних пам'яті.

Імена змінних (numRows, numColumns і videoBase) є позначками, що вказують на ділянки пам'яті, що виділяються – у даному випадку, на простір, що резервується під значення змінних. Програма може звертатися до цього простору, використовуючи позначку як покажчик на відповідне значення в пам'яті. В асемблерних програмах позначки перетворюються в адреси пам'яті, за якими містяться змінні, що дозволяє вам звертатися до пам'яті за іменами, а не за чисельними адресами.

При програмуванні безпосередньо в машинних кодах ви повинні замість позначок визначати дійсні адреси. Використання символьних позначок для визначення розташування змінних у пам'яті є однією з основних переваг мови асемблера.

Змінні розташовуються безпосередньо одна за одною. Знаючи це, можна чинити різні “штуки”. Наприклад, здається, що визначення

```
DATASEG
aTOm        DB      “ABCDEFGHIJKLM”
nTOz        DB      “NOPQRSTUVWXYZ”
```

створюють два символьних рядки з позначками aTOm і nTOz. Однак у пам'яті символи від A до Z зберігаються послідовно, створюючи один рядок букв алфавіту. Позначка nTOz просто вказує на середину рядка, при цьому в пам'яті не створюються два окремих фрагменти.

DB має спеціальну можливість визначати багатобайтові значення від одного до необхідної вам кількості байтів. Рядок складається з окремих ASCII–символів, кожний з яких займає один байт, отже, директива DB є в асемблері простим інструментом для визначення рядків символів і байт, розділяючи їх комами:

```
DATASEG
```

```
perfectTen      DB      1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
theTime         DB      9, 0            ; тобто 9:00,
theDate         DB      12, 15, 98      ;тобто 12/15/1998.
```

Крім того, можна комбінувати символьні і байтові значення, створюючи дворядкові змінні з ASCII–символами повернення каретки (13) і кінця рядка (10). Наступний приклад показує, що символьні рядки можуть міститись як в одинарних, так і в подвійних лапках:

```
combo           DB      'Line#1', 13, 10, "Line#2".
```

Деякі мови, зокрема Pascal, розрізняють одинарні символи і рядки символів. В асемблері символи і рядки відрізняються винятково довжиною. В асемблерних рядках відсутні такі допоміжні значення як байт довжини чи символ кінця рядка, якщо, звичайно ви самі не розмістили їх там.

Рядки розглядаються окремо при вивченні команд асемблера, спеціально створених для роботи з ними. Зараз необхідно запам'ятати, що на відміну від мов високого рівня, рядки в асемблері є просто набором послідовних значень у пам'яті, створених директивою DB.

5.3.5 Тіло програми

Після сегмента даних розташовується тіло програми, відоме за назвою кодового сегмента – області пам'яті, що містить код асемблерної програми, яка виконується. Всередині цієї області можна виділити чотири стовпчики тексту: позначки, мнемоніки, операнди і коментарі. Кожен стовпчик має свою важливу функцію, яку краще розглянути на прикладі. Кількість пропусків між колонками в тексті програми довільна. Найчастіше стовпчики вирівнюють за допомогою одно– чи дворазового натискання клавіші табуляції в редакторі.

Якщо ваш редактор дозволяє вибирати між вставкою символів табуляції і вставкою пропусків, використовуйте табуляцію, задавши її в

кожному восьмому стовпчику, що є стандартною установкою для більшості редакторів. Вставка символів табуляції дозволяє легко вирівнювати стовпчики. При цьому ви зможете повторно редагувати текст кожного стовпчика, не зміщуючи текст в інших колонках. Звичайно, при бажанні, ви можете вставляти і пропуски між колонками – асемблер ніяких розходжень при цьому не робить.

Розглянемо приклад сегментів даних і коду і виділимо в ньому чотири стовпчики (рисунок 5.7).

```
Позначка   Мнемоніка       Операнд    Коментар
           DATASEG
ExCode     DB              0          ;Байтова змінна
           CODESEG
Start:     mov             ax, @data  ;Пересилання в DS адреси
           mov             ds, ax     ;сегмента даних
           jmp             Exit       ;Перехід на позначку Exit
           mov             cx, 10     ;Цей рядок пропускається!
Exit:      mov             ah, 04Ch   ;Функція DOS: вихід з програми
           mov             al, [exCode] ;Повертає код виходу програми
                ;в AL
           int             21h        ;Виклик DOS. Зупинка програми
           END             Start      ;Кінець програми/точка входу
```

Рисунок 5.7 – Чотири стовпчики в програмі мовою асемблера

Це незакінчена програма, так що не намагайтеся її транслювати. Вона містить основні елементи кодового сегмента мови Асемблера. У сегменті даних для подальшого використання визначається однобайтова змінна з ім'ям exCode зі значенням 0.

Після директиви CODESEG на рисунку 5.7 представлено кілька рядків, розділених на позначки, мнемоніку, операнди і стовпчик коментарів. У першому стовпчику знаходяться дві позначки – Start: і Exit:. Позначки позначають місця в програмі, на які можуть посилатися інші команди і директиви. Для рядків без позначок цей стовпчик не заповнюється. У кодовому сегменті позначка завжди закінчується двокрапкою (:), у сегменті

даних двокрапка не використовується (наприклад, у позначці exCode). Просто запам'ятаєте це правило, за яким не криється будь–який логічний зміст.

В другому стовпчику містяться мнемоніки. Під кожним мнемонічним формулюванням у цьому стовпчику ховається одна машинна команда: mov – для Move (пересилання даних), jmp – для Jump (безумовний перехід), int – для Interrupt (переривання) і т. ін. Деякі мнемоніки запам'ятовуються легко: dec – для Decrement (декремент), shl – для Shift Left (зсув вліво), ror – для Rotate Right (циклічний зсув вправо). Інші здаються справою рук божевільної друкарки: jcxz – для Jump if cx is Zero (перехід, якщо CX дорівнює 0) і rcr – Rotate through Carry Right (циклічний зсув вправо через перенесення). У деяких випадках мнемоніка цілком збігається з командою: out – для Out (виведення байта чи слова до порту), push – для Push (занесення слова до стека), pop – для Pop (діставання слова зі стека).

Третій стовпчик на рисунку 5.7 містить операнди, що обробляються мнемонічними командами. Деякі команди не вимагають операндів, у цьому випадку третій стовпчик залишається порожнім. Багато команд вимагають двох операндів, інші – тільки одного. Жодна з команд процесора i8086 не вимагає більше двох операндів. Перший операнд звичайно називається призначенням, другий – джерелом. Операнди можуть мати різний вигляд і вивчаються при розгляді кожної конкретної мнемонічної команди [38].

Четверта (остання) колонка є необов'язковою і, якщо включається в програму, то повинна починатися з крапки з комою (;). Turbo Assembler ігнорує всі символи від крапки з комою до кінця рядка, надаючи вам місце для розміщення короткого коментаря, що описує виконувані в даному рядку дії. При роботі обов'язково додавайте чіткі коментарі, які цілком описують вашу програму. Саме так треба починати, особливо, якщо мова асемблера є

новою для вас, оскільки програми, написані на ній, є дуже складними для читання.

Іноді ви можете побачити рядок в асемблерній програмі, що починається з крапки з комою відразу в першому стовпчику. Такі, більш довгі коментарі часто використовуються програмістами для опису пояснювальним коментарем:

```
;-----------------------------------------------------------------
-
; Призначення : Передбачення виграшних номерів лотереї
; Система :       IBM PC / Turbo Assembler Ideal Mode
; Автор :           Олексій Калязін
;-----------------------------------------------------------------
```

5.3.6 Заключення

Останньою частиною програми на мові Асемблер є заключення–рядок, що інформує Turbo Assembler про досягнення кінця програми. У заключенні використовується єдина директива END. Повторюючи останній рядок (рисунок 5.7), розглянемо типове заключення:

```
END      Start   ; Кінець програми / Точка входу.
```

Директива END позначає кінець тексту вихідної програми. Асемблер ігнорує будь–який текст нижче цього рядка і, між іншим, тут можна розмістити додаткові коментарі. Праворуч від директиви END ви маєте визначити позначку з якої ви хочете розпочати виконання програми. Зазвичай ця позначка співпадає з позначкою, яка вказує на першу команду, що йде за директивою CODESEG.

Ви можете розпочинати виконання програми з будь–якого місця, хоча для цього навряд чи існують особливі причини.

5.3.7 Префіксні команди

Машинні команди становлять більшу частину кодів асемблерної програми. В полі операції (мнемоніки) може бути присутня префіксна команда. Наприклад, команда STOSB (Store String Byte – записати рядок байтів) може включати префіксну команду REP (Repeat – повторити). У цьому випадку команди REP STOSB можуть бути закодовані в полі операції одного асемблерного твердження.

5.3.8 Перепризначення сегментів

Зазвичай звернення до змінних за прямою адресою виконуються відносно сегмента, який адресується за допомогою регістра ds. Щоб це змінити, можна визначити сегментне перепризначення в такий спосіб:

```
mov  ch, [es:OverByte].
```

Ця команда завантажує в регістр ch байт з позначкою OverByte, записаний у сегменті, який адресується регістром es. Команда перепризначення сегмента ds на es потрібна для того, щоб змінити використовувану процесором за умовчуванням адресацію базового сегмента ds. Можна також використовувати аналогічні перепризначення і для доступу до даних в інших сегментах. Нижче наводиться три відповідних приклади:

```
mov  dh,[cs:CodeByte]          ; dh <– байт із кодового сегмента ,
mov  dh,[ss:StackByte]         ; dh <– байт зі стекового сегмента,
mov  dh,[ds:DataByte]          ; dh <– байт із сегмента даних ???.
```

У першому рядку в dh завантажується байт із кодового сегмента. Оскільки більшість змінних знаходиться в сегменті даних, звернення до даних, записаних у кодовому сегменті, є вкрай рідким. Другий рядок завантажує байт із сегмента стека. Хоча це і припустимо, але на практиці відбувається нечасто. У третьому рядку зайво визначається сегмент ds (дані

за прямою адресою завжди за умовчуванням розташовуються в сегменті, який адресується регістром ds). Нижче наведено кілька додаткових порад, які допоможуть вам правильно використовувати перепризначення:

- незважаючи на те, що ви визначаєте перепризначення як частину звернення до даних, у дійсності воно займає байт машинного коду і вставляється тільки перед командою, на яку впливає. Перепризначення є командними префіксами і змінюють поводження наступної команди, що виконується;
- перепризначення діє тільки в межах однієї команди. Потрібно використовувати перепризначення при кожнім зверненні до даних, розташованих у сегментах, відмінних від установлених за умовчуванням для даної команди;
- у режимі Ideal Turbo Assembler адреса, включаючи сегментне перепризначення, цілком береться в дужки. Хоча режим MASM забезпечує більш вільний стиль, прості синтаксичні вимоги режиму Ideal цілком сумісні з режимом MASM;
- ви повинні бути впевнені, що змінні дійсно розташовуються в обумовлених вами сегментах, і стежити, щоб адреси цих сегментів знаходилися в сегментних регістрах es і ds. Регістр стека ss і регістр кодового сегмента cs ініціалізації не вимагають.

5.4 Використання в асемблерних програмах макросів

При програмуванні мовою асемблера дуже часто доводиться повторюватися, переписуючи ті ж самі послідовності команд. Щоб зменшити кількість повторів у програмі групу команд можна зберігати в іменованому макросі і потім використовувати його всякий раз, коли вам необхідно виконувати визначену послідовність команд. Коли Turbo Assembler зустрічає макроім'я, він заміняє його послідовністю команд із макросу. Макрос можна визначити або в будь–якім місці програми, або

розмістити в окремому текстовому файлі, а потім завантажувати цей файл під час асемблерування. Щоб це зробити необхідно на початку асемблерної програми вставити директиву включення макросу INCLUDE ”macros.mac” (прочитати бібліотеку макросів) (рисунок 5.8).

```
; Кафедра АУТС ;Системне програмування
; Програмування  динаміка і робота з макросами
IDEAL
MODEL small
STACK 256
INCLUDE "macros.mac"   ;підключення файлу з макросами
DATASEG
_100Hz DW    11930      ;початкове значення лічильника ПТ
len_snd      DW    400        ;тривалість звучання
CODESEG
Start:                  ;(0)
;ініціалізація регістра ds
 mov ax,@data                ;ax<-@data (1)
 mov ds,ax              ;da<-ax        (1)
;програмування таймера
 mov al,0B6h                 ;al<-КС=0B6h (2)
 out  43h,al                 ;РКС ПТ(порт 43h)<-al (2)
;завантаження буферного регістра 2 таймера
 mov  ax,[_100Hz]       ;пересилання Nпоч=11930=2E9Ah у ax(3)
 out  42h,al                 ;порт 42h<-МБ11930=9Ah(3)
 mov  al,ah             ;al<=ah=СБ11930=2Eh (3)
 out 42h,al             ;порт 42H<=СБ11930=2Eh (3)
                        ;включення каналу 2 таймера і динаміка
 in al,61h              ;читання стану порту 61h комп'ютера
                        ; (динаміка) (4)
 or al,3                     ;al<-(al v 03h) (4)
 out  61h,al                 ;порт 61h<-al(4)
 delay [len_snd]        ;встановлення затримки (див. macros.mac)
 and al,11111100b       ;маска відключення звуку (15)
 out 61h,al             ;порт 61h<-al (15)
    mov  ax,04C00h           ;ax<-04C00h (16)
 int 21h                     ;виклик переривання DOS 21h (16)
END Start                    ;(17)
```

Рисунок 5.8 – Вихідна програма керування динаміком ПК (sound2.asm)

Найпростіший макрос (рисунок 5.9) починається з директиви MACRO, після чого вказується його ім'я, а в разі потреби, і його параметр.

У програмі SOUND2.asm (рисунок 5.8) макрос використовується для формування затримки часу.

```
MACRO delay time               ;макрос встановлення затримки

  local outer
  push cx                      ;зберігаємо cx в стеку (5)
  mov cx, time                 ;cx<-400=0190h (6)
outer:                      ;зовнішній  цикл
  push cx                      ;(7)
  mov cx,0FFFFh                ;(8)
  loop $                       ;внутрішній цикл, повторити
                                     ;         cx раз (9,10)
  pop cx                       ;поновлюємо cx (11)
  loop outer                ;cx<-cx-1 і, якщо cx<>0, то - перехід на outer,
                               ;інакше - виконання наступної команди
  pop cx                       ;(14)
ENDM
```

Рисунок 5.9 – Макрос встановлення затримки

У записі “MACRO delay time” (рисунок 5.8) delay є ім'ям макросу, а time – його параметром. Значення цього параметра в програмі, яка використовує зазначений макрос, приймає конкретне значення. Так у розглянутій вихідній асемблерній програмі (рисунок 5.8) макрос підключається командою delay [len_snd], де len_snd – ім'я (значення) змінної, яка визначає тривалість звучання.

Щоб створювати локальні позначки, що автоматично нумеруються, усередині макросу, необхідно використовувати директиву LOCAL з ім'ям позначки. Директиву необхідно вставляти після рядка, що відкриває макрос.

У макросі “delay time” використана позначка “outer”, що є “локальною”, тому визначена відповідною директивою асемблера “local”. Компілятор замість позначки “outer” при першому використанні макросу поставить позначку “??0000”. Якщо цей макрос буде використаний ще раз, то компілятор замість позначки “outer” поставить позначку “??0001”. Тобто будуть створюватися різні локальні позначки, що дозволить багаторазово використовувати той самий макрос різними програмами і при цьому не буде виникати конфлікт позначок.

У наведеній вище програмі (рисунок 5.8) у макрос входять команди (блоки 5–14 схеми алгоритму (рисунок 5.10), які реалізують затримку часу, що визначає тривалість звучання динаміка.

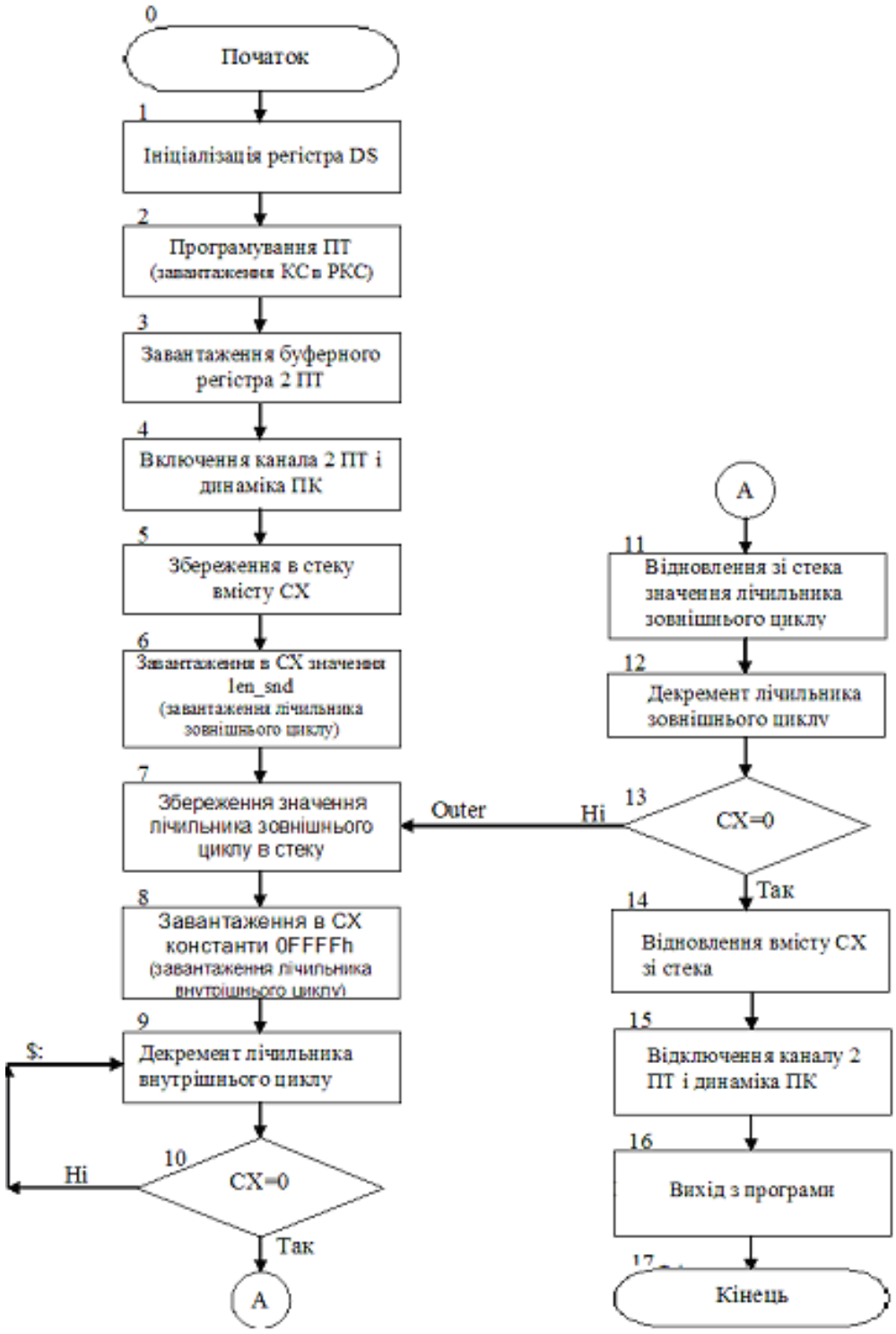


Рисунок 5.10 – Схема алгоритму керування динаміком ПК

На рисунку 5.11 наведено лістинг розглянутої програми, який показує підключення макросу при асемблеруванні.

```
1                                       ;Кафедра АУТС; Системне програмування
   2                                    ;Програмування динаміка і робота з
   3                                    ;макросами
   4                                    IDEAL
   5  0000                              MODEL   small
   6  0000                              STACK   256
   7                                    INCLUDE "macros.mac"    ;підключення файлу з макросами
1  19                MACRO              delay   time            ;макрос встановлення затримки
1  20                                   local   outer
1  21                                   push    cx              ;збереження cx в стеку (5)
1  22                                   mov     cx, time        ;cx<–400=0190h  (6)
1  23                 outer:                                    ;зовнішній цикл
1  24                                   push    cx              ; (7)
1  25                                   mov     cx, 0FFFFh      ; (8)
1  26                                   loop    $               ;внутрішній цикл, повторити
1  27                                                           ;   cx раз (9,10)
1  28                                   pop     cx              ;відновлення cx (11)
1  29                                   loop outer              ;cx<–cx–1 і, якщо cx<>0, то перехід
1                                                               ;на outer, інакше виконання
                                                                 ;наступної команди (12,13)
1  30                                   pop     cx              ;(14)
1  31                                   ENDM
1  32
   41  0100                                    DATASEG
   43  0000 2E9A          _100Hz               DW          11930 ;початкове значення лічильника ПТ
   44  0002 0190          len_snd              DW          400   ;тривалість звучання
   46  0004                                    CODESEG
   47                                   EXTRN StrWrite:proc
   49  0000                             Start:
   50                                                           ;ініціалізація регістра ds
   51  0000 B8 0000s      mov           ax, @data               ;ax<–@data          (1)
   52  0003 8E D8         mov           ds, ax                  ;ds<–ax   (1)
   53                                                           ;програмування таймера
   54  0005 B0 B6         mov           al, 0B6h                ;al<–КС=0B6h (2)
   55  0007 E6 43         out           43h, al                 ;РКС ПТ(порт 43h)<–al (2)
   56                                                           ;завантаження буферного регістра 2 таймера
   57  0009 A1 0000r      mov           ax, [_100Hz]            ;пересилання частоти в ax (3)
   58  000C E6 42         out           42h, al                 ;порт 42h<–МБ11930=9ah (3)
   59  000E 8A C4         mov           al, ah                  ;al<–ah=СБ11930=2Eh (3)
   60  0010 E6 42         out           42h, al                 ;порт 42h<–СБ11930=2Eh (3)
   61                                                           ;включення каналу 2 таймера і динаміка
   62  0012 E4 6          in            al, 61h                 ;зчитування стану порту динаміка (4)
   63  0014 0C 03         or            al, 3                   ;al<–(al\/03h) (4)
  64   0016 E6 61         out           61h, al                 ;і виведення в нього (4)
   66                                   delay   [len_snd]       ;встановлення затримки (див. macros.mac)
1  67  0018 51                        push    cx              ;збереження cx в стеку (5)
1  68  0019 8B 0E 0002r              mov     cx,[len_snd]    ;cx<–400=0190h  (6)
1  69  001D                         ??0000:                 ;зовнішній цикл
1  70  001D 51            push      cx                      ;(7)
1  71  001E B9 FFFF       mov       cx, 0FFFFh              ;(8)
1  72  0021 E2 FE                   loop      $             ;внутрішній цикл, повторити cx раз
                                                             ;(9,10)
1  73  0023 59            pop       cx                      ;відновлення cx (11)
1  74  0024 E2 F7                   loop      ??0000        ;cx<–cx–1 і, якщо cx<>0, то
                                                             ;перехід на outer, інакше виконання
                                                             ;наступної команди (12,13)
1  75  0026 59            pop       cx                ;(14)
   76  0027 24 FC         and       al, 11111100b     ;маска вимикання звука (15)
   77  0029 E6 61         out       61h, al           ;порт 61h<–al (15)
   78  002B B8 004C       mov       ax,04Ch           ;ax<–04Ch (16)
   79  002E C7 06 0021 0000 mov     21h               ;виклик переривання DOS 21h
                                80                          END Start
```

Рисунок 5.11 – Лістинг програми SOUND_2.asm(SOUND_2.lst)

5.5 Використання в асемблерних програмах підпрограм

Одним з найбільш корисних інструментів у мові асемблера є процедура, чи підпрограма – набір взаємозалежних команд, які зазвичай виконують одну операцію, що часто зустрічається. Процедура може виводити на екран рядок символів, підсумовувати набори значень чи ініціалізувати вихідний порт. Деякі підпрограми живуть чудовим життям: грають у шахи чи забезпечують доступ до віддаленого комп'ютера.

Інші мають більш скромну роль: виводять окремий символ чи зчитують код клавіші з клавіатури.

Деякі програмісти створюють довгі підпрограми, що виконують відразу велику кількість роботи, пояснюючи це тим, що використання численних маленьких підпрограм може сповільнювати швидкість виконання програми. Не робіть цього.

Поєднуючи команди у великі підпрограми, ви можете одержати тільки незначне збільшення швидкості, але, що більш імовірно, це призведе до помилкової і важконалагоджуваної програми, розмірковуючи над якою ви можете пошкодувати про те, що стали програмістом.

Гарна підпрограма виконує одну і тільки одну роботу. Гарна підпрограма коротка, наскільки це можливо, і велика, наскільки це необхідно. Гарна підпрограма уміщається на одній чи двох друкованих сторінках. Гарна підпрограма починається не з коду, а з коментарів, що описують її призначення, результати, очікувані вхідні дані і використовувані регістри. У гарній підпрограмі можна розібратися, не знаючи, що робить вся програма. Іншими словами, гарна підпрограма – коротка, витончена і точна.

У програмі керування динаміком ПК SOUND_1.asm (рисунок 5.12) наведено приклад використання підпрограми (процедури) DELAY для

формування затримки часу, що визначає тривалість імпульсу і паузи ($t_{імп}$= $t_п$) імпульсної послідовності, яка викликає звучання динаміка.

На рисунках 5.13, 5.14, і 5.15 відповідно наведено:

- схему алгоритму керування динаміком;
- схему алгоритму затримки часу;
- лістинг програми SOUND_1.asm (SOUND_1.lst).

```
 IDEAL
MODEL small

DATASEG

len_snd  DW  40000
frc_snd  DW  2970

CODESEG
;-------------------------------------------------------------------------
;DELAY – процедура формування затримки часу
;-------------------------------------------------------------------------
;Вхід:  ініціалізована змінна frc_snd
;Вихід: ні
;Регістри, що використовуються: CX
;-------------------------------------------------------------------------
Proc Delay
   mov cx,[frc_snd]   ; (proc 1)
   loop $             ; (proc 2; proc 3)
   retn               ; (proc 4)
EndP Delay

Start:                ; (0)
    mov  ax,@data     ; (1)
    mov  ds,ax        ; (1)
    cli               ; (2)
    in   al,61h       ; (3)
    mov  cx, [len_snd]; (4)
Begin:
    push cx           ; (5)
    or   al,00000010b ; (6)
    out  61h,al       ; (7)
    call Delay        ; (8)
    and  al, 11111101b; (9)
    out  61h,al       ; (10)
```

Рисунок 5.12 – Вихідна програма керування динаміком (SOUND_1.asm)

```
call Delay          ; (10)
pop  cx             ; (11)
loop Begin          ; (12,13)
sti                 ; (14)
mov  ax,04c00h      ; (15)
int  21h            ; (15)
END Start           ; (16)
```

Продовження рисунку 5.12

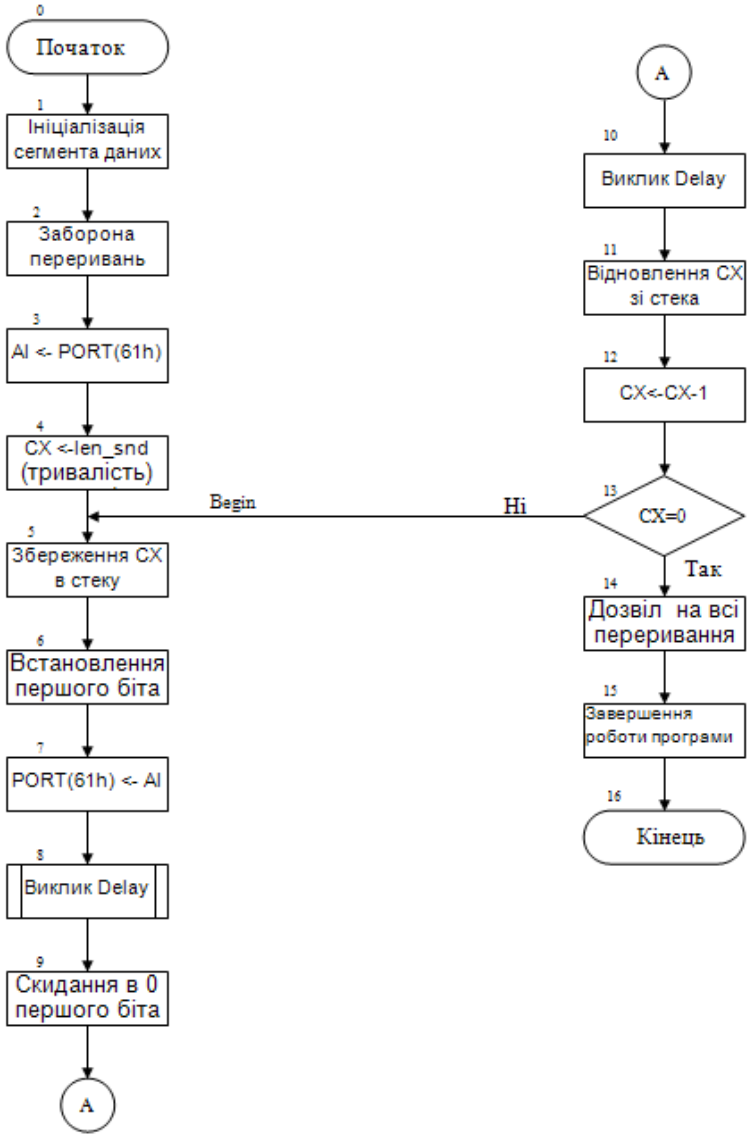


Рисунок 5.13 – Схема алгоритму керування динаміком, якщо шпаруватість Q= 2

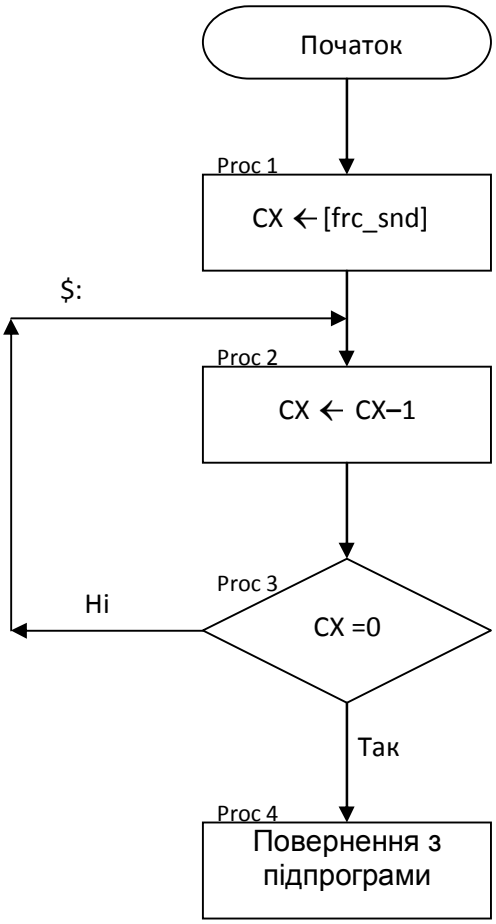


Рисунок 5.14 – Схема алгоритму затримки часу

```
1
2                               IDEAL
3   0000                        MODEL small
4
5   0000                        DATASEG
6
7   0000  9C40             len_snd  DW  40000
8   0002  0B9A             frc_snd  DW  2970
9
10  0004                        CODESEG
11  ;-------------------------------------------------------------------
12                              ;DELAY – процедура формування затримки часу
13  ;-------------------------------------------------------------------
14                              ;Вхід: ініціалізована змінна frc_snd
15                              ;Вихід: немає
16                              ;Регістри,  що використовуються: CX
```

Рисунок 5.15 – Лістинг програми керування динаміком SOUND_1.asm (SOUND_1.lst)

```
17  ;-------------------------------------------------------------------------
18  0000                    Proc Delay
19  0000  8B 0E 0002r          mov cx,[frc_snd]        ; (proc 1)
20  0004  E2 FE                loop $                  ; (proc 2;    proc 3)
21  0006  C3                   retn                    ; (proc 4)
22  0007                    EndP Delay
23
24  0007                    Start:                      ; (0)
25  0007  B8 0000s             mov  ax,@data           ; (1)
26  000A  8E D8                mov  ds,ax              ; (1)
27  000C  FA                   cli                     ; (2)
28  000D  E4 61                in   al,61h             ; (3)
29  000F  8B 0E 0000r          mov  cx, [len_snd]      ; (4)
30  0013                    Begin:
31  0013  51                   push cx                 ; (5)
32  0014  0C 02                or   al,00000010b       ; (6)
33  0016  E6 61                out  61h,al             ; (7)
34  0018  E8 FFE5              call Delay              ; (8)
35  001B  24 FD                and  al, 11111101b      ; (9)
36  001D  E6 61                out  61h,al             ; (10)
37  001F  E8 FFDE              call Delay              ; (10)
38  0022  59                   pop  cx                 ; (11)
39  0023  E2 EE                loop Begin              ; (12,13)
40  0025  FB                   sti                     ; (14)
41
42  0026  B8 4C00              mov  ax,04c00h          ; (15)
43  0029  CD 21                int  21h                ; (15)
44
45                             END Start               ; (16)
```

Продовження рисунку 5.15

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ:

1. Дайте визначення мові Асемблер.
2. Назвіть та дайте визначення основних керуючих програм, які використовуються при створенні програм на мові Асемблер.
3. Поясніть наступні терміни: мнемоніка команди та мнемо код; код операції команди(КОП); операнди; коментар.
4. Наведіть та поясніть формати команд 8-го МП-ра та МК-ра і 16-го МП-ра.
5. Наведіть та поясніть формати даних 8-го МП-ра та МК-ра і 17-го МП-ра.
6. Поясніть, що таке прапорці та яку роль вони відіграють у роботі МПС.
7. Як можна обчислити час виконання окремих команд у МК-рі та МП-рі?
8. Поясніть послідовність створення EXE-програми для МПС на основі МП-ра i8086
9. Поясніть формати лістинга програми на прикладі команд пересилань.
10. Поясніть окремі частини шаблону, який можна використовувати для створення вихідних EXE-програм.
11. Наведіть та поясніть приклади префіксних команд.
12. Як можна виконати перепризначення сегментів?
13. Що таке макроси і як вони використовуються при програмуванні мовою асемблера? Наведіть приклад.
14. Що таке підпрограми і як вони використовуються при створенні асемблерних програм? Наведіть приклад.

ЛІТЕРАТУРА [1, 5, 7 … 10, 14, 15, 17, 36…39]

6 ХАРАКТЕРИСТИКА КОМАНД МІКРОПРОЦЕСОРІВ ТА МІКРОКОНТРОЛЕРІВ

6.1 Мнемоніка команди

Мнемоніка команди – представлення коду операції у вигляді сполучення латинських літер, що мають визначений зміст (використовуються англійські слова або скорочення, наприклад, MOV, PUSH, POP, JMP, CLR, NOP). Мнемоніки введені для полегшення написання програм, і складають основу мови асемблера.

6.2 Код операції команди (КОП)

Код операції команди (КОП) – комбінація бітів (не більше восьми для 8–розрядних МП–в і МК–в), що знаходяться на початку машинного коду команди і визначають тип операції, що підлягає виконанню у конкретний момент часу. КОП витягається з пам'яті і розміщується в програмно недоступний регістр команд. Потім він декодується і визначається тип команди, яка повинна бути виконана. Для 16–х мікропроцесорів КОП частково може знаходитися у другому байті машинного коду команди. Крім КОП в машинний код команди можуть входити адреси та операнди.

6.3 Мнемоніка команди та мнемокод

Мнемоніка команди – представлення коду операції у вигляді сполучення латинських літер, що мають визначений зміст (використовуються англійські слова або скорочення, наприклад, MOV, PUSH, POP, JMP, CLR, NOP). Мнемоніки введені для полегшення написання програм, і складають основу мови асемблера.

Мнемокод включає в себе мнемоніку команди та опис операндів, які беруть участь в операції.

6.4 Машинний код команди

Машинний код команди представляє собою двійковий код команди, який складається з одного або декількох байтів в залежності від типу команди конкретного МП–ра або МК–ра.

6.5 Операнди

Операндами у мікропроцесорній техніці називають дані, які приймають участь у виконанні тієї чи іншої команди (операції). В залежності від способу адресації операндів дані можуть знаходитися у регістрах, пам'яті, у машинному коді команди і т.ін.

6.6 Коментар

Коментар – використовується для полегшення читання програми. Компілятор (асемблер) пропускає коментарі, що показані в програмі через "**;**" після або до мнемоніки команди, не генеруючи при цьому ніякого машинного коду.

6.7 Формати команд

6.7.1 Мікропроцесор i8080

8–розрядний МП–р, наприклад, i8080 керується командами трьох форматів (рисунок 6**.**1).

Однобайтовий формат використовується для представлення команд з неявною, регістровою і непрямою адресацією і дозволяє адресувати операнди, що містяться в регістрах МП і комірках запам'ятовуючого

пристрою (ЗП). Двобайтовий формат команд представляє команди з безпосереднім операндом чи з прямою адресою порту введення/виведення (другий байт команди).

В другому і третьому байтах трьохбайтових команд міститься 16–розрядний безпосередній операнд або 16–розрядна пряма адреса комірки пам'яті.

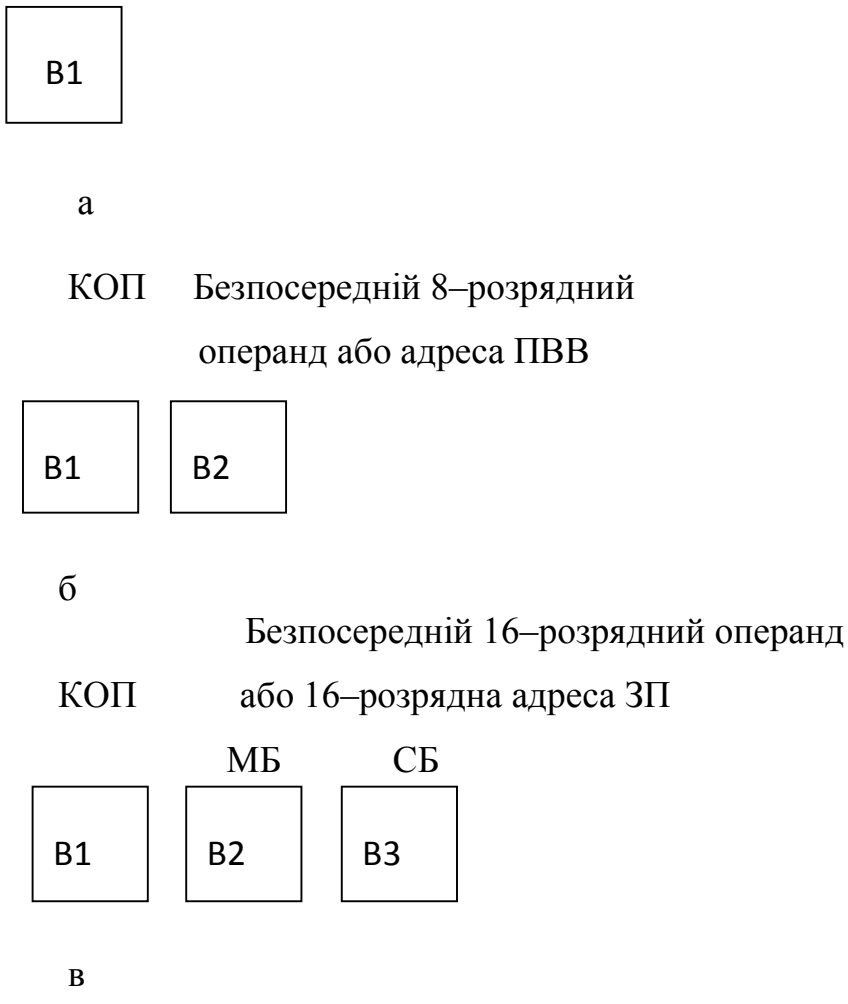


Рисунок 6.1 – Формати команд МП і8080:
а – однобайтовий (В1);
б – двобайтовий (В1,В2);
в – трьохбайтовий (В1,В2,В3).

6.7.2 Мікроконтролер типу МК51

8–розрядний МК типу МК51, наприклад, AT89C51 має команди тринадцяти форматів (типів) (рисунок 6.2).

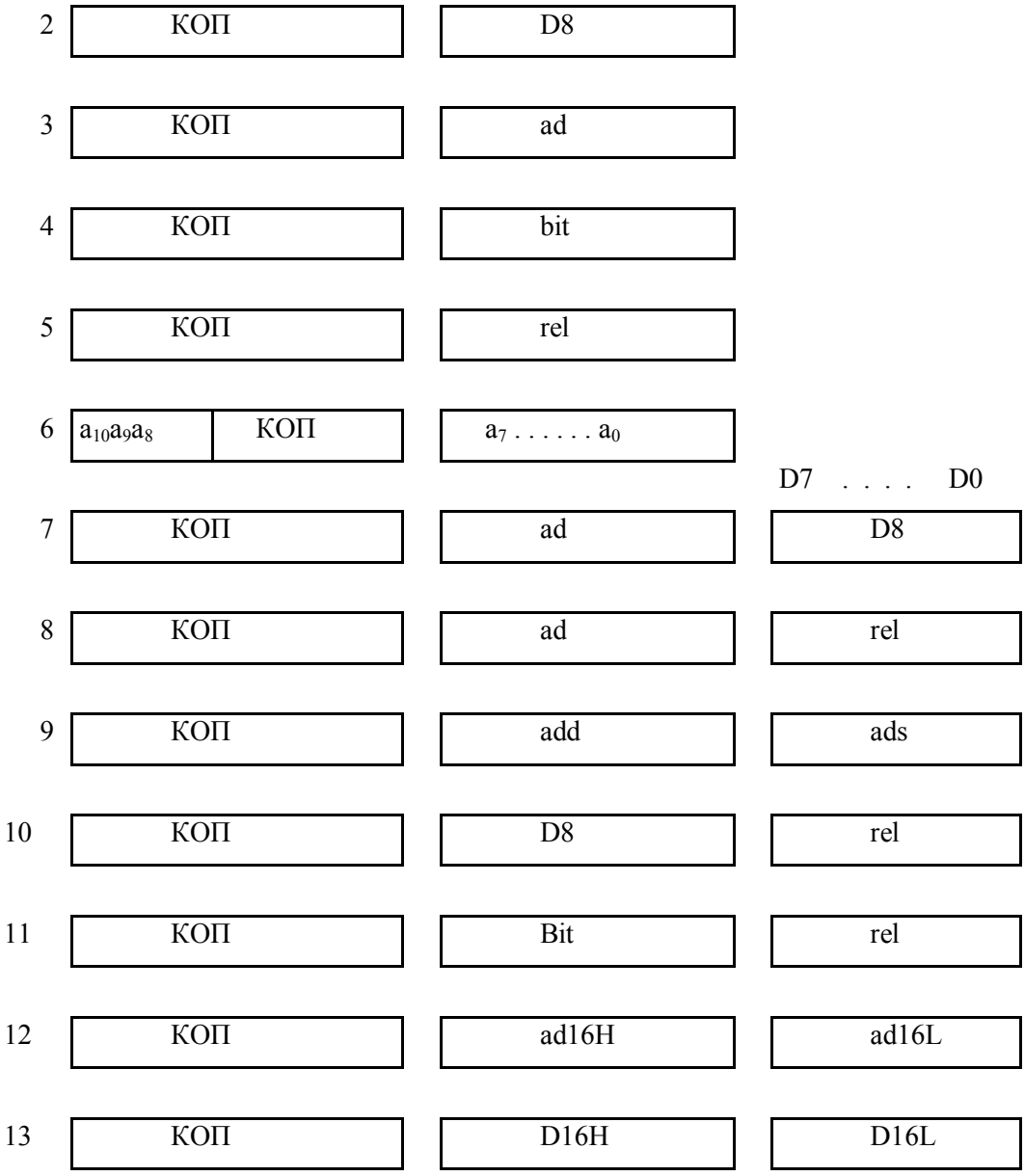


Рисунок 6.2 – Типи команд МК51

Перший байт команди кожного типу (формату) завжди містить КОП. Другий і третій байти включають в себе або адреси операндів, або безпосередні операнди.

Команди першого типу мають довжину 1 байт. Крім КОП в склад команди може входити адреса одного з восьми РЗП поточного банку регістрів або адреса регістра, що використовувався для непрямої адресації пам'яті.

Команди другого типу містять два байти, з яких:

– перший байт виконує функції, що описані вище;
– другий байт є 8–розрядним безпосереднім операндом (БО), який входить в саму команду.

Другий байт команд третього типу являє собою пряму адресу комірки резидентної пам’яті даних (РПД) або одного з 21 регістрів спеціальних функцій (таблиця 6.1).

Другий байт команд четвертого типу містить адресу одного з 128 прямоадресуємих біт РПД (рисунок 6.3) або адресу прямоадресуємих біт регістрів спеціальних функцій, що допускають пряму бітову адресацію (рисунок 6.4).

Другий байт команд п'ятого типу є 8–розрядним числом зі знаком в додатковому коді, яке в командах передачі керування підсумовується з поточним значенням програмного лічильника (адресою наступної команди), чим забезпечується перехід в програмі на (+127) комірок вперед або на (–128) назад відносно наступної команди. Значення цього числа обчислюється компілятором за формулою:

rel = АДРЕСА МІТКИ – АДРЕСА НАСТУПНОЇ КОМАНДИ.

Двобайтова команда шостого типу крім коду операції (КОП) містить одинадцятирозрядну пряму адресу (a_{10}, a_9, …, a_1, a_0), що забезпечує перехід

в програмі всередині сторінки, об'ємом 2 Кбайт або виклик підпрограми в тому ж діапазоні адрес.

Таблиця 6.1 – Регістри спеціальних функцій

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

Другі і треті байти команд (7 … 11) типів є сполученням описаних вище команд:

– прямої адреси байта (ad, ads, add);

– прямої адреси біта (bit);

– 8–розрядного числа зі знаком (rel);

– безпосереднього операнда (D8).

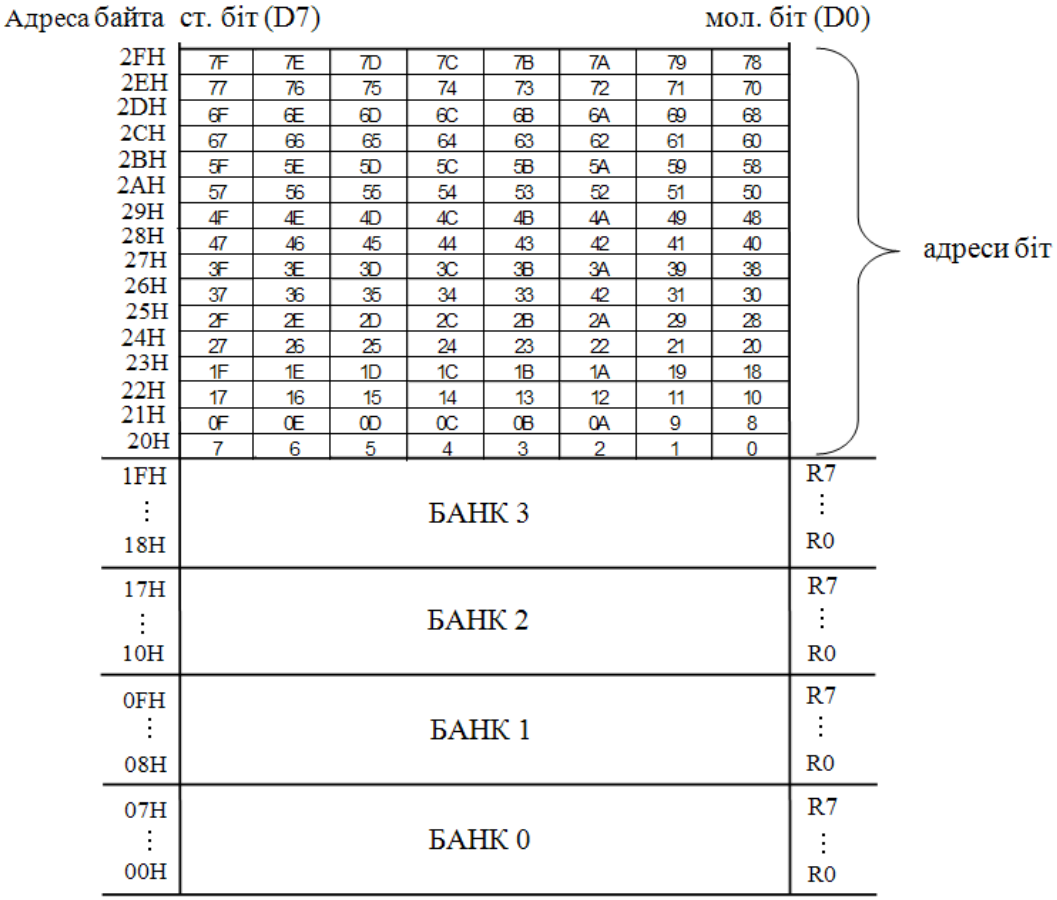


Рисунок 6.3 – Організація бітової адресації ОЗП

Другі і треті байти команд 12–го і 13–го типів представляють відповідно: пряму 16–ну адресу переходу або підпрограми, 16–ний безпосередній операнд.

6.7.3 Формати команд мікропроцесора i8086

Основні формати команд МП–ра i8086 приведено на рисунку 6.5. Команди розташовуються в пам'яті побайтно в комірках з послідовно зростаючими адресами.

Вибираються вони з пам'яті словами (по два байти) і завантажуються в чергу команд блоку сполучення із системною шиною (БСШ) мікропроцесора. Перший байт команди завжди містить код операції (КОП), причому в окремих командах його нульовий розряд несе інформацію про довжину операндів (байт чи слово), а перший розряд задає тип джерела і/чи приймача інформації.

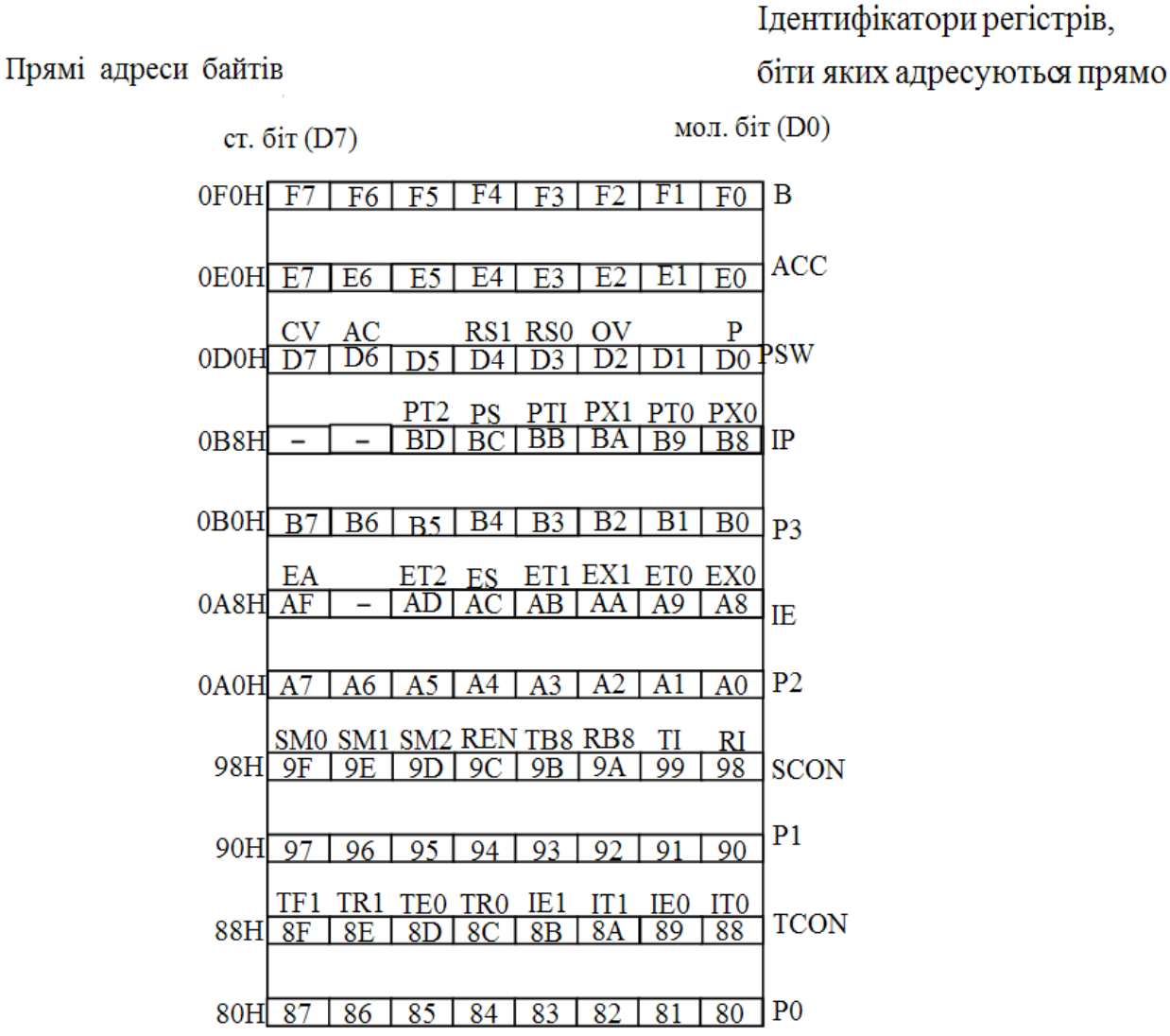


Рисунок 6.4 – Адреси прямоадресуємих біт регістрів спеціальних функцій

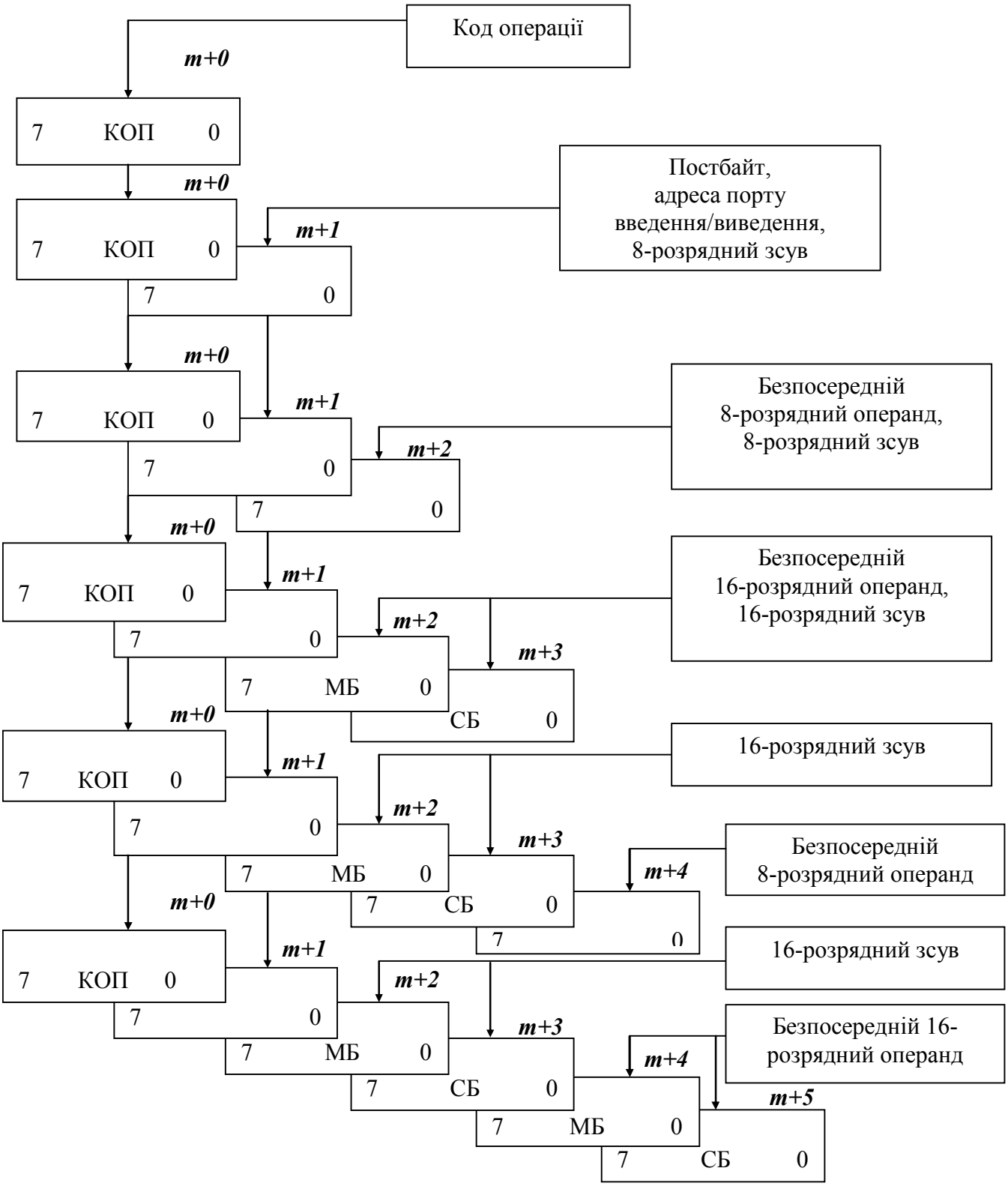


Рисунок 6.5 – Формати команд МП

Другий байт у більшій частині команд може бути трьох типів:

– постбайт, що визначає, відкіля брати операнд (операнди) і/чи куди направляти результат операції. У деяких командах розряди 5...3 постбайта (поле reg) можуть використовуватися для розширення коду операції команди;
– адреса порту введення/виведення, що задає пряму адресу порту в діапазоні 00H…FFH у двобайтних командах;
– 8 – розрядний зсув, що визначає відносну адресу переходу в командах умовної передачі керування.

Наступні байти в довгих командах містять 8–чи 16–розрядний зсув і/чи безпосередній операнд, також 8–чи 16–розрядний. Якщо в командах присутній зсув, і безпосередній операнд, то спочатку розміщується зсув, а потім операнд. Для 16–розрядного зсуву, і для 16–розрядного операнда завжди першим розташовується молодший байт, а другим – старший (адреса старшого байта на одиницю більше адреси молодшого).

8 – розрядний зсув при формуванні ефективної адреси розширюється до 16–розрядного вмістом старшого (знакового) розряду. Наприклад, якщо заданий байт зсуву 7EH=01111110B, то він розширюється до 007EH. Якщо заданий байт A3H=10100011B, то він розширюється до FFA3H. Зсув сприймається мікропроцесором як ціле двійкове число зі знаком у додатковому коді, що лежить у діапазоні: –128 ... +127 для 8–розрядного чи: –32768 ... +32767 для 16–розрядного зсуву.

6.8 Формати даних

Під час роботи з мікропроцесором або мікроконтролером необхідно знати не тільки формати команд, які керують роботою МП або МК, але й формати даних (операндів), що приймають участь у виконанні тієї або іншої

команди (операції). Для 8–розрядного МП–ра або МК–ра формати даних наведено на рисунку 6.6.

В 16–му МП–рі крім названих форматів даних використовуються 16–ні операнди, які можуть представляти собою (рисунок 6.7) :

– ціле додатне число в діапазоні: 0…65535D;

– число зі знаком в діапазоні: (–32768 … +32767);

– шістнадцять незалежних логічних змінних: X1, X2, … X16.

Цей процесор може виконувати операції над двійково–десятковими числами в упакованому, не упакованому форматах, а також числами, що представлено в коді ASCII.

Крім того, у ролі операндів можуть виступати рядки (ланцюжки) даних, елементами яких можуть бути байти або слова [5, 10, 11, 38].

6.9 Довжина команд у байтах і їх розміщення у пам'яті програм

У 8–розрядному МП–рі, наприклад, i8080, багатобайтові команди зберігаються в сусідніх комірках ЗП і адресуються за першим байтом, причому двобайтові слова даних і адреси розташовуються в порядку зростання адреси ЗП – спочатку молодший, а потім старший байти. Як видно з рисунка 6.1, для цього МП–ра команди можуть мати довжину 1,2 або 3 байти. Програма, що керує роботою МП, послідовно, команда за командою, розміщується в комірках пам'яті в порядку зростання їх адрес.

Адреса команди, яка повинна використовуватись, знаходиться у програмному лічильнику PC. Пристрій керування мікропроцесора на підставі прочитаного коду операції, що міститься завжди в першому байті команди, визначає, скільки ще байтів міститься в команді, та керує їх читанням з пам'яті шляхом збільшення на одиницю адреси, що видається при кожному зверненні до ЗП

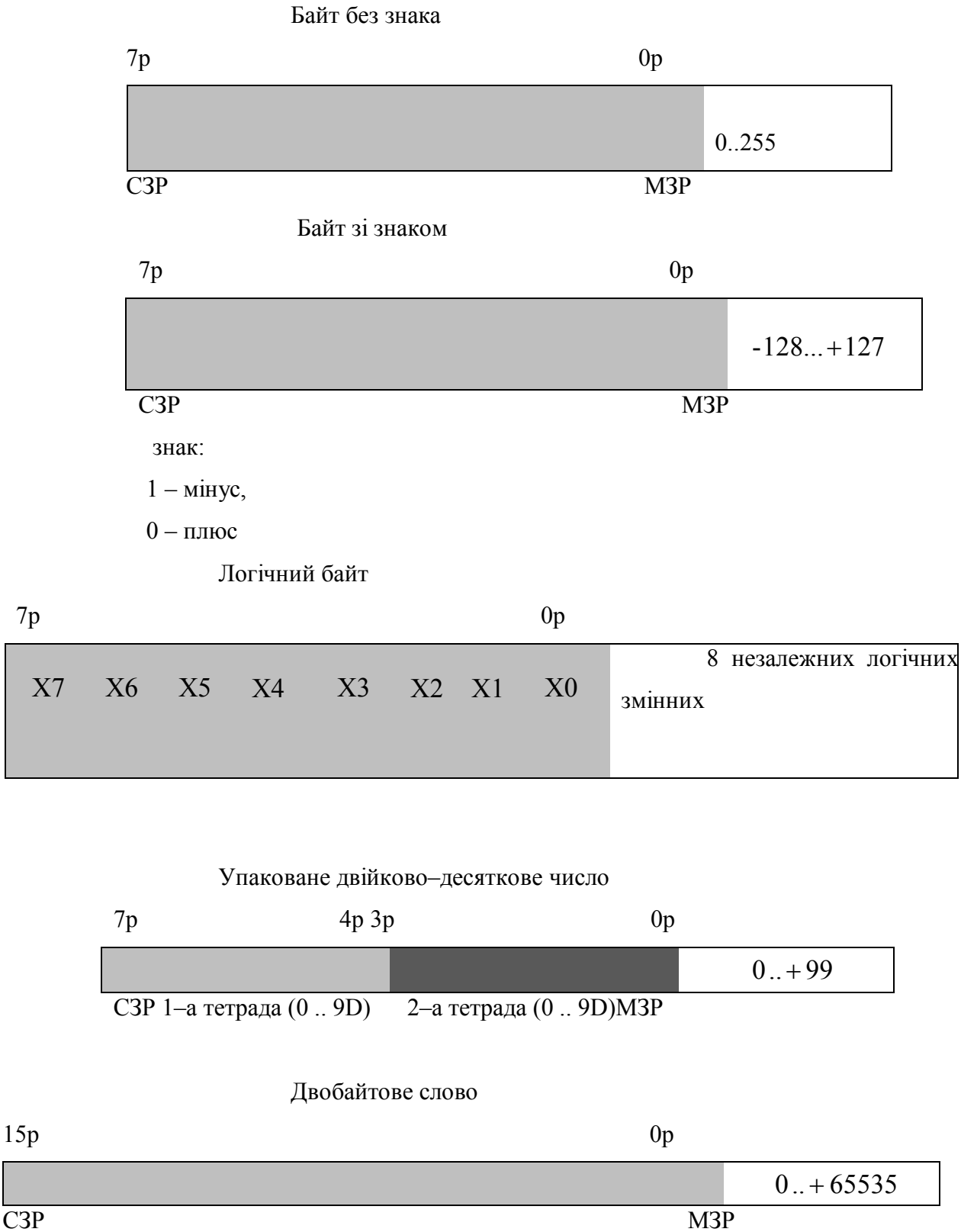


Рисунок 6.6 – Формати (типи) даних 8–розрядного МП–ра і МК–ра

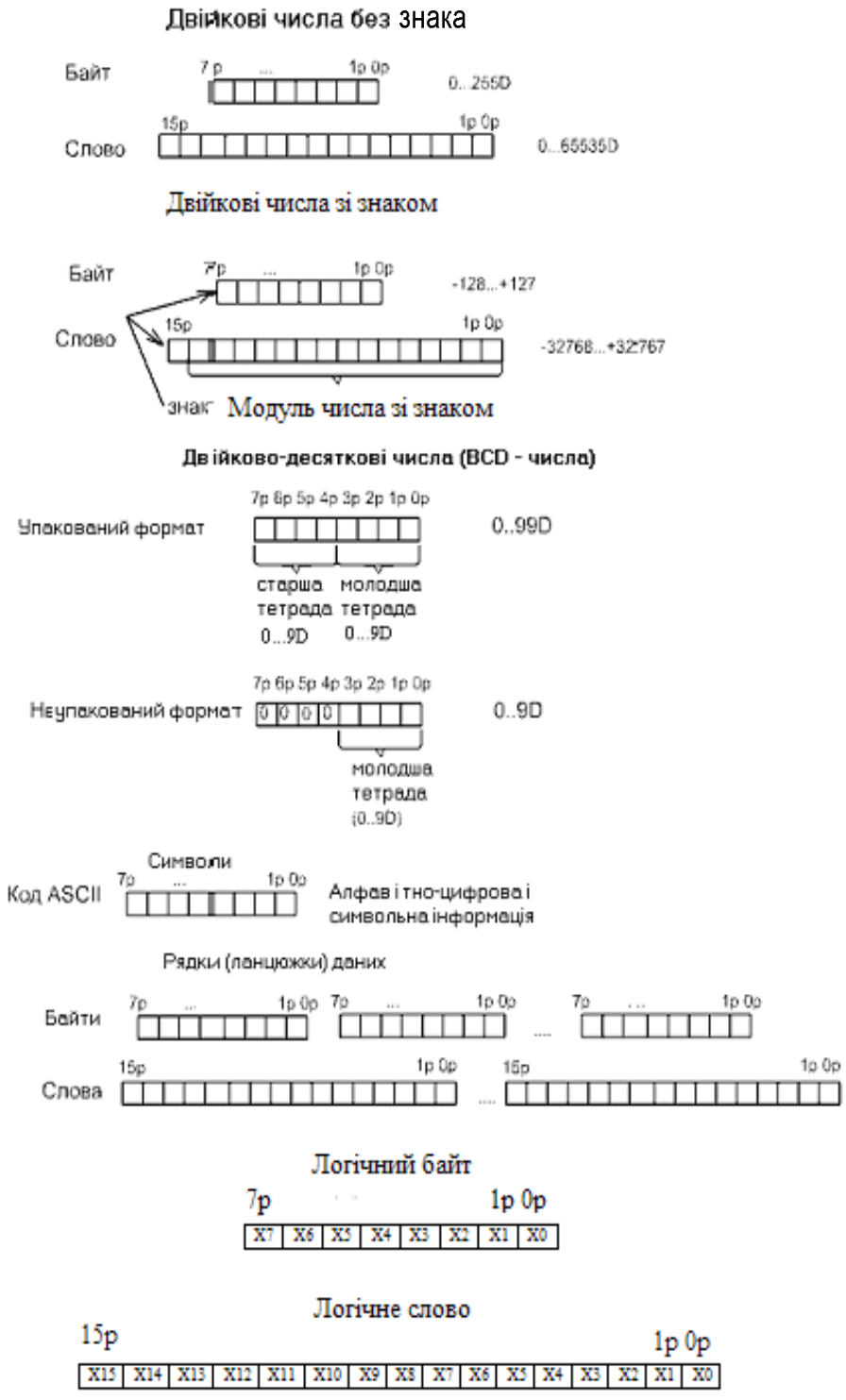


Рисунок 6.7 – Типи і формати даних мікропроцесора і8086

Після читання з пам'яті чергової команди МП формує адресу КОП наступної команди. В МК типу МК51 довжина команд також дорівнює 1, 2 або 3 байти. В пам'яті окремі байти багатобайтових команд також розташовуються один за одним в сусідніх комірках пам'яті, причому перший завжди містить КОП команди. Однак, на відміну від 8–розрядного МП–ра, двобайтові слова даних і адреси розташовуються в іншому порядку: спочатку старший байт, а потім, за більш старшою адресою – молодший байт (рисунок 6.2). Довжина команд 16–го МП–ра і8086 складається від 1 до 6 байт (рисунок 6.5), які розміщуються в пам'яті так само, як в 8–му МП–рі. Так, наприклад, байти найдовшої команди 6 байт розташовуються в такому порядку: 1 байт – КОП; 2 байт – постбайт (використовується для адресації операндів); 3 байт – МБ (зсув); 4 байт – СБ (зсув, який бере участь в обчисленні ефективної адреси операнда ЕА, яка формується у відповідності з форматом постбайта); 5 байт – МБ безпосереднього операнда; 6 байт – СБ безпосереднього операнда.

6.10 Вплив команд на прапорці (ознаки)

Як відзначалося раніше, одним з основних регістрів МП–ра або МК–ра є регістр прапорців (ознак).

Прапорці регістра ознак встановлюються під час виконання ряду команд МП–ра і МК–ра. Під час описання системи команд, яка, як правило, оформлюється у вигляді таблиці, в одній з колонок показується вплив окремих команд на ті або інші прапорці [2,11,15,29,39].

Якщо команда не змінює прапорець, то ставлять риску, якщо змінює, то ставлять плюс, а якщо встановлює у який–небудь стан, то пишуть нуль (0) або одиницю (1). Як правило, команди пересилань не змінюють прапорці, окрім команд, які пересилають дані у регістр прапорців. Частіше прапорці змінюють арифметичні, логічні команди і команди порівняння.

Окремі прапорці (ознаки) аналізуються під час виконання команд умовних переходів, виклику і повернення з підпрограм, що дозволяє передавати керування в програмі в залежності від поточного значення прапорців. В таблиці 6.2 показано вплив на прапорці команд МК типу AT89C51.

Вплив команд на прапорці в 16–му МП–рі i8086 відображено в таблицях 6.3, 6.4.

6.11 Час виконання команд

Вибір з пам'яті і виконання команд в МП, наприклад, i8080, можна розділити на елементарні відрізки часу (рисунок 6.8). Найменший відрізок часу, протягом якого процесор виконує певні дії, називається тактом (Т). Такт дорівнює періоду зовнішніх синхроімпульсів (періоду тактової частоти). Наприклад, при $f_{гті}$ = 2МГц тривалість такту $t_Т$ =500 нс. Початок і кінець кожного такту відповідають переднім фронтам імпульсів F1 двофазного генератора (рисунок 6.8). Час такту $t_Т$=const. Його можна змінювати, змінюючи $f_{ГТІ}$.

Таблиця 6.2 – Команди, що впливають на встановлення прапорців МК

Мнемоніка	Прапорці		
	C	OV	AC
ADD	x	x	x
ADDC	x	x	x
SUBB	x	x	x
MUL	0	x	
DIV	0	x	
DAA	x		
RRC	x		
RLC	x		

Продовження таблиці 6.2

Мнемоніка	Прапорці		
	C	OV	AC
SETB C	1		
CLR C	0		
CPL C	x		
ANL C, bit	x		
ANL C, /bit	x		
ORL C, bit	x		
ORL C, /bit	x		
MOV C, bit	x		
CJNE	x		

Примітки:

1. x – прапорець змінюється та дорівнює 0 або 1 відповідно до результату операції.

2. Операції над вмістом регістра прапорців PSW або його окремих розрядів також впливають на встановлення прапорців.

3. Прапорець паритету (парності) P встановлюється/скидається апаратно і відображає непарне/парне число одиниць в акумуляторі.

З тактів (елементарних відрізків часу) складаються відрізки часу більшої величини, що називаються машинними циклами (МЦ) і які позначаються літерою М. Кожний МЦ відповідає одній операції звернення до пам'яті або ПВВ. Число МЦ в команді визначається числом звернень процесора до пам'яті або ПВВ.

Час виконання команди складає командний цикл (КЦ).

Формат команд розглянутого МП – один, два або три байти. Перший байт кожної команди містить код операції (КОП). Кожна команда включає, як мінімум, один МЦ – цикл вибірки коду операції команди (М1). Для вибірки 2–х і 3–х байтових команд потрібно відповідно два або три МЦ.

Таблиця 6.3 – Команди, які впливають на прапорці у МП–рі i8086

Операція	Команди	Прапорці стану					
		OF	CF	AF	SF	ZF	PF
Додавання, віднімання, порівняння	ADD ADC SUB SBC	+	+	+	+	+	+
	CMP NEG CMPS SCAS	+	+	+	+	+	+
	INC DEC	+	–	+	+	+	+
Множення	MUL IMUL	+	+	?	?	?	?
Ділення	DIV IDIV	?	?	?	?	?	?
Десяткова корекція	DAA DAS	?	+	+	+	+	+
	AAA AAS	?	+	+	?	?	?
	AAM AAD	?	?	?	+	+	+
Зсуви	AND OR XOR TEST	0	0	?	+	+	+
	SHL SHR [1)]	+	+	?	+	+	+
	SHL SHR [2)]	?	+	?	+	+	+
	SAR	0	+	?	+	+	+
	ROL ROR RCL RCR	+	+	–	–	–	–
	ROL ROR RCL RCR	?	+	–	–	–	–
Відновлення прапорців	POPF IRET	+	+	+	+	+	+
	SAHF	–	+	+	+	+	+
Керування прапорцем перенесення	STC	–	1	–	–	–	–
	CLC	–	0	–	–	–	–
	CMC	–	Г	–	–	–	–

Примітки:

«+» – результат операції впливає на прапорець; «–» – не впливає;

1 – встановлюється в «1» ; 0 – скидається в «0»; Г – інвертується; ? – не визначений;

1)– зсув на один розряд; 2) – зсув на декілька розрядів.

Таблиця 6.4 – Вплив команд на прапорці у МП–рі i8086

Операції	Команди	Прапорці керування		
		DF	IF	TF
Відновлення прапорців	POPF IRET	+	+	+
Переривання	INT INTO	–	0	0
Керування прапорцями	STD	1	–	–
	CLD	0	–	–
	STI	–	1	–
	CLI	–	0	–

Примітки:

«+» – результат операції впливає на прапорець; «–» – не впливає;

1 – встановлює в «1» ; 0 – скидає в «0».

Командний цикл для даного МП–ра містить від 1 до 5 машинних циклів, а кожний машинний цикл складається з трьох, чотирьох або п'яти тактів (рисунок 6.8).

В одній з колонок таблиці, які описують систему команд записують кількість тактів, що необхідно для виконання команди. Знаючи число тактів (n_T) і значення тактової частоти МП–ра (f_T) можна визначити час виконання команди $t_K=n_T / f_T$.

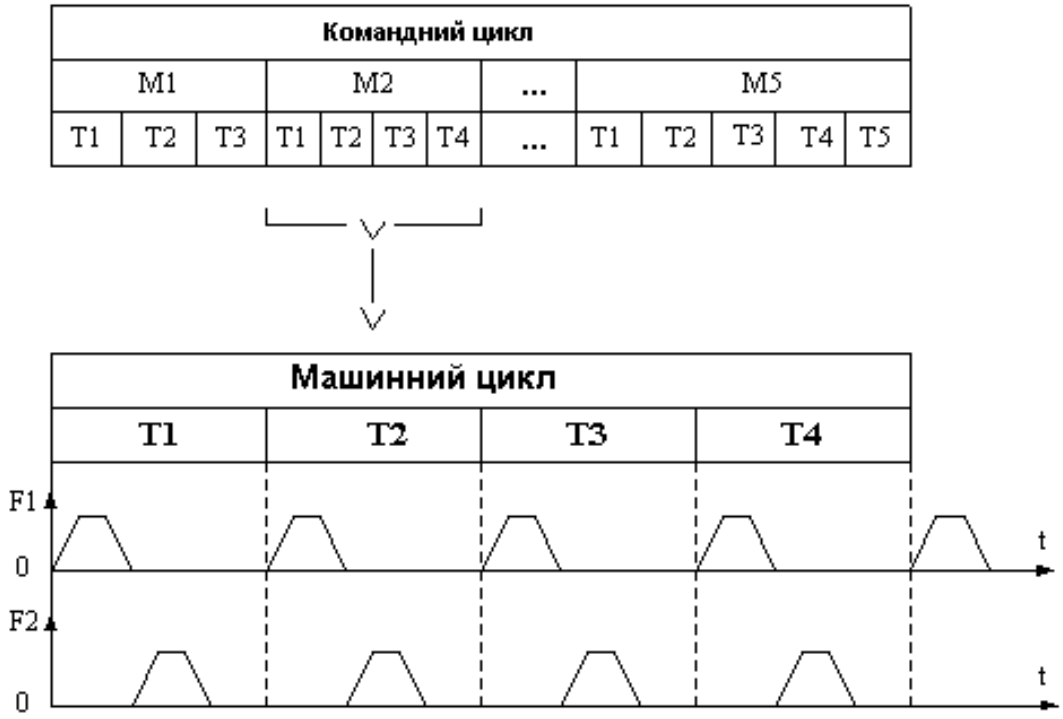


Рисунок 6.8 – Формат командного циклу МП i8080

В МК типу МК51 всі машинні цикли однакові і складаються з 6 станів: (S1, S2, ... , S6) по дві фази в кожному стані (P1, P2). Тривалість однієї фази дорівнює періоду тактової частоти f_{BQ}. Тобто один машинний цикл складається з 12 фаз (12 періодів f_{BQ}). Знаючи значення f_{BQ}, можна визначити $T_{BQ}=1/f_{BQ}$, а отже, і час машинного циклу $t_{МЦ}=T_{BQ}*12$.

Більшість команд МК типу МК51 виконуються за 1 або 2 машинних цикли, а множення і ділення – за 4 машинних цикли. Якщо, наприклад, f_{BQ}=12МГц, то $T_{BQ}=1/12*10^{-6}$ с, $T_{МЦ}=1/12*12*10^{-6}$ =1 мкс, а окремі команди виконуються за 1, 2 або 4 мкс.

Одна з колонок таблиць, в яких приводяться команди та їх характеристики для 16–го МП–ра i8086, містить інформацію про час виконання команд в кількості тактів. В цій колонці може стояти або конкретне число, що відображає час виконання даної команди в тактах, або приводиться вираз:

$$n + T_{BA},$$

де n – конкретне значення,

а T_{BA} – додаткове число тактів, необхідне для обчислення ефективної (виконавчої) адреси операнда в пам'яті.

Це значення визначається з таблиці 6.5 за відомим способом адресації операнда, що застосовується у відповідній команді.

Таблиця 6.5 – Визначення часу обчислення ефективної адреси T_{EA}

Значення T_{EA} в тактах в залежності від способу адресації операндів							
mod	**r/m**	**Тип адресації**	Такти	**mod**	**r/m**	**Тип адресації**	Такти
00	110	Пряма	6	00	000	Базово–індексна (BX)+(SI)	7
00	111	Непряма (BX)	5	00	001	(BX)+(DI)	8
				00	010	(BP)+(SI)	8
				00	011	(BP)+(DI)	7
00	100	Непряма (SI)	5				
00	101	(DI)	5			Базово–індексна з зсувом	
01	110	Базова (BP)+DATA8	9	01	000	(BX)+(SI)+DATA8	11
10	110	(BP)+DATA16	9	10	000	(BX)+(SI)+ DATA16	11
01	111	(BX)+DATA8	9	01	001	(BX)+(DI)+ DATA8	12
10	111	(BX)+DATA16	9	10	001	(BX)+(DI)+ DATA16	12
01	100	Індексна (SI)+DATA8	9	01	010	(BP)+(SI)+ DATA8	12
10	100	(SI)+DATA16	9	10	010	(BP)+(SI)+ DATA16	12
01	101	(DI)+DATA8	9	01	011	(BP)+(DI)+ DATA8	11
10	101	(DI)+DATA16	9	10	011	(BP)+(DI)+ DATA16	11

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Дайте визначення мові Асемблер.
2. Назвіть та дайте визначення основних керуючих програм, які використовуються при створенні програм на мові Асемблер.
3. Поясніть наступні терміни: мнемоніка команди та мнемокод; код операції команди (КОП); операнди; коментар.
4. Наведіть та поясніть формати команд 8–го МП–ра та МК–ра і 16–го МП–ра.
5. Наведіть та поясніть формати даних 8–го МП–ра та МК–ра і 16–го МП–ра.
6. Поясніть, що таке прапорці та яку роль вони відіграють у роботі МПС.
7. Як можна обчислити час виконання окремих команд у МК–рі та МП–рі?
8. Поясніть послідовність створення EXE–програми для МПС на основі МП–ра i8086.
9. Поясніть формат лістинга програми на прикладі команд пересилань.
10. Поясніть окремі частини шаблону, який можна використовувати для створення вихідних EXE–програм.
11. Наведіть та поясніть приклади префіксних команд.
12. Як можна виконати перепризначення сегментів?
13. Що таке макроси і як вони використовуються при програмуванні мовою асемблера? Наведіть приклад.
14. Що таке підпрограми і як вони використовуються при створенні асемблерних програм? Наведіть приклад.

ЛІТЕРАТУРА [1…10, 14, 15, 16, 29, 31, 36…39]

7 СПОСОБИ АДРЕСАЦІЇ ОПЕРАНДІВ

7.1 Способи адресації операндів мікропроцесора i8086

7.1.1 Загальні відомості

Тип звернення (адресації) до операндів (даних, що приймають участь в операції) називають способом адресації. До способів адресації операндів також відносять те, як в командах передачі керування дається вказівка на адресу переходу. У мікропроцесорі ВМ86 застосовуються наступні способи адресації: неявна, регістрова, безпосередня, пряма, непряма, базова, індексна, базово–індексна, стекова, відносна, адресація рядків, адресація портів введення/виведення.

У багатьох командах МП–ра i8086 для адресації операндів використовується постбайт, формат якого розглядається нижче.

7.1.2 Формат постбайта та його вплив на обчислення ефективної (виконавчої) адреси ЕА (ВА)

Адреса, що обчислюється мікропроцесором для адресації операнда в пам'яті чи адреси переходу, називається виконавчою (ефективною) адресою і позначається ЕА. Виконавча адреса ЕА являє собою зсув у сегменті (ціле беззнакове число) і обчислюється в залежності від способу адресації, що для багатьох команд визначається постбайтом (рисунок 7.1).

Дворозрядне поле mod (md) разом з полем r/m задають спосіб формування виконавчої адреси ВА. Поле mod визначає зсув (disp), а r/m – регістри, вміст яких бере участь в обчисленні ЕА. Виконавча адреса є 16–розрядним зсувом у сегменті пам'яті (сегментах даних, стека, коду та екстра).

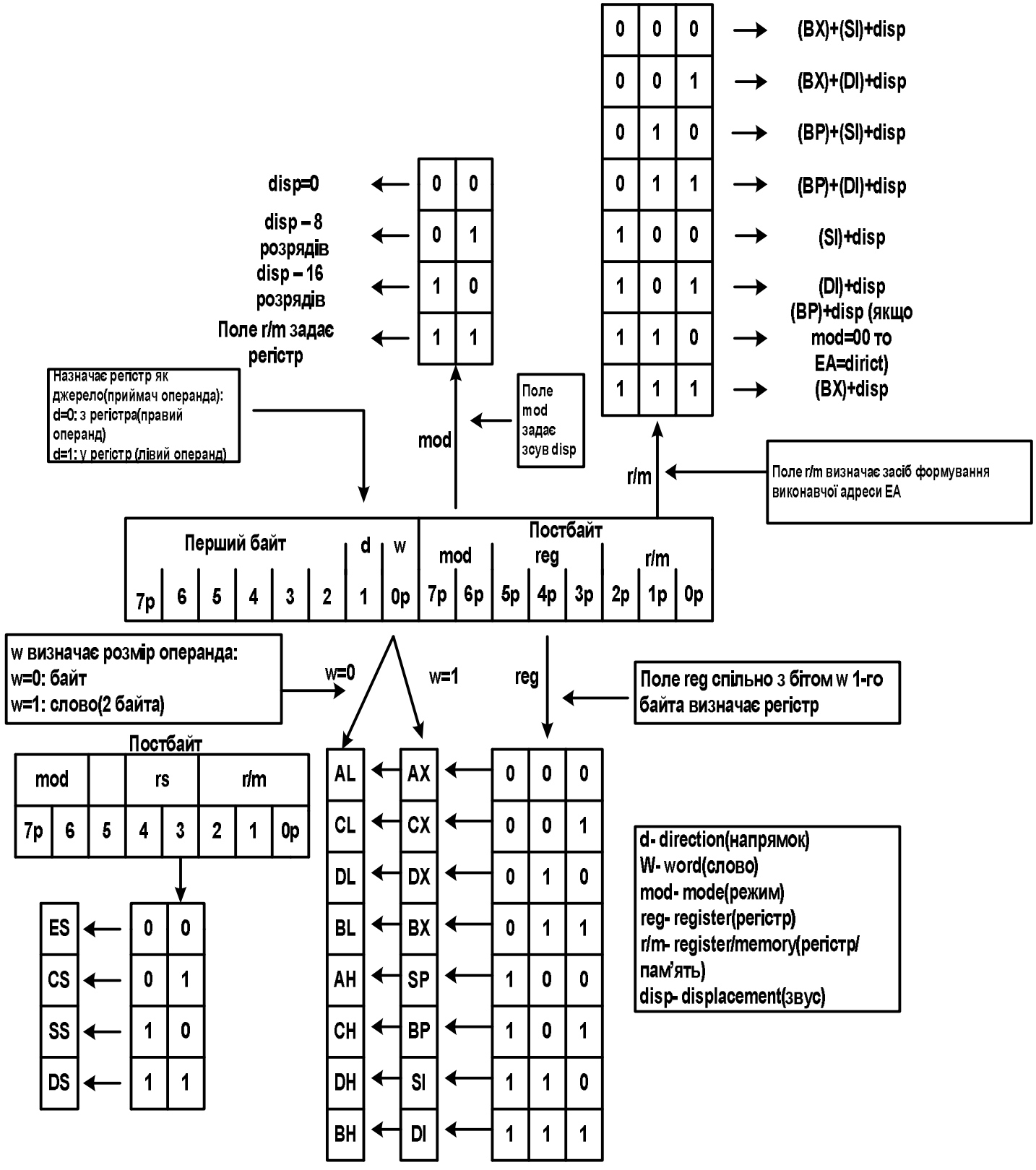


Рисунок 7.1 – Формат постбайта команди та його вплив на спосіб адресації

Якщо при обчисленні ЕА виникає перенесення, то воно ігнорується відповідно до правил додавання чисел у додатковому коді. Таким чином фізична адреса завжди виявляється в межах одного сегмента. Крім ефективної адреси ЕА постбайт може задавати адресу регістра, що приймає участь в операції. Для цього трьохрозрядне поле reg разом з полем w першого байта команди задає або 8–розрядний регістр (при w=0), або 16–розрядний регістр (при w=1). Наприклад, reg=110 і w=1 визначають регістр SI, а той же код у полі reg при w=0 указує на регістр DH (старший байт регістра DX).

Якщо поле mod=11, то поле r/m аналогічно полю reg задає регістр. У таблиці 7.1 приведено різні варіанти адресації за допомогою постбайта.

Таким чином, постбайт визначає дві адреси: регістра та ЕА. У командах пересилання, наприклад, один з них може бути адресою джерела інформації, другий – адресою приймача. Конкретний вибір регістра, що задається полем reg, у якості приймача/джерела інформації залежить від вмісту біта d першого байта команди. При d=0 поле reg задає джерело, а ЕА – приймач. При d=1 поле reg задає приймач, а ЕА – джерело. У двомісних арифметико–логічних операціях поле reg при d=1 задає лівий операнд і приймач результату, а ЕА – правий операнд.
При d=0 адреса ЕА задає лівий операнд і приймач результату, а reg правий операнд. У ряді команд (із другим безпосереднім операндом або з одним операндом) поле reg використовується для розширення коду операції й у цьому випадку трактувати це поле як адресу регістра не можна.

7.1.3 Неявна адресація

При неявній адресації в команді відсутня адреса операнда в явному вигляді. Інформацію про потрібну адресу МП одержує з коду операції команди, тобто говорять що операнд адресується неявно. Так, наприклад, у

розглянутому процесорі можуть адресуватися: регістр – акумулятор, регістр прапорців і т. ін.

Таблиця 7.1 – Режими адресації за допомогою постбайта

mod / r/m	0 0	0 1	1 0	11,r/m=reg	
				W=0	W=1
000	(BX) + (SI)	(BX)+(SI)+disp8	(BX)+(SI)+disp16	AL	AX
001	(BX) + (DI)	(BX)+(DI)+disp8	(BX)+(DI)+disp16	CL	CX
010	(BP) + (SI)	(BP)+(SI)+disp8	(BP)+(SI)+disp16	DL	DX
011	(BP) + (DI)	(BP)+(DI)+disp8	(BP)+(DI)+disp16	BL	BX
100	(SI)	(SI) + disp8	(SI) + disp16	AH	SP
101	(DI)	(DI) + disp8	(DI) + disp16	CH	BP
110	direct = disp16	(BP) + disp8	(BP) + disp16	DH	SI
111	(BX)	(BX) + disp8	(BX) + disp16	BH	DI

Команда XCHG AX, BX; AX↔BX здійснює обмін словами між двома регістрами AX і BX. З машинного коду команди

$$1\,0\,0\,1\,0\,0\,\underbrace{1\,1}_{\text{BX}} = 93\text{H}$$

можна побачити, що один з регістрів – BX, адресується за допомогою регістрової адресації (див. нижче), а другий – AX, адресується неявно.

7.1.4 Регістрова адресація

При регістровій (прямій регістровій) адресації операнд міститься в одному з регістрів МП. Код регістра вказується або в першому байті команди, або в постбайті. Команди такого типу найбільш короткі і виконуються за мінімальний час. На рисунку 7.2 приведено приклади команд із регістровою адресацією.

Регістри загального призначення можуть бути джерелами операндів, їх приймачами чи обома одночасно. Сегментні регістри можуть

використовуватися як джерела або приймачі операндів, тобто пересилати вміст одного із сегментних регістрів в інший однією командою неможливо.

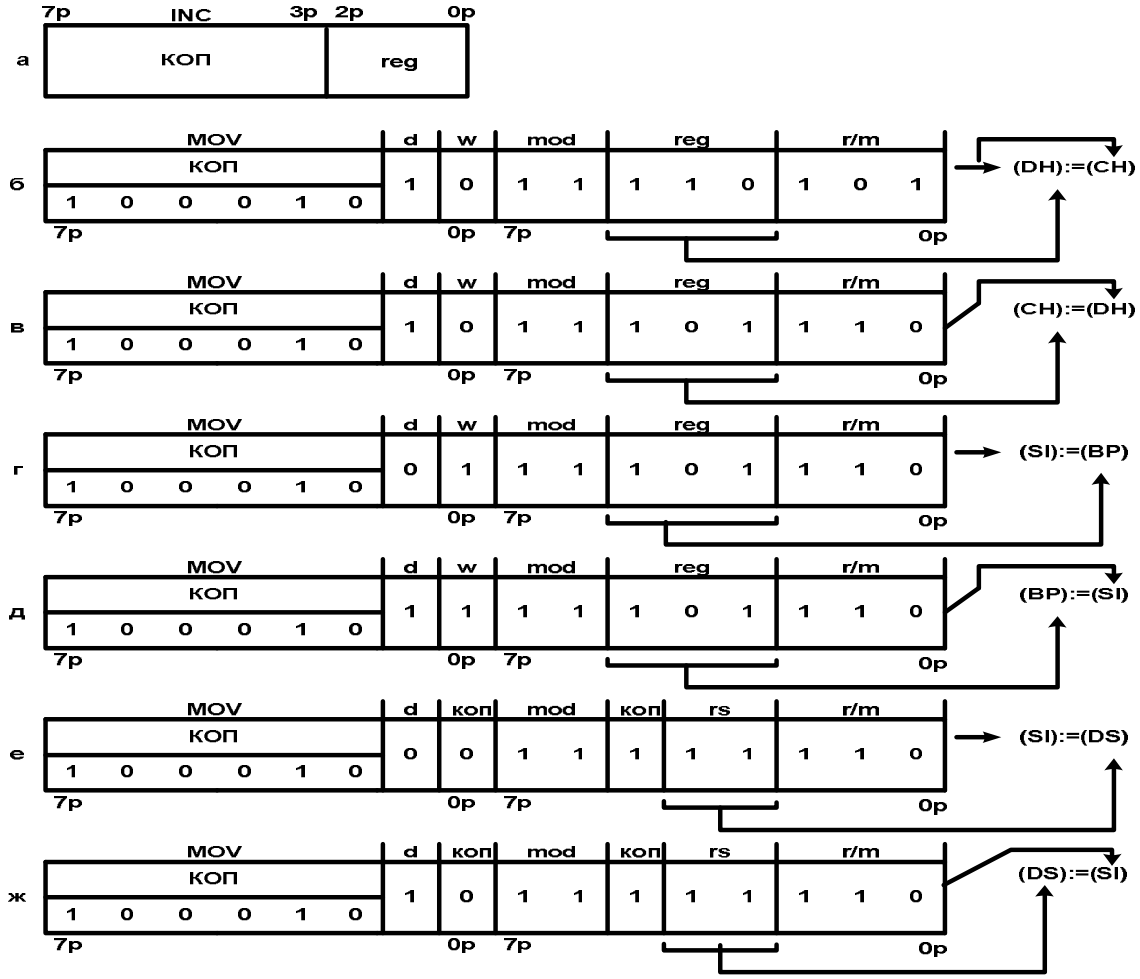
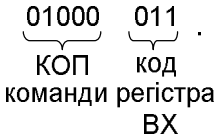


Рисунок 7.2 – Приклади команд з регістровою адресацією

Для виконання цієї задачі необхідно використовувати один з РЗП як проміжний регістр.

Команда INC reg (рисунок 7.2, а) виконує інкремент вмісту одного з 16–розрядних РЗП: AX,CX,DX, BX, SP, BP, SI чи DI. У поле reg компілятор розміщує трьохрозрядний код регістра, що вказується в мнемоніці команди.

Наприклад, мнемоніці INC BX буде відповідати машинний код



На рисунку 7.2, б показано машинний код команди MOV DH,CH; DH← CH. У цій команді біт w=0, тому що операнд має довжину 8 біт. Біт d=1 говорить про те, що операнд–регістр DH, що адресується полем reg постбайта, є приймачем інформації (у команді пересилання стоїть зліва).

На рисунку 7.2, в приведено машинний код аналогічної команди, де операнди помінялися місцями: MOV CH, DH; CH← DH.

На рисунку 7.2, г, д відповідно показано машинні коди команд: MOV SI, BP; SI←BP та MOV BP, SI; BP←SI. Довжина операндів дорівнює шістнадцяти бітам, тому поле w=1.

На рисунку 7.2, е, ж відповідно приведені коди команд: MOV SI, DS; SI←DS і MOV DS, SI; DS←SI, у яких один з операндів знаходиться в сегментному регістрі DS.

7.1.5 Безпосередня адресація

При безпосередній адресації операнд (data) входить у саму команду і витягується з пам'яті при виконанні команди відразу ж за її кодом операції.

Безпосередні операнди можуть мати довжину 8 чи 16 біт (рисунок 7.3).

На рисунку 7.3, а показано машинний код команди додавання ADD AL,85H; AL←AL+85H. Безпосередній операнд 85H додається до вмісту акумулятора AL і результат поміщається в AL, що адресується неявно. Оскільки в мнемоніці команди зазначено 8–розрядний регістр AL, то компілятор встановить біт W=0.

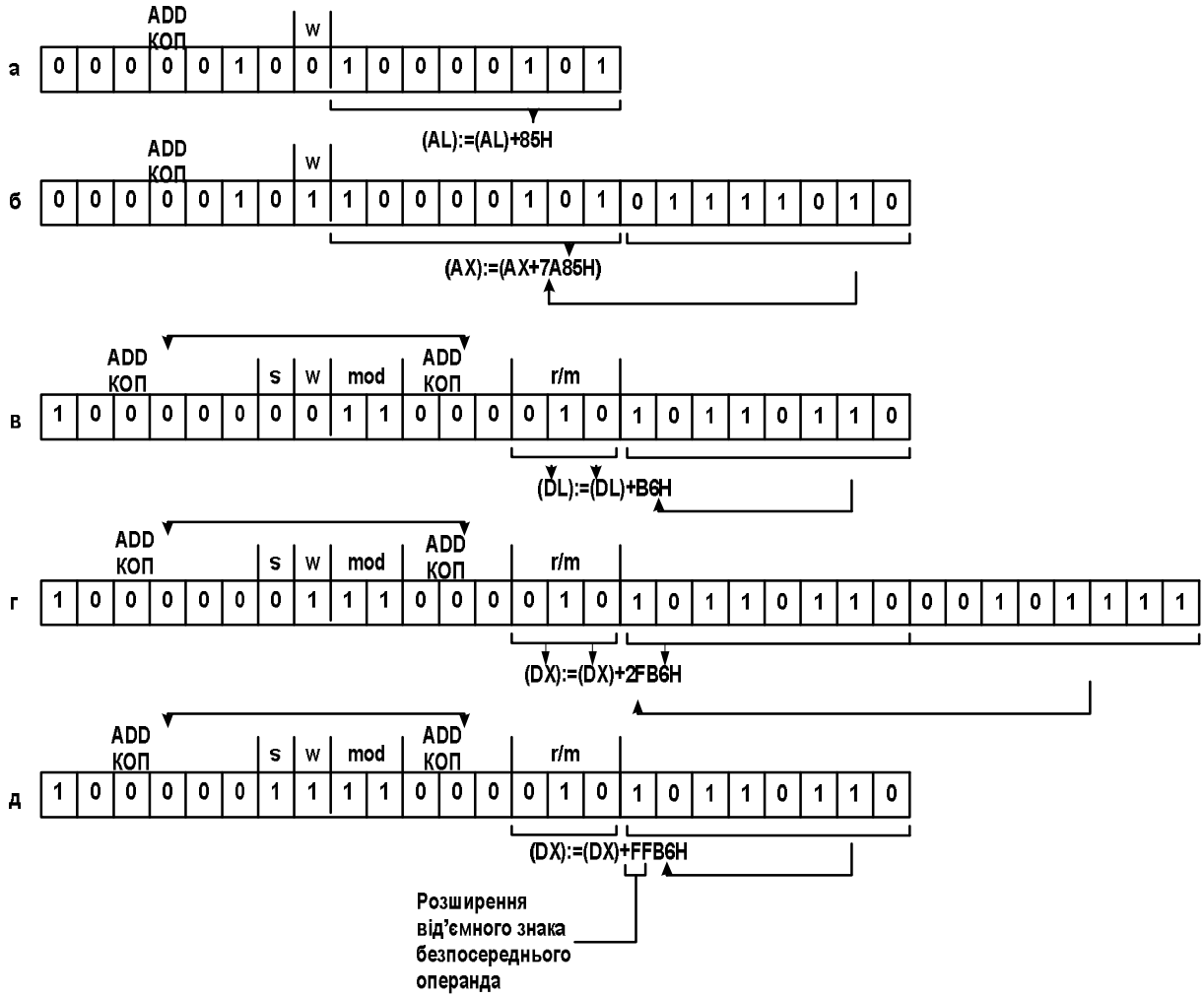


Рисунок 7.3 – Приклади команд з безпосередньою адресацією

На рисунку 7.3, б наведено машинний код команди додавання ADD AX, 7A85H; AX←AX+7A85H. Безпосередній операнд (БО) 7A85H складається з вмістом акумулятора AX, що адресується неявно, і результат додавання розміщується в AX. Оскільки в мнемоніці команди зазначено 16–розрядний регістр AX, компілятор встановить біт W=1. Команда має довжину 3 байти, що зберігаються в пам'яті в такому порядку: код операції, молодший байт БО, старший байт БО.

На рисунку 7.3, в показано машинний код команди додавання ADD DL,0B6H; DL←DL+0B6H. Безпосередній операнд 0B6H додається до вмісту регістра DL і результат операції розміщується в DL. Другий операнд (регістр DL) адресується за допомогою полів mod=11 і r/m=010 постбайту.

Через те, що поле reg у цьому випадку не використовується, воно доповнює код операції команди. Оскільки в мнемоніці команди зазначено 8–розрядний регістр DL, то компілятор установить біти S=W=0.

На рисунку 7.3, г наведено машинний код команди додавання ADD DX,2FB6H**;** DX←DX+2FB6H. Безпосередній 16–розрядний операнд 2FB6H додається до вмісту 16–розрядного регістра DX і результат розміщується в DX. Другий операнд (регістр DX) адресується за допомогою полів mod=11 і r/m=010 постбайта. Через те, що поле reg у цьому випадку для адресації не використовується, воно доповнює код операції команди. Оскільки в мнемоніці команди обидва операнди зазначені як 16–розрядні, то компілятор установить біти: S=0 , W=1.

На рисунку 7.3, д показано машинний код команди додавання ADD DX,FFB6H; DX← DX+FFB6H. Вміст регістра DX, що адресується за допомогою полів mod=11 і r/m=010 постбайта, додається до 16–розрядного безпосереднього операнда. Старший байт БО: FFH є розширенням знакового розряду молодшого байта: B6H. У машинному коді команди розміщується байт B6, що при виконанні команди розширюється до 16–розрядного операнда значенням знакового (старшого) розряду БО (у нашому прикладі дорівнює 1). У цьому випадку компілятор встановлює біти S=W=1, що вказує процесору на необхідність виконання описаних вище дій. Застосований механізм дозволяє економити одну комірку пам’яті.

Безпосередня адресація не може використовуватися для завантаження констант у сегментні регістри і включення констант у стек. У цьому випадку використовується проміжне завантаження констант в один з регістрів загального призначення. Відзначимо, що при записі констант (безпосередніх операндів) у шістнадцятковому коді мовою асемблер перед константою, що починається з букви, повинен стояти нуль. Безпосередні

операнди застосуються для ініціалізації регістрів загального призначення та комірок пам'яті, як еталонні значення в командах порівняння і т. ін.

7.1.6 Пряма адресація

При прямій (абсолютній) адресації адреса операнда EA міститься в самій команді (рисунок 7.4).

На рисунку 7.4, а приведено машинний код команди MOV AL, [dataByte]; AL← M(DS*16+offset dataByte).

Іншими словами дана команда пересилає в акумулятор AL значення 8–розрядної змінної з ім'ям dataByte, що розташована в сегменті даних зі зміщенням (offset dataByte) відносно початку сегмента, що дорівнює 0001H. У машинному коді цієї команди адреса (зміщення) змінної в сегменті даних зберігається в другому і третьому байтах команди.

Рисунок 7.4, б містить машинний код команди MOV AX, [dataWord]; AL ← M(DS*16+offset dataWord), AH ← M(DS*16+offset dataWord+1).

У цій команді в регістр AX пересилається значення 16–розрядної змінної dataWord, розташованої у двох комірках сегменту даних зі зміщеннями відносно початку сегмента: offset DataWord (дорівнює 0002H, адресує молодший байт змінної, що пересилається в AL (молодшу частину акумулятора AX)); (offset dataWord+1) (дорівнює 0003H, адресує старший байт змінної, що пересилається в AH (старшу частину акумулятора AX)).

Символічні імена dataWord, dataByte і т. ін. повинні бути визначені в програмі мовою асемблера.

У розглянутих двох прикладах пряма адреса (зміщення у сегменті даних) була розміщена в другому і третьому байтах коду команди.

Пряма адресація можлива також з використанням постбайта, у якому поле mod=00, а r/m=110 (рисунку 7.4, в). За постбайтом розміщується 16–

розрядна пряма адреса (зміщення операнда, що адресується в сегменті даних).

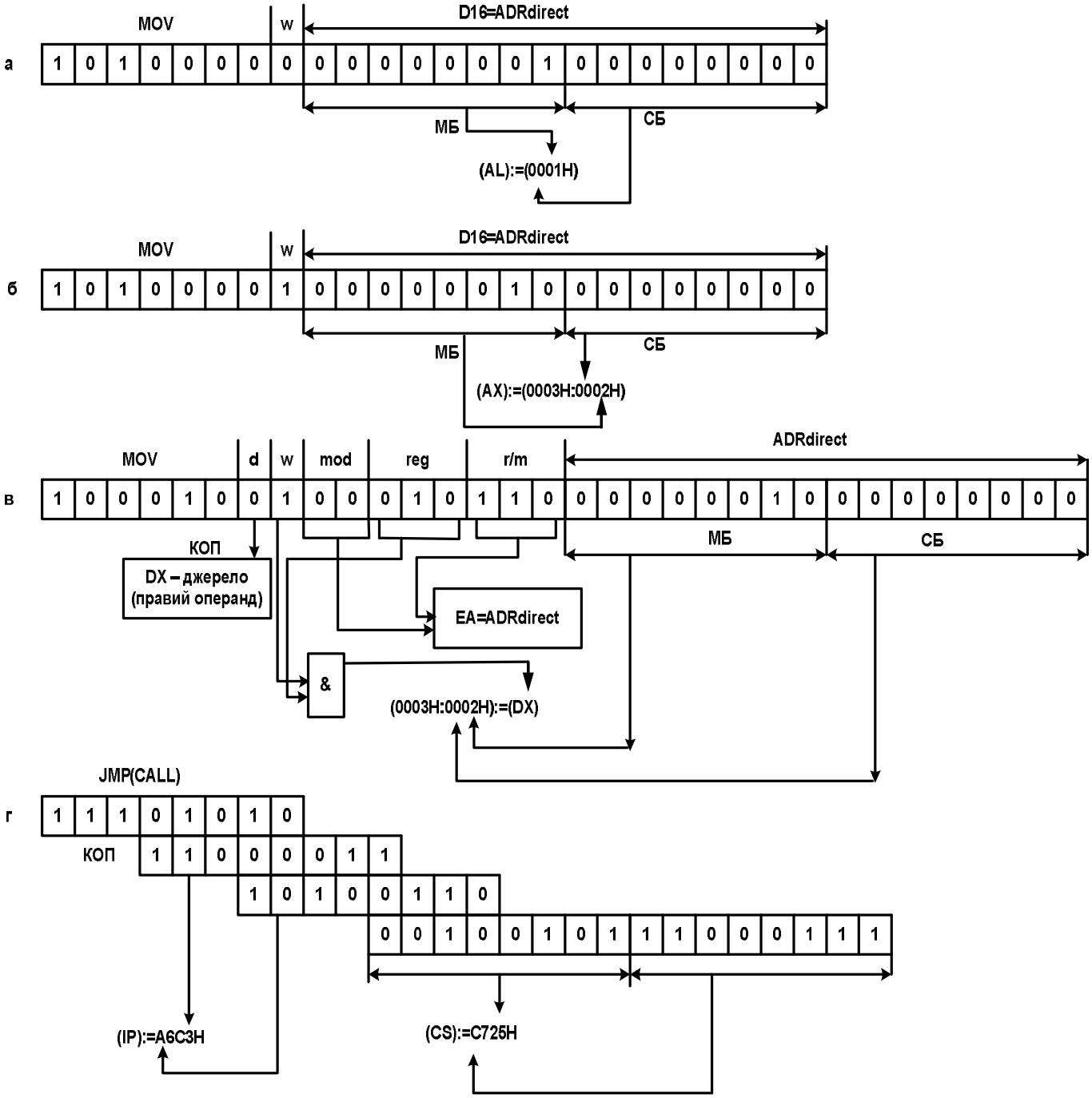


Рисунок 7.4–Приклади команд з прямою адресацією

На рисунку 7.4, в показано машинний код команди MOV [dataWord], DX; M(DSx16+offset dataWord)←DL, M(DSx16+offset dataWord+1)←DH.

Ця команда пересилає вміст 16–розрядного регістра DX в дві комірки сегмента даних, виділених для збереження змінної dataWord. Молодший байт цієї змінної має зміщення (offset dataWord) у сегменті даних, що дорівнює 0002H, а старший байт – зміщення 0003H. Регістр DX адресується за допомогою регістрової адресації (поле reg постбайта, що дорівнює 010).

Слід зазначити особливість прямої адресації при написанні програм мовою Асемблер. У деяких командах, наприклад, INC, MUL, ROR та ін. за іменем змінної неможливо визначити розмір операнда (змінної) – байт або слово. Це приклади так званих “анонімних” звернень до пам'яті. Щоб усунути цю неоднозначність, компілятору (асемблеру) потрібна додаткова інформація, яку повідомляє спеціальний оператор. Якщо, наприклад, необхідно зробити інкремент 8–розрядної змінної dataByte, необхідно записати INC [Byte dataByte]. Щоб зсунути на один розряд вправо 16–розрядну змінну dataWord необхідно записати ROR [Word dataWord], 1.

Пряма адреса в командах передачі керування (наприклад, JMP,CALL) задає адресу (зсув) у поточному сегменті команд CS (JMP NEAR LABEL). Існують команди передачі керування з довгою (4 байти) прямою адресою для переходів між сегментами. У таких командах (рисунок 7.4, г) другий і третій байти визначають новий вміст регістра IP, а четвертий і п’ятий адресу нового сегмента команд CS (JMP FAR LABEL).

7.1.7 Непряма адресація

У цьому режимі ефективна адреса операнда EA знаходиться в одному з регістрів BX, SI, DI або BP (рисунок 7.5). Отже, адреси пам'яті можна обчислювати і змінювати під час виконання програми. Шляхом зміни вмісту названих регістрів одна й та сама команда може звертатися до різноманітних комірок пам'яті. Використання регістра в якості джерела EA вказується взяттям його імені в квадратні дужки, наприклад:

ADD AX , [DI]; AX← AX + M (DS*16+DI).

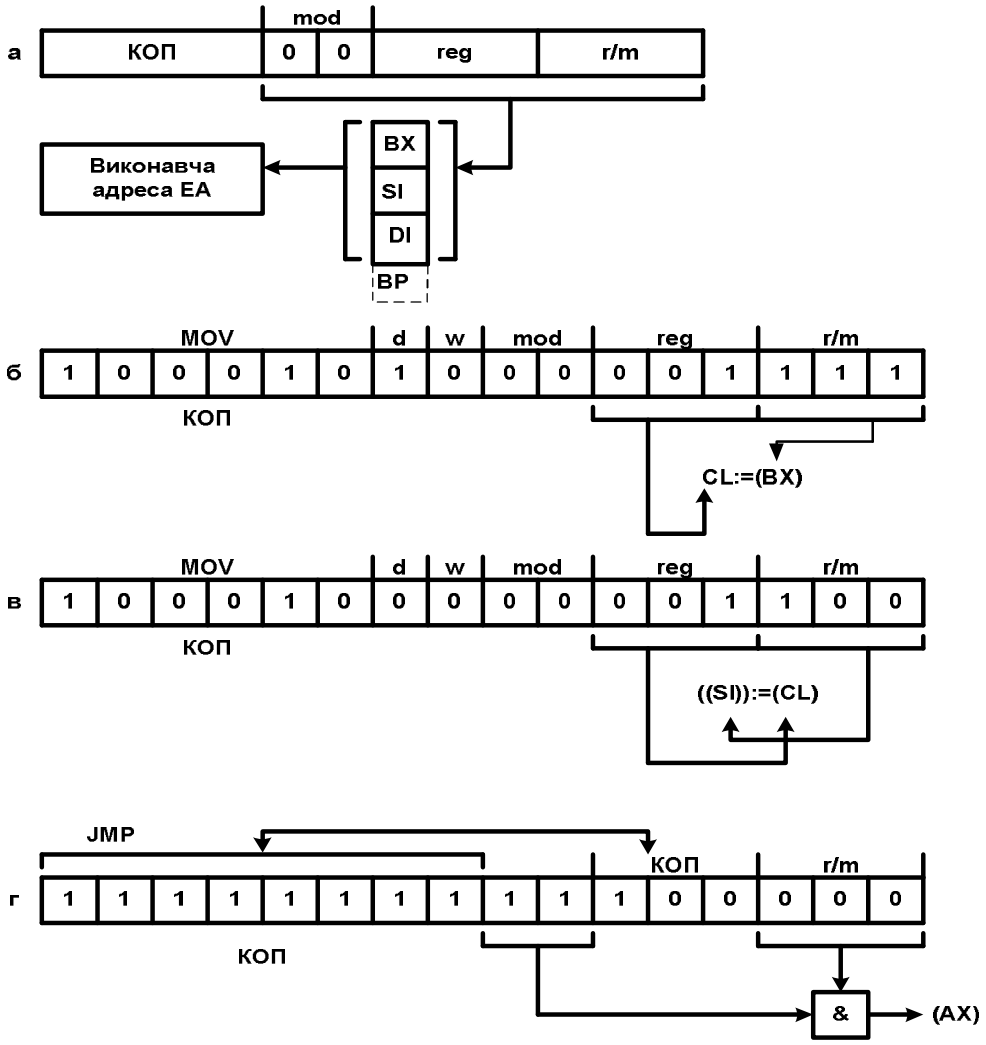


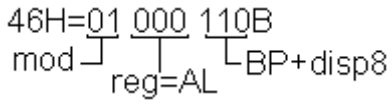
Рисунок 7.5 – Приклади команд з непрямою адресацією

Непряма (непряма регістрова) адресація реалізується за допомогою постбайта (рисунок 7.5, б, в).

Якщо непряма адреса зберігається в одному із регістрів BX, SI або DI (рисунок 7.5, а), то поле mod=00, r/m=100 (регістр SI),101(регістр DI) або 111 (регістр BX).

Якщо для збереження непрямої адреси використовується регістр BP, наприклад, виконується команда OR AL, [BP]; AL← AL ∨ M (SS*16+BP), то компілятор сформує наступні три байти:

1–й байт (код операції) – 0AH= 00001010B;

2–й байт (постбайт) – 

;

3–й байт (8–розрядний зсув) – disp8=00H=00000000B.

Оскільки комбінація: mod=00, r/m=100 використовується при прямій адресації (див 7.1), то компілятор встановив комбінацію: mod=01 і r/m=110, що вказує на те, що ефективна адреса операнда EA=BP+disp8.

Зсув disp8 входить у третій байт машинного коду команди і дорівнює 00H. Регістр BP для непрямої адресації не рекомендують використовувати через те, що довжина машинного коду команди збільшується на один байт.

На рисунку 7.5, б наведено машинний код команди

MOV CL, [BX]; CL← M (DS*16+BX).

На рисунку 7.5, в показано машинний код команди

MOV [SI], CL; M(DS*16+SI) ←CL.

Непряма адресація може використовуватися також у командах передачі керування (JMP, CALL). Вміст будь–якого з 16 – розрядних регістрів загального призначення (AX, BX, CX, DX, SI, DI, BP, SP) у цьому випадку є зсувом у кодовому сегменті CS. Тобто вміст названого регістра при виконанні команди завантажується у вказівник команд (програмний лічильник) IP.

На рисунку 7.5, г показано машинний код команди JMP AX, яка передає керування команді, що знаходиться в поточному сегменті коду CS зі зсувом, що дорівнює вмісту регістра AX. При виконанні цієї команди значення AX завантажується в регістр IP, після чого програма виконується, починаючи з цієї нової адреси.

7.1.8 Базова, індексна і базово–індексна адресації

7.1.8.1 Загальні відомості

При обчисленні ЕА часто користуються поняттями: **база** та **індекс**.

У видах адресації, що розглядаються нижче, в якості бази та індексу можуть використовуватися:

- вміст одного з регістрів: BX, BP, SI або DI;
- 16–розрядний зсув (disp16).

Різниця між базовою та індексною адресацією дуже умовна. Вирази для обчислення ЕА за формою однакові (рисунки 7.6, а; 7.7, а):

$$EA = \left.\begin{matrix} BP \\ BX \\ SI \\ DI \end{matrix}\right\} + зсув(disp).$$

Але при базовій адресації значення BP, BX, SI або DI є базою, а зсув: disp8 або disp16 – індексом.

При індексній адресації роль бази виконує disp16, а зміст BP, BX, SI або DI є індексом.

7.1.8.2 Базова адресація

При базовій адресації виконавча адреса ЕА може бути сумою вмісту регістрів BX∨BP∨SI∨DI і зсуву: disp8∨disp16 (число зі знаком в додатковому коді).

При базовій адресації в якості бази рекомендується використовувати регістр BX або BP (рисунок 7.6, а).

Якщо при обчисленні ЕА використовуються регістри BX, SI або DI, то автоматично вибирається поточний сегмент даних DS, а якщо база міститься в регістрі BP, то поточний сегмент – SS.

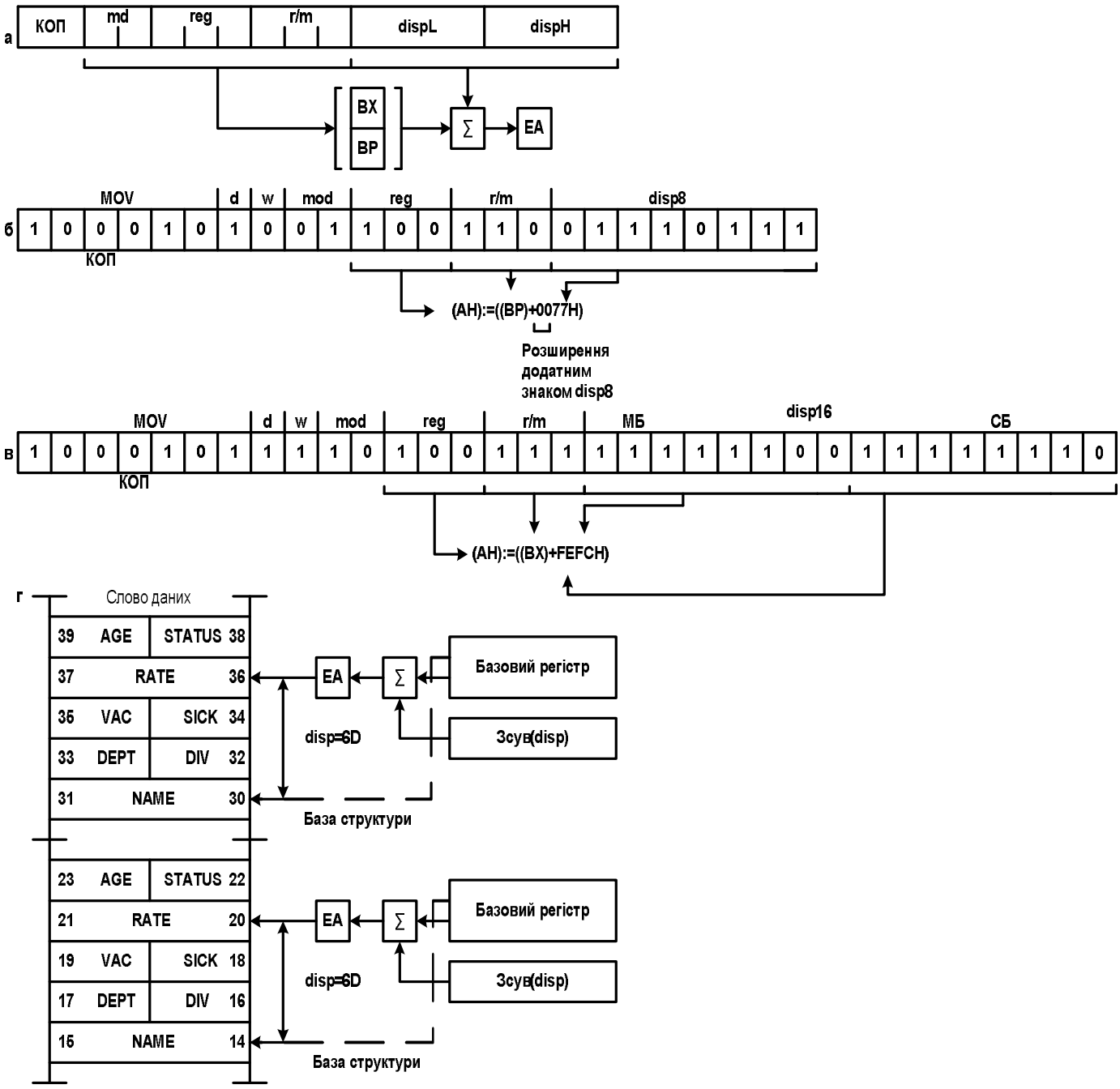


Рисунок 7.6 – Приклади команд з базовою адресацією

Існує можливість перепризначити сегменти DS і SS командою – префіксом заміни сегмента (див. підрозділ 5.3.8).

Якщо використовується зсув disp8, то це число зі знаком в діапазоні (–128…+127), яке при обчисленні EA розширюється до 16–ти розрядів значенням знакового (старшого) біта disp8. Короткі зсуви застосовують для економії одного байта пам'яті програм, якщо індекс вкладається в діапазон:

–128…+127 і немає сенсу використовувати disp16 (рисунок 7.6, б): MOV AH, [BP+77H]; AH←M(SS*16+BP+0077H).

В іншому прикладі (рисунок 7.6, в) 16–розрядний зсув 0FEFCH безпосередньо використовується при обчисленні ЕА:

MOV AH, [BX+0FEFCH]; AH←M(DS*16+BX+0FEFCH).

Головне застосування базової адресації пов'язане з обробкою однотипних структур даних, коли зсув (номер) елемента структури відомий при асемблеруванні програми. Це дозволяє звертатися до тих самих елементів однотипних структур даних, розташованих у різних областях пам'яті. Модифікація вмісту базового регістра забезпечує доступ до цих областей. Таким чином, при базовій адресації: індекс=const, а база=var і обчислюється при виконанні програми (рисунок 7.6, г).

7.1.8.3 Індексна адресація

При індексній адресації ЕА може обчислюватися як сума додатного 16–розрядного зсуву (disp16) та вмісту регістра SI∨DI∨BP∨BX. При індексній адресації в якості індексу рекомендується використовувати регістр SI або DI (рисунок 7.7, а, б). Зсув disp16=const виконує функцію бази при обчисленні ЕА, а зміст одного з названих вище регістрів виконує функцію індексу**,** який змінюється (зсув відносно постійної бази). Індексну адресацію зручно застосовувати для звернення до елементів масивів (рисунок 7.7, в). Зсув disp16 визначає відому при асемблеруванні початкову (базову) адресу масиву в сегменті, а значення регістра визначає положення потрібного елемента в масиві. Прості маніпуляції з вмістом регістра, що виконує функцію індексу, дозволяють звертатися до будь–якого елемента масиву.Таким чином, при індексній адресації: база=const, а індекс=var і обчислюється при виконанні програми.

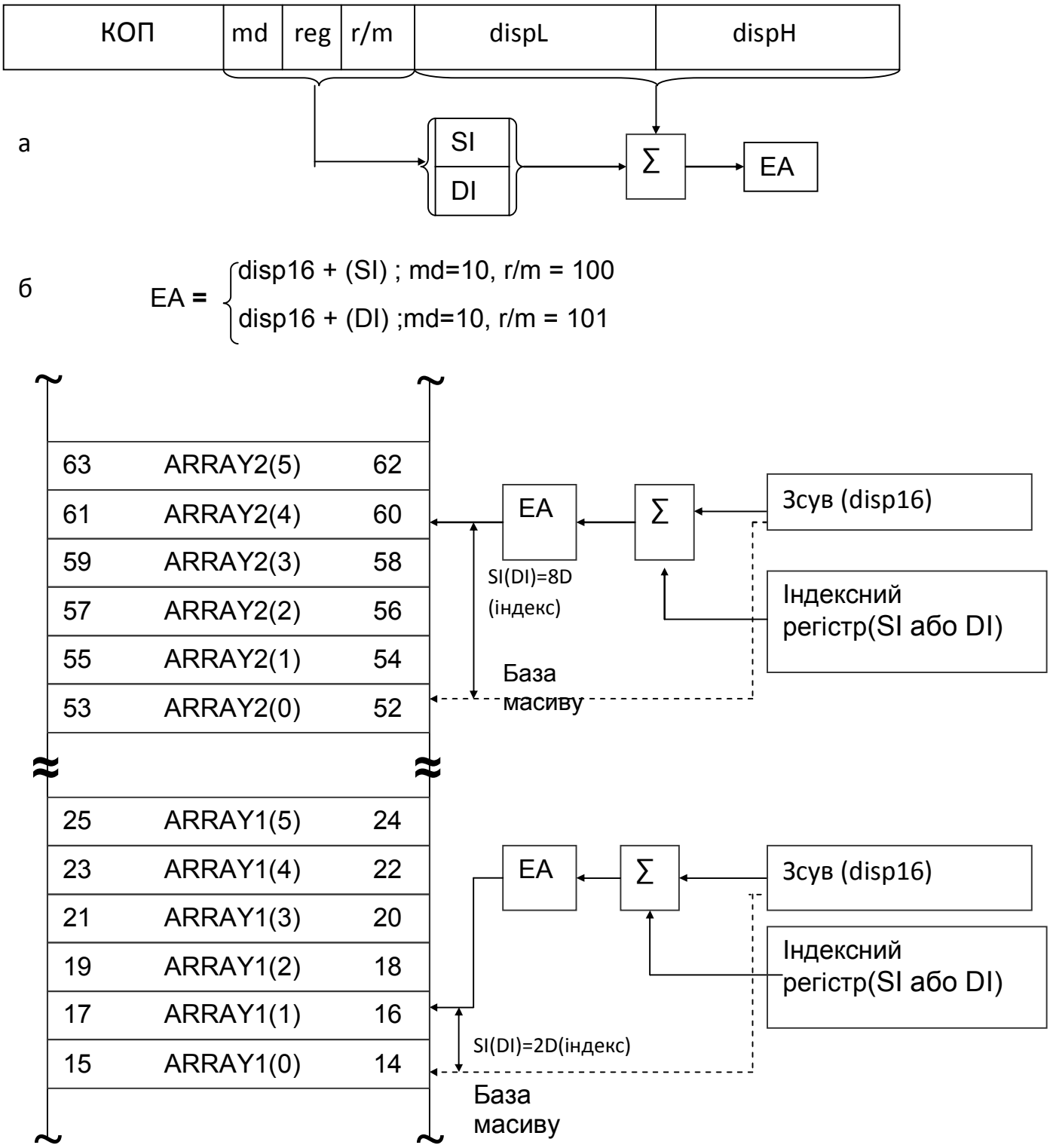


Рисунок 7.7 – Приклади команд з індексною адресацією

7.1.8.4 Базово–індексна адресація

Базово–індексна адресація генерує виконавчу адресу EA, яка є сумою значень базового та індексного регістрів (рисунок 7.8).

Така адресація є гнучким засобом доступу до найрізноманітніших ділянок пам'яті, тому що дві компоненти адреси можна визначати і модифікувати при виконанні програми. При використанні базового регістра BX виконавча адреса EA є зсувом у сегменті даних DS, а при використанні регістра BP – у сегменті стеку SS.

Прикладні програми можуть містити префіксні команди, які дозволяють перепризначати EA відносно інших сегментів пам'яті.

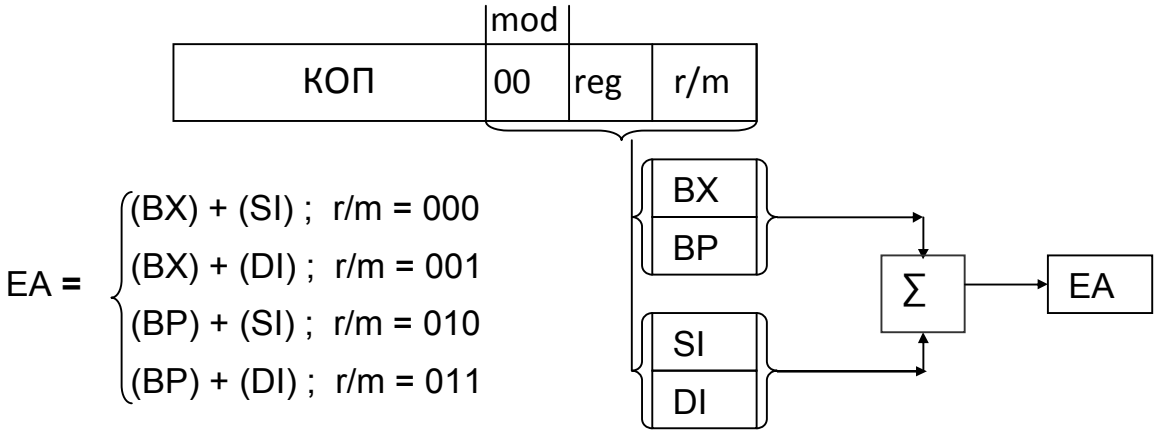


Рисунок 7.8 – Базово – індексна адресація

При базово–індексній адресації поле mod постбайта дорівнює 00, а r/m задає комбінації регістрів, що використовуються при обчисленні EA.

Таким чином, при базово–індексній адресації: база=var і індекс=var, які можуть обчислюватися програмою і пересилатися в регістри, що використовуються при цьому в якості бази та індексу.

Будь–який з чотирьох названих вище регістрів може бути базою або індексом, але рекомендується використовувати:

– BX∨BP у якості бази,

– SI∨DI у якості індексу.

Як приклад, базово–індексну адресацію можна використовувати для адресації елементів двомірних масивів, що розташовані на початку сегмента.

7.1.8.5 Базово – індексна адресація зі зсувом (зміщенням)

У випадку базово–індексної адресації зі зсувом в обчисленні ЕА бере участь вміст регістрів BX, BP, SI, DI і зсув, що може бути 8/16–розрядним числом зі знаком (рисунок 7.9).

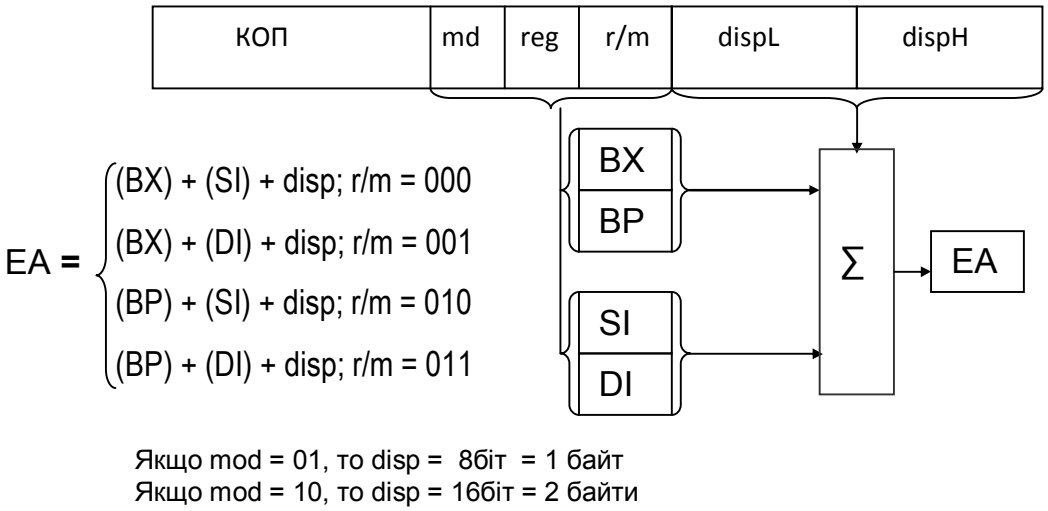


Рисунок 7.9 – Базово-індексна адресація зі зміщенням

Використання зсуву ще більше збільшує гнучкість програмування у порівнянні з базово–індексною адресацією. Наприклад, базово–індексна адресація зі зсувом може застосовуватися для забезпечення доступу до елементів масивів однотипних структур даних, розташованих у сегменті стека (рисунок 7.10) або даних.

Регістр BP є зсувом початку структури в сегменті стека відносно початку сегмента стека SS. Відстань початку масиву від початку структури дорівнює зсуву, що задано в команді. Вміст індексного регістра вибирає конкретний елемент масиву даних у стековій структурі.

Таким чином, при базово–індексній адресації зі зсувом можуть використовуватися:

– BX∨BP=var – початок структури (база);

– disp=const – початок масиву в структурі (зсув);

– SI∨DI=var – адреса елемента в масиві (індекс).

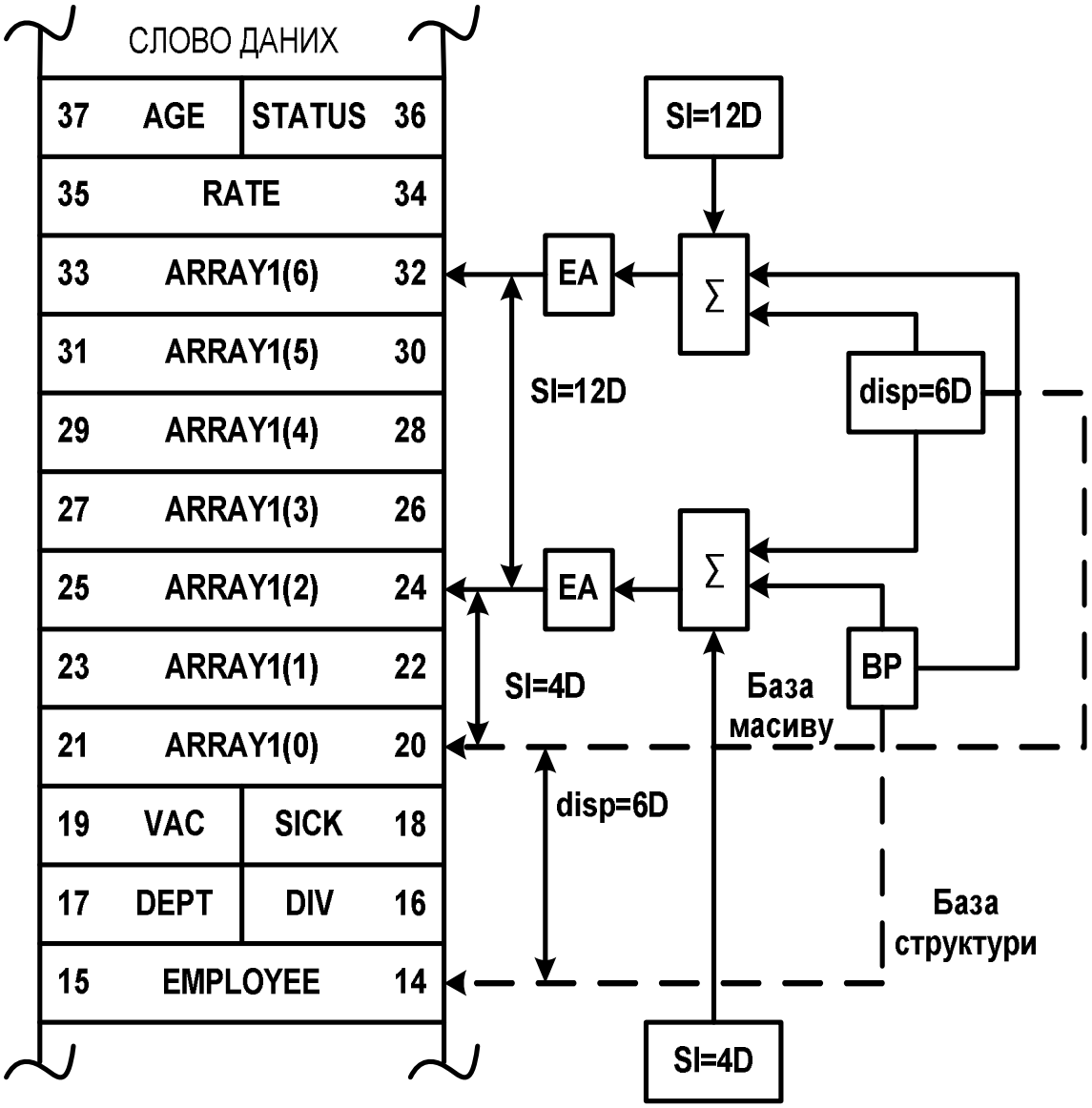


Рисунок 7.10 – Звернення до елемента масиву структури, розташованої в сегменті стеку за допомогою базово–індексної адресації зі зсувом

7.1.9 Стекова адресація

Стекова адресація виділена в самостійний вид адресації умовно, щоб підкреслити специфіку роботи зі спеціальною областю пам'яті, яка називається стеком. При записі до стеку (команда PUSH) стек розширюється, а при читанні зі стеку (команда POP) стек стискається.

В будь–якому випадку операція виконується відносно крайньої (останньої) заповненої комірки, яка називається “верхівкою стеку”.

Формально стек – це пам'ять з організацією типу “FILO–останній увійшов–перший вийшов”. У МП i8086 базова адреса сегменту пам'яті, що виділено під стек, зберігається в регістрі SS. Адреса верхівки стека знаходиться в регістрі–покажчику стека SP. Для адресації структур даних, що поміщаються у стек, використовується регістр BP, який бере участь в обчисленні виконавчої адреси (зсуву) у сегменті стека.

На рисунку 7.11 показано машинний код і особливості виконання команди

PUSH AX; $M(SSx16+(SP_{ПОЧ}–1)) \leftarrow AH$,

$M(SSx16+(SP_{ПОЧ}–2)) \leftarrow AL$,

$SP \leftarrow SP_{ПОЧ}–2$,

де $SP_{ПОЧ}$ – вміст регістра–покажчика стека SP до виконання команди.

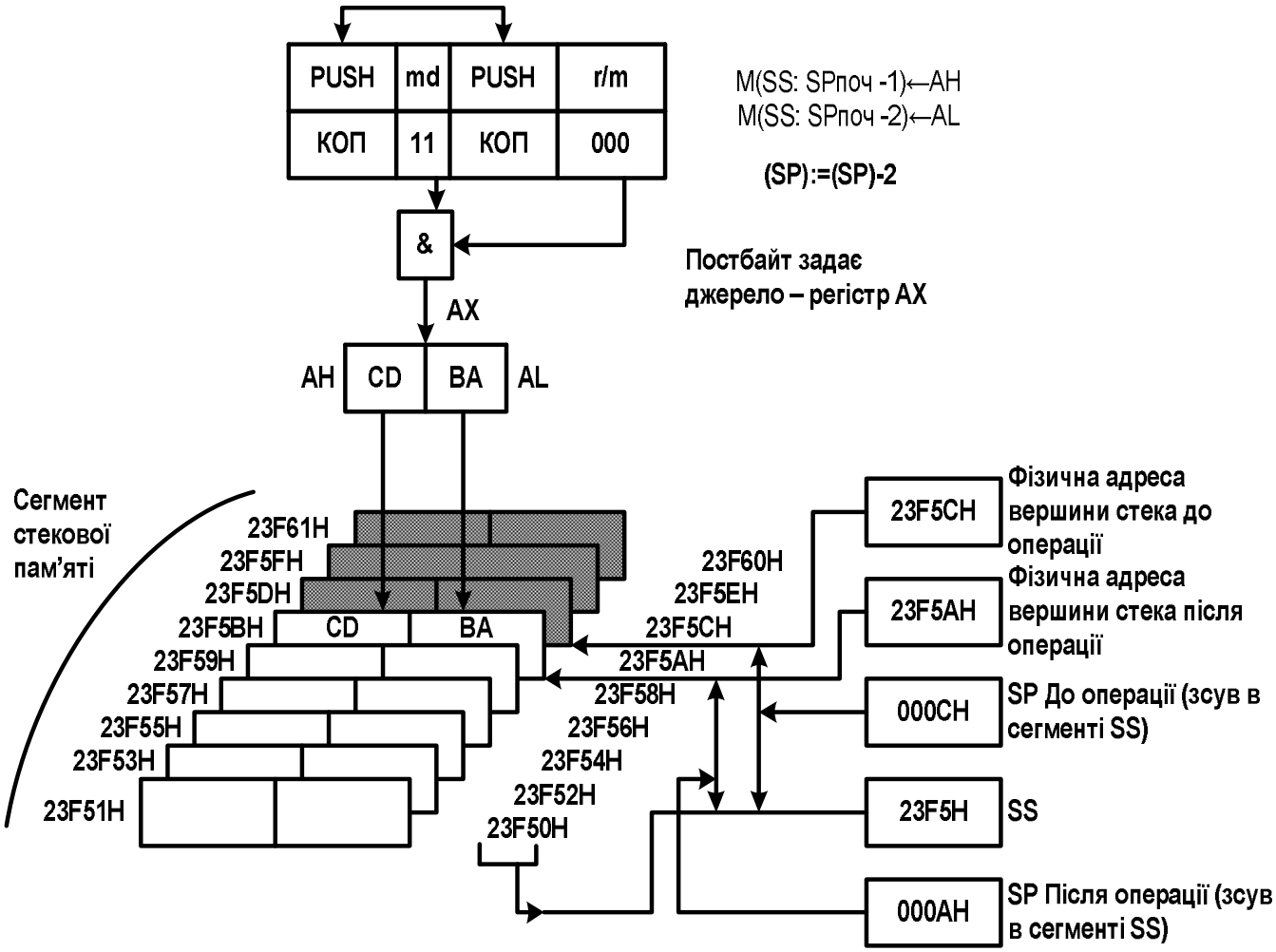


Рисунок 7.11 – Стекова адресація

7.1.10 Відносна адресація

Відносна адресація застосовується в командах передачі керування, наприклад JMP, CALL, LOOP. При цьому способі адресації в команді задається зсув відносно вмісту адреси команд, тобто відносно самої команди. Коротка відносна адреса використовується в командах безумовного та умовного переходів, циклу і задається 8–розрядним зсувом у діапазоні: –128…+127, який при обчисленні фізичної адреси розширюється до 16 розрядів знаковим бітом зсуву. На рисунку 7.12, а наведено приклад команди безумовного переходу з короткою відносною адресою.

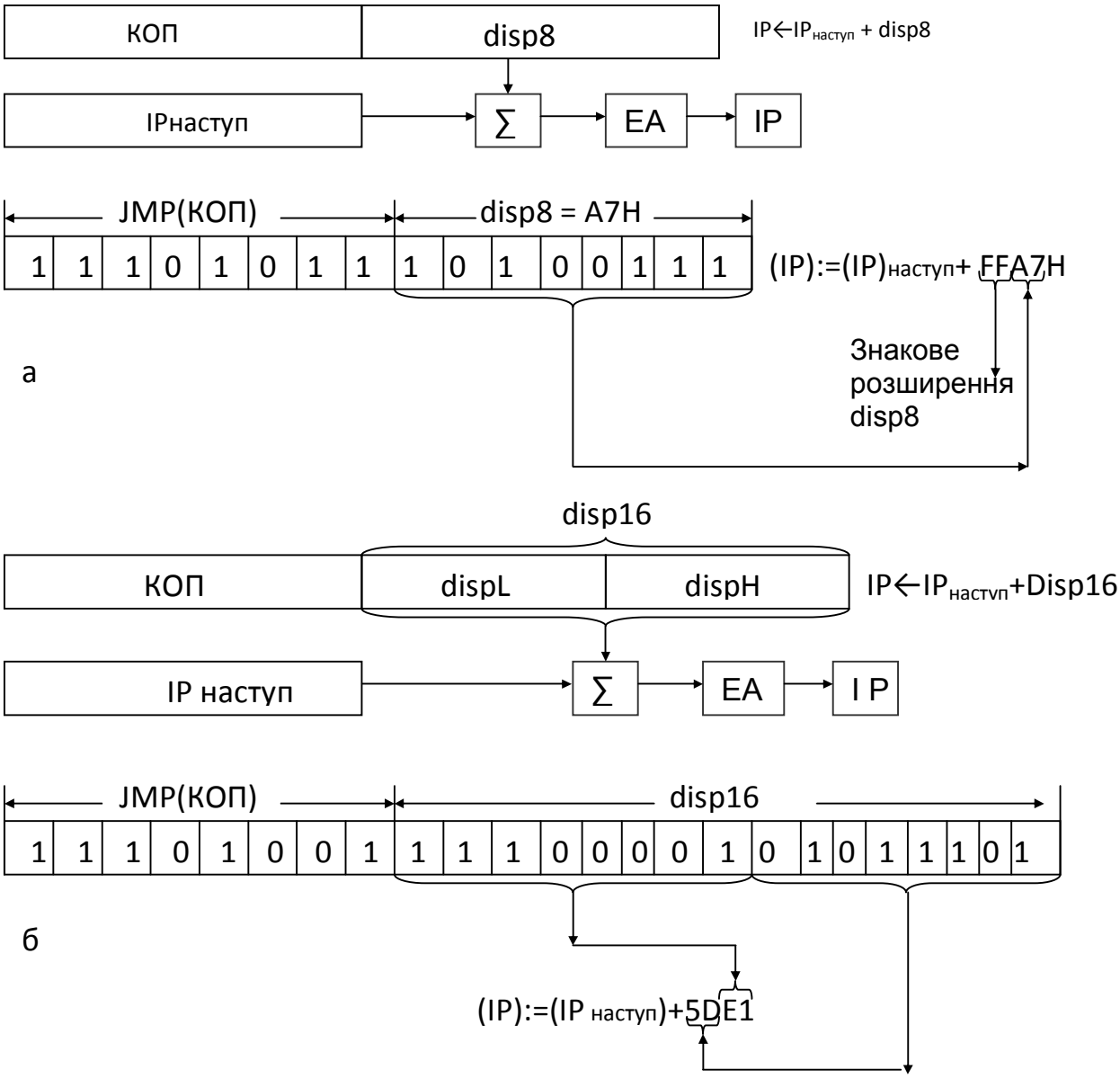


Рисунок 7.12 – Відносна адресація

Другий байт команди (disp8=A7H=1010 0111B) розширюється до значення FFA7H і додається до змісту вказівника команд IPнаступ., де IPнаступ. – адреса (зсув) наступної за JMP команди. Додавання FFA7H еквівалентно відніманню 59H=01011001B=89D з адреси, що міститься в IP (A7H є додатковим кодом числа: 89D). Після виконання даної команди МП перейде у програмі на 89 байт назад відносно адреси наступної команди (IPнаступ.).

На рисунку 7.12, а в якості IPнаступ. виступає вміст лічильника команд IP після витягування команди JMP із пам'яті, тобто IPнаступ. адресує наступну команду.

На рисунку 7.12, б наведено приклад команди переходу з 16–розрядним зсувом, який визначає безумовний перехід вперед по програмі на 5DE1H=24033 байти відносно адреси наступної команди.

У програмі, що написана мовою Асемблер, вказується не значення зсуву, а мітка тієї команди, якій передається керування. Необхідне значення зсуву disp автоматично обчислює системна програма–компілятор, що також, як і мова програмування, має назву “асемблер”.

7.1.11 Адресація рядків даних

Цей вид адресації використовується при виконанні операцій із рядками (ланцюжками) даних. Рядком (ланцюжком) називають послідовність байтів або слів, що розміщуються у сусідніх комірках пам'яті (наприклад, рядок, що вводиться з терміналу). При обробці елементів рядків апаратно передбачається, що регістр SI адресує байт або слово рядка джерела (звідси походить його назва “індекс джерела”), а індексний регістр DI – байт або слово рядка приймача, «індекс приймача» (рисунок 7.13).

Для формування адрес наступних елементів рядків МП автоматично корегує вміст регістрів SI і DI (інкрементує, якщо прапорець DF=0, або декрементує, якщо DF=1) при переході до наступних елементів рядків.

Рядок–джерело може знаходитися в будь–якому сегменті (за замовчуванням приймається сегмент даних DS), а рядок–приймач тільки в додатковому сегменті даних, логічна базова адреса якого знаходиться в регістрі ES.

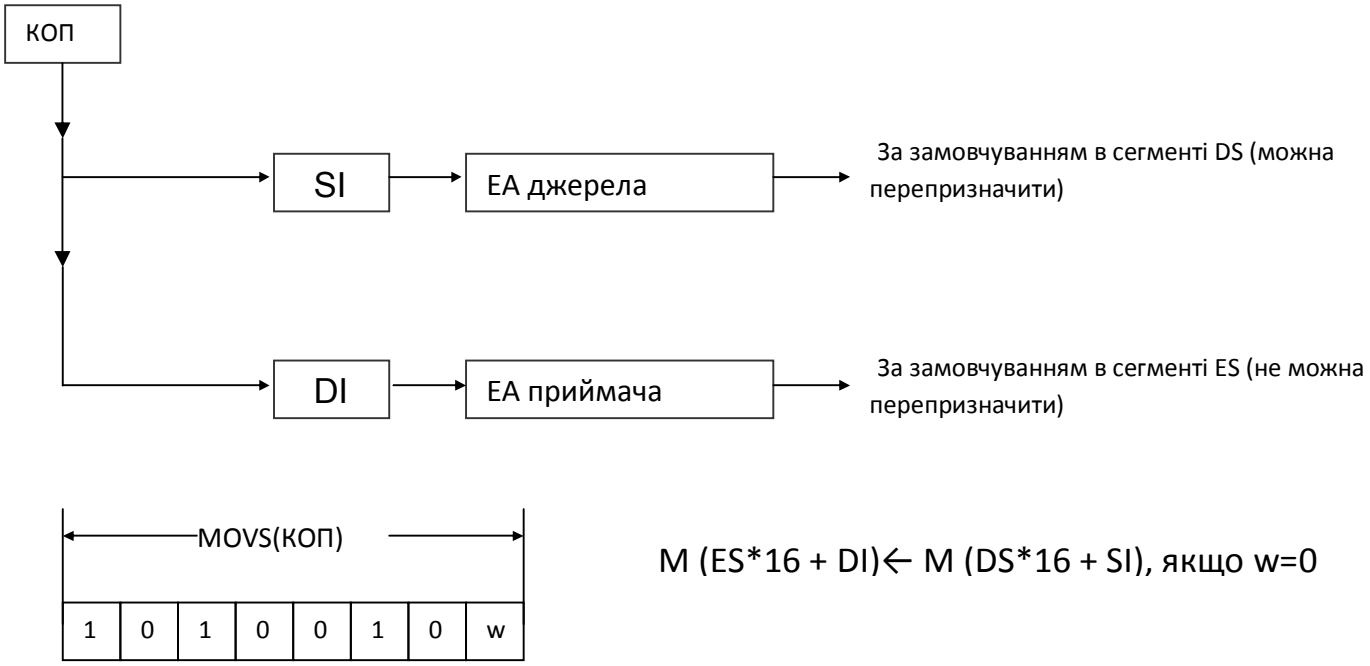


Рисунок 7.13– Адресація рядків даних

На рисунку 7.13 показано машинний код однобайтної команди MOVS – пересилання елемента рядка–джерела (байт/слово) на місце елемента рядка–приймача. В залежності від довжини елемента, ця команда мовою Асемблер має вигляд:

MOVSB; M(ES*16+DI)←M(DS*16+SI), якщо довжина операнда дорівнює 8 бітам (1 байт);

код операції команди MOVSB: 10100100B=A4H(W=0);

MOVSW; M(ES*16+DI)←M(DS*16+SI),

M(ES*16+DI+1)←M(DS*16+SI+1), якщо довжина операнда дорівнює 16 бітам (2 байти);

код операції команди MOVSW – 10100101B=A5H(W=1).

В залежності від значення прапорця DF при виконанні команди MOVS змінюється вміст регістрів SI і DI :

– при виконанні команди MOVSB – на ±1 (DF=0/1);

– при виконанні команди MOVSW – на ±2 (DF=0/1).

За допомогою однієї команди MOVS можна переслати тільки один елемент рядка. Для організації циклічного виконання команди MOVS (а також інших команд обробки рядків) використовується однобайтна команда–префікс REP (повторити). Вона записується перед головною командою (наприклад, REP MOVSB) і забезпечує її виконання N разів. Число повторень N попередньо записується в регістр CX. Після пересилання кожного елемента вміст CX зменшується на одиницю. Коли після чергового декремента CX=0, виконання команди закінчується.

7.1.12 Адресація портів введення/виведення

Мікропроцесор i8086, крім можливості адресації 1 Мбайта комірок пам'яті, може додатково адресувати: 64 К пристроїв (портів) введення/виведення (ПВВ), які обмінюються 8–розрядними даними або 32К ПВВ, які обмінюються 16–розрядними даними. Для адресації зовнішніх пристроїв (ЗВПР), розташованих в області адрес ЗВПР, використовується пряма і непряма адресації. При прямій адресації адреса ЗВПР 8–розрядна, що дозволяє адресувати: 256 8–розрядних або 128 16–розрядних портів введення/виведення (рисунок 7.14, а).

Наприклад, команда IN AL, 20H; AL←PORT(20H) вводить байт від зовнішнього пристрою (ЗВПР) в акумулятор AL через порт з адресою 20H. Ця команда займає 2 комірки пам'яті і має машинний код: 11100100B=E4H – 1–й байт; 00100000B=20H – 2–й байт.

Непряма адресація портів (рисунок 7.14, б) аналогічна непрямій адресації операндів. Адреса зовнішнього пристрою розміщується в регістрі DX, що дозволяє адресувати 65536 8–розрядних або 32768 16–розрядних портів введення/виведення. Вміст регістра DX можна змінювати в процесі

виконання програми і у такий спосіб звертатися до групи пристроїв введення/виведення в циклі.

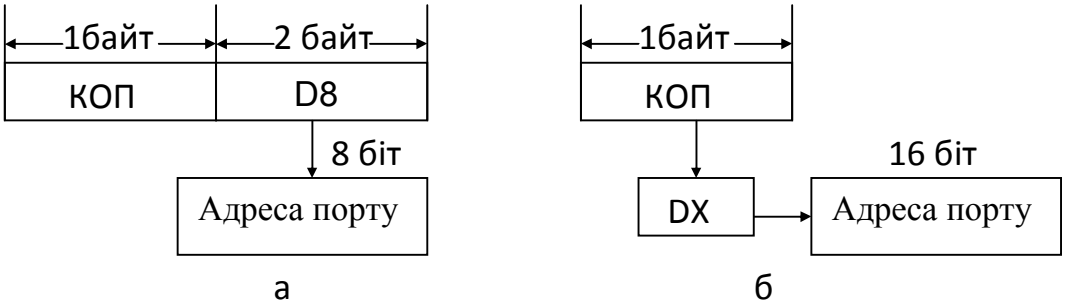


Рисунок 7.14 – Адресація портів введення/виведення

Наприклад, команда IN AL, DX; AL← PORT(DX) вводить байт від ЗВПР в акумулятор AL через порт, адреса якого міститься в регістрі DX. Дана команда займає одну комірку пам'яті і має машинний код 11101100B=ECH.

Крім розглянутої, може також використовуватись адресація ЗВПР як комірки пам'яті, що виконується з застосуванням зазначених вище способів адресації ЗП.

7.1.13 Спосіб адресація операндів і час виконання команд

При розробці програмного забезпечення, особливо для МПС реального часу, важливим є час виконання тієї чи іншої команди. Тому для кожного формату команди МП вказується кількість тактів n, необхідних для її виконання. Той самий формат деяких команд може бути використано для задання різних способів адресації і, отже, різноманітних способів обчислення виконавчої (ефективної) адреси (ВА/ЕА). Кількість тактів, необхідних для виконання такої команди

$$n_T=n+T_{BA},$$

де n – постійне значення, яке визначається з довідкових таблиць;

T_{BA} – кількість тактів, необхідних для обчислення ефективної адреси.

Час виконання кожної команди в секундах визначається виразом

$$T_K = (n + T_{BA}) \cdot t_T ,$$

де T_K – час виконання команди в секундах;

n =const, для конкретної команди береться з таблиць;

T_{BA} – час обчислення виконавчої (ефективної) адреси в тактах;

t_T – тривалість одного такту в секундах.

Значення t_T визначаються співвідношенням

$$t_T = 1/f_T,$$

де f_T – величина високої тактової частоти, що формується системним генератором.

Для визначення T_{BA} можна користуватися даними таблиці 7.15.

7.2 Призначення та перепризначення сегментів у МПС на основі МП–ра i8086

Оскільки в кожний момент часу програмі доступні чотири поточних сегменти: CS, DS, SS чи ES, має бути обговорена відповідність між джерелом адреси в сегменті (зсувом) і типом сегмента. За відсутності додаткових вказівок прийняті наступні призначення сегментних регістрів (рисунок 7.15).

Іншими словами, рисунок 7.15 задає правила, відповідно до яких для обчислення фізичної адреси комірки пам'яті мікропроцесор вибирає сегментний регістр за відомим джерелом зсуву в сегменті.

Таблиця 7.15 – Визначення часу обчислення ефективної адреси T_{EA}

Значення T_{EA} в тактах в залежності від способу адресації операндів							
mod	**r/m**	**Тип адресації**	Такти	**mod**	**r/m**	**Тип адресації**	Такти
00	110	Пряма	6	00	000	Базово–індексна (BX)+(SI)	7
00	111	Непряма (BX)	5	00	001	(BX)+(DI)	8
				00	010	(BP)+(SI)	8
				00	011	(BP)+(DI)	7
00	100	Непряма (SI)	5				
00	101	(DI)	5			Базово–індексна з зсувом	
01	110	Базова (BP)+DATA8	9	01	000	(BX)+(SI)+DATA8	11
10	110	(BP)+DATA16	9	10	000	(BX)+(SI)+ DATA16	11
01	111	(BX)+DATA8	9	01	001	(BX)+(DI)+ DATA8	12
10	111	(BX)+DATA16	9	10	001	(BX)+(DI)+ DATA16	12
01	100	Індексна (SI)+DATA8	9	01	010	(BP)+(SI)+ DATA8	12
10	100	(SI)+DATA16	9	10	010	(BP)+(SI)+ DATA16	12
01	101	(DI)+DATA8	9	01	011	(BP)+(DI)+ DATA8	11
10	101	(DI)+DATA16	9	10	011	(BP)+(DI)+ DATA16	11

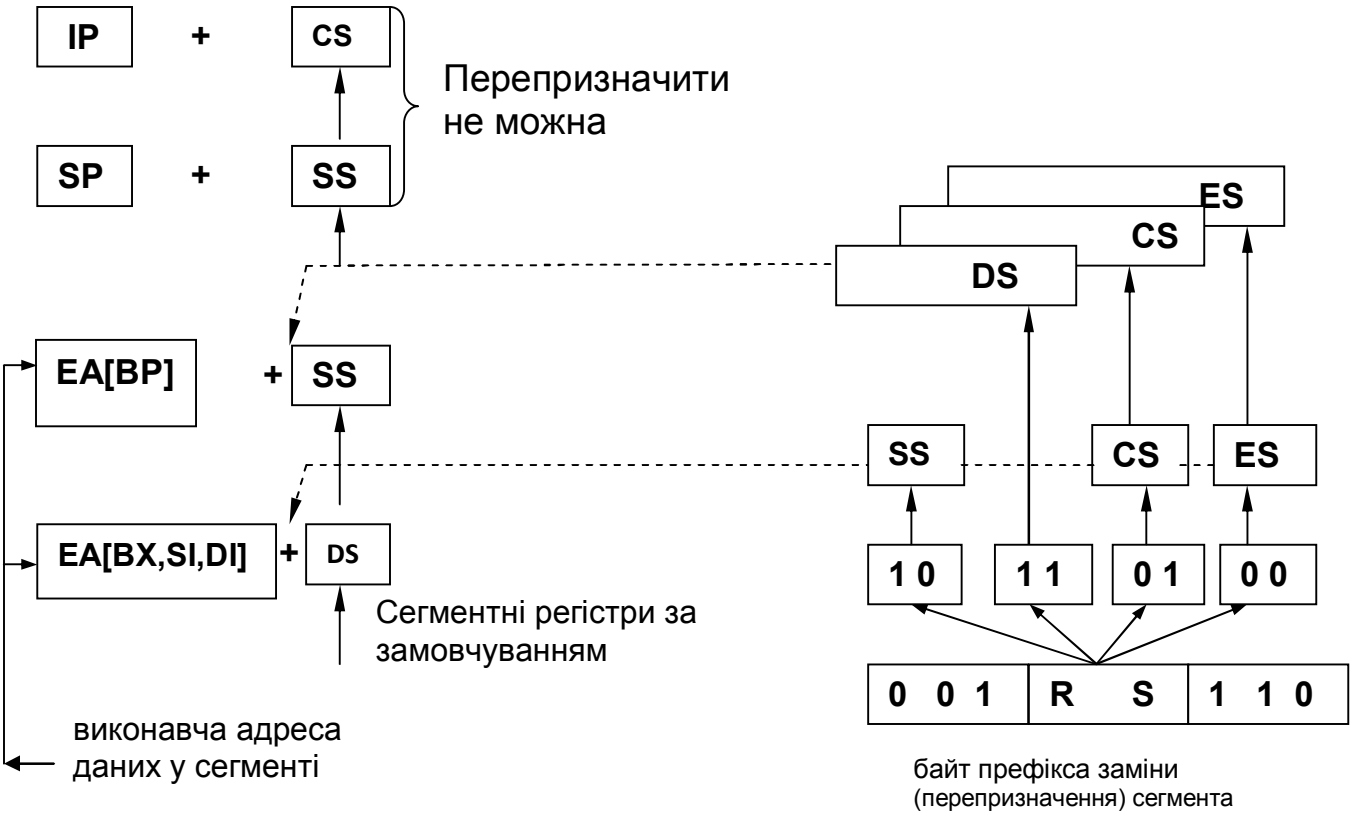


Рисунок 7.15 – Вибір типу сегмента пам’яті при обчисленні фізичної адреси

При читанні команд із пам'яті зсув зчитується з регістра – покажчика команд (програмного лічильника) IP. У цьому випадку при обчисленні фізичної адреси використовується сегментний регістр CS.

Якщо відбувається звернення до стеку командами PUSH чи POP, то значення зсуву в сегменті залежить від вмісту регістра – покажчика стека SP, а положення стекового сегмента в пам'яті визначається вмістом сегментного регістра SS.

При зверненні до даних, розташованих у поточних сегментах даних чи стека, мікропроцесор формує так звану виконавчу (ефективну) 16–розрядну адресу даних ВА (ЕА), що є зсувом відносно початку обраного сегмента. ЕА являє собою ціле беззнакове число.

Значення ЕА в залежності від зазначеного в коді операції команди способу адресації операндів, може визначатися вмістом одного з регістрів BX, BP, SI, DI, їхніми комбінаціями та константами.

Якщо для обчислення ЕА використовувався регістр – покажчик бази BP, то отримана ефективна адреса є зсувом у сегменті стека, положення якого в пам'яті визначається вмістом сегментного регістра SS.

Якщо для обчислення ЕА використовувалися регістри BX, SI, чи DI, то ефективна адреса є зсувом у сегменті даних, положення якого в пам'яті визначається вмістом сегментного регістра DS.

Перші два призначення (CS: IP) і (SS:SP) неможливо змінити. Можна перепризначити лише сегментні регістри, а також відповідно сегменти пам'яті при розрахунку фізичної адреси за виконавчою адресою ЕА. Це робиться за допомогою однобайтного префікса заміни сегмента (рисунок 7.15), що в машинному коді повинен передувати команді, що звертається до нестандартного сегмента пам'яті (див 5.3.8).

7.3 Способи адресації операндів мікроконтролера типу МК–51

У МК типу МК51 існують наступні способи адресації операндів–даних що беруть участь в операціях:

- неявна;
- регістрова (пряма регістрова);
- пряма (байтова і бітова);
- безпосередня;
- непряма;
- відносна;
- базово–індексна.

При **неявній** адресації інформацію про адресу операнда, що бере участь в операції, пристрій керування МК–ра одержує з коду операції команди. Таким чином часто адресуються акумулятор і прапорець перенесення, наприклад, INC A, CLR C.

При **регістровій** адресації команда містить трьохрозрядну пряму адресу одного з восьми робочих регістрів поточного банку регістрів загального призначення (РЗП) резидентної пам’яті даних (РПД). Для вибору робочого (поточного) банку використовуються розряди PSW3, PSW4 (RS0, RS1) регістра прапорців. Приклади команд з регістровою адресацією:

MOV A, R5; XCH A, R3.

Пряма байтова адресація застосовується для звернення до комірок РПД і до регістрів спеціальних функцій. В цьому випадку команда включає в себе пряму 8–розрядну адресу операнда, що при описанні команд мікроконтролера позначається ad, наприклад: ADD A, ad; DEC ad і т. ін.

Пряма бітова адресація використовується для звернення до окремо адресуємих 128 бітів РПД і до окремо адресуємих бітів регістрів спеціальних функцій (див. програмістську модель МК–ра). При цьому команда містить пряму 8–розрядну адресу біта, що бере участь в операції. Ця адреса при описі команд позначається bit, наприклад: SET bit;
MOV C, bit і т. ін.

При **безпосередній** адресації операнд входить до складу самої команди і витягується з пам'яті при виконанні команди одразу ж після її коду операції. Безпосередній операнд позначається #D8, #D16, наприклад: MOV A, #D8; MOV DPTR, #D16.

При **непрямій** адресації в команді міститься посилання на регістр, в якому міститься адреса операнда. Ця адресація може використовуватися для звернення до комірок РПД або зовнішньої пам’яті даних ЗПД. В якості регістрів–покажчиків РПД використовуються регістри R0, R1 робочого

(поточного) банку РЗП. Це, наприклад, команди MOV A, @Ri; MOV ad, @Ri і т.ін.

В якості регістрів–покажчиків ЗПД застосовуються ті ж R0 і R1, що дозволяє вибрати одну з 256 комірок зовнішньої пам'яті даних, або 16–розрядний регістр–покажчик даних (DPTR), який забезпечує адресацію однієї з 65536=64 К восьмирозрядних комірок ЗПД. Наприклад, команди MOVX A, @Ri; MOVX @DPTR, A і т.ін.

Відносна адресація застосовується в командах переходів для обчислення адреси команди, на яку передається керуванння. При цьому в команді задається 8–розрядний зсув відносно адреси самої команди, яке при описі команд позначається як rel. Зсув сприймається як число зі знаком, що представлено у додатковому коді. Це дозволяє здійснювати переходи на +127 байт вперед і на –128 байт назад відносно адреси наступної команди. Наприклад, команди JNZ rel; DJNZ ad, rel і т. ін., де rel – 8–бітний зсув. Зсув обчислюється компілятором шляхом віднімання з адреси мітки команди, на яку потрібно перейти, адреси наступної команди.

При **базово–індексній** адресації адреса операнда в пам’яті програм обчислюється як сума 16–розрядної базової адреси, що міститься в регістрах DPTR або PC, і 8–бітного індексу, що знаходиться в акумуляторі. Це дозволяє звертатися до елементів таблиць, масивів і т.ін. Наприклад, команди MOVC A, @A+DPTR; MOVC A,@A+PC.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Дайте визначення способу адресації.
2. Що називається виконавчою (ефективною) адресою?
3. Що являє собою ефективна адреса?
4. Для чого використовується постбайт?
5. Звідки МП–р одержує інформацію про потрібну адресу при неявній адресації?
6. Де міститься операнд при регістровій адресації?
7. Яку довжину можуть мати безпосередні операнди у МП–рі i8086?
8. Який тип адресації має команда MOV AL, [dataByte]?
9. Де міститься ефективна адреса при непрямій адресації?
10. Що виконує роль бази при базовій та індексній адресації?
11. Яке головне застосування базової адресації?
12. В якому випадку використовується індексна адресація?
13. Дайте визначення стека.
14. Для чого використовується стек?
15. В яких командах може використовуватись відносна адресація?
16. Які регістри використовуються при адресації рядків даних?
17. Яким співвідношенням визначається тривалість одного такту?
18. Які способи адресації застосовуються у МП–рі i8086?

ЛІТЕРАТУРА [1…5, 10, 11, 15; 16; 29; 39]

8 КОМАНДИ МІКРОПРОЦЕСОРІВ ТА МІКРОКОНТРОЛЕРІВ

8.1 Мікропроцесор i8080

8–розрядний МП–р i8080 має 78 базових команд, які в залежності від їх функціонального призначення умовно розбиті на наступні групи [2 … 4, 29, 31]:

- пересилання;
- обмін байтами;
- введення/виведення;
- арифметичні і логічні команди з одним операндом;
- арифметичні і логічні команди з двома операндами;
- команди порівняння;
- зсув вмісту акумулятора;
- команди передачі керування;
- команди виклику підпрограм;
- команди повернення з підпрограм;
- команди керування мікропроцесором (спеціальні команди).

8.2 Мікропроцесор i8086

8.2.1 Загальні відомості

Всі команди МП i8086, в залежності від їх функціонального призначення, умовно можуть бути розбиті на такі групи: пересилання; робота зі стеком; введення/виведення; обмін; трансляція; завантаження; пересилання прапорців; арифметичні; корекція; порівняння; логічні; зсув; обробка рядків; умовні і безумовні переходи; організація циклів; виклик і

повернення з підпрограм; програмні переривання; керування мікропроцесором.

Всі названі групи команд об'єднано в п’ять таблиць, до яких включено основні (базові) команди (мнемоніки). По кожній команді в таблицях міститься основна та довідкова інформація: номер команди в межах таблиці; мнемоніка; функція/алгоритм, який виконує команда (коментар); формат команди; число байт; час виконання в тактах.

У таблицях відсутня ще одна важлива характеристика команди вплив на окремі прапорці (ознаки), на якій зупинимося при більш докладному розгляді кожної групи команд.

8.2.2 Команди пересилання даних

Команди даної групи (таблиця 8.1) мають мнемоніку MOV і здійснюють пересилання байта або слова від джерела операнда до місця призначення (приймач). В якості джерела і приймача можуть служити один з з регістрів загального призначення (РЗП), сегментні регістри або комірки пам'яті.

Крім того, джерелом можуть бути безпосередні операнди.

У залежності від способу адресації операндів команди пересилання мають довжину від 2–х до 6–ти байтів і на прапорці не впливають.

8.2.3 Команди роботи зі стеком

Ці команди (таблиця 8.1) мають мнемоніки PUSH/POP і служать для занесення 16–ти розрядного слова, що міститься в регістрі або памяті, у стек або читання цього слова зі стека. В якості 16–ти розрядного операнда при роботі зі стеком можуть виступати вміст одного з РЗП, сегментних регістрів, регістра прапорців і двох сусідніх комірок пам'яті.

Таблиця 8.1 – Команди пересилань і введення/виведення

№	Мне–моніка	Функція (алгоритм)	Формат				Число байт	Число тактів
			1 байт	2 байт	3 байт	4 байт		
1	MOV	r/m ← reg	1000100w	mdreg r/m			2...4	9+Тва
2	MOV	reg ← r/m	1000101w	mdreg r/m			2...4	2 /8+Тва
3	MOV	r/m ← sr	10001100	md0sr r/m			2...4	2 / 9+Тва
4	MOV	sr ← r/m	10001110	md0sr r/m	В якості sr не можна використовувати регістр CS		2...4	2 / 8+Тва
5	MOV	A ← M(addr)	1010000w	addr L	addr H		3	10
6	MOV	M(addr) ← A	1010001w	addr L	addr H		3	10
7	MOV	r/m ← data	1100011w	md000 r/m	data L	data H, w=1	3...6	10+Тва
8	MOV	reg ← data	1011wreg	data L	data H, w=1		2,3	4
9	PUSH	стек ← r/m	11111111	md110 r/m			2...4	10/16+$T_{ва}$
10	PUSH	стек ← reg	01010reg				1	10
11	PUSH	стек ← sr	000sr110				1	10
12	PUSHF	стек ← F	10011100				1	10
13	POP	r/m ← стек	10001111	md000 r/m			2...4	8/17+$T_{ва}$
14	POP	reg ← стек	01011reg				1	8
15	POP	sr ← стек	000sr111				1	8
16	POPF	F ← стек	10011101				1	8
17	XCHG	r/m ↔ reg	1000011w	mdreg r/m			2...4	4/17+$T_{ва}$
18	XCHG	AX ↔ reg	10010reg				1	3
19	XLAT	AL←M(BX+AL)	11010111				1	11
20	LEA	reg ← EA	10001101	mdreg r/m			2...4	2+$T_{ва}$
21	LDS	DS, reg ←r/m	11000101	mdreg r/m			2...4	16+$T_{ва}$
22	LES	ES, reg ←r/m	11000100	mdreg r/m			2...4	16+$T_{ва}$
23	LAHF	AH ← F(7...0)	10011111				1	4
24	SAHF	F(7...0) ← AH	10011110				1	4
25	IN	A ←port(addr)	1110010w	addr			2	10
26	IN	A ←port(DX)	1110110w				1	8
27	OUT	port(addr) ←A	1110011w	addr			2	10
28	OUT	port(DX) ←A	1110111w				1	8

Примітка:

data - байт або слово даних;

addr - 16-розрядна адреса запам'ятовуючого пристрою (ЗП) або 8- розрядна адреса зовнішнього пристрою (ЗВПР);

addr L – молодший байт адреси;

addr H - старший байт адреси;

M(addr) - мітка ЗП з адресою addr;

port(addr) - порт ЗВПР з адресою addr.

SR	REG
00	ES
01	CS
10	SS
11	DS

reg	w = 0	w = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

w = 0 : data = 8 біт

w = 1 : data = 16 біт

Довжина команди від 1 до 4–х байт. На прапорці вони не впливають, крім команди POPF, яка завантажує із стека регістр прапорців.

8.2.4 Команди введення/виведення

Крім команд роботи з пам'яттю, процесор має окрему групу команд обміну з пристроями введення/виведення (портами) (таблиця 8.1).
Для введення байта/слова з порту служить команда IN, яка розташовує дані в акумуляторі (в AL або AX, в залежності від довжини операнда).
Номер порту може бути задано як безпосередньо у другому байті команди, так і непрямо в регістрі DX, причому тільки регістр DX з всіх РЗП може використовуватися для цієї мети.

Перший формат команд введення/виведення дозволяє адресувати 2^8=256 8–розрядних портів або 128 16–розрядних. Непряме задання порту, хоч і вимагає попереднього завантаження його номера в DX, дозволяє організувати програмні цикли, в яких використовується номер портів введення/виведення, що змінюється. Кількість портів, що адресуються в цьому випадку: 65536 8– розрядних або 32768 16–розрядних.

Команда виведення OUT служить для пересилання даних (байта або слова) з акумулятора (з AL або AX) в порт виведення. Ця команда, як і команда IN, має 2 формати, які визначають спосіб адресації порту виведення. Довжина команд IN, OUT один/два байти, на прапорці вони не впливають.

8.2.5 Команди обміну

Команда XCHG (таблиця 8.1) викликає обмін байтами/словами між двома джерелами, в якості яких можуть служити регістри і пам'ять. Сегментні регістри і регістри прапорців при обміні використовуватися не можуть. Довжина команд від 1 до 4 байт. На прапорці вони не впливають.

8.2.6 Команда трансляції (перетворення кодів)

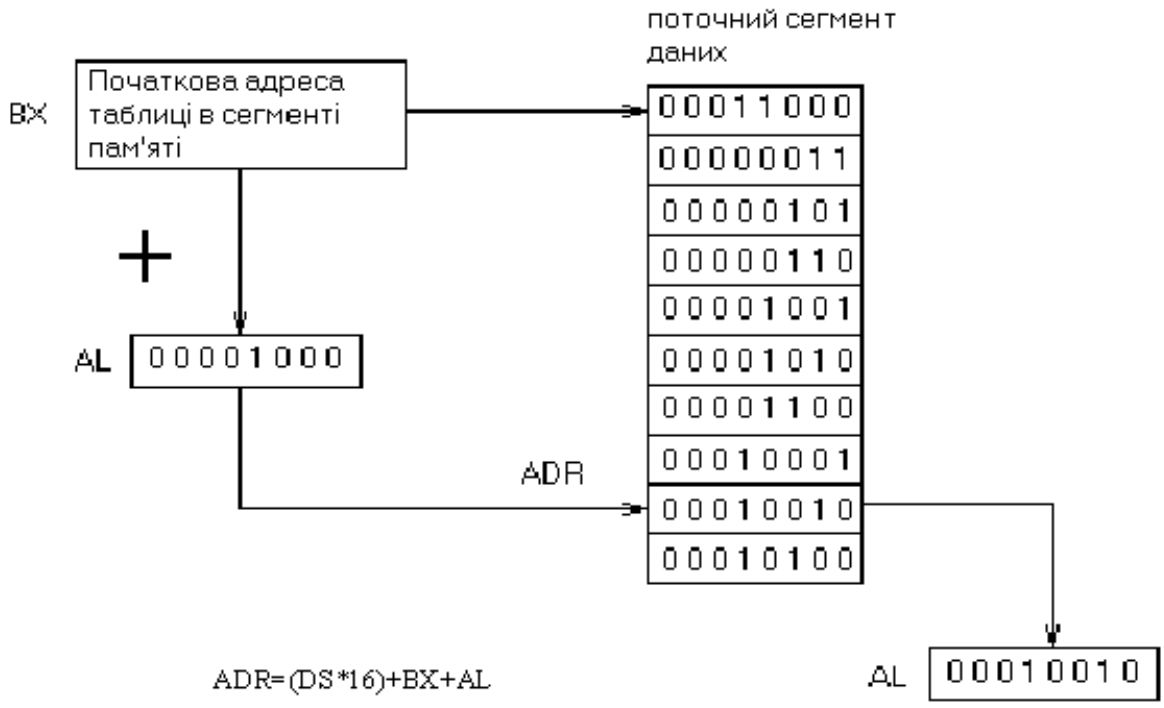
Однобайтна команда XLAT; AL←M(DS*16+BX+AL) (таблиця 8.1) орієнтована на табличні перетворення кодів. Таблиця розміром не більше за 256 байт розміщується в пам'яті (за умовчанням в сегменті DS). Початкова адреса таблиці завантажується в регістр BX. Перед виконанням команди в AL в двійковому коді міститься номер рядка таблиці, з якого байт пересилається в AL. Розглянемо, наприклад, перетворення десяткових цифр (номери рядка таблиці) в код "2 з 5" (таблиця 8.2).

Таблиця 8.2 – Перетворення кодів

Цифри	Код 2 з 5
0	11000
1	00011
2	00101
3	00110
4	01001
5	01010
6	01100
7	10001
8	10010
9	10100

Для перекодування двійкового представлення деякої цифри К в код "2 з 5" (К=0, 1, 2, …, 9) досить виконати наступні операції (рисунок 8.1):

- завантажити початкову адресу таблиці коду "2 з 5" до BX;
- завантажити двійковий код цифри К (наприклад К=8D=00001000B) до AL;
- виконати команду XLAT, яка перешле вміст К–го рядка таблиці до AL (для К=8 до AL пересилається код xxx10010B, де xxx– початкове значення 3–х старших розрядів елемента пам'яті, в якому зберігається код). У прикладі, що розглядається xxx=000.



–

Рисунок 8.1 – Приклад використання команди XLAT

8.2.7 Команди завантаження

Ці команди (таблиця 8.1) включають три мнемоніки:

- LEA (завантажити виконавчу (ефективну) адресу операнда EA);
- LDS (завантажити покажчик до DS та reg);
- LES (завантажити покажчик до ES та reg),

і виступають засобом керування механізмом адресації операндів.

За командою LEA в 16–розрядний РЗП, що адресується полем reg другого байта команди (постбайта), завантажується ефективна адреса операнда EA, обчислена у відповідності зі значеннями полів md і r/m пост байта. Використання команди LEA зручне при складанні підпрограм, працюючих з параметрами. У цьому випадку перед викликом підпрограми виділений регістр завантажується адресою змінної, яку заздалегідь записано в пам'ять як параметр.

Наприклад, підпрограма оперує з параметром, адреса якого міститься в РЗП – BX. Якщо перед викликом цієї підпрограми виконати команду LEA

BX, [ALPHA], де ALPHA ім'я елемента пам'яті, що містить змінну ALPHA, то підпрограма буде використовувати як параметр змінну з комірки пам’яті з ім'ям ALPHA. Якщо ж перед викликом підпрограми виконати команду LEA BX, [BETA], то як значення параметра підпрограма буде використовувати значення змінної з комірки пам'яті з ім'ям BETA.

Потрібно підкреслити, що дана команда дозволяє обчислювати EA за значенням двох РЗП та зміщенням.

Цю команду зручно використовувати перед виконанням команди XLAT (див. вище) для обчислення базової адреси таблиці в поточному сегменті даних і завантаження цієї адреси до регістра BX.

Команди LDS і LES використовуються в основному при зверненні до даних, що знаходяться поза поточними сегментами DS або ES. При цьому виникає необхідність змінити базову 16–розрядну адресу сегмента (базу) і 16–розрядне зміщення в сегменті, що звуться покажчиком. Для цього покажчик заздалегідь завантажується в чотири суміжні елементи пам'яті в такому порядку:

- молодший байт (МБ) зміщення;
- старший байт (СБ) зміщення;
- МБ бази (адреси сегмента);
- СБ бази бази (адреси сегмента).

При виконанні команди LDS/LES адреса (зміщення) першого байта покажчика в сегменті даних обчислюється за значеннями полів md і r/m постбайта. Після цього два перших байти покажчика (зміщення) завантажуються в регістр, що адресується полем reg постбайта, а два старших байти (база) в регістр DS/ES.

Команди завантаження прапорців не змінюють.

8.2.8 Команди пересилання прапорців

У МП–рі, що розглядається, є чотири однобайтних команди (таблиця 8.1), які пересилають вміст регістра прапорців RF:

– LAHF (завантажити AH прапорцями, №23);
– SAHF (переслати AH до регістра RF, №24);
– PUSHF (записати RF до стека, №12);
– POPF (прочитати RF зі стека, №16).

За командою LAHF здійснюється пересилання молодшого байта регістра прапорців RF в AH, а за командою SAHF– зворотне пересилання. Ці команди введено в систему команд i8086 для спрощення програмної сумісності з 8–розрядним процесором i8080. Зокрема, без їх використання для реалізації команд PUSH PSW і POP PSW МП–ра i8080 за допомогою команд МП–ра i8086 було би потрібно по 9 байт пам'яті, а використання, наприклад, команди LAHF дозволяє при реалізації команд PUSH PSW обійтися усього двома байтами: LAHF, PUSH AX.

Команда PUSHF вміщує вміст регістра RF до стека, причому спочатку записується старший байт RF, а потім – молодший.

Очевидно, що дві з цих чотирьох команд (SAHF і POPF) впливають на прапорці.

8.2.9 Арифметичні команди

8.2.9.1 Загальні відомості

Мікропроцесор i8086 виконує чотири основні арифметичні операції: додавання, віднімання, множення і ділення над 8– і 16– розрядними даними зі знаком і без знака (таблиця 8.3). Числа зі знаком представлено в МП в додатковому коді і їх діапазон при 16–ти розрядах, наприклад, становить від: – 32768 до + 32767. Для реалізації арифметичних дій над числами,

представленими в двійково–десятковому коді, введено спеціальні команди корекції результатів. Всі арифметичні команди впливають на прапорці.

8.2.9.2 Команди додавання

Операції додавання включають три мнемоніки: ADD (додати), ADC (додати з урахуванням перенесення) і INC (збільшити на одиницю) (таблиця 8.3).

Команда ADD (№ 1…4) здійснює додавання операнда джерела до операнда місця призначення. Джерелом і місцем призначення звичайно є регістр або пам'ять, причому джерелом можуть бути також дані (константи), безпосередньо представлені в команді. У першому форматі (№ 1, 2) поле d визначає місце призначення результатів додавання: при d = 1 це регістр, а при d = 0 пам'ять. У другому форматі (№ 3) поле s визначає число байтів даних, безпосередньо представлених в команді. При s = 0 команда складається з 3 (w=0) або 4 (w=1) байт, причому в двох останніх записано 16–розрядну константу.

При s = 1 (w=1) команда складається з 3 байт і останній, третій, містить 8–розрядну константу, яка заздалегідь (тобто до додавання) розширяється знаковим розрядом до 16 розрядів, якщо довжина чисел, що додаються дорівнює шістнадцяти бітам (w=1). Нагадаємо, що розширення знаком складається в розповсюдженні значення знакового розряду молодшого байта на весь старший байт, наприклад константу 5AH буде розширено до 005AH, а константу 8AH – до FF8AH.

Встановлення прапорців SF, ZF, PF, AF, і CF відповідно до результату операції додавання здійснюється так само, як і в МП–рі i8080, але додатково введено прапорець переповнення OF, що використовується при виконання дій над числами зі знаком.

Таблиця 8.3 – Арифметичні команди

№	Мне-моніка	Функція (алгоритм)	Формат				Число байт	Число тактів
			1 байт	2 байт	3 байт	4 байт		
1	ADD	r/m←r/m + reg	0000000w	mdreg r/m			2...4	16+$T_{ва}$
2	ADD	reg←reg + r/m	0000001w	mdreg r/m			2...4	3 /9+$T_{ва}$
3	ADD	r/m←r/m+data	100000sw	md000 r/m	data L	data H,sw=01	3...6	4/17+$T_{ва}$
4	ADD	A ←A + data	0000010w	data L	data H ,w=1		2...3	4
5	ADC	r/m← r/m+reg+CF	0001000w	mdreg r/m			2...4	16+$T_{ва}$
6	ADC	reg← reg+r/m +CF	0001001w	mdreg r/m			2...4	3/ 9+$T_{ва}$
7	ADC	r/m ←r/m+data+CF	100000sw	md010 r/m	data L	data H,sw=01	3...6	4/17+$T_{ва}$
8	ADC	A←A+data+CF	0001010w	data L	data H, w=1		2, 3	4
9	INC	r/m ← r/m + 1	1111111w	md000 r/m			2...4	3/15+$T_{ва}$
10	INC	reg ← reg + 1	01000reg	reg – тільки 16–ти розрядний			1	2
11	AAA	ASCII корекція для plus	00110111	Якщо ((AL)&0FH)>9 або (AF=1), тоді (AL)←(AL)+6; (AH)←(AH)+1; F=1; (AC_F)←1; (AL)←(AL)&0FH			1	4
12	DAA	2–10 корекція для додавання	00100111	Якщо ((AL)&0FH)>9 або (AF=1), тоді (AL)←(AL)+6, AF+1; якщо (AL)>9FH або (C_F)=1, то (AL)←(AL)+60H, F←1			1	4
13	SUB	r/m ← r/m – reg	0010100w	mdreg r/m			2...4	16+$T_{ва}$
14	SUB	reg ← reg – r/m	0010101w	mdreg r/m			2...4	3 /9+$T_{ва}$
15	SUB	r/m ← r/m – data	100000sw	md101 r/m	data L	data H,sw=01	3...6	4/17+$T_{ва}$
16	SUB	A ← A – data	0010110w	data L	data H, w=1		2, 3	4
17	SBB	r/m←r/m – reg – CF	0001100w	mdreg r/m			2...4	16+$T_{ва}$
18	SBB	reg ← reg– r/m– CF	0001101w	mdreg r/m			2...4	3/9+$T_{ва}$
19	SBB	r/m←r/m–data– CF	100000sw	md011 r/m	data L	data H,sw=01	3...6	4/17+$T_{ва}$
20	SBB	A ← A – data – CF	0001110w	data L	data H, w=1		2...3	4
21	DEC	r/m ← r/m – 1	1111111w	md001 r/m			2...4	3/15+$T_{ва}$
22	DEC	reg ← reg – 1	01001reg	reg – тільки 16–розрядний			1	2
23	NEG	r/m ← 0 – r/m	1111011w	md011 r/g	Змінює знак (формує доповнення до двох)		2...4	3/16+$T_{ва}$
24	CMP	r/m – reg	0011100w	mdreg r/m	Якщо r/m=reg, тоді Z_F=1 Якщо r/m<reg, тоді C_F=1 Якщо r/m>reg, тоді $Z_F = C_F = 0$		2...4	9+$T_{ва}$
25	CMP	reg – r/m	0011101w	mdreg r/m	Якщо reg=r/m, тоді Z_F=1 Якщо reg<r/m, тоді C_F=1 Якщо reg>r/m, тоді $Z_F = C_F = 0$		2...4	3/9+$T_{ва}$

Продовження таблиці 8.3

№	Мне-моніка	Функція (алгоритм)	Формат	Число байт	Число тактів	Число байт	Число тактів
26	CMP	r/m – data	100000sw	md111 r/m	data L data H, sw = 01 Якщо r/m = data, то Z_F=1 Якщо r/m <data, то C_F=1 Якщо r/m>data, то $Z_F = C_F = 0$	3...6	Регістри 4; Пам'ять w = 0 10+$T_{ва}$; w = 1 17+$T_{ва}$;
27	CMP	A – data	0011110w	data L	data H, w = 1 Якщо A=data, тоді Z_F = 1 Якщо A<data, тоді C_F = 1 Якщо A>data, тоді $Z_F = C_F = 0$	2, 3	4
28	AAS	ASCII корекція для віднімання	00111111	Якщо ((AL)&0FH)>9 або (AF=1), тоді (AL)←(AL)–6, (AH)←(AH)–1, C_F←1; AF ←1, (AL) ← (AL)&0FH		1	4
29	AAM	ASCII корекція для множення	11010100	00001010	(AL)←залишок від (AL):10 (AH)←частка від (AL):10	2	83
30	DAS	2–10 корекція для вирахування	00101111	Якщо ((AL)&0FH)>9 або (AF=1), тоді (AL)←(AL)–6, AF ←1; Якщо (AL)>9FH або (C_F) = 1, тоді (AL)←(AL)–60H, C_F ←1		1	4
31	MUL (множення цілих чисел без знака)	AX←AL*r/m, w=0 DX,AX←AX*r/m, W = 1	1111011w	md100 r/m	Якщо AH<>0, тоді $C_F = O_F = 1$, Якщо AH = 0, тоді $C_F = O_F = 0$, Якщо DX<>0, тоді $C_F = O_F = 1$, Якщо DX = 0, тоді $C_F = O_F = 0$.	від 2 до 4	reg8·reg8 70–77; reg16·reg16 118–133; Mem8· reg8 76..83+$T_{ва}$; Mem16·reg16 124..139+$T_{ва}$;
32	IMUL (множення чисел зі знаком)	AX←AL*r/m,w=0 DX,AX←AX*r/m, W = 1	1111011w	md101 r/m	Якщо AH<>0, тоді $C_F = O_F = 1$, Якщо AH = 0, тоді $C_F = O_F = 0$, Якщо DX<>0, тоді $C_F = O_F = 1$, Якщо DX = 0, тоді $C_F = O_F = 0$.	від 2 до 4	reg8·reg8 80–98; reg16·reg16 128–154; Mem8· reg8 86..104+$T_{ва}$; Mem16·reg16 138..164+$T_{ва}$;
33	DIV (ділення цілих чисел без знака)	AX: r/m , w = 0 AH – залишок AL – частка DX,AX : r/m, w =1 DX – залишок AX – частка	1111011w	md110 r/m	Якщо частка перевищує розміри AL або AX, чи якщо дільник = 0, то генерується переривання типу 0.	від 2 до 4	reg16:reg8 80..90; reg32:reg16 144..162; reg16:mem8 86..96+$T_{ва}$; reg32:mem16 150..168+$T_{ва}$

Продовження таблиці 8. 3

№	Мне-моніка	Функція (алгоритм)	Формат	Число байт	Число тактів	Число байт	Число тактів
34	IDIV (ділення цілих чисел зі знаком)	AX: r/m , w = 0 AH – залишок AL – частка DX,AX : r/m, w =1 DX – залишок AX – частка	1111011w	md111 r/m	Якщо частка перевищує розміри AL або AX, чи якщо дільник = 0, то генерується переривання типу 0.	від 2 до 4	reg16:reg8 101..112; reg32:reg16 165..184 reg16:mem8 107..118+$T_{ва}$; reg32:mem16 175..194+$T_{ва}$;
35	AAD	ASCII корекція для ділення Перетворює два неупакованих BCD–числа, що містяться в AX, в двійкове значення	11010101	00001010	(AL) ← (AH)x10+(AL) (AH) ← 0	2	60
36	CBW	AH ← sign AL	10011000	Знак AL переходить в усі біти AH		1	2
37	CWD	DX ← sign AX	10011001	Знак AX переходить в усі біти DX		1	5

Примітка: w = 0 : data = 8 біт ; w = 1 : data = 16 біт;

s = 0 : data = 16 біт; s = 1 : sign data = 8 біт.

В таблиці 8.4 наведено приклад додавання 8–розрядних даних, що показує відмінність в модифікації прапорців CF і OF в залежності від результату.

Команда ADC (№ 5…8) виконує операцію додавання двох операндів з додаванням значення прапорця CF. Вона використовується для організації додавання багаторозрядних чисел, двійкове представлення яких перевищує 16 розрядів. Спочатку за командою ADD підсумовуються молодші розряди, а потім при додаванні більш старших розрядів чисел використовується команда ADC, яка дозволяє враховувати значення перенесення. В якості джерела та місця призначення операндів може служити регістр або пам'ять. Команда ADC має формати, аналогічні форматам команди ADD. За

допомогою команди ADC здійснюється також побайтове додавання при w = 0 багатобайтових чисел, як це робилося у i8080.

Команда INC (№ 9, 10) спричиняє збільшення на одиницю (інкремент) вмісту регістра або пам'яті. Необхідно зазначити, що ця команда, так само як і команда DEC, впливає на всі прапорці, крім CF.

Таблиця 8.4 – Приклади встановлення прапорців CF і OF

Доданки і результат в двійковій системі	Числа без знака у десятковій системі	Числа зі знаком у десятковій системі	Доданки і результат в двійковій системі	Числа без знака у десятковій системі	Числа зі знаком у десятковій системі
00000101 + 00101010 00101111	5 + 42 47 CF = 0	+5 + +42 +47 OF = 0	00100101 + 01101001 10001110	37 + 105 142 CF = 0	+37 + +105 −114 OF = 1
00000011 + 11111111 00000010	3 + 255 2 CF = 1	+3 + −1 +2 OF = 0	10000101 + 11110111 01111100	133 + 247 124 CF = 1	−123 + −9 +124 OF = 1

8.2.9.3 Команди віднімання

Операції віднімання включають чотири мнемоніки: SUB (відняти), SBB (відняти з урахуванням позики), DEC (зменшити на одиницю) і NEG (змінити знак) (таблиця 8.3).

За командіою SUB (№ 13…16) відбувається віднімання операнда джерела з операнда місця призначення. Як і в команді додавання, операнди можуть знаходитися в регістрах або пам'яті. В якості числа, що віднімається, може також служити операнд (константа), заданий безпосередньо в команді.

Команда SBB (№ 17…20) необхідна для віднімання операндів з урахуванням позики, тобто нарівні з операндами у відніманні бере участь значення прапорця CF.

Команда DEC (№ 21,22) виконує зменшення на одиницю (декремент) вмісту регістра або пам'яті і має два формати, як і команда INC.

Команда NEG (№ 23) змінює знак операнда, причому використовується представлення операнда в додатковому коді. Наприклад, якщо операнд є: – 1 (11111111), то команда NEG змінить його на +1 (00000001).

8.2.9.4 Команди множення

Операції множення містять два мнемоніки: MUL (помножити) і IMUL (помножити числа зі знаком) (таблиця 8.3).

За командою MUL (№ 31) відбувається множення без знака вмісту акумулятора (AL або AX) на операнд джерела, а результат подвійної довжини повертається до акумулятора і регістра, що використовується для його розширення (до AL і AH у разі 8–розрядних операндів або до AX і DX у разі 16–розрядних операндів). Команда впливає тільки на два прапорці CF і OF, які встановлюються в «1», якщо старша половина результату відмінна від нуля.

На відміну від операцій додавання і віднімання звичайне множення чисел, представлених в двійковій системі числення, дає правильні результати тільки для чисел без знака. Наприклад, якщо розглядати множення 8–розрядних чисел

11111111B x 11111111B = 1111111000000001B, як 255 · 255 = 65 025, то результат буде правильним. Якщо ж ці числа, що перемножуються розглядати як числа зі знаком (– 1)х(– 1), то результат –511 буде невірним. Коли операнди, що перемножуються, і результат розглядаються як числа зі знаком, використовується команда IMUL (№ 32).

8.2.9.5 Команди ділення

Операції ділення містять два мнемоніки: DIV (розділити) і IDIV (розділити числа зі знаком) (таблиця 8.3).

За командою DIV (№ 33) відбувається ділення без знака операнда подвійної довжини, що знаходиться в акумуляторі і регістрі, що використовується для розширення акумулятора (в AL і AH у разі 8–розрядних операндів або в AX і DX у разі 16–розрядних операндів) на операнд із заданого джерела. Результат – частка заноситься до акумулятора (AL або AX), а залишок – до регістра розширення акумулятора (AH або DX). При виконанні операції ділення прапорці приймають довільні значення. При додатному результаті, що перевищує максимально допустиме значення, або при від'ємному результаті, яке менше мінімального значення, результат–частка і залишок будуть мати невизначені значення і відбудеться переривання типу 0.

Особливість команди ділення IDIV (№ 34) полягає в тому, що результат – частка і залишок завжди мають однакові знаки. Наприклад, при діленні числа 47 на +3 з двох можливих результатів: 15 із залишком 2 і 16 із залишком – 1, буде отримано перший результат. Дробові значення результату округлюються до найближчого цілого. Значення прапорців при виконанні команди IDIV також не визначені.

8.2.9.6 Команди розширення операндів

До групи арифметичних операцій відносяться також дві команди, які здійснюють розширення знаком 8– і 16– розрядних операндів (таблиця 8.3). Ці команди грають допоміжну роль при підготовці операнда, що використовується як ділене. Оскільки ділене при діленні 8–розрядних операндів розміщується в 16–розрядному акумуляторі AX (а при діленні 16–

розрядних операндів в 32–розрядному складеному регістрі DX, AX), необхідно при підготовці діленого заповнити AH (або DX). Для чисел без знака вказані регістри заповнюються нулями. При діленні чисел зі знаком перед заповненням вказаних регістрів потрібно аналізувати знак діленого і заповнити регістр або нулями, якщо ділене є додатним, або одиницями, якщо ділене від'ємне. Для цього використовуються команди CBW (розширення знаком байта до слова) (№36) і CWD (розширення знаком слова до подвійного слова) (№37). За командою CBW (CWD) знак AL (або AX), тобто самий старший розряд регістра, записується у всі розряди регістра AH (або DX).

8.2.9.7 Команди корекції

8.2.9.7.1 Загальні відомості

У основі двійково–десяткового представлення чисел лежить принцип кодування кожної десяткової цифри групою з чотирьох бітів (тетрадою). Оскільки чотирма бітами можна закодувати шістнадцять різних комбінацій, а десяткових цифр тільки десять, останні шість кодів в двійково–десятковому представленні не використовуються (таблиця 8.5).

Ця обставина може бути причиною отримання неправильних результатів при виконанні звичайних арифметичних операцій, наприклад таких, як додавання і віднімання. Для подібних ситуацій МП–р, що розглядається, має спеціальні команди десяткової корекції результатів. Такими командами є:

DAA, DAS, AAA, AAS, AAM, AAD (таблиця 8.3).

Застосування цих команд залежить від виду арифметичної операції, що виконується над двійково–десятковими числами (додавання, віднімання, множення або ділення) і формату представлення двійково–десяткових

чисел. Існує два формати представлення двійково–десяткових чисел [10, 28, 39]:

- упакований;
- не упакований.

При використанні упакованого формату в одному байті можна записати (упакувати) дві двійково–десяткові цифри.

При застосуванні не упакованого формату в одному байті записується одна двійково–десяткова цифра (молодша тетрада байта), а чотири старших біти дорівнюють нулю.

Таблиця 8. 5 – Двійково–десяткове представлення чисел

№	Двійково–десятковий код	Десяткова цифра
1	0 0 0 0	0
2	0 0 0 1	1
3	0 0 1 0	2
4	0 0 1 1	3
5	0 1 0 0	4
6	0 1 0 1	5
7	0 1 1 0	6
8	0 1 1 1	7
9	1 0 0 0	8
10	1 0 0 1	9
11	1 0 1 0	не використовується
12	1 0 1 1	не використовується
13	1 1 0 0	не використовується
14	1 1 0 1	не використовується
15	1 1 1 0	не використовується
16	1 1 1 1	не використовується

Крім названих двох форматів, що застосовуються в двійково–десятковій системі числення, існує код ASCII (американський стандартний код для обміну інформацією), який застосовується при обміні інформацією

в обчислювальних системах [10, 28, 38]. Представлення чисел в коді ASCII відображає таблиця 8.6.

Таблиця 8.6– Представлення чисел за допомогою коду ASCII

Цифра	Код	Шістнадця–ткове представлення	Цифра	Код	Шістнадцяткове представлення
0	00110000	30	5	00110101	35
1	00110001	31	6	00110110	36
2	00110010	32	7	00110111	37
3	00110011	33	8	00111000	38
4	00110100	34	9	00111001	39

Відмінність коду ASCII від не упакованого двійково–десяткового формату при представленні чисел полягає в наявності коду 3D (0011) в старшій тетраді.

8.2.9.7.2 Команди корекції після додавання і віднімання двійково–десяткових чисел в упакованому форматі

Команда DAA (№ 12) служить для корекції результату після додавання двійково–десяткових чисел в упакованому форматі. У процесі виконання цієї команди виконуються наступні дії:

якщо (AL) $\wedge$ (0FH) > 9 або (AF)=1, то

(AL) $\leftarrow$ (AL) + 6,

(AF) $\leftarrow$ 1.

Якщо (AL) > 9FH або (CF)=1, то

(AL) $\leftarrow$ (AL) + 60H,

(CF) $\leftarrow$ 1.

Вона впливає на прапорці AF, PF, ZF, SF, і CF. Стан прапорця OF не визначено.

Приклад

Нехай регістри AL і BL містить 29H і 13H – упаковані двійково–десяткові записи десяткових чисел 29 і 13 відповідно і виконується наступна послідовність команд:

ADD AL, BL; AL ← AL+BL,
DAA.

Після виконання першої команди в регістрі AL буде записано 3CH, яке не є правильним двійково–десятковим числом, що відображає суму операндів. Команда DAA записує в регістр AL скориговане число 42H, яке є правильним двійково–десятковим записом десяткового числа 42.

Команда DAS (№ 30) виконує корекцію результату після віднімання двійково–десяткових чисел в упакованому форматі. У процесі виконання цієї команди виконуються наступні дії:

якщо (AL) ∧ (0FH) > 9 або (AF)=1, то
(AL) ← (AL) – 6,
(AF) ← 1.
Якщо (AL) > 9FH або (CF)=1, то
(AL) ← (AL) – 60H,
(CF) ← 1.

За командою DAS вміст регістра AL замінюється правильним двійково–десятковим числом в упакованому форматі. Команда розглядає вміст регістра AL як результат віднімання двох упакованих двійково–десяткових чисел і впливає на прапорці AF, PF, ZF, SF і CF. Стан прапорця переповнення OF не визначено.

Приклад

Нехай регістри AL і BL містять упаковані двійково–десяткові коди 44H і 27H десяткових чисел 44 і 27 відповідно і виконується наступна послідовність команд:

SUB AL, BL; AL ← AL – BL,
DAS.

Після виконання першої команди вміст регістра AL (1DH) не є правильним двійково–десятковим записом результату. Команда DAS перетворює це число в 17H – правильний двійково–десятковий запис десяткового числа 17.

Корекція після множення і ділення двійково–десяткових чисел в упакованому форматі неможлива.

8.2.9.7.3 Команди корекції після додавання, віднімання, множення і ділення двійково–десяткових чисел в не упакованому форматі

Для корекції результатів арифметичних дій над BCD–числами в не упакованому форматі використовуються чотири команди: AAA (корекція для додавання), AAS (корекція для віднімання), AAM (корекція для множення), AAD (корекція для ділення).

Команда AAA (№11) виконує корекцію результату після додавання BCD–чисел в не упакованому форматі. У процесі виконання команди виконуються наступні дії:

(AL)∧ 0FH) > 9 або (AF)=1, то
(AL) ← (AL) + 6,
(AH) ← (AH) + 1,
(CF) ← 1,

(AF) ← 1,

(AL) ← (AL) ∧ 0FH.

За цією командою вміст регістра AL перетворюється в число в двійково–десятковому не упакованому форматі з нульовими старшими чотирма бітами. Вміст регістра AL являє собою результат додавання двох чисел в двійково–десятковому не упакованому виді. Команда впливає на прапорці допоміжного перенесення AF і перенесення CF. Стан прапорців переповнення OF, знака SF, нуля ZF і паритету PF не визначено.

Приклад

Нехай регістри AX і CL містять коди 0008H і 04H відповідно і виконується наступна послідовність команд:

ADD AL, CL; AL ← AL + CL,

AAA.

Після виконання команди ADD AL, CL регістр AX містить число 000CH (десяткове число 12), яке не є правильним записом в двійково–десятковому вигляді. Після виконання команди AAA в регістрі AX міститься двійково–десяткове число в не упакованому форматі (0102H, яке є правильним записом в цьому форматі десяткового числа 12).

У спеціальній літературі [5, 11, 38] при описі команди AAA зазначається, що вона підганяє чисельну суму двох розпакованих BCD–цифр до розпакованого BCD–формату, який легко перетворюється в код ASCII. Якщо скористатися останнім прикладом, то це перетворення можна виконати за допомогою команди:

OR AX, 3030H; AX ← AX + 3030H.

Команда AAS (№ 28) виконує корекцію результату після віднімання BCD–чисел в не упакованому форматі. У процесі виконання команди виконуються наступні дії:

якщо ((AL)∧ 0FH) > 9 або (AF)=1, то

(AL) ← (AL) – 6,

(AH) ← (AH) –1,

(CF) ← 1,

(AF) ← 1,

(AL) ← (AL) ∧ 0FH).

Операндами команди є не упаковані двійково–десяткові числа. За командою вміст регістра AL перетворюється в не упаковане двійково–десяткове число з нульовими старшими чотирма бітами. Вміст регістра AL розглядається як результат віднімання двох не упакованих двійково–десяткових чисел. Команда впливає на прапорці AF, CF. Стан прапорців OF, SF, ZF, і PF не визначені.

Приклад

Нехай регістри AX і CL містять числа 0105H і 08H, тобто двійково–десяткові представлення десяткових чисел 15 і 8 відповідно, і виконується наступна послідовність команд:

SUB AL, CL; AL ← AL – CL,

AAS.

Після виконання команди SUB AL, CL в регістрі AX знаходиться число 01FDH, яке не є правильним двійково–десятковим представленням результату. Команда AAS перетворює вміст регістра AX в не упаковане двійково–десяткове число 0007H.

Цей результат командою OR AX, 3030H; AX ← AX + 3030H перетворюється в формат ASCII коду.

Команда AAM (№ 29) виконує корекцію результату після множення BCD–чисел в не упакованому форматі.

В процесі виконання команди виконуються наступні дії:

– (AL) ← залишок від ділення (AL): 10;

– (AH) ← частка від ділення (AL): 10.

Команда коригує результат попереднього множення двох не упакованих двійково–десяткових чисел. Вміст регістра AX перетворюється в правильне двійково–десяткове число. Вміст регістра AX розглядається як результат множення двох не упакованих двійково–десяткових чисел. Для того, щоб корекція результату проводилася правильно, старші напівбайти операндів, що перемножуються, повинні дорівнювати 0. Команда впливає на прапорці SF, ZF і PF. Стан прапорців OF, AF і CF не визначено.

Приклад

Нехай регістри AL і CL містять числа 08H і 09H відповідно і виконується наступна послідовність команд:

MUL CL; AX ← ALxCL,

AAM.

Після виконання команди MUL CL регістр AL містить число 48H (десяткове 72), яке не є не упакованим двійково–десятковим числом. Виконання команди AAM призводить до отримання в регістрі AX числа 0702H, яке є правильним двійково–десятковим числом в не упакованому форматі (шістнадцяткове число 0702H є не упакованим двійково–десятковим представленням десяткового числа 72).

Останній результат командою OR AX, 3030H; AX ← AX + 3030H легко перетворюється в формат ASCII коду.

Команда AAD (№ 35) виконує перетворення двох не упакованих BCD–чисел, розташованих в AX, в двійкове 8–розрядне число.

У процесі виконання команди виконуються наступні дії:

– (AL) ← (AH) x 10 + (AL),

– (AH) ← 0.

Вміст регістра AX розглядається як не упаковане двійково–десяткове число. За командою AAD це число перед діленням на не упакований дільник перетворюється в нормальне беззнакове ціле двійкове число для отримання результату в правильному не упакованому двійково–десятковому форматі. Для отримання правильного результату в подальшій команді ділення DIV регістр AH повинен завжди містити нуль. Команда впливає на прапорці паритету PF, знака SF, нуля ZF. Стан прапорців допоміжного перенесення AF, перенесення CF, переповнення OF не визначено.

Приклад

Нехай регістри AX і DL містять коди 0205H і 06H відповідно і виконується наступна послідовність команд:

```
AAD,
DIV DL.
```

Оскільки регістр AX містить правильне не упаковане двійково–десяткове число, команда AAD перетворює вміст регістра AX в число 0019H (десяткове 25), яке є шістнадцятковим представленням беззнакового цілого не упакованого двійково–десяткового числа 0205H. Тому після виконання команди DIV DL в регістрі AL буде знаходитися результат від ділення 04H, а в регістрі AH – залишок 01H, які є правильними двійково–десятковими числами.

Особливості використання команд корекції розглянемо на прикладі множення трьохрозрядного числа 638 на 4. Нехай число 638 дано в не упакованому коді в комірках a3, a2 і a1 відповідно. Множник 4 зберігається в комірці b, а результат множення потрібно розмістити в комірках c4, c3, c2 і c1:

$$\begin{array}{r} a3 \quad a2 \quad a1, \\ \times \\ \hline b, \\ \hline c4 \quad c3 \quad c2 \quad c1. \end{array}$$

Програма множення буде нагадувати дії, які проводяться при звичайному “ручному” множенні:

1. a1 x b → AX
2. AAM
3. AL → c1
4. AH → c2
5. a2 x b → AX
6. AAM
7. AL + c2 → AL
8. AAA
9. AL → c2
10. AH → c3
11. a3 x b → AX
12. AAM
13. AL + c3 →AL
14. AAA
15. AL → c3

16. AH → c4.

На кроках 2, 6 і 12 команда AAM здійснює корекцію результату множення, розташовуючи скориговану молодшу цифру результату множення в AL, а старшу в AH. На кроках 8 і 14 за допомогою команди AAA проводиться корекція результату додавання молодшої цифри поточного результату множення зі старшою цифрою результату попереднього множення.

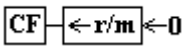
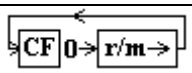
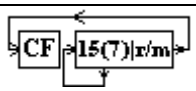
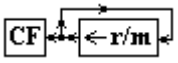
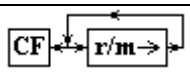
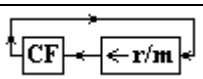
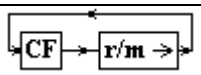
8.2.9.8 Команди порівняння

Команди порівняння CMP необхідні для порівняння двох операндів шляхом віднімання значення другого операнда від першого операнда. На відміну від звичайного віднімання отримана різниця нікуди не заноситься, а результатом операції порівняння є значення прапорців CF та ZF, які встановлюються в залежності від співвідношення операндів, що порівнюються (таблиця 8.3). Команда CMP має формати, аналогічні команді віднімання: № 24 … 27.

8.2.10 Логічні команди

Логічні команди (таблиця 8.7) використовуються для реалізації чотирьох основних булевих функцій: AND (порозрядне логічне І), OR (порозрядне логічне АБО), XOR (виключне АБО) (порозрядна логічна сума за модулем 2), і NOT (порозрядне логічне НЕ). Сюди також відноситься команда TEST (перевірка), яка полягає в порозрядному логічному множенні (І) операндів без занесення результату множення в місце призначення і необхідна для аналізу вмісту джерела за значеннями прапорців.

Таблиця 8.7 – Команди логічні, зсуву та обробки рядків

№	Мне-моніка	Функція (алгоритм)	Формат				Число байт	Число Тактів
			1 байт	2 байт	3 байт	4 байт		
1	NOT	r/m ← $\overline{\text{r/m}}$	1111011w	md010 r/m	(не змінює прапорці)		2...4	3/16+$T_{ва}$
2	AND	r/m ← r/m&reg	0010000w	mdreg r/m			2...4	16+$T_{ва}$
3	AND	reg ← reg&r/m	0010001w	mdreg r/m			2...4	3/9+$T_{ва}$
4	AND	r/m ← r/m&data	1000000w	md100 r/m	data L	data H, w=1	3...6	4/17+$T_{ва}$
5	AND	A ← A&data	0010010w	data L	data H, w=1		2, 3	4
6	OR	r/m ← r/m ∨ reg	0000100w	mdreg r/m			2...4	16+$T_{ва}$
7	OR	reg ← reg ∨ r/m	0000101w	mdreg r/m			2...4	3/9+$T_{ва}$
8	OR	r/m ← r/m∨data	1000000w	md001 r/m	data L	data H, w=1	3...6	4/17+$T_{ва}$
9	OR	A ← A ∨ data	0000110w	data L	data H, w=1		2, 3	4
10	XOR	r/m ← r/m⊕reg	0011000w	mdreg r/m			2...4	16+$T_{ва}$
11	XOR	reg ← reg⊕r/m	0011001w	mdreg r/m			2...4	3/9+$T_{ва}$
12	XOR	r/m ← r/m⊕data	1000000w	md110 r/m	data L	data H, w=1	3...6	4/17+$T_{ва}$
13	XOR	A ← A⊕data	0011010w	data L	data H, w=1		2, 3	4
14	TEST	r/m & reg	1000010w	mdreg r/m	(результат кон´юнкції відкидається, впливає на прапорці)		2...4	3/9+$T_{ва}$
15	TEST	r/m & data	1111011w	md000 r/m	data L	data H, w=1	3...6	4/10+$T_{ва}$
16	TEST	A & data	1010100w	data L	data H, w=1		2, 3	4
17	SHL / SAL		110100vw	md100 r/m	Логічний/арифметичний зсув вліво (множення на 2 чисел без знака/зі знаком)		2...4	
18	SHR		110100vw	md101 r/m	Логічний зсув вправо (ділення на 2 чисел без знака)		2...4	
19	SAR		110100vw	md111 r/m	Арифметичний зсув вправо (ділення на 2 чисел зі знаком)		2...4	2 (reg-один раз) 15+$T_{ва}$ (Mem-один раз) 4·CL+8 (reg-CL раз) 4·CL+ +20+$T_{ва}$ (Mem-CL раз)
20	ROL		110100vw	md000 r/m	Циклічний зсув вліво з копіюванням в C_F значення старшого біта		2...4	
21	ROR		110100vw	md001 r/m	Циклічний зсув вправо з копіюванням в C_F значення молодшого біта		2...4	
22	RCL		110100vw	md010 r/m	Циклічний зсув вліво з включенням C_F в ланцюг зсуву		2...4	
23	RCR		110100vw	md011 r/m	Циклічний зсув вправо з включенням C_F в ланцюг зсуву		2...4	

Продовження таблиці 8.7

24	MOVS	M(DI)←M(SI)	1010010w	(MOVSB → w=0; MOVSW→w=1)	1	18/ (9+17)*N
25	CMPS	M(DI) — M(SI)	1010011w	(CMPSB→ w = 0; CMPSW→w = 1)	1	
26	SCAS	A — M(DI)	1010111w	(A–M(ESx16+DI))	1	15/ (9+15)*N
24	MOVS	M(DI) ←M(SI)	1010010w	(MOVSB → w=0; MOVSW→w=1)	1	18/ (9+17)*N
25	CMPS	M(DI) — M(SI)	1010011w	(CMPSB→ w = 0; CMPSW→w = 1)	1	
26	SCAS	A — M(DI)	1010111w	(A–M(ESx16+DI))	1	15/ (9+15)*N
27	LODS	A ← M(SI)	1010110w	LODSB→w = 0 A←M(DSx16+SI) LODSW→w = 1	1	12/(9+ +13) *N
28	STOS	M(DI) ← A	1010101w	STOSB→w = 0 M(ESx16+DI)←A STOSW→w = 1	1	11/ (9+10)*N
29	REP	Повторювати команду, доки CX<>0	1111001x	Префікс команд MOVS or STOS	1	Додатково 6 тактів на цикл
30	REPE REPZ	Повторювати команду, доки CX<>0; ZF=1	1111001z z = 1	Префікс команд CMPS or SCAS	1	
31	REPNE REPNZ	Повторювати команду, доки CX<>0; ZF=0	1111001z z = 0	Префікс команд CMPS or SCAS	1	

Примітка:

v = 0 : число зсувів = 1;

v = 1 : число зсувів = (CL);

z = 1: REPE/REPZ –повторювати команду, доки CX<>0; ZF = 1 (доки однакові);

z = 0: REPNE/REPNZ – повторювати команду, доки CX<>0; ZF = 0 (доки не однакові);

x – може приймати довільне значення;

N – число елементів рядка, які обробляються;

& – логічне ’’ І ’’;

$\vee$ – логічне ’’ АБО’’;

$\oplus$– додавання за модулем два.

Всі двоопернадні команди AND (№ 2…4), OR (№ 6…8), XOR (№ 10…12) і TEST (№ 14…16) мають по три однакових формати і співпадають за часом виконання. Крім цього, команди AND (№ 5), OR (№ 9) та XOR (№ 13) мають четвертий формат, в якому перший операнд (акумулятор: AL або AX) адресується неявно, а другий (безпосередній операнд) входить до

складу самої команди. Однооперандна команда NOT (№ 1) здійснює інвертування операнда, має один формат і прапорці не змінює.

Команди AND, OR, XOR і TEST впливають на прапорці PF, SF і ZF. Прапорці CF і OF завжди скидаються в нуль, оскільки відсутні перенесення і переповнення відповідно. Прапорець AF не визначено. Нижче наведено приклади використання: ANL (кон'юнкція, І).

Приклад: ANL AL, #D8; AL←AL $\wedge$ #D8. Скинути в 0 парні біти акумулятора AL (інші біти залишити без зміни).

```
          7p 6  5  4   3  2  1 0p
    AL→ X X X X  X X X X      ; x=0/1; x*0=0; x*1=x,
∧ #D8 → 1  0  1  0   1  0  1 0    ; #D8 = AAh,
-------------------------------
    AL← X 0  X 0  X 0  X  0.
```

ORL (диз'юнкція, АБО).

Приклад: ORL AL, #D8; AL←AL $\vee$ #D8. Встановити у 1 непарні біти акумулятора А (інші біти залишити без зміни).

```
          7p 6  5  4   3  2  1 0p
    AL→ X X X X  X X X X      ; x=0/1; x+1=1; x+0=x,
∨ #D8→ 1  0  1  0   1  0  1  0    ; #D8 = AAh,
-------------------------------
    AL← 1  X 1  X  1  X 1 X.
```

XRL (виключаюче АБО, суматор за модулем 2, $\oplus$ – псевдоплюс).

Приклад: XRL AL, #D8; AL←AL $\oplus$ #D8. Проінвертувати старшу тетраду акумулятора AL (інші біти залишити без зміни).

```
          7p 6  5  4   3  2  1 0p
    AL→ X X X X  X X X X      ; x=0/1; x⊕0=x; x⊕1= x̄,
⊕ #D8→ 1 1  1  1   0  0  0  0    ; #D8 = F0h,
-------------------------------
    AL← X̄ X̄ X̄ X̄  X X  X X
```

CPL AL (побітова інверсія акумулятора AL).

Приклад: CPL AL, AL← $\overline{A}$.

AL → 0110 1101B – до виконання CPL,

AL ← 1001 0010B – після виконання CPL.

8.2.11 Команди зсуву

Процесор, що розглядається, може виконувати зсув трьох типів (таблиця 8.7):

- логічний (вліво/вправо), команди SHL (№ 17) і SHR (№ 18) відповідно;
- арифметичний (вліво/вправо), команди SAL (№ 17) і SAR (№ 19) відповідно;
- циклічний (вліво/вправо), команди ROL (№ 20), RCL (№ 22), ROR (№ 21), RCR (№23) відповідно.

Особливості виконання різних типів зсувів пояснюються в таблиці 8.7 під час опису функції, що виконується названими командами. Довжина операнда, що зсувається визначається бітом W першого байта (W=0 – 8–ми розрядний операнд, W=1 – 16–розрядний операнд). Значення біта V визначає кількість зсувів, що здійснюються однією командою. Якщо V =0, то здійснюється зсув на один розряд, а при V=1 відповідна команда зсуву буде виконана n раз (n>=1), де число зсувів n вказується в регістрі–лічильнику CL. Приклад використання команди SAR при n=3 приведено на рисунку 8.1.

Рисунок 8.1 – Приклад використання команди SAR при n=3

Команди логічного і арифметичного зсувів можуть застосовуватися

передусім для подвоєння чисел і ділення на 2. Для подвоєння числа без знака досить зсунути всі його розряди на один розряд вліво, заповнивши молодший біт нулем (команда SHL (№17)). Оскільки старший біт при зсуві передається в розряд CF, аналіз значення цього прапорця дозволяє судити про точність результату. Наприклад, подвоєння числа 65 (01000001) шляхом зсуву вліво дає точний результат 130 (10000010), що визначається значенням CF=0. Подальше подвоєння дасть неправильний результат 4 (00000100), що визначається значенням CF=1. Аналогічно використовуючи логічний зсув вправо (команда SHR(№18)), здійснюється ділення числа без знака на 2. Наприклад зсуваючи число 13 (00001101) вправо, отримуємо 6 (00000110) і CF=1, що свідчить про неточний результат. Аналогічні операції для чисел зі знаком реалізовуються за допомогою команд SAL (арифметичний зсув вліво) і SAR (арифметичний зсув вправо).

Відмінність ділення на 2 чисел без знака і чисел зі знаком полягає в тому, що за командою SHR старший біт заповнюється нулем, а за командою SAR значення в старшому біті не змінюється і передається в наступний молодший розряд. Наприклад, зсув SAR числа (–120) (10001000B) дає результат (–60) (11000100B). Команди SHL і SAL реалізують ті ж самі дії, тому коди операцій цих команд однакові.

Для здійснення циклічних зсувів вліво і вправо, без участі або з участю прапорця CF використовуються команди ROL (циклічний зсув вліво), ROR (циклічний зсув вправо), RCL (циклічний зсув вліво через CF) і RCR (циклічний зсув вправо через CF).

8.2.12 Команди обробки рядків

Рядком називають послідовність байтів або слів, що розміщуються в суміжних комірках пам'яті. Прикладом може служити рядок, який вводиться з термінала в МПС. Операції над рядками (таблиця 8.7)

виконуються над кожним елементом рядка (байтом або словом), так що час їх виконання пропорційний числу елементів в рядку. У МП–ра і8086 є декілька однобайтних команд що виконують прості операції над елементами рядками. Ці команди значно прискорюють маніпуляції над рядками завдяки зменшенню часу, що затрачується на обробку кожного елемента і на допоміжні дії, які необхідно виконувати при обробці послідовності елементів. До допоміжних дій відносяться: переадресація елемента рядка, зменшення на одиницю лічильника числа елементів рядка, що обробляється і перевірка досягнення лічильником нуля. Для ілюстрації часових затрат на обробку елементів рядків розглянемо процес пересилання рядка з однієї області пам'яті в іншу. Для вказівки початкової адреси першого елемента початкового рядка і рядка результату використовуються регістри SI (індекс джерела) і DI (індекс місця призначення). Відповідно, число елементів рядків, що пересилаються, задається в регістрі–лічильнику CX. Пересилання рядка полягає у виконанні наступних кроків (при описі кожного кроку вказана відповідна команда і число тактів, необхідних для її виконання):

a) Якщо CX=0, то пересилання закінчено;
b) Переслати байт з комірки з непрямою адресою, вказаною в SI, до AL: MOV AL, [SI]; AL ← M(DS:SI); 8T+5T;
c) Запам'ятати AL в елементі пам'яті з непрямою адресою, вказаною DI: MOV [DI], AL; M(ES:DI) ← AL; 9T+5T;
d) Збільшити SI на "1": INC SI; SI ← SI+1; 2T;
e) Збільшити DI на "1": INC DI; DI ← DI+1; 2T;
f) Зменшити CX на "1": DEC CX; CX ← CX–1; 2T;

перейти до: кроку b), якщо CX≠0 (ZF=0); JNZ Мітка; 8 тактів;

перейти до кроку a), якщо CX=0 (ZF=1); 4 такти.

Власне пересилання байта виконується на кроках b і c. На кроках d і e здійснюється обчислення адрес джерела і місця призначення для наступного елемента рядка. Нарешті, кроки a, f і g служать для визначення числа оброблених елементів. Число тактів, які витрачаються на виконання кроків b…e, дорівнює 31. Дії, які виконуються за допомогою вказаних чотирьох команд протягом кроків b…e, можуть бути виконані за допомогою однієї команди MOVSB (№24) (переслати однобайтний елемент рядка), що вимагає тільки 18 тактів. Для організації циклічного виконання команди MOVS (а також інших команд обробки рядків) використовується однобайтова команда–префікс REP (повторити). Вона записується перед основною командою (наприклад, REP MOVS) і забезпечує її виконання N–раз. Число повторень N заздалегідь записується до регістра CX і на кожному кроці зміст CX зменшується на одиницю. Використовуються три мнемоніки команди–префікса: REP, REPE/REPZ і REPNE/REPNZ (№29,30,31). Перша мнемоніка аналізує тільки одну умову – продовження повторень, доки CX≠0. У тих випадках, коли виникає необхідність додатково аналізувати умову ZF=1 (або ZF=0), використовується друга (або третя) мнемоніка. Команда REP дозволяє ще більше скоротити час на обробку рядка за рахунок поєднання операцій (CX)<–(CX)–1 з перевіркою досягнення лічильником нуля.

У ряді випадків виникає необхідність в пересиланні рядка в зворотному порядку починаючи не з першого, а з останнього її елемента. Наприклад, потрібно переслати рядок, що містить 10 байт і розташований в пам'яті в діапазоні адрес (SI)=200 … (SI)=209 в область пам'яті в діапазоні адрес (DI)=205 … (DI)=214. Ясно, що таке пересилання не можна виконувати починаючи з першого елемента, тому що області пам'яті з вихідним рядком і рядком–результатом перекриваються і перше ж пересилання "зіпсує" значення п'ятого елемента вихідного рядка.

Необхідне пересилання рядків можна здійснити починаючи з останнього елемента початкового рядка. У цьому випадку потрібно встановити початкові адреси елементів рядків рівними (SI)=209 і (DI)=214. Крім того, при пересиланні кожного елемента необхідно не збільшувати, а зменшувати на одиницю вміст індексних регістрів SI і DI. Напрям передачі встановлюється за допомогою прапорця DF: при значенні DF=1 відбувається автодекрементування індексних регістрів, а при значенні DF=0 – автоінкрементування.

Коли в якості елементів рядків використовуються не байти, а слова, відповідне збільшення або зменшення значень індексних регістрів здійснюється на два, тобто при DF=1: SI←SI–2; DI←DI–2, а при DF=0: SI←SI+2; DI←DI+2. Крім команди MOVS для дії з рядками є ще чотири команди: CMPS (№25 – порівняння елементів рядків), SCAS (№26 – сканування елементів рядка), LODS (№27 – завантаження елемента рядка) і STOS (№28 заповнення елемента рядка).

Команда CMPS (№25) дозволяє здійснити поелементне порівняння двох рядків, один з яких розташовується в пам'яті з непрямою адресою, вказаною в SI, а другий – з непрямою адресою, вказаною в DI. За командою CMPS відбувається віднімання елемента рядка з адресою в SI з елемента рядка з адресою в DI.

При виконанні операції порівняння результат віднімання не фіксується, а встановлюються відповідні значення прапорців (ZF та CF), завдяки яким визначається результат порівняння. Аналогічно з командою MOVS при порівнянні змінюються значення індексних регістрів за правилом: SI←SI±Δ; DI←DI±Δ, де “+” використовується при DF=0; “–” – при DF=1; Δ=1 – при W=0; Δ=2 при W=1.

Для циклічного повторення команди CMPS використовується префікс повторення REPNE/REPNZ або REPE/REPZ (повторювати доки елементи,

що порівнюються, не рівні (ZF=0) або доки елементи, що порівнюються, рівні (ZF=1)).

За командою сканування SCAS (№26) відбувається порівняння значення елемента рядка, розташованого з адреси, яка вказана в DI, зі значенням AL. При цьому також здійснюється операція віднімання, результат якої не фіксується. Одне з можливих застосувань команди сканування складається у пошуку елемента рядка, що дорівнює заданому зразку. Зразковий елемент завантажується в AL і потім організується цикл сканування з використанням префікса повторення REPNE/REPNZ. Як і при виконанні попередніх команд, кожний раз за командою SCAS відбувається автоінкремент (або автодекремент) адреси елемента: (DI)←(DI)±Δ.

Дві наступні команди LODS (завантаження елемента рядка, №27) і STOS (запис елемента рядка, №28), дозволяють завантажувати елементи рядка в акумулятор і записувати вміст акумулятора в рядок. При виконанні цих команд здійснюється підготовка адреси наступного елемента рядка, тобто модифікується вміст відповідного індексного регістра. Однак повторення команди LODS звичайно не використовується, а повторення команди STOS, що організується за допомогою префікса REP, застосовується при завантаженні рядка константою, яку заздалегідь вміщено в акумулятор.

Розглянуті вище команди роботи з рядками реалізують відносно прості операції, на базі яких можна організувати більш складні операції обробки рядків.

8.2.13 Команди безумовних переходів

Команди безумовних переходів мають мнемоніку JMP (таблиця 8.8).

Команда JMP дозволяє здійснити перехід в будь–яку точку програми, розташовану як в поточному програмному сегменті, так і в іншому сегменті

програм. При переході в межах поточного програмного сегмента використовуються перші три формати команди JMP (№1...3).

Таблиця 8.8 – Команди передачі керування мікропроцесором

№	Мнемоніка	Функція (алгоритм)	Формат					Число байт	Число тактів
			1 байт	2 байт	3 байт	4 байт			
1	JMP	IP←IP+disp16	11101001	disp L	disp H	(disp16 – число зі знаком)		3	15
2	JMP	IP ← IP+disp8	11101011	disp8	(disp8–число зі знаком)			2	15
3	JMP	IP ← r/m	11111111	md100 r/m				2...4	11 / 18+$T_{ва}$
4	JMP	IP,CS←ip,cs	11101010	ip_L	ip_H	cs_L	cs_H(5–й байт)	5	15
5	JMP	IP,CS ← r/m	11111111	md101 r/m				2...4	24+$T_{ва}$
6	CALL	Стек ← IP, IP ← IP + disp	11101000	disp L	disp H			3	19
7	CALL	Стек ← IP, IP ← r/m	11111111	md010 r/m				2...4	16 / 21+$T_{ва}$
8	CALL	Стек ← IP,CS IP,CS←ip ,cs	10011010	ip_L	ip_H	cs_L	cs_H(5–й байт)	5	28
9	CALL	Стек ← IP,CS IP,CS ← r/m	11111111	md011 r/m				2...4	37+$T_{ва}$
10	RETN	IP ← стек	11000011					1	8
11	RETN	IP ← стек SP←SP + data	11000010	data L	data H			3	17
12	RETF	CS,IP ← стек	11001011					1	12
13	RETF	CS,IP ← стек SP ← SP+data	11001010	data L	data H			3	18
14	JCC (псевдо–мнемо–ніка–див. таблицю 8.9)	IP ← IP+disp8, якщо виконується умова: cccc	0111cccc	disp8	disp8 – число зі знаком в додатковому коді			2	8 / 4
15	LOOP	CX ← CX–1 IP ← IP+disp8, якщо (CX)<>0	11100010	disp8				2	17 / 5
16	LOOPZ LOOPE	CX ← CX–1 IP ← IP+disp8, якщо (CX)<>0,ZF=1	11100001	disp8				2	18 / 6

Продовження таблиці 8.8

17	LOOPNZ LOOPNE	CX ← CX–1 IP ← IP+disp8, якщо (CX)<>0,ZF=0	11100000	disp8			2	18 / 6
18	JCXZ	IP ← IP+disp8, якщо (CX) = 0	11100011	disp8			2	18 / 6
19	INTN (INT type)	Стек←F,CS,IP IP ← (N*4); CS ← (N*4+2)	11001101	N(type)	(Порядок запису в стек: F(RF); CS; IP)		2	52
20	INT3	Стек←F,CS,IP IP ← (3*4+2); CS ← (3*4+2)	11001100				1	52
21	INTO	Стек←F,CS,IP; IP ← (4*4); CS ←(4*4+2), якщо OF = 1	11001110				1	52 або 4
22	IRET	F,CS,IP←стек	11001111	Порядок читання зі стека: IP; CS; F(RF).			1	24
23	CLC	CF ← 0	11111000				1	2
24	CMC	CF ← $\overline{CF}$	11110101				1	2
25	STC	CF ← 1	11111001				1	2
26	CLD	DF ← 0	11111100				1	2
27	STD	DF ← 1	11111101				1	2
28	CLI	IF ← 0	11111010				1	2
29	STI	IF ← 1	11111011				1	2
30	HLT	Зупинка МП–ра	11110100				1	2
31	WAIT	Переведення МП в стан очікування	10011011				1	3
32	ESC	Зовнішня команда для співпроцесора	11011xxx	mdYYY r/m			2...4	8+$T_{ва}$
33	LOCK	Блокування шин МПС	11110000				1	2
34	NOP	Немає операції	10010000				1	2

Примітка:

disp – зсув; L – молодший байт; H – старший байт; cccc – 4–розрядний код умови (див. таблицю 8. 9); N – тип переривання (0…255); xxx – вказує

номер співпроцесора; YYY – вказує номер (код операції), який повинен виконувати співпроцесор.

Перший формат команд безумовних переходів (таблиця 8.8) забезпечує перехід в довільну точку програми в межах поточного програмного сегмента, для чого до вмісту IP додається в додатковому коді 16–розрядний зсув, старший розряд якого є знаковим (діапазон переходів відносно адреси наступної команди: –32768…+32767). Другий, скорочений формат дозволяє перейти до точки програми, що знаходиться на відстані не більш ніж на: –128 … +127 від адреси команди, наступної за JMP. Нарешті, третій формат здійснює завантаження покажчика команд 16–розрядним числом, яке або розміщено за виконавчою адресою EA, що визначається постбайтом або знаходиться в одному з РЗП. Останній перехід називається непрямим, тому що використовується непряма адресація (адреса знаходиться в одному з РЗП або в двох комірках пам’яті).

Для реалізації безумовного переходу до точки програми, розташованої поза поточним програмним сегментом, коли потрібне перезавантаження сегментного регістра CS, використовуються четвертий і п'ятий формати JMP. Четвертий формат визначає прямий міжсегментний перехід, при якому у другому і третьому байтах машинного коду команди вказане нове значення регістра IP (PC), а в четвертому і п'ятому байтах – нове значення регістра CS. П'ятий формат за допомогою постбайта дозволяє визначити виконавчу адресу EA, за якою в пам’яті знаходяться нове значення регістра IP (PC) (в байтах пам'яті з адресами EA, EA+1) і нове значення регістра CS (в байтах пам'яті EA+2, EA+3).

8.2.14 Команди умовних переходів

Команди умовних переходів здійснюють передачу керування в залежності від результатів виконання попередніх операцій (таблиця 8.9).

Існують 16 різновидів команд умовних переходів, які в таблиці 8.8, №14 показано одним рядком.

Таблиця 8.9 – Команди умовних переходів

№	Мнемоніка	Умова переходу	Значення прапорців	КОП команди
Для чисел зі знаком				
1	JL / JNGE	Менше / не більше і не дорівнює $(< / \not\geq)$	SF ⊕ OF = 1	1100
2	JNL / JGE	Не меньше / більше або дорівнює $(\not< / \geq)$	SF ⊕ OF = 0	1101
3	JG / JNLE	Більше /не менше і не дорівнює $(> / \not\leq)$	(SF⊕ OF) ∨ ZF = 0	1111
4	JNG / JLE	Не більше / менше або дорівнює $(\not> / \leq)$	(SF⊕ OF) ∨ ZF = 1	1110
Для чисел без знака				
5	JB / JNAE / JC	Менше / не більше і не дорівнює $(< / \not\geq)$	CF = 1	0010
6	JNB / JAE / JNC	Не менше / більше або дорівнює $(\not< / \geq)$	CF = 0	0011
7	JA / JNBE	Більше /не менше і не дорівнює $(> / \not\leq)$	CF ∨ ZF = 0	0111
8	JNA / JBE	Не більше / менше або дорівнює $(\not> / \leq)$	CF ∨ ZF = 1	0110
Інші умовні переходи				
9	JE / JZ	Дорівнює / за нулем	ZF = 1	0100
10	JNE / JNZ	Не дорівнює / якщо не нуль	ZF = 0	0101
11	JS	За мінусом	SF = 1	1000
12	JNS	За плюсом	SF = 0	1001
13	JO	За переповненням	OF = 1	0000
14	JNO	За відсутністю переповнення	OF = 0	0001
15	JP / JPE	За парністю	PF = 1	1010
16	JNP / JPO	За непарністю	PF = 0	1011

Всі команди умовних переходів виконуються за: 8 тактів, якщо умова виконуються (є перехід); 4 такти, якщо умова не виконується (немає переходу).

Мнемоніки всіх команд умовних переходів мають першу літеру J (JMP), а наступні букви, які в рядку №14 позначено як СССС, у конкретних командах будуть мати те, чи інше значення в залежності від виконуваної

команди умовного переходу (таблиця 8.9). Чотири молодших біти першого байта машинного коду команди в таблиці 8.8, №14 позначено як CCCC, а в конкретній команді будуть приймати значення: від 0000 до 1111 (таблиця 8.9).

Розрізнюють три різновиди умовних переходів, які використовуються для: встановлення співвідношень чисел зі знаком, чисел без знака та переходів в залежності від значень окремих прапорців (таблиця 8.9). У перших двох різновидах для одних і тих же співвідношень між числами вибираються різні мнемоніки команд, оскільки одним і тим же співвідношенням чисел зі знаком і чисел без знака відповідають різні значення прапорців.

У мнемоніках команд умовних переходів при порівнянні чисел зі знаком для позначення умови "більше" використовується буква G (Greater – більше), а для позначення – "менше" буква L (Less – менше). Для аналогічних умов при порівнянні чисел без знака використовуються відповідно букви A (Above – над) і B (Below – під). Умови рівності позначаються буквою E (Equal – дорівнює), а не виконання деякої умови – буквою N (Not – не). Потрібно зазначити, що допускається використання двох різних мнемонік для кожної команди; наприклад, мнемоніки JL і JNGE – еквівалентні, оскільки умови "менше" і "не більше або дорівнює" – ідентичні.

Повний список мнемонік команд умовних переходів, умови переходів, коди операцій, а також відповідні булеві комбінації прапорців і їх значення наведено в таблиці 8.9.

Всі команди умовних переходів мають однаковий двобайтовий формат (таблиця 8.8, № 14), в першому байті якого задається код операцій (КОП), а у другому – 8–розрядний зсув, яке розглядається як число зі знаком і, отже, дозволяє здійснювати переходи в діапазоні від (– 128) до (+127) відносно

адреси наступної команди. При необхідності більш віддаленого ("далекого") переходу при виконанні умови використовується додатково команда безумовного переходу.

Час виконання кожної з команд умовних переходів вказано для двох випадків:

- умова виконана і керування дійсно передається у відповідності зі значенням другого байта команди;
- умова не виконана, так що керування передається наступній команді.

8.2.15 Команди організації циклів

Команди організації циклів (таблиця 8.8) введено в МП–р і8086 для зручності виконання обчислювальних циклів. До них відносяться наступні мнемоніки: LOOP (цикл, доки (CX)≠0), LOOPNZ/LOOPNE (цикл, доки (CX) ≠0 і не нуль/не дорівнює), LOOPZ/LOOPE (цикл, доки (CX) ≠0 і нуль/дорівнює) і JCXZ (перехід по нулю у регістрі CX) (команди з номерами 15…18 в таблиці 8.8). Кожна з цих команд має двобайтовий формат, у другому байті якого вказується 8–розрядний зсув, що використовується для організації переходу. Цей зсув розглядається як число зі знаком і перед обчисленням адреси переходу воно розширяється знаком до 16 розрядів. Вказане першим в таблиці 8.8 число тактів, що дорівнює 17/18, відповідає випадку, коли умова переходу в цих командах виконується. При не виконанні умови названі команди виконуються за п'ять/шість тактів. Як лічильник циклів використовується регістр CX, вміст якого в процесі виконання кожної з названих команд організації циклів (крім JCXZ) зменшується на одиницю (декрементується).

8.2.16 Команди виклику і повернення з підпрограм

Команда CALL (таблиця 8.8) дозволяє викликати підпрограму, розташовану як в поточному програмному сегменті, так і в іншій області пам'яті. Вона має такі ж формати, що і команда JMP, за винятком укороченого (команди з номерами 6…9 в таблиці 8.8). На відміну від команди JMP аналогічного формату за командою CALL перед зміною значень IP або IP і CS відбувається автоматичний запис в стек поточних значень цих регістрів, що забезпечує запам'ятовування точки повернення з підпрограми.

Для повернення з підпрограми використовується команда RET, під дією якої відбувається передача керування за адресою повернення, занесеному в стек при виконанні попередньої команди CALL. При поверненні з підпрограм, розташованих в поточному програмному сегменті, застосовуються перші два формати команди RET (№ 10, 11), причому другий формат відрізняється від першого тим, що до вмісту покажчика стека додається константа, що записана у 2–му і 3–му байтах команди. Це дозволяє одночасно з поверненням з підпрограми скидати параметри, записані в стек при виконання цієї підпрограми і які не використовуються надалі.

Для міжсегментного повернення застосовуються третій і четвертий формати RET (№ 12, 13), які забезпечують відновлення зі стека вмісту як регістра CS, так і програмного лічильника IP.

8.2.17 Команди програмних переривань

Команди програмних переривань (таблиця 8.8) включають три мнемоніки: INT (переривання), INTO (переривання за переповненням) і IRET (повернення з підпрограми обробки переривання).

Команда переривання INT type (№ 19) має двобайтовий формат, другий байт якого містить 8–розрядне число, що визначає тип (type) або рівень переривання. За командою INT type процесор переходить до виконання підпрограми обслуговування переривання вказаного рівня, причому автоматично виконуються дії, необхідні для забезпечення повернення в точку переривання. Ці дії полягають в наступному: вміст регістра прапорців F записується в стек (PUSHF), скидаються прапорці IF і TF (IF←0, TF←0), поточні значення регістра CS і покажчика команд IP записуються в стек (стек←CS, стек←IP).

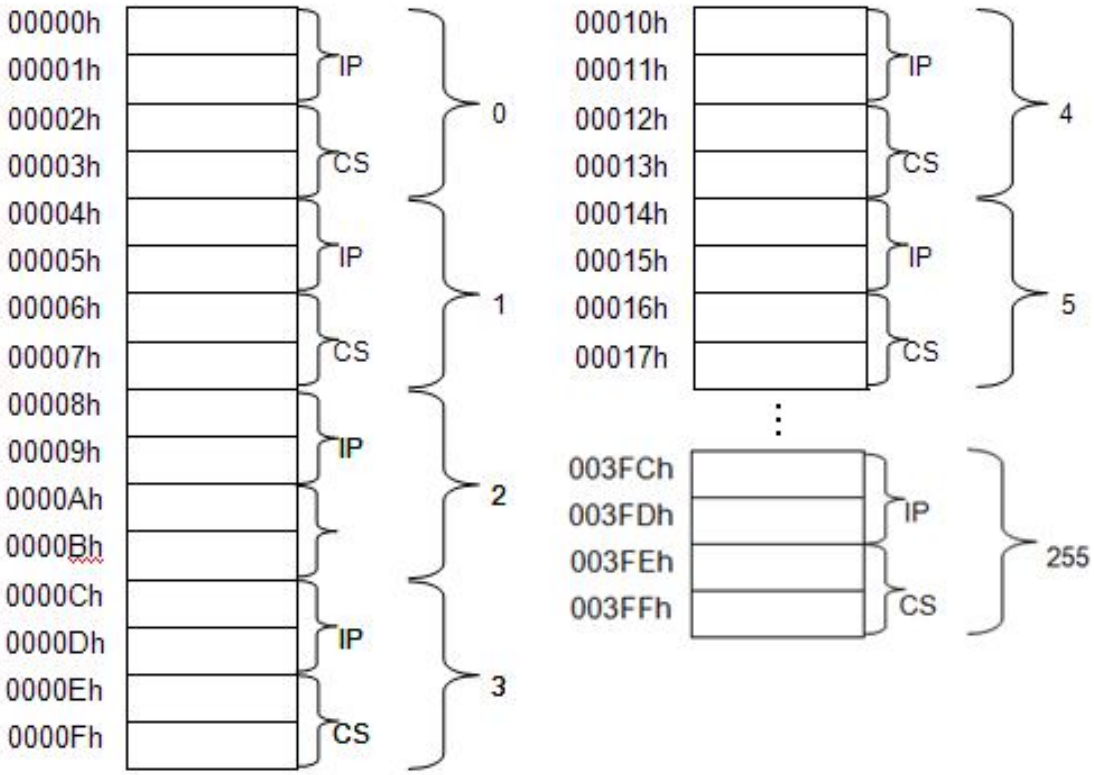
Для визначення початкової адреси підпрограми обслуговування переривання у відповідності зі значенням type використовується таблиця 8.10.

Для кожного з 256 (0…255) рівнів переривань в цій таблиці відведено по чотири байти: перші два байти визначають значення покажчика команд IP, другі значення сегментного регістра CS. Ця четвірка байтів визначає початкові адреси підпрограм обслуговування переривань (пари значень CS, IP), які повинні бути заздалегідь записані до комірок пам'яті за абсолютними адресами 00000H…003FFH. Адреса входу до таблиці 8.10 відповідна вказаному в команді INT type рівню переривання, визначається в МП таким чином. Після запам'ятовування у стек у поточних значень регістра прапорців F, регістрів CS і IP здійснюються завантаження: IP_L←M(type*4); IP_h←M(type*4+1); CS_L←M(type*4+2); CS_h←M(type*4+3). Нові значення CS і IP визначають початкову адресу необхідної підпрограми обслуговування переривання.

Команда переривання INT 3 (№ 20) має однобайтовий формат і не вимагає спеціальної вказівки рівня переривання в машинному коді команди. Код цієї команди (1 байт) автоматично сприймається процесором як переривання третього рівня (type=3) і звичайно використовується в

програмах як контрольна точка. Код команди (0CCh) можна вставити до програми, щоб змусити відлагоджувача робити зупинку в різних місцях з метою перегляду стану програми.

Таблиця 8.10 – Таблиця адрес підпрограм–обробників переривань



Команда переривання при переповненні INTO викликає перехід на обслуговування переривання четвертого рівня (type=4) у випадку, коли значення прапорця переповнення OF=1. Команда INTO звичайно використовується після арифметичних команд над числами зі знаком. Частіше за все декілька перших рівнів переривань (до 32) резервуються під обробку ряду специфічних ситуацій, таких, наприклад, як спроба ділення на нуль, переповнення і тому подібних. У таблиці 8.11 вказано призначення перших п'яти зарезервованих рівнів.

Особливість обробки переривань зарезервованих рівнів полягає в тому, що процесор переходить до їх обслуговування незалежно від значення

прапорця дозволу переривань IF.

Таблиця 8.11 Зарезервовані рівні переривань

Номер рівня	**Адреса таблиці**	**Призначення**
0	00000H	Обробка ситуації "ділення на нуль"
1	00004H	Робота в покроковому режимі (TF=1)
2	00008H	Обслуговування запиту переривання, що не маскується (за входом NMI)
3	0000CH	Обслуговування запиту переривання за однобайтовою командою INT3 (type=3)
4	00010H	Обробка ситуації "переповнення" (INTO)

Однобайтова команда IRET ставиться в кінці кожної підпрограми обслуговування переривання і забезпечує повернення з переривання. За цією командою процесор витягує з стека значення покажчика команд IP і сегментного регістра CS, а також відновлює колишній вміст регістра прапорців F (як і за командою POPF). При переході на підпрограму обслуговування переривання при необхідності вміст інших регістрів МП, шо відповідає програмі, що переривається, може бути запам'ятовано в стеку за допомогою команд запису до стека (PUSH), а потім відновлено при поверненні з підпрограми за допомогою команд читання зі стека (POP).

8.2.18 Команди керування мікропроцесором

8.2.18.1 Загальні відомості

У групі команд керування мікропроцесором розрізнюють три типи: операції з прапорцями, встановлення МП в особливі стани і синхронізація з

співпроцесорами (таблиця 8.8).

8.2.18.2 Операції з прапорцями

Ці команди містяться в таблиці 8.8 і включають 7 мнемонік: STC (№25, встановлення прапорця перенесення CF), CMC (№24, інверсія прапорця перенесення CF), CLC (№23, скидання прапорця перенесення CF), STD (№27, встановлення прапорця напряму DF), CLD (№26, скидання прапорця напряму DF), STI (№29) і CLI (№28), відповідно встановлення і скидання прапорця дозволу переривання IF. Призначення перерахованих команд очевидне. Так команди STC, CMC і CLC дозволяють задавати необхідне початкове значення прапорця CF при різних арифметичних і логічних перетвореннях даних і зсувах. Команди STD і CLD використовуються при обробці рядків для задання напряму обробки: від першого елемента рядка до останнього або навпаки. Нарешті, команди STI і CLI служать для керування системою переривань і дозволяють відповідно дозволити або заборонити зовнішнє переривання у будь–якій частині програми.

8.2.18.3 Команди встановлення МП в особливі стани

Ці команди (таблиця 8.8) містять дві мнемоніки: HLT (№30, зупинка) і WAIT (№31, очікування), які переводять процесор відповідно в стан зупинки або очікування.

Знаходячись в будь–кому з цих станів, процесор не виконує ніяких дій (вводяться такти очікування TW на часових діаграмах роботи МП), доки не з’являться певні зовнішні впливи. З стану зупинки процесор може бути виведено двома способами: шляхом початкового скидання (сигналом за входом RESET) або зовнішнім перериванням (сигналом запиту за входом INT). При першому способі процесор перейде до виконання основної програми з початку, у другому до виконання підпрограми обслуговування

переривання відповідного рівня. При виконанні команди HLT вміст покажчика команд IP автоматично збільшується на одиницю, так що після виконання підпрограми обслуговування переривання процесор перейде до виконання наступною за HLT команди.

Основний спосіб виведення з стану очікування полягає в подачі сигналу низького рівня (логічний 0) на вхід мікропроцесора TEST. Таким чином, час знаходження в стані очікування визначається моментами виконання команди WAIT і появою активного нуля на вході TEST. Керування очікуванням за допомогою цього механізму дозволяє здійснити синхронізацію, тобто узгодити роботу процесора з різними зовнішніми пристроями (наприклад, з співпроцесорами або з пристроями, що мають меншу швидкодію). У випадку, коли сигнал на вході TEST активний (логічний 0) в момент виконання команди WAIT, процесор буде знаходитися в стані очікування протягом трьох тактів ГТІ (час виконання команди WAIT). Дії входу TEST схожі на дії входу READY. Відмінність в тому, що READY – вхід для апаратної перевірки готовності пам'яті або ПВВ до обміну з МП–ром, а TEST – вхід для програмної перевірки.

Другий спосіб виведення з стану очікування полягає в подачі запиту переривання по входу INT. Однак в цьому випадку процесор виходить з стану очікування тільки тимчасово. За командою WAIT не відбувається автоматичне нарощування покажчика команд IP, внаслідок чого після виконання відповідної підпрограми обслуговування переривання процесор знову перейде до виконання команди WAIT, тобто перейде в стан очікування. Таким чином, МП–р i8086 на відміну від i8080 може виконувати підпрограми переривання під час очікування сигналу готовності від зовнішнього пристрою. Важливо відмітити, що при поновленні роботи процесора після очікування (за сигналом TEST=0) зовнішні переривання не будуть обслуговуватися, доки не виконається наступна за WAIT команда.

8.2.18.4 Команди синхронізації з співпроцесорами

Ці команди використовуються під час проектування багатопроцесорних систем на основі МП–ра і8086 . Хоч система команд МП і8086 досить розвинена, проте в ній відсутні деякі команди, характерні для високопродуктивних систем. Відсутні команди процесора можна реалізувати за допомогою відповідних підпрограм. Однак більш ефективним рішенням є використання співпроцесорів (спеціалізованих процесорів), призначених для розширення функцій основного процесора. Наприклад, відсутні в системі команд операції над числами з плаваючою комою можуть бути виконано, наприклад, за допомогою співпроцесора і8087.

Для організації сумісної роботи основного процесора системи з співпроцесорами служить команда ESC (№32, таблиця 8.8). Перший байт команди містить код операції ESC, що дорівнює 11011, і трьохрозрядне поле ххх; другий байт має структуру постбайта, в якому поле *reg* позначено як ууу.

Поле ххх вказує номер того співпроцесора багатопроцесорної системи, який повинен виконати відповідну операцію, а поле *ууу* – номер (код) цієї операції. Для організації взаємодії основного процесора зі співпроцесорами останні повинні стежити за появою в потоку команд основного процесора команди ESC, яка вкаже, який співпроцесор і яку операцію повинен виконувати. У загальному випадку поля ххх *і* ууу дозволяють задати 64 комбінації 6–розрядних двійкових кодів. Наприклад, можна побудувати систему з одним співпроцесором, який виконує 64 різні операції, або з 8–ю співпроцесорами, кожний з яких буде виконувати до 8 операцій, і т. ін.

Поля *mod* і *r/m* 2–го байта команди ESC використовуються для

задавання адреси операнда, який буде приймати участь у виконанні команди ESC. За вмістом цих полів основний процесор витягує операнд з пам'яті і виставляє його значення на ШД як операнд для співпроцесора. Таким чином, основний процесор видає всю необхідну інформацію для роботи відповідного співпроцесора: момент включення в роботу (поява коду операції ESC), номер співпроцесора (поле xxx), код операції (поле yyy) і операнд (видається на ШД).

Як приклад використання арифметичного співпроцесора розглянемо процес перемноження двох чисел з плаваючою комою. Нехай числа знаходяться в сегменті даних зі зсувами вказаними в індексних регістрах *SI* і *DI*, а результат потрібно вмістити на місце другого операнда. Співпроцесор № 1 (x=001), виконує операції yyy, які закодовані таким чином: 000 – завантаження (LOAD) даних до акумулятора співпроцесора; 001 – множення (MUL) чисел з плаваючою комою; 010 – запам'ятовування (STO) акумулятора співпроцесора в пам'яті на місце другого операнда.

Відповідний фрагмент програми для основного процесора представлено на рисунку 8.2. Кожній команді ESC передує команда WAIT, яка синхронізує роботу основного процесора з співпроцесором. Тут припускається, що сигнал готовності співпроцесора (логічний 0) подається на вхід TEST.

Є ще одна команда, яка може застосовуватися при роботі з співпроцесором команда LOCK. Вона є однобайтовим префіксом блокування шин (машинний код 11110000B=F0H), при виконанні якої на однойменному виході мікропроцесора формується логічний нуль на час виконання наступної команди. Цей сигнал на виході LOCK повідомляє співпроцесорам багатопроцесорної системи, що при виконанні наступної команди забороняється робити запит системної шини.

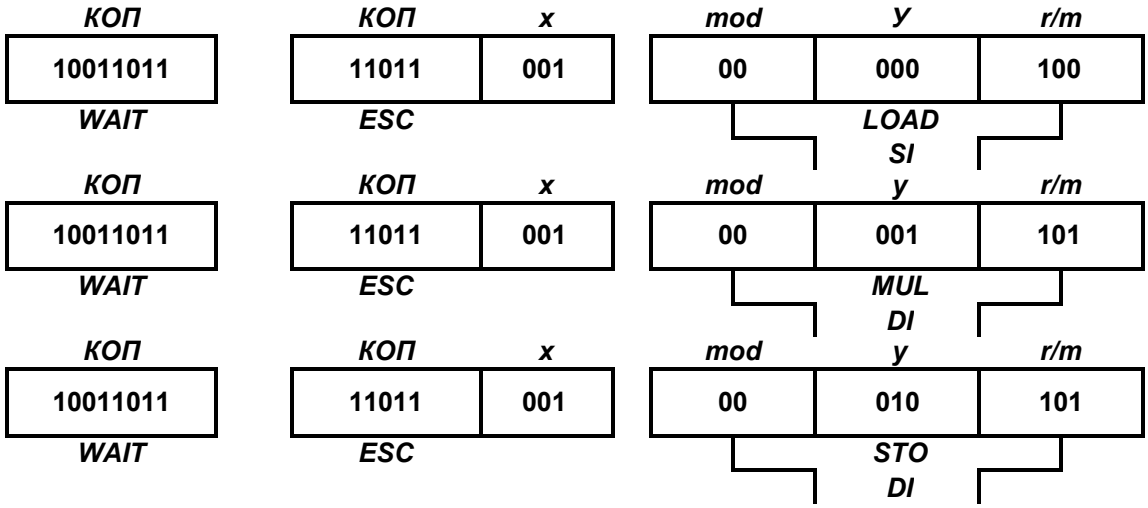


Рисунок 8.2 – Фрагмент програми множення чисел з плаваючою комою

Проектування багатопроцесорних систем часто пов'язано з розв'язанням питання керування доступом до спільної пам'яті, яка зветься базою даних (БД). Для запобігання конфліктним ситуаціям і помилкам в роботі системи потрібно організувати керування так, щоб в кожний момент часу тільки один процесор мав доступ до спільної БД. Для цього необхідно, по–перше, мати ознаку дозволу доступу і, по–друге, виключити можливість аналізу цієї ознаки одночасно декількома процесорами системи. Ознака дозволу доступу має два значення: "0" – доступ дозволено і "1" – доступ заборонено. Для зберігання цієї ознаки виділяється деяка комірка пам'яті, яку визначимо SEMAPHOR. Доки один процесор аналізує вміст комірки SEMAPHOR, робота інших процесорів системи блокується за допомогою префікса LOCK. Цей префікс може передувати будь–якій команді в програмі і спричиняє появу активного сигналу на виході LOCK центрального процесора, який використовується для блокування інших процесорів. Нижче, як приклад використання префікса LOCK, представлено фрагмент програми керування доступом до спільної бази даних:

```
CHECK:    MOV AL, 1          ; підготовка заборони доступу
```

```
                    LOCK  XCHG       ; читання значення ознаки дозволу з
                    [SEMAPHOR], AL   ;одночасною забороною доступу і
                                     ;блокуванням
                    TEST AL, AL      ; аналіз значення ознаки дозволу доступу
                    JNZ CHECK        ; перехід до повторної перевірки значення
                                     ;ознаки
                                     ;команди, що здійснюють допуск до БД
……                                   ;іншим процесорам
                    MOV              ;встановлення ознаки в стан –  доступ
                    [SEMAPHOR], 0    ;дозволено
```

8.2.18.5 Порожня операція

Окреме місце в системі команд мікропроцесора, що розглядається, займає однобайтна команда NOP – відсутність операції (таблиця 8.8). За цією командою МП–р не виконує ніяких дій. Команда не впливає на прапорці і не має операндів.

Вона використовується для задавання часової затримки, тривалістю два такти, або для резервування однієї комірки пам'яті.

Машинний код команди NOP співпадає з кодом команди XCHG AX, AX, за якою також не виконується ніяких дій.

8.3 Мікроконтролер типу МК51

8.3.1 Загальні відомості

111 основних команд мікроконтролера МК51 розбито на п'ять групп [15, 16, 18]:

- команди передачі даних;
- арифметичні команди;
- логічні операції;
- операції з бітами;

– команди передачі керування.

Таблиці команд, які наведено нижче, дають достатньо повну характеристику кожної з базових команд мікроконтролера:

– назва команди (у стислій формі описує функцію, яку команди виконують);
– мнемокод (включає в себе мнемоніку команди та опис операндів, що беруть участь в операції);
– код операції (КОП);
– тип команди (позначається як Т);
– довжину команди в байтах (позначається як Б);
– кількість машинних циклів, необхідних для виконання команди (позначається як Ц);
– коментар (операція).

8.3.2 Символічні позначення, які використовуються при описі команд мікроконтролера

При описі команд мікроконтролера (таблиці 8.12…8.15) використовуються наступні символічні позначення:

– A – регістр–акумулятор (8 біт);
– Rn – один з регістрів загального призначення поточного банку РЗП (R0/R1/R2/R3/R4/R5/R6/R7; n=0, 1,…,7). RS1, RS0 – два біти регістра прапорців – PSW;

Вибір поточного банку РЗП		
№банку	RS1	RS0
0	0	0
1	0	1
2	1	0
3	1	1

– @ Ri – непряма адресація. Операнд знаходиться в резидентній пам'яті даних (РПД) або в зовнішній пам'яті даних (ЗПД) за адресою, що міститься в одному з двох РЗП поточного банку: R0/R1(Ri, де i=0/1);
– ad; add; ads – позначення прямоадресуємого байта, що міститься в

РПД або в одному з 21 регістрів спеціальних функцій (РСФ);

- # D8 – 8–розрядний безпосередній операнд (константа);
- DPTR – 16– розрядний регістр МК–ра;
- PC – програмний лічильник – 16– розрядний регістр МК–ра;
- @ DPTR – непряма адресація. Операнд знаходиться у зовнішній пам'яті даних (ЗПД) за адресою, що міститься в регістрі DPTR;
- @ A + DPTR; @ A + PC: базова–індексна (непряма) адресація. Операнд знаходиться в пам'яті програм (ПП) за адресою, яка обчислюється відповідно як сума: (A + DPTR) або (A + PC);
- B – регістр–розширювач акумулятора (8 біт);
- Bit – позначення прямоадресуемого біта, що міститься в одному з 128 бітів РПД (8 * 16 = 128) або в 11 регістрах спеціальних функцій (див. програмістську модель);
- C – прапорець перенесення / позики регістра PSW;
- ad16; ad11 – відповідно 16–ти або 11–ти розрядна пряма адреса пам'яті программ;
- rel – 8–ми розрядне число зі знаком: (–128 ... 127), яке при виконанні переходів підсумовується з адресою наступної команди:
  rel = ADR мітки – ADR наступної команди.

8.3.3 Команди передачі данних

Більшу частину команд даної групи (таблиця 8.12) складають команди передачі та обміну байтами. Команди пересилання входять і в групу команд роботи з окремими бітами. Всі команди даної групи не модифікують прапорці результату, за винятком команд завантаження регістра PSW, тригера (прапорця) C і акумулятора (встановлюється прапорець паритету).

На рисунку 8.3 зображено граф шляхів передачі даних в МК–рі.

Таблиця 8.12 – Група команд передачі даних

№	Назва команди	Мнемокод	КОП	Т	Б	Ц	Операція
1	Пересилання до акумулятора з регістра (n=0..7)	**MOV A, Rn**	11101rrr	1	1	1	**(A) ← (Rn)**
2	Пересилання до акумулятора прямоадресуємого байта	**MOV A, ad**	11100101	3	2	1	**(A) ← (ad)**
3	Пересилання до акумулятора байта з РПД (i=0,1)	**MOV A, @Ri**	1110011i	1	1	1	**(A) ← РПД(Ri)**
4	Завантаження до акумулятора константи	**MOV A, #D8**	01110100	2	2	1	**(A) ← D8**
5	Пересилання до регістра (n=0..7) з акумулятора	**MOV Rn, A**	11111rrr	1	1	1	**(Rn) ← (A)**
6	Пересилання до регістра прямоадресуємого байта	**MOV Rn, ad**	10101rrr	3	2	2	**(Rn) ← (ad)**
7	Завантаження до регістра (n=0..7) константи	**MOV Rn, #D8**	01111rrr	2	2	1	**(Rn) ← D8**
8	Пересилання акумулятора за прямою адресою	**MOV ad, A**	11110101	3	2	1	**(ad) ← (A)**
9	Пересилання регістра за прямою адресою	**MOV ad, Rn**	10001rrr	3	2	2	**(ad) ← (Rn)**
10	Пересилання прямоадресуюємого байта за прямою адресою	**MOV add, ads**	10000101	9	3	2	**(add) ← (ads)**
11	Пересилання байта з РПД за прямою адресою	**MOV ad, @Ri**	1000011i	3	2	2	**(ad) ← РПД(Ri)**
12	Пересилання константи за прямою адресою	**MOV ad, #D8**	01110101	7	3	2	**(ad) ← D8**
13	Пересилання до РПД з акумулятора	**MOV @Ri, A**	1111011i	1	1	1	**РПД(Ri) ← (A)**
14	Пересилання до РПД прямоадресуємого байта	**MOV @Ri, ad**	0110011i	3	2	2	**РПД(Ri) ← (ad)**
15	Пересилання до РПД константи	**MOV @Ri, #D8**	0111011i	2	2	1	**РПД(Ri) ← D8**
16	Завантаження регістра–покажчика даних	**MOV DPTR, #D16**	10010000	13	3	2	**(DPTR) ← D16**
17	Пересилання до акумулятора байта з ПП	**MOVC A, @A+DPTR**	10010011	1	1	2	**(A) ←ПП((A) + (DPTR))**
18	Пересилання до акумулятора байта з ПП	**MOVC A, @A+PC**	10000011	1	1	2	**(PC) ← (PC) + 1** **(A) ← ПП((A) + (PC))**
19	Пересилання до акумулятора байта з ЗПД	**MOVX A, @Ri**	1110001i	1	1	2	**(A) ← ЗПД(Ri)**
20	Пересилання до акумулятора байта з розширеної ЗПД	**MOVX A, @DPTR**	11100000	1	1	2	**(A) ← ЗПД(DPTR)**
21	Пересилання до ЗПД з акумулятора	**MOVX @Ri, A**	1111001i	1	1	2	**ЗПД(Ri) ← (A)**
22	Пересилання до розширеної ЗПД з акумулятора	**MOVX @DPTR, A**	11110000	1	1	2	**ЗПД(DPTR) ← (A)**
23	Завантаження до стека	**PUSH ad**	11000000	3	2	2	**(SP) ← (SP) + 1** **РПД(SP) ← (ad)**
24	Вилучення зі стека	**POP ad**	11010000	3	2	2	**(ad) ← РПД(SP)** **(SP) ← (SP) – 1**
25	Обмін акумулятора з регістром	**XCH A, Rn**	11001rrr	1	1	1	**(A) ↔ (Rn)**
26	Обмін акумулятора з прямоадресуємим байтом	**XCH A, ad**	11000101	3	2	1	**(A) ↔ (ad)**
27	Обмін акумулятора з байтом з РПД	**XCH A, @Ri**	1100011i	1	1	1	**(A) ↔ РПД(Ri)**
28	Обмін молодшої тетради акумулятора з молодшою тетрадою байта РПД	**XCHD A, @Ri**	1101011i	1	1	1	**$(A_{0–3}) \leftrightarrow РПД(Ri)_{0–3}$**

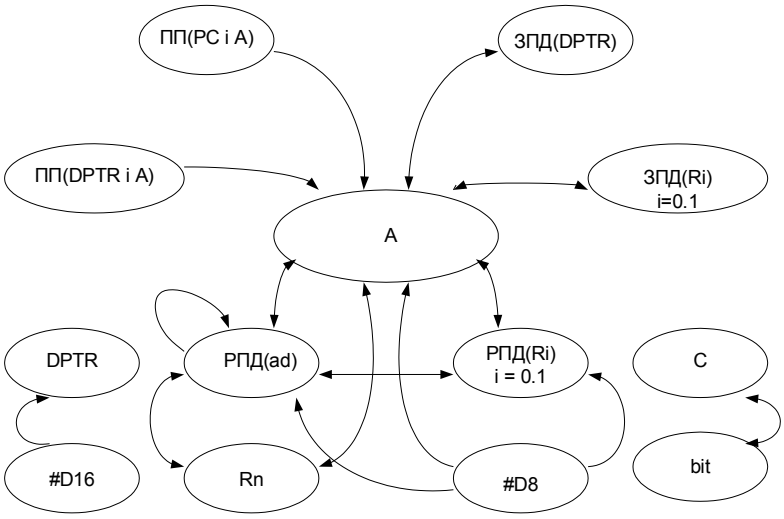


Рисунок 8.3 – Граф шляхів передачі даних в МК–рі

Окремою вершиною на цьому графі зображено акумулятор А, тому що він бере участь у більшості пересилань і адресується за допомогою неявної або прямої адресації. Проте деякі пересилання виконуються без участі акумулятора.

В число команд пересилань входять операції зі стеком, який організується в РПД.

Для адресації комірок стека використовується регістр–покажчик стека (SP), який дозволяє адресувати будь–яку комірку внутрішнього ОЗП. Запис інформації в стек проводиться командою PUSH ad, а зчитування зі стека – POP ad. У початковоому стані SP адресує “верхівку” стека – останню комірку стекової пам’яті у яку записано інформацію. Довжина операнда при роботі зі стеком дорівнює 8 біт. Перед записом у стек вміст SP інкрементується, а після зчитування – декрементується.

8.3.4 Арифметичні команди

В наборі команд МК–ра (таблиця 8.13) є такі арифметичні операції: додавання, додавання з урахуванням прапорця перенесення, віднімання з позикою, множення і ділення, інкремент, декремент та десяткова корекція акумулятора.

В АЛП проводяться дії над цілими числами. У таких операціях над двома операндами, як: додавання ADD, додавання з перенесенням ADDC та віднімання з позикою SUBB акумулятор зберігає перший операнд і приймає результат операції.

Другим операндом може бути регістр обраного банку робочих регістрів, регістр резидентної пам’яті даних із непрямою і прямою адресацією або байт безпосередніх даних. Вказані операції впливають на прапорці: переповнення, перенесення, допоміжного перенесення і прапорець парності в слові стану процесора PSW.

Використання розряду перенесення дозволяє багаторазово підвищити точність при виконанні операцій додавання ADDC і віднімання SUBB.

Виконання операцій додавання і віднімання з урахуванням знаку може бути здійснено за допомогою прапорця переповнення OV регістра PSW. Прапорець допоміжного перенесення AC забезпечує виконання арифметичних операцій у двійково–десятковому коді і використовується командою десяткової корекції DA A.

Операції інкремента і декремента на прапорці не впливають.

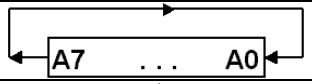
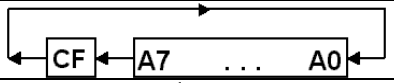
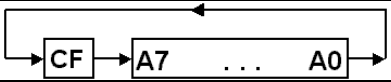
8.3.5 Логічні команди

АЛП МК–ра дає можливість виконувати логічні операції над байтовими і бітовими операндами. В таблицю 8.14 включено команди виконання логічних операцій над вмістом 8–розрядних даних. Операції над бітовими операндами описано в таблиці 8.15.

Таблиця 8.13 – Група команд арифметичних операцій

№	Назва команди	Мнемокод	КОП	Т	Б	Ц	Операція
1	Додавання акумулятора з регістром (n=0..7)	**ADD A, Rn**	00101rrr	1	1	1	**(A) ← (A) + (Rn)**
2	Додавання акумулятора з прямоадресуємим байтом	**ADD A, ad**	00100101	3	2	1	**(A) ← (A) + (ad)**
3	Додавання акумулятора з байтом з РПД (i=0,1)	**ADD A, @Ri**	0010011i	1	1	1	**(A) ← (A) + РПД(Ri)**
4	Додавання акумулятора з константою	**ADD A, #D8**	00100100	2	2	1	**(A) ← (A) + D8**
5	Додавання акумулятора з регістром і перенесенням	**ADDC A, Rn**	00111rrr	1	1	1	**(A) ← (A) + (Rn) + (C)**
6	Додавання акумулятора з прямоадресуємим байтом і перенесенням	**ADDC A, ad**	00110101	3	2	1	**(A) ← (A) + (ad) + (C)**
7	Додавання акумулятора з байтом з РПД і перенесенням	**ADDC A, @Ri**	0011011i	1	1	1	**(A) ← (A) + РПД(Ri) + (C)**
8	Додавання акумулятора з константою і перенесенням	**ADDC A, #D8**	00110100	2	2	1	**(A) ← (A) + D8 + (C)**
9	Десяткова корекція акумулятора (після додавання двох операндів, що представлено в упакованому BCD–коді)	**DA A**	11010100	1	1	1	**Якщо (A_{3-0}) > 9V (AC)=1, то (A_{7-0}) ←(A_{7-0}) + 6, якщо (A_{7-4}) > 9V (C)=1, то (A_{7-4}) ← (A_{7-4}) + 6**
10	Віднімання з акумулятора регістра і позики	**SUBB A, Rn**	10011rrr	1	1	1	**(A) ← (A) – (C) – (Rn)**
11	Віднімання з акумулятора прямоадресуємого байта і позики	**SUBB A, ad**	10010101	3	2	1	**(A) ← (A) – (C) – (ad)**
12	Віднімання з акумулятора байта РПД і позики	**SUBB A, @Ri**	1001011i	1	1	1	**(A) ← (A) – (C) – РПД(Ri)**
13	Віднімання з акумулятора константи і позики	**SUBB A, #D8**	10010100	2	2	1	**(A) ← (A) – (C) – D8**
14	Інкремент акумулятора	**INC A**	00000100	1	1	1	**(A) ← (A) + 1**
15	Інкремент регістра	**INC Rn**	00001rrr	1	1	1	**(Rn) ← (Rn) + 1**
16	Інкремент прямоадресуюємого байта	**INC ad**	00000101	3	2	1	**(ad) ← (ad) + 1**
17	Інкремент байта в РПД	**INC @Ri**	0000011i	1	1	1	**РПД(Ri) ← РПД(Ri) + 1**
18	Інкремент покажчика даних	**INC DPTR**	10100011	1	1	2	**(DPTR) ← (DPTR) + 1**
19	Декремент акумулятора	**DEC A**	00010100	1	1	1	**(A) ← (A) – 1**
20	Декремент регістра	**DEC Rn**	00011rrr	1	1	1	**(Rn) ← (Rn) – 1**
21	Декремент прямоадресуюємого байта	**DEC ad**	00010101	3	2	1	**(ad) ← (ad) – 1**
22	Декремент байта в РПД	**DEC @Ri**	0001011i	1	1	1	**РПД(Ri) ← РПД(Ri) – 1**
23	Множення акумулятора на регістр B	**MUL AB**	10100100	1	1	4	**Якщо (A)x(B) > 255, то (B)(A) ← (A)x(B); якщо (A)x(B) <255, то (A) ←(A)x(B)**
24	Ділення аккумулятора на регістр B	**DIV AB**	10000100	1	1	4	**(A)←(A)/(B)—цела частина, (B) ← R ((A) / (B)) —залишок від ділення**

Таблиця 8.14 – Група логічних команд

№	Назва команди	Мнемокод	КОП	Т	Б	Ц	Операція
1	Логічне І акумулятора і регістра	**ANL A, Rn**	01011rrr	1	1	1	**(A) ← (A) Λ (Rn)**
2	Логічне І акумулятора і прямоадресуємого байта	**ANL A, ad**	01010101	3	2	1	**(A) ← (A) Λ (ad)**
3	Логічне І акумулятора і байта з РПД	**ANL A, @Ri**	0101011i	1	1	1	**(A) ← (A) Λ РПД(Ri)**
4	Логічне І акумулятора і константи	**ANL A, #D8**	01010100	2	2	1	**(A) ← (A) Λ D8**
5	Логічне І прямоадресуємого байта і акумулятора	**ANL ad, A**	01010010	3	2	1	**(ad) ← (ad) Λ (A)**
6	Логічне І прямоадресуємого байта і константи	**ANL ad, #D8**	01010011	7	3	2	**(ad) ← (ad) Λ D8**
7	Логічне АБО акумулятора і регістра	**ORL A, Rn**	01001rrr	1	1	1	**(A) ← (A) V (Rn)**
8	Логічне АБО акумулятора і прямоадресуємого байта	**ORL A, ad**	01000101	3	2	1	**(A) ← (A) V (ad)**
9	Логічне АБО акумулятора і байта з РПД	**ORL A, @Ri**	0100011i	1	1	1	**(A) ← (A) V РПД(Ri)**
10	Логічне АБО акумулятора і константи	**ORL A, #D8**	01000100	2	2	1	**(A) ← (A) V D8**
11	Логічне АБО прямоадресуємого байта і акумулятора	**ORL ad, A**	01000010	3	2	1	**(ad) ← (ad) V (A)**
12	Логічне АБО прямоадресуємого байта і константи	**ORL ad, #D8**	01000011	7	3	2	**(ad) ← (ad) V D8**
13	Виключне АБО акумулятора і регістра	**XRL A, Rn**	01101rrr	1	1	1	**(A) ← (A) ⊕ (Rn)**
14	Виключне АБО акумулятора і прямоадресуємого байта	**XRL A, ad**	01100101	3	2	1	**(A) ← (A) ⊕ (ad)**
15	Виключне АБО акумулятора і байта РПД	**XRL A, @Ri**	0110011i	1	1	1	**(A) ← (A) ⊕ РПД(Ri)**
16	Виключне АБО акумулятора і константи	**XRL A, #D8**	01100100	2	2	1	**(A) ← (A) ⊕ D8**
17	Виключне АБО прямоадресуємого байта і акумулятора	**XRL ad, A**	01100010	3	2	1	**(ad) ← (ad) ⊕ (A)**
18	Виключне АБО прямоадресуємого байта і константи	**XRL ad, #D8**	01100011	7	3	2	**(ad) ← (ad) ⊕ D8**
19	Скидання акумулятора	**CLR A**	11100100	1	1	1	**(A) ← 0**
20	Інверсія акумулятора	**CPL A**	11110100	1	1	1	**(A) ← ($\overline{A}$)**
21	Зсув акумулятора вліво циклічний 	**RL A**	00100011	1	1	1	**(A_{n+1}) ← (A_n), n=0..6, (A_0) ← (A_7)**
22	Зсув акумулятора вліво циклічний через перенесення 	**RLC A**	00110011	1	1	1	**(A_{n+1})← (A_n), n=0..6, (A_0) ← (C), (C) ← (A_7)**
23	Зсув акумулятора вправо циклічний 	**RR A**	00000011	1	1	1	**(A_n) ← (A_{n+1}), n=0..6, (A_7) ← (A_0)**
24	Зсув акумулятора вправо циклічний через перенесення 	**RRC A**	00010011	1	1	1	**(A_n) ← (A_{n+1}), n=0..6, (A_7) ← (C), (C) ← (A_0)**
25	Обмін місцями тетрад у акумуляторі	**SWAP A**	11000100	1	1	1	**(A_{3-0}) ↔ (A_{7-4})**

Таблиця 8.15 – Група команд операцій з бітами

№	Название команди	Мнемокод	КОП	Т	Б	Ц	Операція
1	Скидання перенесення	**CLR C**	11000011	1	1	1	**(C) ← 0**
2	Скидання прямоадресуємого біта	**CLR bit**	11000010	4	2	1	**(b) ← 0**
3	Встановлення перенесення	**SETB C**	11010011	1	1	1	**(C) ← 1**
4	Встановлення прямоадресуємого біта	**SETB bit**	11010010	4	2	1	**(b) ← 1**
5	Інверсія перенесення	**CPL C**	10110011	1	1	1	**(C) ← $\overline{C}$**
6	Інверсія прямоадресуємого біта	**CPL bit**	10110010	4	2	1	**(b) ← $\overline{b}$**
7	Логічне І перенесення і прямоадресуємого біта	**ANL C, bit**	10000010	4	2	2	**(C) ← (C) Λ (b)**
8	Логічне І перенесення та інверсії прямоадресуємого біта	**ANL C,/bit**	10110000	4	2	2	**(C) ← (C) Λ ($\overline{b}$)**
9	Логічне АБО перенесення і прямоадресуємого біта	**ORL C, bit**	01110010	4	2	2	**(C) ← (C) V (b)**
10	Логічне АБО перенесення й інверсії прямоадресуємого біта	**ORL C,/bit**	10100000	4	2	2	**(C) ← (C) V ($\overline{b}$)**
11	Пересилання прямоадресуємого біта у перенесення	**MOV C, bit**	10100010	4	2	1	**(C) ← (b)**
12	Пересилання перенесення у прямоадресуємий біт	**MOV bit, C**	10010010	4	2	2	**(b) ← (C)**

Система команд МК–ра дозволяє реалізувати такі логічні операції: “І”, “АБО”, “ВИКЛЮЧНЕ АБО” над операндом в регістрі–акумуляторі А і байтом–джерелом. Другим операндом (байтом–джерелом) при цьому можуть бути регістр у обраному банку робочих регістрів, регістр внутрішнього ОЗП (РПД), що адресується за допомогою непрямої адресації, комірки внутрішнього ОЗП, що адресуються прямо, і регістри спеціального призначення, а також безпосередні операнди.

Існують логічні операції, що виконуються тільки в акумуляторі: скидання й інвертування усіх восьми розрядів А; циклічний зсув вліво і вправо; циклічний зсув вліво і вправо з урахуванням прапорця перенесення; обмін місцями старшої і молодшої тетрад акумулятора.

Приклади використання логічних команд наведено у 8.2.10.

8.3.6 Операції з бітами

До складу МК–ра входить так званий “бітовий” процесор, що забезпечує виконання ряду операцій над бітами (таблиця 8.15).

Бітовий процесор є частиною архітектури МК–ра сімейства МК51 і його можна розглядати як незалежний процесор для побітової обробки. Бітовий процесор виконує свій набір команд, має свій ОЗП, що адресується побітово, і свій пристрій введення/виведення.

Команди, що оперують із бітами, забезпечують пряму адресацію 128 бітів у шістнадцятьох комірках внутрішнього ОЗП (комірки з адресами 20H–2FH) і пряму побітову адресацію регістрів спеціального призначення, адреси яких кратні восьми: P0 (80H), TCON (88H), P1 (90H), SCON (98H), P2 (A0H), IE (A8H), P3 (B0H), IP (B8H), PSW (D0H), A (E0H), B (F0H).

Кожен з бітів, що адресуються окремо, може бути встановлені у “лог. 1”, скинуті у “лог. 0”, інвертовані та перевірені. Можуть бути реалізовані переходи: якщо біт встановлено; якщо біт не встановлено; перехід, якщо

біт встановлено, із скиданням цього біта; біт може бути перезаписано у (із) розряду перенесення. Між будь–яким бітом, що адресується прямо, і прапорцем перенесення можуть бути виконані логічні операції “І”, “АБО”, а результат заноситься у прапорець перенесення. Команди побітової обробки забезпечують реалізацію складних функцій комбінаторної логіки та оптимізацію програм користувача.

8.3.7 Команди передачі керування

До даної групи команд (таблиця 8.16) відносяться команди, що забезпечують умовні і безумовні переходи, виклик підпрограм і повернення з них, а також команда пустої операції NOP.

Асемблер припускає використання узагальненого імені команд JMP і CALL, які в процесі трансляції заміняються оптимальними за форматом командами переходу (AJMP, SJMP, LJMP) або виклику (ACALL, LCALL). У більшості команд використовується пряма адресація, тобто адреса переходу цілком (або її частина) знаходиться у самій команді передачі керування. Можна виділити три різновиди команд переходів за вказаною розрядністю адреси переходу.

Довгий перехід. Це перехід по всьому адресному просторі ПП. В команді міститься повна 16–бітна адреса переходу ad16. Трьохбайтні команди довгого переходу містять у мнемоніці букву L (Long). Всього існує дві такі команди: LJMP – довгий перехід і LCALL – довгий виклик підпрограми. На практиці рідко виникає необхідність переходу в межах всього адресного простору і частіше використовуються скорочені команди переходу, що займають менше місця в пам’яті.

Таблиця 8.16 – Група команд передачі керування

№	Назва команди	Мнемокод	КОП	Т	Б	Ц	Операція
1	Довгий перехід у повному обсязі пам'яті програм	**LJMP ad16**	00000010	12	3	2	**(PC) ← ad16**
2	Абсолютний перехід всередині сторінки у 2 Кбайт	**AJMP ad11**	$a_{10}a_9a_8$00001	6	2	2	**(PC) ← (PC) + 2** **(PC_{10-0}) ← ad11**
3	Короткий відносний перехід всередині сторінки у 256 байт	**SJMP rel**	10000000	5	2	2	**(PC) ← (PC) + 2** **(PC) ← (PC) + rel**
4	Непрямий перехід	**JMP @A + DPTR**	01110011	1	1	2	**(PC) ← (A) + (DPTR)**
5	Перехід, якщо акумулятор дорівнює нулю	**JZ rel**	01100000	5	2	2	**(PC) ← (PC) + 2, якщо (A) = 0,** **тоді (PC) ← (PC) + rel**
6	Перехід, якщо акумулятор не дорівнює нулю	**JNZ rel**	01110000	5	2	2	**(PC) ← (PC) + 2, якщо (A) ≠ 0,** **тоді (PC) ← (PC) + rel**
7	Перехід, якщо перенесення дорівнює одиниці	**JC rel**	01000000	5	2	2	**(PC) ← (PC) + 2, якщо (C) = 1,** **тоді (PC) ← (PC) + rel**
8	Перехід, якщо перенесення дорівнює нулю	**JNC rel**	01010000	5	2	2	**(PC) ← (PC) + 2, якщо (C) = 0,** **тоді (PC) ← (PC) + rel**
9	Перехід, якщо прямоадресуємий біт дорівнює одиниці	**JB bit, rel**	00100000	11	3	2	**(PC) ← (PC) + 3, якщо (b) = 1,** **тоді (PC) ← (PC) + rel**
10	Перехід, якщо прямоадресуємий біт дорівнює нулю	**JNB bit, rel**	00110000	11	3	2	**(PC) ← (PC) + 3, якщо (b) = 0,** **тоді (PC) ← (PC) + rel**
11	Перехід, якщо прямоадресуємий біт встановлено, з наступним скиданням біта	**JBC bit, rel**	00010000	11	3	2	**(PC) ← (PC) + 3, якщо (b) = 1,** **тоді (b) ← 0 и (PC) ← (PC) + rel**
12	Декремент Rn і перехід, якщо не нуль	**DJNZ Rn, rel**	11011rrr	5	2	2	**(PC) ← (PC) + 2, (Rn) ← (Rn) –1,** **якщо (Rn) ≠ 0,то (PC)←(PC) +rel**
13	Декремент прямоадресуємого байта і перехід, якщо не нуль	**DJNZ ad, rel**	11010101	8	3	2	**(PC) ← (PC) + 3, (ad) ← (ad) –1,** **якщо (ad) ≠ 0,то (PC) ←(PC) +rel**
14	Порівняння акумулятора з прямоадресуємим байтом і перехід, якщо не дорівнює	**CJNE A, ad, rel**	10110101	8	3	2	**(PC) ← (PC) + 3, якщо (A) ≠ (ad),** **то (PC)←(PC)+rel,якщо(A)< (ad),** **то (C) ←1, інакше (C) ← 0**

Продовження таблиці 8.16

№	Назва команди	Мнемокод	КОП	Т	Б	Ц	Операція
15	Порівняння акумулятора з константою і перехід, якщо не дорівнює	**CJNE A, #D8, rel**	10110100	10	3	2	**(PC) ← (PC) + 3, якщо (A) ≠ D8, то (PC)←(PC)+ rel,якщо(A) < D8, то (C) ← 1, інакше (C) ← 0**
16	Порівняння регістра з константою і перехід, якщо не дорівнює	**CJNE Rn, #D8, rel**	10111rrr	10	3	2	**(PC) ← (PC) + 3, якщо (Rn) ≠ D8, то (PC)←(PC)+rel,якщо(Rn)< D8, то (C) ← 1, інакше (C) ← 0**
17	Порівняння байта в РПД з константою і перехід, якщо не дорівнює	**CJNE @Ri, #D8,rel**	1011011i	10	3	2	**(PC) ← (PC) + 3, якщо РПД(Ri) ≠ D8, то (PC) ← (PC) + rel, якщо РПД(Ri) < D8, то (C) ←1, інакше (C) ← 0**
18	Довгий виклик підпрограми	**LCALL ad16**	00010010	12	3	2	**(PC) ← (PC) +3, (SP) ← (SP) + 1, РПД(SP)←(PC_{7-0}), (SP) ← (SP)+ 1, РПД(SP) ← (PC_{8-15}), (PC) ← ad16**
19	Абсолютний виклик підпрограми в межах сторінки у 2 Кбайт	**ACALL ad11**	$a_{10}a_9a_8$10001	6	2	2	**(PC) ← (PC) +2, (SP) ← (SP) + 1, РПД(SP) ← (PC_{7-0}), (SP)← (SP)+ 1, РПД(SP) ← (PC_{15-8}), (PC) ← ad11**
20	Повернення з підпрограми	**RET**	00100010	1	1	2	**(PC_{15-8}) ←РПД(SP),(SP)←(SP)–1, (PC_{7-0}) ←РПД(SP),(SP) ← (SP)–1**
21	Повернення з підпрограми обробки переривання	**RETI**	00110010	1	1	2	**(PC_{15-8}) ←РПД(SP),(SP)←(SP)–1, (PC_{7-0}) ← РПД(SP),(SP) ←(SP)–1**
22	Холоста команда	**NOP**	00000000	1	1	1	**(PC) ← (PC) + 1**

Абсолютний перехід. Це перехід в межах однієї сторінки пам'яті програм розміром 2048байт. Такі команди містять тільки 11 молодших біт адреси переходу ad11. Команди абсолютного переходу мають формат 2 байти. Початкова буква мнемоніки – A (Absolute). При виконанні команди в обчисленій адресі наступної по порядку команди: (PC) = (PC) + 2) одинадцять молодших біт замінюються на ad11 із тіла команди абсолютного переходу. Тобто, відбуватиметься перехід у поточній сторінці, номер якої визначається п'ятьма старшими розрядами адреси наступної команди. Очевидно, що число сторінок дорівнює 2^5=32.

Відносний перехід. Це короткий відносний перехід дозволяє передати керування в межах від –128 до +127 байт щодо адреси наступної команди (команди, що іде наступною по порядку за командою відносного переходу). Існує одна команда безумовного короткого відносного переходу SJMP (Short). Всі команди умовного переходу використовують даний метод адресації. Відносна адреса переходу позначається rel, яке міститься у другому байті команди.

Непрямий перехід. Команда JMP @A+DPTR дозволяє передавати керування за непрямою адресою. Ця команда зручна тим, що надає можливість організації переходу за адресою, що обчислюється самою програмою і невідома при написанні вихідного тексту програми.

Умовні переходи. Розвинута система умовних переходів надає можливість здійснювати переходи за такими умовами: вміст акумулятора дорівнює нулю (JZ); вміст акумулятора не дорівнює нулю (JNZ); перенесення дорівнює одиниці (JC); перенесення дорівнює нулю (JNC); біт, що адресується прямо, дорівнює одиниці (JB); біт, що адресується прямо, дорівнює нулю (JNB). Для організації програмних циклів зручно користуватись командою DJNZ. Як лічильник циклів може використовуватись регістр або байт, що адресується прямо (наприклад, комірка РПД). Команда порівняння CJNE ефективно використовується в

процедурах очікування якоїсь події. Наприклад, команда

```
WAIT:  CJNE  A, P0, WAIT
```

буде виконуватись доти, доки на лініях порту 0 не встановиться інформація, що збігається з вмістом акумулятора. Всі команди передачі керування, за винятком CJNE, не впливають на прапорці. Команда CJNE встановлює прапорець C, якщо перший операнд виявляється меншим ніж другий.

Підпрограми. Для звернення до підпрограм необхідно використовувати команди виклику підпрограм (LCALL, ACALL). Ці команди на відміну від команд переходу (LJMP, AJMP) зберігають у стеку адресу повернення в основну програму. Для повернення з підпрограми необхідно виконати команду RET. Команда RETI відрізняється від команди RET тим, що дозволяє переривання з рівнем пріоритету, що обслуговується. Тому цю команду необхідно застосовувати наприкінці підпрограм, що викликані перериваннями.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Назвіть та опишіть логічні команди МП–ра і8086 та МК–ра типу МК51.
2. Поясніть використання логічних команд для: встановлення в1, складання в0, інвертування потрібних бітів операнда–джерела. При цьому інші біти повинні не змінюватись.
3. Назвіть та опишіть команди обробки рядків МП–ра і8086.
4. Назвіть та поясніть команди переходів МП–ра і8086 та МК–ра типу МК51.
5. Назвіть та поясніть команди організації циклів МП–ра і8086.
6. Назвіть та поясніть команди виклику та повернення з підпрограм МП–ра і8086 та МК–ра типу МК51.
7. Назвіть та опишіть команди роботи з бітами МК–ра типу МК51.
8. Які чотири булеві функції реалізуються за допомогою логічних команд?
9. Опишіть особливості виконання зсувів: логічного, циклічного, арифметичного.
10. Для чого можуть застосовуватись команди логічного і арифметичного зсувів?
11. Яку дію виконує команда CMPS?
12. У якому випадку використовуються четвертий і п’ятий формати JMP у мікропроцесорі і8086?
13. Які мнемоніки включають команди програмних переривань мікропроцесора і8086?
14. Опишіть два способи виведення мікропроцесора із стану очікування.

ЛІТЕРАТУРА [2, 5, 10 … 12, 14, 15, 16, 18, 29, 39]

ПРЕДМЕТНИЙ ПОКАЗЧИК

СПИСОК ЛІТЕРАТУРИ

1. Гилмор Ч. Введение в микропроцессорную технику/ Пер. с англ.–М.: Энергоатомиздат, 1984.
2. К.Г. Самофалов, О.В. Викторов, А.К. Кузняк. Микропроцессоры. – К.: Техшка, 1986.
3. Кабалеков Б.А. Микропроцессоры и их применение в системах передачи и обработки сигналов. Учеб. пособие для вузов. – М.: Радио и связь, 1988.
4. Кабалеков Б.А. Цифровые устройства и микропроцессорные системы. – М.: Телеком, 2000.
5. Микропроцессорный комплект К1810. Структура, программирование, применение. Справочная книга. Под ред. Ю.М. Казаринова. – М.: Высш. шк., 1990.
6. Микропроцессорные ситемы. Учебное пособие для вузов. Б. К. Александров та другие. –СПб.: Политехника, 2002.
7. Микропроцессоры: в 3–х книгах. Учебник для вузов. П.В.Нестеров и др., под ред. Л.Н. Преснухина. – М.: Высшая шк., 1986.
8. Микро–ЭВМ/ Пер. с англ., под ред. А. Дирксена. – М.: Энергоатомиздат, 1982.
9. Р. Токхайм. Микропроцессоры. Курс и упражнения/ Пер. с англ.–М.: Энергоатомидат, 1988.
10. Злобин В.К., Григорьев В.Л. Программирование арифметических операций в микропроцессорах. Учеб. Пособие для тех. Вузов. – М.: Высш. шк., 1991.
11. Дао Л. Программирование микропроцессора 8088. Пер. с англ. – М.:Мир, 1988.
12. Башков Е.А. Аппаратное и программное обеспечение зарубежных микро–ЭВМ. Учеб. Пособие. –К.: Высш. шк., 1991.

13. Проектирование микропроцессорной электронно–вычислительной аппаратуры: Справочник/В.Г.Артюхов. А.А.Будняк и др. – К.: Техніка, 1988.
14. Лихтциндер Б.Я., Кузнецов В.Н. Микропроцессоры и вычислительные устройства в радиотехнике. – К.: Вища шк., 1988.
15. В.В. Сташин и др. Проектирование цифровых устройств на однокристальных микроконтроллерах. – М.: Энергоатомиздат,1990.
16. Боборыкин А.В. и др. Однокристальные микро–ЭВМ. –М.: “МИКАП”, 1994.
17. Микроконтроллеры. Выпуск 2: однокристальные микроконтроллеры PIC16x8x и др. Перевод с англ. Б.Я. Прохоренко.– М.:ДОДЭКА, 2000.
18. А.О.Новацький, П.М.Повідайко. Організація та застосування однокристальної мікроЕОМ МК51: Навчальний посібник. – Житомир, 2001.
19. Хвощ С.Т. и др. Микропроцессоры и микро–ЭВМ в системах автоматического управления. Справочник. – Л.: Машиностроение, 1987.
20. Клингман Э. Проектирование микропроцессорных систем. Пер. с англ. – М.: Мир, 1990.
21. Артвик Б.А. Сопряжение микро–ЭВМ с внешними устройствами. Пер. с англ. – М.: Машиностроение, 1983.
22. Вершинин О.Е. Применение микропроцессоров для автоматизации технологических процессов. — Л.: Энергоатомиздат, 1986.
23. Вуд М.А. Микропроцессоры в вопросах и ответах. – М.: Энергоатомидат, 1990.
24. М. Тули. Справочное пособие по цифровой электронике. – .:Энергоатомидат, 1990.

25. Алексенко А.Г. и др. Проектирование радиоэлектронной аппаратуры. – М.: Радио и связь, 1984.
26. В.Л. Горбунов, Д.И. Панфилов. Микропроцессоры. Лабораторный практикум. – М.: Высш. шк., 1984.
27. В.А. Кравец, А.Я. Шпильберг. Зарубежные ЭВМ. – Харьков1991.
28. Гольденберг Л.М. и др. Цифровые устройства и микропроцессорные системы. Задачи и упражнения. Учеб. пособие для вузов. – М.: Радио и связь, 1992.
29. Методические указания к выполнению курсовых проектов по дисциплине «ППМПС» , раздел «Применение микропроцессора К 1810 ВМ86».–К.: КПИ, 1992.
30. Методические указания к лабораторным работам по курсу «ППМПС».–К: КПИ, 2002.
31. Методические указания к выполнению курсовых проектов по дисциплине “ Проектирование и применение микропроцессорных систем ”. Раздел “ Проектирование модулей микропроцессора и параллельного ввода–вывода ”.–К.: КПИ, 1990.
32. Методические указания к выполнению курсовых проектов по дисциплине “ Проектирование модулей последовательного ввода–вывода и формирования временных интервалов (таймера) ”. – К..–КПИ. 1992.
33. Методические указания к выполнению курсовых проектов по дисциплине “ Проектирование и применение микропроцессорных систем ”. Раздел “ Проектирование контроллеров прерываний и прямого доступа к памяти ”. – К.: КПИ, 1992.
34. Методические указания к выполнению курсовых проектов по дисциплине “ Проектирование и применение микропроцессорных систем ” (задания на КП, требования к объему, указания к оформлению, рекомендации по проектированию типовой

гипотетической МПС контроля, управления и отображения).–К.: КПИ, 1991.
35. Федорков Б.Г. Телец В.А. Микросхемы ЦАП и АЦП. – Л.:Энергоатомиздат, 1990.
36. Зубков С.В. "Assembler. Для DOS, Windows и UNIX" 1999.
37. Фролов Ф.В., Фролов Г.В. "Аппаратное обеспечение персонального компьютера" БСП. том 33. 1998.
38. Рудаков П.И., Финогенов К.Г. "Программируем на языке ассемблера IBM PC", 2–е издание, 1997.
39. Том Сван "Освоение Turbo Assembler" 2–е издание, 1996.