МІНІСТЕРСТВО ОСВІТИ І НАУКІ УКРАЇНИ

Національний авіаційний університет

“МІКРОПРОЦЕСОРИ В СИСТЕМАХ ТЕХНІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ”
(Архітектура і система команд)

Частина 1

Навчльний посібник

Київ 2013

УДК 681. 325. 5-181 4(076. 5)
ББК З 973. 26Я73-1
Ш 752

В. А. Швець, Т. В. Мелешко, В. В. Шестакова

Рецензент д.т.н. професор В. П. Квасніков

Затверджено на засіданні науково-методично-редакційною радою Інституту інформаційно-діагностичних систем 1 листопада 2012 року.

Ш752 Мікропроцесорні пристрої в системах захисту інформації. Навчальний посібник / В. А.Швець, Т. В. Мелешко, В. В. Шестакова – К.:НАУ, 2013. - 221 с.

Навчальний посібник складений відповідно до програми курсу, містить у собі опис мікропроцесорів, КР1810 і К1816 (архітектуру, принцип роботи, повну систему команд).

Призначений для студентів спеціальності 6.170102 "Системи технічного захисту інформації"

ББК З 973.26Я

Поява ЕОМ не відзначалася сполохами ядерних вибухів. Про це довго не писалося в газетах. Настання нової ери проходило в тиші сильно засекречених дослідницьких лабораторій. І навіть творці перших ЕОМ не віддавали собі звіту в значенні того, що вони робили.
Н. Н. Моісеєв, академік.

Це маленьке чудо, і я вважаю, що мікропроцесор зробить величезний вплив на розвиток обчислювальної техніки. По суті, ми тепер починаємо на усе дивитися іншими очима.
Д. Слотник,
творець мережі ARPA
і суперобчислювача ILLIAC-4.

1. ІСТОРІЯ РОЗВИТКУ МІКРОПРОЦЕСОРІВ

Приблизно за 25 років до того, як корпорація Intel випустила перший у світі функціонально завершений однокристальний мікропроцесор (МП) 4004, поклавши тим самим початок мікропроцесорному буму, відбулася подія, що радикально змінила наш світ, (був створений транзистор (1946 р.).

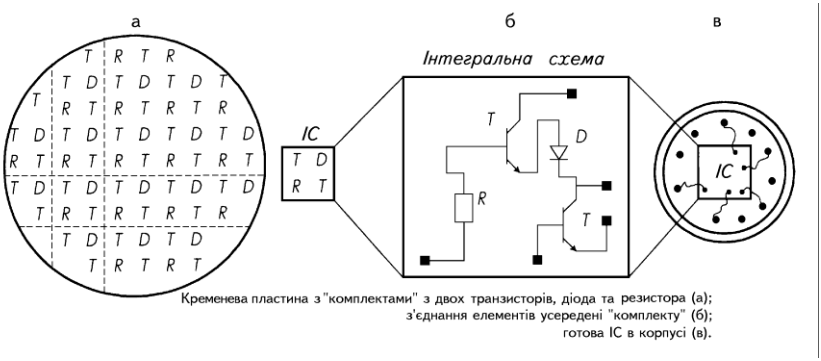
Транзистор є невід'ємною частиною будь-якої мікросхеми, а також МП.

Можливість інтеграції декількох напівпровідникових елементів на одній подложці дала можливість створенню малих інтегральних схем. Ці елементи з'єднувалися між собою за допомогою напилювання металевих смужок (рис. 1.1).

Далі пластина розрізалася на окремі кристали, що поміщалися в корпус. У результаті утворювався готовий

функціональний вузол. Перші мікросхеми містили від одиниць до декількох десятків елементів. Це були малі інтегральні схеми. Дуже швидко з'явилися прилади середнього ступеня інтеграції 100-1000 елементів.

Протягом 80-х років замінилося два покоління мікросхем: великі – до 10000 елементів на кристал і надвеликі – до 100 тисяч елементів. На початку 90-х ультра великі мікросхеми містили до 1 млн. транзисторів, а МП Pentium Pro 200 MHz - 5,5 млн. транзисторів. Кількість елементів на однім кристалі може досягти по над 10 млн. транзисторів.



Кременева пластина з "комплектами" з двох транзисторів, діода та резистора (а); з'єднання елементів усередені "комплекту" (б); готова IC в корпусі (в).

Рис. 1.1.

Технологія великих інтегральних схем дозволила одержувати дуже значні арифметико-логічні пристрої, або компоненти, наприклад мікропроцесори.

Історія створила всі передумови для появи МП. Довго чекати не довелося.

33 рокі тому компанія Intel почала випуск сімейства інтегральних мікросхем 4004 (комплект Intel MCS-4), призначених для побудови калькуляторів, (рис. 1.2).

Це були перші обчислювальні прилади, що забезпечували паралельну обробку під мікропрограмним керуванням. До складу сімейства входив 4-розрядний мікропроцесор із усіма властивими йому гідностями: універсальність, гнучкість, мініатюрність, мікропотужність, низька вартість.

Такий прилад виконував 18 команд, мав період машинного циклу 10,8 мкс при роботі зі стандартною двофазною системою синхроімпульсів. Адресний простір обмежувався 4 КВ постійної пам'яті (ROM), 1 КВ оперативної пам'яті (RAM) і 128 каналами вводу-виводу (I/O).

Мікропроцесор виник не на порожньому місці, його появі передувало десятиліття розвитку мікроелектроніки, що до початку 70-х років домоглася істотних успіхів у створенні інтегральних мікросхем. І все ж вихід 4004-го став революційною подією, що призвела до корінного зламу уявлень про можливості мікроелектронної та обчислювальної техніки. Обчислювальна техніка вступила в бурхливу мікропроцесорну еру, що характеризується гонкою технологій і високою швидкістю удосконалювання технічного рівня виробів. Переваги мікропроцесорів були миттєво оцінені, і буквально в лічені місяці до випуску аналогічних ВІС приступили інші компанії.

Питання про пріоритет створення першого у світі мікропроцесора вже до 1974 р. цілком прояснилося. От як писав про це часопис "Electronics" у 1974 р.: "Intel достеменно належить честь використання ідеї мікропроцесора, і вона була першою, хто випустив їх на ринок, хоча великі заслуги належать також іншим фірмам і окремим особам, що тим або іншим шляхом сприяли розвитку техніки великих інтегральних схем. Варто пригадати компанію Viatron, що у 1968 р. здивувала світ, повідомивши про свій намір створити систему обробки даних на основі власного 8-розрядного мікропроцесора, керованого примітивною програмою, що записана в постійний запам'ятовуючий пристрій (ПЗП). Проте ця компанія зустрілася із серйозними фінансовими труднощами і через два роки збанкрутувала. Тим часом, General E1ectric розробила однокристальний базовий логічний блок і впритул підійшла до створення мікропроцесора".

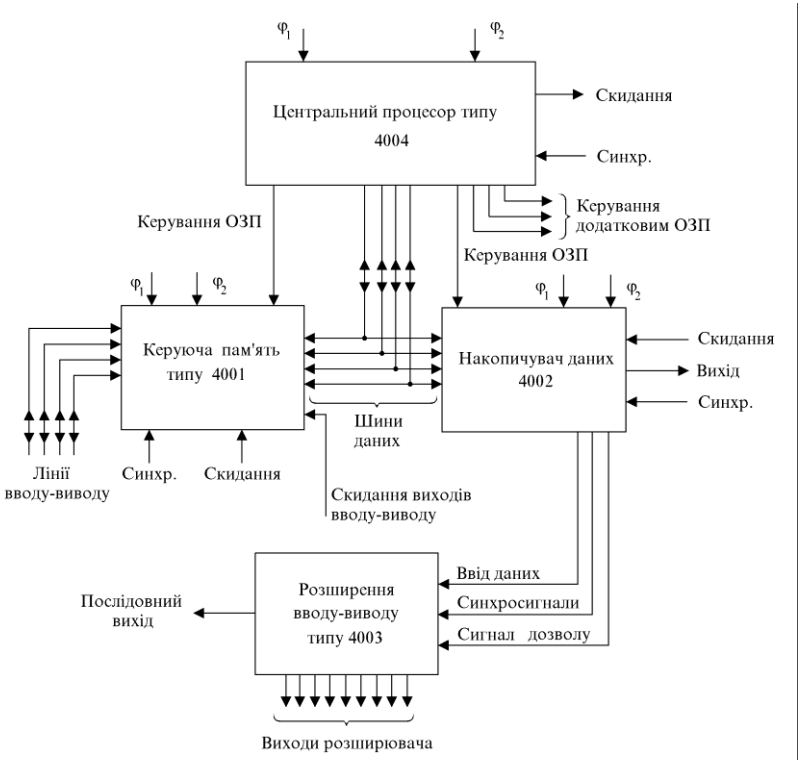


Рис. 1.2.

Вже в 1971 р. Intel не приховувала своїх досягнень. Вона існувала усього чотири роки і була відома мікросхемами пам'яті, коли в гучній рекламній кампанії проголосила "нову еру в мікроелектроніці", випустивши на ринок перший комерційний мікропроцесор. Цього разу реклама виявилася пророцтвом, а не безсоромним обманом покупців, як це іноді трапляється. Один кристал 4004 мав майже такі ж обчислювальні можливості, як і ENIAC випуску 1946 р. із його 18 000 електронних ламп. Мікропроцесор був дешевим (біля $200) і мав малюсінькі розміри, що дозволяло розроблювачам створювати принципово нові системи. Скеп-

тики пророкували, що ринок однокристального процесора буде такий же малюсінький, як і він самий, але дійсність перевершила всі чекання.

Ідея 4004-го виникла в одного з фундаторів компанії Intel Роберта Нойса (Robert Noyce). Він запропонував використовувати технологічний процес виготовлення кристалів пам'яті для створення логічних чипів. Але якими саме повинні бути ці логічні пристрої? Це питання вирішив щасливий випадок. У 1969 р. японська компанія Busicom замовила Intel розробку комплекту з 12 спеціалізованих чипів для настільних калькуляторів. Замість цього Intel запропонувала будувати калькулятори на основі однокристального мікропроцесора, що згодом набув популярності як Intel 4004. Конструкторську розробку виконали Тед Хофф (Ted Hoff), Стен Мейзор (Stan Mazor) і Федеріко Феджин (Federico Faggin). Вони запозичили багато рішень з архітектури великих ЕОМ. Проте, щоб розмістити процесор на однім кристалі, їм довелось знизити його розрядність до чотирьох. Це дозволило зменшити кількість транзисторів і упакувати процесор у 16-контактний корпус - найбільший у той час (для порівняння, сучасний Pentium має 64-розрядну зовнішню шину і поміщається в 296-контактному корпусі). Процесор міг виконувати 45 інструкцій, і для програмованого калькулятора компанії Busicom до комплекту MCS-4 достатньо було додати чотири кристала пам'яті по 256 байт.

Чип 4004 відчинив дорогу більш потужним і швидкодіючим мікропроцесорам. Як тільки розробка надійшла в серійне виробництво, Тед Хофф і Стен Мейзор приступили до створення його 8-розрядної версії - 8008. Це був перший мікропроцесор, що мав систему переривань і мультиплексовану шину адреса-дані, але функціонував він незадовільно. І вже в 1974 р. з'явився, також 8-розрядний, кристал 8080. Він містив 6000 транзисторів, виконував більш 250 команд, причому деякі з них працювали з парами регістрів - із 16-

розрядними даними. Його адресний простір 64 КВ по тим часам був величезним (сучасний Pentium Pro може адресуватися до 64 ТВ віртуальної пам'яті).

З виходом 8-розрядних чипів Intel відчинилися найширші перспективи для розробки дешевого програмованого устаткування. І події не змусили себе чекати. Незабаром Гарі Кілделл (Gary Kildall) із Digital Research створив для процесора 8080 операційну систему CP/M. Цей software значно спростив процес розробки і налагодження прикладного програмного забезпечення і, разом із чипом 8080, фактично відчинив небачені раніше можливості створення доступних масовому споживачу комп'ютерів.

Першим персональним комп'ютером прийнято вважати мікроЕОМ Altair 8800 компанії MITS, що надійшла в продаж у 1975 р. і вже була оснащена операційною системою CP/M. Машина випускалася у вигляді набору "Зроби сам" або як закінчений виріб і призначалася для любителів складної технічної іграшки. Надалі з'явився розширений варіант пристрою для бухгалтерських і інженерних розрахунків.

Багато компаній із середини 70-х років приступили до виробництва особистих, домашніх, любительських мікроЕОМ, але головний успіх дістався компанії Apple, яка випустила в 1977 р. однойменну ПЕОМ, що відрізнялася зробленим програмним забезпеченням, розрахованим на самого непідготовленого користувача. Дружнє програмне забезпечення (friendly software) розширило коло потенційних покупців, обмежене раніше любителями електроніки і професіоналами.

Мікропроцесорна гонка набирала усе більше прискорення. Вже згадуваний Федеріко Феджин і Масатоши Шима (Masatoshi Shima) заснували компанію Zilog, що блискавично налагодила випуск чипа Z80 - декілька удосконаленого аналога 8080, сумісного і з ним, і з операційною системою

CP/M. мікропроцесор Z80 мав умонтовану програму регенерації динамічного ОЗУ і разом із CP/M склав "ідеальну пару" для масового виробництва дешевих ПК.

Незабаром дебютувала Motorola із своїм 6800-м, стартували Texas Instruments, National Semiconductor і Fairchild, розробивши свої власні мікропроцесори; більшість із них використовувалося для вбудовування в "інтелектуальні" прилади.

Нарешті, на новий і надзвичайно динамічний сектор комп'ютерного ринку звернула увагу корпорація IBM - гігант обчислювальної індустрії. Випустивши в 1981 р. свій перший IBM PC, вона відразу стала законодавцем мод для виробників персональних комп'ютерів і понині займає лідируюче місце на всесвітньому комп'ютерному ринку.

Удосконалений 16-розрядний чип з'явився в 1978 р. одночасно в "двох особах" - 8086 і 8088. Перший чип містив 29000 транзисторів, більш розвинену систему команд, а також мав у десять разів більшу продуктивність, чим його 8-розрядний попередник. Суворо дотримуючись принципу програмної сумісності, Intel зберіг 16-розрядний адресний регістр, властивий чипу 8080, але для розширення адресного простору до 1 MB додав сегментний регістр, тим самим створивши нову ідеологію адресації, що застосовується і сьогодні.

Ці близнюки знаходили світову популярність завдяки обчислювальним системам класу Extended Technology - знаменитим PC XT компанії IBM із прекрасним DOS-ом і незабутньою оболонкою Norton Commander.

Поки покупці були зайняті конструюванням потужних прикладних систем, що базуються на мікропроцесорі 8086, фірма Intel розробила й представила на ринок мікропроцесор 8088 (різновид 8086 із 8-бітною шиною даних) і математичний сопроцесор 8087. У 1981 р., коли фірма IBM вибрала мікропроцесор 8088 для свого персонального ком-

п'ютера, архітектура 8086/8088 перетворилася в промисловий стандарт для персональних комп'ютерів.

Спираючись на архітектуру 8086 і з огляду на запити ринку, фірма Intel розробила два мікропроцесори - 80186 і 80286. Мікропроцесор 80186, який відрізнявся високим ступенем інтеграції, був розроблений для ринку умонтованих керуючих систем.

Мікропроцесор 80286 призначався для ринку персональних комп'ютерів і робочих станцій, де були потрібні програмна сумісність і висока продуктивність. У високопродуктивному мікропроцесорі 80286 уперше були мікросхемно реалізовані такі сучасні можливості, як багатозадачність і керування віртуальною пам'яттю. Для нього були розроблені такі периферійні компоненти, як математичний сопроцесор lntel287, мікросхеми вводу-виводу, контролер DMA та інші периферійні пристрої системи.

У 1985 р. фірма Intel представила 32-бітний мікропроцесор Intel386. Він швидко завоював місце на ринку завдяки своїй високій продуктивності, завершеній 32-бітній архітектурі і підтримці сторінкової віртуальної пам'яті для сучасних операційних систем. Крім того, у ньому забезпечена сумісність по об'єктному коду для більшості програм, написаних для процесорів з архітектурою 8086 і 80286. Периферійні компоненти для Intel386 оптимізовані на досягнення високої продуктивності 32-бітних систем і містять у собі такі основні 32-бітні периферійні пристрої, як контролер DMA 82380, контролери кеш-пам'яті 82385, 82395 і математичний сопроцесор Intel387. Кожне нове покоління архітектури, що розвивається фірмою Intel, забезпечує значний зріст продуктивності.

У 1988 являє фірма Intel запропонувала мікропроцесор Intel386SX, що подає собою здешевлений варіант популярного ЦП Intel386 із 16-бітною шиною даних. Він виконує всі 32-бітні програми, написані для ЦП Intel386.

До кінця 1988 р. мікропроцесор Intel386SX був добре прийнятий широким колом користувачів.

У 1989 р. фірма Intel представила мікропроцесор Intel486DX. Він об'єднує 1,2 мільйона транзисторів, має продуктивність, що вдвічі перевищує продуктивність ЦП Intel386, і володіє 100%-ою сумісністю зі старими виробами сімейства Intel386.

У 1990 р. фірмою Intel був поданий перший член сімейства архітектури Intel386 із високим ступенем інтеграції - мікропроцесор Intel386SL. При об'єднанні ЦП Intel386SL з однокристальною мікросхемою 82360SL, що задовольняє стандарту ISA, забезпечується мале споживання енергії і габарити, необхідні в портативних персональних комп'ютерах. Водночас забезпечується повна сумісність із програмним забезпеченням попереднього покоління.

У 1991 р. фірма Intel представила мікропроцесор Intel486SX. Цей процесор забезпечує перехід на технологію Intel486 для малих офісних систем. У 1991 р. фірма Intel також представила мікропроцесор Intel486DX із робочою частотою 50 МГц, що підвищив продуктивність сімейства Intel486DX на 50 відсотків.

У 1992 р. фірма Intel представила мікропроцесор Intel 486DX2, що дозволив створювати високопродуктивні системи в настільному виконанні. Мікропроцесор Intel486DX2 використовує технологію "подвоєння швидкості", що забезпечує зручність конструювання і 100 %-у сумісність.

У 1993 р. Intel випустила процесор п'ятого покоління Pentium, із якого почалася нова ера розвитку мікропроцесорів.

2. МІКРОПРОЦЕСОР КР1810ВМ86 (І8086)

2.1 Призначення і функціональні особливості

Мікросхема К1810ВМ86 являє собою однокристальний високопродуктивний 16-розрядний мікропроцесор, виконаний по *n*-канальній МОН-технології. Мікросхема розміщена в дворядному 40-вивідному корпусі "Монтаж 2-40". Мікропроцесор (МП), має властивості як 8-, так і 16-розрядного процесора. Він розглядає пам'ять як послідовність 8-розрядних байтів даних, проте для підвищення продуктивності процесор має зв'язок із пам'яттю по 16-розрядній магістралі даних.

Мікропроцесор має можливість адресації до одного Мбайта пам'яті.

Мікропроцесор виконує операції над бітами, байтами, словами (два байти).

Мікропроцесор виконує дії 8- і 16-розрядної, знакової і беззнакової, двійкової або десяткової арифметики, у тому числі множення і ділення.

Мікропроцесор має 14 внутрішніх 16-розрядних регістрів із симетричними операціями.

Можливі 24 режими адресації операндів.

Мікропроцесор має гнучку структуру переривань: апаратних і програмних. Можливо до 256 типів пріоритетних переривань, програми обслуговування переривань викликаються векторним способом.

Центральний процесорний пристрій (ЦПП) МП К1810ВМ86 має високу продуктивність завдяки паралельному виконанню операцій опрацювання й обертання. Суміщення операцій досягається завдяки наявності в МП блока попередньої вибірки команд.

Відмінною рисою МП К1810ВМ86 є можливість вибору одного з двох режимів роботи (мінімального або мак-

симального) шляхом підключення виводу 33 ($MN/\overline{MX}$) або до джерела живлення Ucc, або до виводу *GND*.

Мікропроцесор, що працює в мінімальному режимі, використовується в системах, що мають не складну конфігурацію, сам виробляє всі необхідні сигнали керування периферійними пристроями.

Якщо вивід ($MN/\overline{MX}$) залучений до корпуса, то МП К1810ВМ86 перевизначає виводи 24 - 31 для роботи в максимальному режимі і застосовується в складі системи складної конфігурації. При цьому він виробляє сигнали $\overline{ST0},\overline{ST1},\overline{ST2}$, у яких закодована інформація про виконуваний центральним процесорним пристроєм цикл каналу, декодує ці сигнали і виробляє усі керуючі сигнали.

Завдяки наявності вмонтованої схеми керування доступом до магістралі в МП, що працює в мінімальному режимі, можливий прямий доступ до пам'яті. Зовнішній контролер прямого доступу до пам'яті може направити запит на вхід ЦПП на використання каналу для прямої передачі даних з пристрою вводу-виводу (ПВВ) у пам'ять. При одержанні сигналу захоплення *HLD*, ЦПП закінчує поточний цикл каналу і видає сигнал підтвердження захоплення *HLDA* надаючи контролеру прямого доступу свій канал.

Архітектурною особливістю ЦПП К1810ВМ86 є можливість координувати взаємодію декількох процесорів, що спрощує побудову на його базі мультипроцесорних систем. У цих системах можливо застосування двох типів процесорів: незалежних, тобто таких що виконують власний потік команд і допоміжних процесорів (сопроцесори). Сопроцесор переглядає команди, що вибираються головним процесором, визнає деякі з них "своїми" і виконує їх. Мікросхему К1810ВМ86 можна використовувати в якості незалежного процесора, а в ролі сопроцесора можуть виступати спеціальні процесори вводу/виводу, опрацювання даних та ін.

2.2 Призначення виводів мікропроцесора

Призначення виводів МП у мінімальному і максимальному режимах наведене в табл.2.1, розташування виводів показане на рис.2.1.

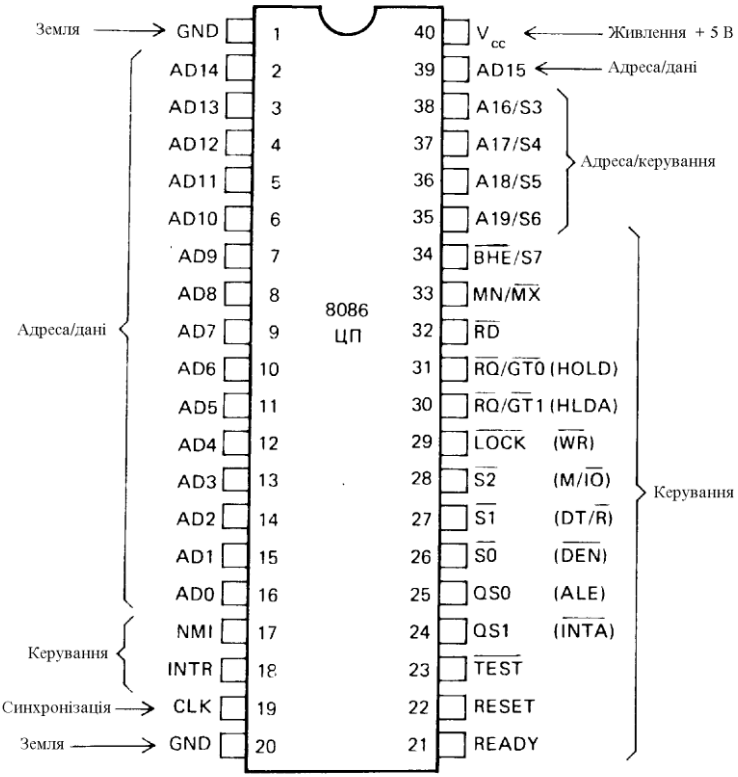


Рис. 2.1

Функції виводів, характерні тільки для максимального режиму наведені в табл. 2.2, для мінімального - у табл.2.3.

Таблиця 2.1

№ виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
2-16, 39	*A/D0-A/D15*	Вхід / вихід із трьома станами	Канал адреси/даних. Ці виводи утворять мультиплексований канал адреси пам'яті або виводу/вводу (у такті T1) і даних (у тактах T2,T3,Tw, T4). Сигнал A0 схожий на сигнал $\overline{BHE}$, але призначений для вибору молодшого байта каналу даних (вводи *D7 - D0*). Коли при операціях із пам'яттю або УВВ пересилається байт даних по молодшій половині каналу даних, 8-розрядні УВВ використовують сигнал A0 як умову вибору пристрою. Ці сигнали при підтвердженні переривання і підтвердженні захоплення знаходяться в третьому стані.
35 - 38	*A19/ ST6, A18/ ST5, A17/ ST4, A16/ ST3.*	Вихід із трьома станами.	Канал адреси/стани. У такті T1 - це чотири старших розряди адреси пам'яті. При роботі з УВВ на ці виходи видаються сигнали низького рівня. При роботі з ЗУ й УВ у тактах T2, T3, Tw і T4 на цих виходах присутня інформація про стан. Стан тригера дозволу переривання (*ST5*) обновляється в кожному такті. Сигнали *ST4*, *ST3* вказують, який регістр використовується для звертання до даних. У третій стан виводи переводяться при підтвердженні захоплення.
34	$\overline{BHE}$ / $\overline{ST\,7}$	Вихід із трьома станами	Дозвіл передачі по старшій половині каналу A/D/Стан. У такті T1 сигнал $\overline{BHE}$ повинний вико-

Таблиця 2.1(продовження)

№ виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
			ристовуватися для дозволу передачі даних по старших розрядах каналу даних (*D15-D8*), 8-розрядного пристрою залучені до старшої половини каналу звичайно використовують цей сигнал як умову вибору пристрою. Коли байт даних повинний передаватися по старшій половині каналу, то в циклах читання, запису і підтвердження переривання в такті Т1 видається сигнал $\overline{BHE}$ низького рівня. У тактах Т2,Т3 і Т4 видається сигнал стана *ST1*. При захопленні сигнал знаходиться в третьому стані.
32	$\overline{R}$	Вихід з трьома станами	Читання. Строб читання вказує, що процесор виконує цикл читання ЗУ або УВВ, у залежності від стана виводу $\overline{ST\ 2}$. Сигнал використовується для читання пристроїв, залучених до локального каналу МП К1810ВМ86. Активний сигнал $\overline{R}$ низького рівня видається в тактах Т2,Т3 і Tw кожного циклу читання. Якщо локальний канал знаходиться у високоімпедансному стані, у такті Т2 видається сигнал $\overline{R}$ високого рівня. Сигнал $\overline{R}$ при підтвердженні захоплення знаходиться в третьому стані.
22	*RDY*	Вхід	Готовність. Сигнал служить підтвердженням того, що ЗП, що адресується, або ПВВ закінчить передачу даних у поточному

Таблиця 2.1(продовження)

№ виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
			циклі. Сигнал активний у стані високого рівня. Вхід *RDY* не синхронізований.
18	*INT*	Вхід	Запит переривання. Потенційний вхід із запам'ятовуванням запиту на внутрішньому тригері, що опитується в останньому такті кожної команди з метою визначення необхідності виконання циклу підтвердження переривання. Адреса програми обслуговування переривання визначається по вектору з таблиці, розташованої в області системної пам'яті. Переривання може бути маскувано програмою установки тригера дозволу переривання. Вхід *INT* внутрішньо синхронізований.
23	$\overline{TEST}$	Вхід	Перевірка. Стан вхідного сигналу опитується при команді чекання *WAIT*. Якщо на вході $\overline{TEST}$ сигнал низького рівня, то процесор продовжує опрацювання, у противному випадку процесор очікує в "холостому" стані. Вхід внутрішньо синхронізований за переднім фронтом кожного імпульсу *CLC*.
21	*CLR (RESET)*	Вхід	Установка. Сигнал служить для установки внутрішніх схем процесора в початковий стан. При подачі на вхід *CLR* сигналу високого рівня процесор негайно припиняє свої дії. Сигнал повинний підтримуватися на виході не меньш ніж 4 такти. Робота процесора відновлюється коли сиг-

Таблиця 2.1(закінчення)

№ виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
			нал знімається з входу *CLR*. Сигнал на виході *CLR* внутрішньо синхронізований.
17	*NMI*	Вхід	Немаскований запит переривання. Імпульсний вхід по якому визивається переривання типу 2. Адреса програми обслуговування переривання визначається зі вектором з таблиці векторів у системній області пам'яті. Позитивний перепад напруги на вході викликає переривання наприкінці поточної команди. Сигнал внутрішньо синхронізований.
19	*CLC*	Вхід	Тактовий імпульс. Сигнали забезпечують синхронізацію процесора.
40	*Ucc*		Напруга джерела живлення.
1, 20	*GND*		Корпус.
33	$NM / \overline{MX}$	Вхід	Режим керування мінімальний / максимальний.

Таблиця 2.2

№ виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
26 – 28	$\overline{ST\ 2}$, $\overline{ST\ 1}$, $\overline{ST\ 0}$	Вихід із трьома станами	Стан циклу каналу. Сигнал видається в тактах Т4,Т1 і Т2 і в такті Т3 або Tw. При наявності сигналу готовності (*RDY*= "1") ці сигнали переходять у пасивний стан (1,1,1). Код стана призначений для зовнішнього контролера каналу, що повинний виробляти всі сигнали керування, обміном інформації з ЗП й ПВВ. Всяка зміна сигналів

Таблиця 2.2(продовження)

№ виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
			стана $\overline{ST\ 2}, \overline{ST\ 1}, \overline{ST\ 0}$ в такті Т4 використовується як указівка про початок циклу каналу, а повернення до пасивного стану вказує на закінчення циклу.
30, 31	$\overline{RQ}/\overline{E0}$, $\overline{RQ}/\overline{E1}$	Вхід/вихід	Запит/дозвіл доступу до магістралі. Виводи використовуються іншими пристроями, щоб процесор примусити звільнити локальний канал наприкінці поточного процесорного циклу каналу. Обидва виводи двонаправленні, причому вивід $\overline{RQ}/\overline{E0}$ має більш високий пріоритет, ніж вивід $\overline{RQ}/\overline{E1}$. Процес запит/дозвіл доступу до магістралі виглядає так: 1. Імпульс тривалість один такт *CLC* від пристрою, який потребує керування локальним каналом, указує процесору на запит локального каналу. 2. У такті Т4 або в холостому такті (Т5) МП К1810ВМ86 видає пристрою, від якого наійшов запит, імпульс тривалістю один такт *CLC* , повідомляючи цим, що він дозволяє переведення локального каналу у високоімпедансний стан і що в наступному такті він ввійде в стан "Підтвердження захоплення". Під час підтвердження захоплення ЦПП логічно відключається від локального каналу. 3. Імпульс тривалістю один такт *CLC* пристрою, від якого на-

Таблица 2.2(продовження)

№ виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
			дійшов запит, що володіє каналом, указує МП К1810ВМ86, що запит "захоплення" підходить до кінця і що МП може повернути собі керування каналом *у* наступному такті CLC. Всяка зміна пристрою, що володіє каналом, являє собою послідовність 3-х імпульсів. Після кожної зміни "власника" каналу повинний бути один холостий такт. Якщо ЦПП при одержанні запиту виконувало цикл звертання до пам'яті, то воно повинно звільнити канал у такті Т4 циклу, коли дотримані наступні умови: а) запит з'явився не пізніше такту Т2; б) поточний цикл не є пересилкою молодшого байта слова (по непарній адресі); в) поточний цикл не є першим циклом підтвердження переривання. 4. Поточна команда не заблокована. Якщо запит з'явився в холостому такті, то можливі два виходи: а) у наступному такті канал буде звільнений; б) не пізніше трьох тактів буде запущений новий цикл пам'яті;
29	$\overline{LOCK}$	Вихід із трьома ста-	Канал зайнятий. Сигнал на цьому виході забороняє зовнішнім

Таблиця 2.2(закінчення)

№ виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
		нами	"власникам" системного каналу захоплювати керування системним каналом, поки сигнал знаходиться в стані низького рівня. Сигнал $\overline{LOCK}$ видається по команді *LOCK* (префікс блокування) і підтримується на виході до кінця наступної команди. При підтвердженні захоплення сигнал знаходиться в третьому стані.
24, 25	*QS0, QS1*	Вихід	Стан черги команд. Код стана вказує на стан черги команд у тому такті, після якого була виконана дія над чергою.

Таблиця 2.3

Номер виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
29	$\overline{W}$	Вихід із трьома станами	Запис. Вказує, що процесор виконує запис у ЗП або в ПВВ, у залежності від стані сигналу $M / \overline{IO}$. Сигнал низького рівня видається в тактах Т2, Т3 і Tw кожного циклу запису. При підтвердженні сигнал $\overline{W}$ знаходиться у високоімпендансному стані.
28	$M / \overline{IO}$	Вихід із трьома станами	Ознака звертання до ЗП або ПВВ. Сигнал є логічним еквівалентом сигналу в максимальному режимі. Використовується для того, щоб відрізнити доступ до ЗУ від доступу

Таблиця 2.3(продовження)

Номер виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
			до ПВВ. Сигнал $M / \overline{IO}$ встановлюється в такті Т4 попереднього циклу каналу і підтримується до завершального такту Т4 попереднього каналу. При підтвердженні захоплення локального каналу сигнал $\overline{RQ} / \overline{EI}$ знаходиться у високоімпедансному стані
27	$OP / \overline{IP}$	Вихід із трьома станами	Видача/прийом даних. Використовується для керування напрямком передачі даних через шинні формувачи. Логічним еквівалентом сигналу $OP / \overline{IP}$ в максимальному режимі є сигнал $\overline{ST1}$. Тимчасова діаграма сигналу $OP / \overline{IP}$ збігається з діаграмою сигналу $M / \overline{IO}$. При підтвердженні захоплення локального каналу сигнал $OP / \overline{IP}$ знаходиться в третьому стані.
26	$\overline{DE}$	Вихід із трьома станами	Дозвіл передачі даних. Використовується в системах мінімальної конфігурації з застосуванням шинних формувачів. Сигнал $\overline{DE}$ низького рівня видається в кожному циклі звертання до ЗП й ПВВ і в циклах підтвердження переривання. У циклах читання та підтвердження переривання видається від середини такту Т2 до середини такту Т4, а в циклі запису – від початку такту Т2 до середини такту Т4. При

Таблиця 2.3(закінчення)

Номер виводу	Позначення виводу	Тип виводу	Найменування і призначення виводу
			підтвердженні сигнал $\overline{DE}$ знаходиться в третьому стані.
25	*STB*	Вихід	Строб адреси. Використовується при записі адреси в регістр адреси. Видається в такті Т1 будь-якого циклу.
24	$\overline{INTA}$	Вихід	Підтвердження переривання. Використовується як строб читання в циклах підтвердження переривання. Сигнал низького рівня видається в тактах Т2,Т3 і Tw кожного циклу підтвердження переривання.
31, 30	*HLD, HLDA*	Вхід, вихід	Захоплення, підтвердження захоплення. Сигнал *HLD* свідчить про те, що інший процесор запитує «захоплення» локального каналу. Процесор, що одержав запит «захоплення», видає сигнал підтвердження *HLDA* у середині такту Т4 або холостого такту Т5. Одночасно з видачею сигналу *HLDA* процесор переводить у третій стан локальний канал і канал керування. Виявивши перехід сигналу *HLD* у стан низького рівня, процесор переводить у стан низького рівня сигнал *HLDA* , а коли процесору буде потрібно виконати інший цикл каналу, він поверне собі керування локальним каналом і каналом керування.

2.3 Структурна організація

2.3.1 Архітектура центрального процесорного пристрою

МП К1810ВМ86 при виконанні програм повинен насамперед одержувати команди з зовнішньої пам'яті, а потім виконувати ці команди, використовуючи свої обчислювальні можливості. Архітектура ЦПП, орієнтована на виконання цих двох основних функцій, відрізняється наявністю двох основних частин – двох незалежних пристроїв (рис. 2.2), що працюють асинхронно.

Пристрій обробки (ПО), призначений для декодування і виконання команд, складається з АЛП на базі комбінаційного 16-розрядного суматора з послідовно-паралельним переносом, трьох тимчасових регістрів для проміжного збереження операндів і результату операції і регістра ознак. Обмін даними здійснюється через регістри загального призначення (РЗП). Функція керування виконанням команд покладена на мікропрограмний пристрій керування який декодує команди і виробляє необхідні сигнали керування.

З метою керування складною структурою переривань у МП запроваджена схема обробки запитів переривань, на яку надходять як зовнішні запити (по виводах *NMI* і *INT*), так і внутрішні, що вироблюються при деяких умовах.

Крім того, є схема керування доступом до магістралі, що дозволяє застосувати процесор у режимі прямого доступу до пам'яті й у мультипроцесорних системах.

Пристрій обробки (ПО), проте, не має безпосереднього зв'язку із "зовнішнім світом". Цей зв'язок здійснює інша група пристроїв, що можна назвати пристроєм сполучення каналу (ПСК).

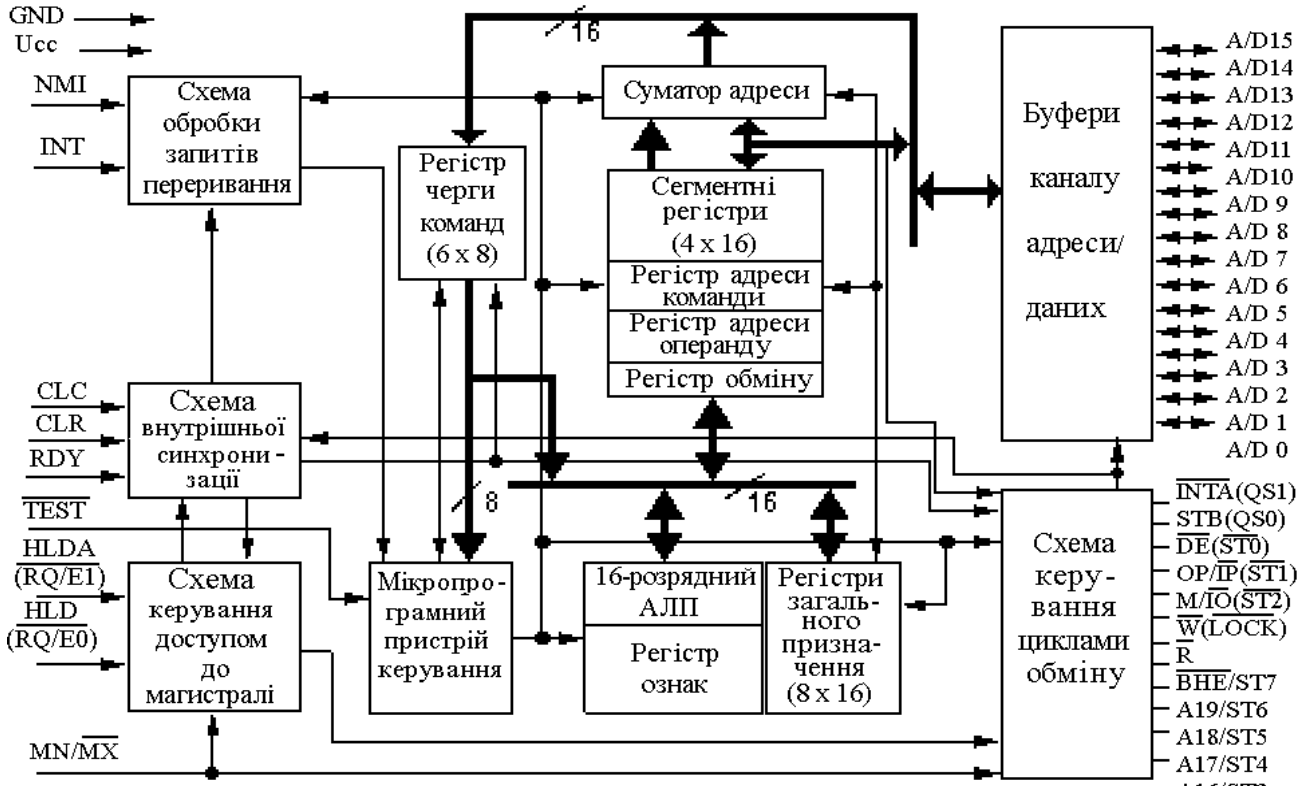

GND
Ucc
NMI
INT
Схема обробки запитів переривання
Регістр черги команд (6 x 8)
Суматор адреси
Сегментні регістри (4 x 16)
Регістр адреси команди
Регістр адреси операнду
Регістр обміну
Буфери каналу адреси/ даних
A/D15
A/D14
A/D13
A/D12
A/D11
A/D10
A/D 9
A/D 8
A/D 7
A/D 6
A/D 5
A/D 4
A/D 3
A/D 2
A/D 1
A/D 0
16
8
16
CLC
CLR
RDY
TEST
Схема внутрішньої синхрони - зації
HLDA
(RQ/E1)
HLD
(RQ/E0)
Схема керування доступом до магістралі
Мікропро - грамний пристрій керування
16-розрядний АЛП
Регістр ознак
Регістри загаль- ного призна- чення (8 x 16)
Схема керу- вання циклами обміну
INTA(QS1)
STB(QS0)
DE(ST0)
OP/IP(ST1)
M/IO(ST2)
W(LOCK)
R
BHE/ST7
A19/ST6
A18/ST5
A17/ST4
A16/ST3
MN/MX

Рис. 2.2

До складу ПСК входить блок обчислення адреси, в якому є спеціальний суматор адреси, сегментні регістри і регістр адреси команди. Суматор адреси призначений для обчислення фізичної адреси команд і даних шляхом додавання ефективної адреси до зміста сегментних регістрів (базової адреси поточного сегмента), у деяких випадках у ньому провадиться додавання до констант. Обрані з пам'яті команди і дані пересилаються в ПО через регістр обміну, що може використовуватися як один 16-розрядний регістр, або як два 8-розрядних регістри.

Схема внутрішньої синхронізації перетворює тактові імпульси, що надходять ззовні, у дві послідовності синхроімпульсів, що неперекриваються, які розподіляються на всі внутрішні пристрої процесора. Крім того, ця схема забезпечує синхронізацію (узгодження швидкостей обміну інформацією) ЦПП з повільно діючими приладами ЗП й ПВВ. Для цієї цілі в мікросхемі передбачений зовнішній вивід *RDY* ("готовність"), на який подаються сигнали з периферійних пристроїв. ЦПП опитує вхід *RDY* у такті Т3 будь-якого машинного циклу. При наявності сигналу *RDY* процесор закінчує машинний цикл, виробляючи такт Т4, але якщо сигнал *RDY* відсутній, то між третім і четвертим тактами вводиться такт чекання, необхідний для успішного завершення поточного циклу обміну з периферією.

Для установки МП у вихідний стан на вхід *CLR* подається сигнал скидання.

Керування функціонуванням 16-розрядного мультиплексованого каналу адреси/даних здійснює схема керування циклами каналу, що виробляє всі необхідні сигнали керування записом або читанням ЗП або записом або читанням ПВВ, підтвердженням переривання й ін.

2.3.2 Пристрій сполучення каналу

Схема попередньої вибірки команд. Розподіл функцій вибірки і виконання команд між двома незалежними частинами ЦПП призводить до розпаралелювання роботи МП і підвищенню його продуктивності. В момент, коли в ПО буде потрібна така команда, вона вже буде знаходиться у внутрішній черзі команд.

Пристрій спряження з каналом прагне виконати цикл вибірки слова з пам'яті всякий раз, коли в черзі звільняється, щонайменше, два байти. Пристрій обробки витягає коди команд із черги в міру потреби.

Попередня черга команд МП К1810ВМ86 складається з трьох 16-розрядних попередніх регістрів команд. Отже, МП може берегти до шести байтів коду команд. Черга організована за принципом: "першим прийшов – першим обслуговуєшся". Шістибайтна глибина попередньої черги дозволяє задовольняти запити пристрою обробки в кодах команди настільки ефективно, що ЦПП майже не простоює в чеканні вибірки команд, ЦПП не перетворюється в монопольного користувача системним каналом.

Якщо в черзі звільняється два байти і відсутній запит на доступ до каналу за операндом пам'яті, то схема попередньої вибірки вибирає з пам'яті наступні байти команди. За одне звертання звичайно вибирається два байти (якщо слово розміщене по парній адресі).

У попередній черзі утримується, як правило, не менше одного байта з потоку команд і пристрою обробки не доводиться чекати вибірки команди. Виконання команд відбувається в тій логічній послідовності, у якій написана програма, оскільки в черзі знаходяться ті команди, що зберігалися в комірках пам'яті, що безпосередньо слідують за поточною командою. При передачі керування в іншу комірку пам'яті, хід виконання програм порушується. Пристрій сполучення з каналом звільняє чергу, вибирає команду за

адресою переходу, передає її ЦПП, а після цього починає нове заповнення черги.

При поверненні з підпрограми або з переривання також відбувається реініціалізація черги команд, адреси яких обчислюються автоматично за допомогою схеми утворення адресних констант, наявні в суматорі адреси.

Якщо процесору необхідно виконати цикл читання або запису, попередня вибірка команд припиняється на час циклу.

Регістр адреси команди і суматор адреси. Шістнадцятирозрядний регістр адреси команди (покажчик команди IP) відповідає програмному лічильнику (PC) МП КР580ИК80А. Пристрій спряження каналу записує в нього зміщення (у байтах) наступної команди від початку поточного сегмента коду. Регістр IP указує наступну команду, що буде виконуватися по закінченні поточної команди і вибірки її з пам'яті пристроєм спряження каналу. Якщо вміст регістра IP пересилається в стек, то покажчик команди автоматично настроюється на адресу наступної команди. Програміст не має прямого доступу до покажчика команд, але може змінити вміст покажчика непрямим шляхом. Деякі команди викликають запис умісту покажчика команд у стек і видачу його зі стека.

У регістрі адреси команди, як було сказано вище, зберігається зміщення команди від початку сегмента коду. У регістрі сегмента коду утримується початкова адреса сегмента коду. Базова адреса сегмента коду і зміщення утворять логічну адресу команди. Базова адреса і зміщення являють собою 16-розрядні беззнакові величини. Найперший (початковий) байт сегмента має нульове зміщення.

Фізична адреса відрізняється від логічної тим, що він адресує конкретну комірку області пам'яті до одного Мбайта. Значення 20-розрядної фізичної адреси може змінюва-

тися від *0H* до *FFFFFH*. Будь-які пересилки між ЦПП і ЗП здійснюються по фізичній адресі.

Програміст найчастіше усього має справу з логічними адресами, не цікавлячись конкретним розташуванням своєї програми по комірках пам'яті, що значно полегшує динамічне керування пам'яттю. Самі різні логічні адреси можуть указувати на ту саму фізичну комірку пам'яті.

При будь-якому звертанні ПСК до пам'яті при вибірці команд і запису або читанні операндів провадиться перетворення логічної адреси у фізичну. Фізична адреса утворюється в суматорі адреси шляхом зсуву базової адреси сегмента на чотири розряди вліво і додавання його до величини зміщення. Відзначимо, що додавання провадиться за модулем 64 Кбайта, тобто якщо отримана величина перевищує значення самої старшої адреси сегмента і виходить за його межі, то фізична адреса знову відраховується від початку сегмента за величиною перевищення. Так утвориться 20-розрядна адреса будь-якої фізичної комірки пам'яті. Сказане вище відноситься до обчислення фізичної адреси команди й операндів; при обчисленні адреси команди в якості базової адреси використовується вміст регістра сегмента коду, а при обчисленні адреси операнда – уміст регістрів сегмента даних і сегмента ES.

2.3.3 Пристрій обробки

Команди, що заздалегідь були обрані з пам'яті і розташовані у внутрішній попередній черзі ПСК, за запитом пристрою обробки надходять у нього. У ПО відбувається декодування і виконання команд. Байти коду команди подаються в мікропрограмний пристрій керування, що містить ПЗП мікрокоманд. У результаті на виході виробляються мікрокоманди, що управляють процесом обробки, інакше, виконанням поточної команди. У складі ПО є АЛП з трьома тимчасовими регістрами для проміжного збереження

операндів, а також регістр ознак для збереження різноманітних ознак результату виконання арифметичних і логічних операцій. АЛП маніпулює регістрами загального призначення й операндами команди.

Для прискорення внутрішніх пересилок даних усі регістри в магістралі даних в ПО мають 26 розрядів. Пристрій обробки не має прямого виходу на системний канал. Зв'язок із «зовнішнім світом» здійснюється ПСК, що і постачає ПО командами. Коли команді потрібний доступ до пам'яті або до ПВВ, то ПО також звертається до ПСК, що по другому запиті одержує і видає дані. Пристрій обробки оперує 16-розрядними адресами, але має доступ до усього простору пам'яті, що адресується, завдяки переміщенню адрес, виконаному в ПСК.

Регістри загального призначення. Центральний процесорний пристрій має внутрішню область пам'яті. Цей ЗП складається з восьми 16-розрядних регістрів загального призначення (рис.2.3), що розбиті на дві підгрупи (регістри даних і індексні регістри) по чотири регістри в кожній.

Регістри даних, іноді називають групою *H* і *L*, відрізняються від інших регістрів тим, що кожний регістр даних можна використовувати як 16-розрядний і як два 8-розрядних регістри, оскільки їх старші і молодші половини адресуються окремо. Інші регістри ЦПП завжди використовуються тільки як 16-розрядні регістри.

Регістри даних можна використовувати без обмеження в більшості арифметичних і логічних операцій. Крім того, у деяких командах передбачається (неявно) використання командою визначених регістрів, що дає виграш у довжині команди і дозволяє одержувати компактні, але потужні коди. Неявне призначення всіх регістрів загального призначення наведене в табл. 2.4.

Рис.2.3

Таблиця 2.4

Регістр	Призначення
AX	Множення, ділення і ввід/вивід слів
AL	Множення, ділення і ввід/вивід байтів, перекодування, десяткова арифметика
AH	Множення, ділення байтів
BX	Перекодування
CX	Операції з рядками, цикли
CL	Зсув і циклічні зсуви на різне число розрядів
DX	Множення, ділення слів, неявне ввід/вивід
SP	Операції зі стеком
SI	Операції з рядками
DI	Операції з рядками

Інша група регістрів, яку іноді називають групою *I* і *P*, містить у собі два покажчики (бази і стека) і два індексних регістри (джерела і приймача). Покажчики й індексні регістри також можуть брати участь у більшості арифметичних і логічних операцій.

Регістри *SP, SI* і *DI* також неявно використовуються в деяких командах (див. табл. 2.4).

Всі вісім регістрів загального призначення підходять під визначення "акумулятор", що використовувалося в МП КР580ИК80А.

Регістр АХ. Регістр *AX* також називається акумулятором. Регістр *AX* завжди застосовується в операціях множення і ділення, а також призводить до максимальної ефективності при використанні в деяких арифметичних, логічних операціях і операціях пересилки даних.

Молодші 8 бітів регістра *AX* відомі також як регістр *AL* (мнемоніка для A-Low, тобто А-молодший), а старші 8 бітів регістра АХ відомі також як регістр АН (мнемоніка для A-High, тобто А-старший). Цей розподіл зручно використовувати при роботі з даними розміром в один байт, оскільки в цьому випадку *AX* може служити як два окремих регістри. Такий фрагмент програми встановлює *AH* рівними 0, копіює це значення в *AL*, а потім додає до *AL* одиницю:

```
mov ah,0
mov al,ah
inc al
```

кінцевим результатом цього фрагмента є установка регістра *AL* у 1. Регістри *BX, CX* і *DX* також можуть служити або як один 16- бітовий регістр, або як два 8-бітових регістри.

Регістр ВХ. Регістр *BX* може служити покажчиком на комірки пам'яті. 16- бітове значення, збережене у *BX*, може бути використано як частина адреси комірки пам'яті, до якої виконується доступ. Наприклад, наступний фраг-

мент програми завантажує в *AL* вміст комірки пам'яті з адресою 9:

```
mov ax,0
mov ds,ax
mov bx,9
mov al,[bx]
```

Зверніть увагу на те, що перед звертанням до комірки пам'яті, на яку указує *BX*, ми завантажили в регістр *DS* 0 (через *AX*). Це викликано сегментированою природою пам'яті 1810ВМ86. По умовчанню при використанні регістра BX у якості покажчика пам'яті, пам'ять, що адресується їм, залежить *від* умісту регістра DS.

Як і регістри *AX*, *CX* і *DX*, регістр *BX* може розглядатися як два 8-бітових регістри, *BH* і *BL*.

Регістр CX. Спеціалізація регістра *CX* полягає в його використанні в якості лічильника. Припустимо, що ви хочете повторити виконання деякого блока команд 10 разів. Це можна зробити у вигляді:

```
mov cx,10
Loop:    ;початок циклу
.
(повторювані команди)
.
sub cx,1
jnz Loop
```

Не звертайте уваги на незнайомі вам елементи програми: суть тут полягає в тому, що команди між міткою Loop (початок циклу) і командою *JNZ* повторюється доти, поки вміст *CX* не стане рівним 0. Відзначимо, що для реверсивного рахунку регістром *CX* і для оберненого переходу на мітку Loop, якщо вміст його не дорівнює 0 потрібно дві команди: sub cx,1 і jnz Loop.

Реверсивний рахунок і зациклювання є елементами програм, що зустрічаються часто, тому 1810ВМ86 має спе-

ціальну команду, що дозволяє організувати цикл більш швидко і компактно. Не дивовижно, що ця команда називається *LOOP* (цикл). Команда *LOOP* віднімає від *CX* одиницю і виконує перехід, якщо його вміст не стає рівним 0, усе в одній команді. Нижче приводиться еквівалент попередньому прикладу:

```
mov cx,10
BeginigOfLoop:
.
повторення команди
.
loop BeginingOfLoop
```

запам'ятайте, що регістр *CX* спеціально призначений для організації лічильників і циклів.

Як і регістри *AX*, *BX* і *DX*, регістр *CX* може розглядатися як два 8-бітових регістри. *CH* і *CL*.

Регістр DX. Регістр *DX* є єдиним регістром, що може бути використаний як покажчик адрес вводу/виводу (*I/O*) командами *IN* і *OUT*. Дійсно, не існує засобу адресації портів *I/O* від 256 до 65,535, не використовуючи *DX*. Наприклад, наступний фрагмент програми записує в порт *I/O* 1000 значення 62:

```
mov al, 62
mov dx, 1000
out dx, al
```

Інші унікальні властивості *DX* пов'язані з операціями ділення і множення. При діленні 32-бітового діленого на 16-бітовий дільник старші 16 біт діленого повинні бути розташовані в *DX*; після ділення залишок записується в *DX* (молодші 16 біт діленого розташовані в *AX*, і частка також розташована в *AX*). Аналогічним образом, при множенні двох 16-бітових множників старші 16 біт добутку записуються в *DX* (молодші 16 біт поміщаються в *AX*).

Як і регістри *AX*, *BX* і *CX*, регістр *DX* може розглядатися як два 8-бітових регістри, *DH* і *DL*.

Регістр SI. Як і регістр *BX*, регістр *SI* може використовуватися як покажчик пам'яті. Наприклад,

```
mov ax, 0
mov ds, ax
mov si, 20
mov al, [si]
```

завантажує в *AL* 8-бітове значення, збережене за адресою 20. При роботі з рядковими командами 1810ВМ86 регістр *SI* стає винятково потужним засобом. Наприклад:

```
cld
mov ax, 0
mov ds, ax
mov si, 20
lodsb
```

не тільки завантажує в *AX* значення комірки пам'яті з адресою 20, але і додає до 81 одиницю. Ця властивість може бути дуже ефективною при доступі до послідовності суміжних комірок пам'яті, наприклад, до рядка тексту. Ще краще організувати автоматичне повторення рядкових команд будь-яке число разів, щоб виконати однією командою сотні і навіть тисячі дій.

Регістр DI. Регістр *DI* схожий на регістр *SI* у тому плані, що він може бути використаний у якості покажчика пам'яті і має спеціальні властивості при роботі з потужними рядковими командами. Наприклад:

```
mov ax, 0
mov ds, ax
mov di, 1024
add bl, [si]
```

складе 8-бітове значення, збережене за адресою 1024, із вмістом *BL*. Регістр *DI* трохи відрізняється від регістра *SI* при роботі з рядковими командами; якщо *SI* завжди служить для

рядкових команд як покажчик на вихідну комірку пам'яті, то *DI* завжди служить покажчиком на комірку пам'яті призначення результату. Крім того, при роботі з рядковими командами *SI* звичайно адресує пам'ять щодо сегментного регістра *ES*, (при використанні *SI* і *DI* як покажчик пам'яті не рядковими командами вони завжди виконують адресацію щодо *DS*). Наприклад:

```
cld
mov dx, 0
mov es, dx
mov di, 2048
stosb
```

використовує рядкову команду *STOSB* для розміщення значення з регістра *AL* на адресу пам'яті, регістр *DI,* і для додавання до *DI* одиниці. Тут ми трохи забігаємо вперед; спочатку нам потрібно дізнатися про сегменти і сегментні регістри, і тільки після цього ви зможете почати вивчення рядкових команд.

Регістр BP. Як і регістри *BX, SI* і *DI*, регістр *BP* може бути використаний як покажчик до пам'яті, проте з деякою відмінністю. Якщо регістри *BX, SI* і *DI* звичайно діють як покажчики на комірки пам'яті відносно сегментного регістра *DS* (або, при використанні *DI* із рядковими командами, стосовно регістра *ES*). BP указує щодо *SS*, регістра стекового сегмента.

Стек, або стековий сегмент, розташован в сегменті, на який покажчиком служить регістр *SS*. З іншого боку, дані звичайно поміщаються в сегменті даних, на який указує регістр *DS*. Оскільки *BX, SI* і *DI* звичайно вказують на сегмент даних, ефективного засобу використання *AX, SI* або *DI* для вказівки на параметри, передані на стек, не існує, оскільки стек звичайно знаходиться зовсім в іншому сегменті.

Регістр *BP* вирішує цю проблему, виконуючи адресацію стекового сегмента. Наприклад:

push bp
mov bp, sp
mov ax, [bp+4]

Регістр *BP* призначений для підтримки параметрів, локальних змінних та інших задач адресації пам'яті на базі стека.

Регістр SP. Регістр *SP*, також відомий як покажчик стека, є найменш універсальним із усіх регістрів загального призначення, оскільки він практично завжди служить тільки для однієї задачі: обслуговування стека. Стек це область пам'яті, у якій дані поміщаються і витягаються звідти в режимі "останнім увійшов - першим вийшов"; тобто, останнє значення, приміщене на стек, буде першим, одержуваним звідти при читанні стека.

Регістр *SP* у кожний момент часу вказує на вершину стека: вершина стека це адреса, у якій буде записане значення, що поміщається на стек.

Дія, пов'язана з записом значення в стек, називається "помістити на стек" і виконується командою *PUSH*. Аналогічним образом, дія витяг значення зі стека називається "зняти зі стека" і виконується командою *POP*.

Хоча 1810ВМ86 дозволяє поміщати в *SP* будь-які значення, а також додавати та віднімати дани, збережені у *SP*, як і для всіх інших регістрів загального призначення, ніколи не варто робити цього, не знаючи в достеменно, до чого це призведе. Змінивши значення *SP*, ви тим самим зміните адресу вершини стека, що швидко призведе до порушення роботи машини.

Чому це відбудеться? Справа в тому, що поміщення на стек і зняття зі стека це не єдині засоби роботи з ним. Стек використовується кожен раз коли викликають або виходять з підпрограми (процедури або функції). Це означає, що стек може знадобитися в будь-який момент, якщо ви зміните *SP*, навіть на час виконання декількох команд, то

при спробі використання його деякими системними ресурсами його стан буде невірним.

Коротше говорячи, не зачіпайте *SP*, поки не будете достеменно знати, що ви робите. Спокійно користуйтеся розташуванням на стек, зняттям із стека, викликами підпрограм і поверненням на них, але не намагайтеся прямо змінити стан *SP*. Інші сім регістрів загального призначення дозволяють пряму модифікацію в будь-який час.

Покажчик команд. Покажчик команд *IP* (*PC*) завжди містить зміщення в пам'яті, по якому розташована наступна команда. Як тільки виконана одна команда, покажчик команд переміщається таким чином, щоб указувати на команду по наступній адресі пам'яті і є наступною командою що буде виканона, проте деякі команди, наприклад, виклики і переходи, можуть завантажувати в покажчик команд інші значення, тим самим виконуючи розгалудження до іншої дільниці програми.

У покажчик команд не може бути прямо записане або зчитане значення; нове значення може бути завантажене в покажчик команд виняткова командами розгалуження.

Самий по собі покажчик команд не може цілком визначати адресу, у якій знаходиться наступна команда. Картина знову ускладнюється внаслідок сегментированої природи адресації пам'яті 1810ВМ86. Для витягу команди потрібна базова адреса, що лежить у сегментному регістрі *CS*, а покажчик команд дає зміщення щодо цієї базової адреси.

Регістр ознак. Вихід АЛП процесора пов'язаний із регістром ознак, із 16 розрядів якого використовується біля 9. З них 6 розрядів використовуються для відбитка деяких специфічних властивостей результату арифметичних і логічних операцій. У системі команд МП К1810ВМ86 є група команд, що дозволяють змінити порядок виконання програми в залежності від стана цих розрядів, тобто від результату попередньої операції. Різноманітні команди впливають на

ознаки по різному. Як результат операції впливає на регістр ознак і як використовується та або інша ознака результату описано нижче.

Якщо ознака допоміжного переносу *AF*=1, це значить, що був перенос із третього розряду в четвертий, або позика з четвертого розряду в третій розряд молодшого байта 16-розрядного числа. Ознака допоміжного переносу (*AF*) використовується командами десяткової арифметики.

Якщо ознака переносу *CF*=1, це означає, що був перенос або позика зі старшого результату. Ознака використовується командами додавання і віднімання багато байтних чисел. Команди циклічного зсуву можуть ізолювати розряд, що зсовується, у пам'яті або регістрі шляхом розміщення його в розряді ознаки переносу.

Якщо ознака переповнювання *OF*=1, це значить, що відбулося арифметичне переповнювання, тобто зникла значуща цифра, коли розрядність результату перевищує розрядність приймача результату. У системі команд МП К1810ВМ86 є команда *INTO* (переривання по переповнюванню), що виробляє програмне переривання при наявності ознаки переповнювання.

Якщо ознака знака *SF*=1, то старший розряд результату дорівнює "1" (результат від'ємний). У противному випадку, якщо *SF*=0, знак результату додатний. Нагадаємо, що в МП К1810ВМ86 двійкові від'ємні числа подані в додатковому коді.

Якщо ознака парності *PF*=1, значить результат парний. Ознака використовується для виявлення збоїв при передачі даних.

Якщо ознака нуля *ZF*=1, значить результат операцій дорівнює нулю.

У регістрі ознак використовуються ще три додаткових розряди, доступних програмісту. Ці розряди регістра

ознак можна використовувати для керування діями процесора шляхом запису в них "0" або "1".

Запис у розряд ознаки напрямку (*DF*) логічної одиниці викликає автодекремент при виконанні операцій із рядками даних. Це значить, що рядки обробляються від старших адрес до молодших (справа ліворуч). Запис у *DF* нуля викликає автоінкремент, тобто обробку рядків зліва праворуч.

Якщо ознака дозволу переривання *IF*=1, то процесор реагує на зовнішні запити переривання, що маскуються. Запис у тригер дозволу переривання нуля забороняє ці переривання. Слід зазначити, що ознака дозволу переривання *IF* не впливає на внутрішні та переривання, що не маскуються.

Якщо ознака покрокового режиму *IF*=1, то процесор переходить у покроковий режим виконання програми. В такому режимі після виконання кожної команди автоматично генерується переривання. Покроковий режим зручний для налагодження програм, коли доводиться контролювати крок за кроком виконання команди.

Як і регістри загального призначення, кожний із сегментних регістрів грає особливу роль. Регістр *CS* указує на програмні коди, регістр *DS* указує на дані, регістр *SS* є вказівкою на стек, а регістр *ES* це регістр універсального призначення ("надлишковий") і може служити покажчиком на усе, що може знадобитися. Тепер розглянемо сегментні регістри більш докладно.

Сегментні регістри. Область пам'яті, безпосередньо що адресується МП К1810ВМ86, досягає одного мегабайта. Вона розділена на логічні сегменти, кожний із яких містить до 64 Кбайт. ЦПП має прямий доступ до чотирьох сегментів одночасно завдяки наявності сегментних регістрів, в яких зберігаються базові (початкові) адреси сегментів.

Регістр сегмента коду (*CS*) містить базову адресу сегмента коду, із якого вибираються команди. Регістр сегмента даних (*DS*) показує поточний сегмент даних. Регістр сегмента стека (*CS*) указує на поточний сегмент стека. З комірками цього сегмента провадяться стекові операції. Регістр додаткового сегмента (*ES*) указує на початок додаткового сегмента, що також використовується для збереження даних.

Сегментні регістри доступні програмісту і їх вміст може змінюватися по деяких командах.

Регістр CS. Регістр *CS* указує на початок блока пам'яті розміром 64Кб, або сегмент програми, у якому знаходиться наступна виконувана команда. Наступна виконувана команда розташовується в сегменті програми зі зміщенням, обумовленим регістром *IP*: тобто, за адресою сегмент:зміщення CS*:IP*. 1810ВМ86 не може витягти команду ні відкіля, крім сегмента, обумовленого в *CS*.

Регістр *CS* може бути змінений при відповідній зміні номера команди, включаючи конкретні переходи, виклики та повернення з підпрограм. Регістр *CS* не може бути прямо завантажений яким-небудь значенням ні при яких обставинах.

Ніякі режими адресації пам'яті або покажчики пам'яті, за винятком *IP*, звичайно, відносно *CS* не працюють.

Регістр DS. Регістр *DS* указує на початок сегмента даних, що представляє собою блок пам'яті розміром 64Кб, у якому знаходиться велика частина операндів пам'яті. Звичайно відносно *DS* працюють зміщення пам'яті, обумовлені регістрами *BX*, *SI* або *DI*, як прямі адреси пам'яті. У цілому сегмент даних працює так, як припускається його назвою: це сегмент, у котрому звичайно знаходиться поточний набір даних програми.

Регістр ES. Регістр *ES* указує на початок блока пам'яті розміром 64Кб, відомого як надлишковий сегмент. Як

припускається його ім'ям, цей сегмент не призначений спеціально для жодної цілі, і доступний у разі потреби. Іноді надлишковий сегмент використовується для організації додаткового блока пам'яті розміром 64Кб для збереження даних, проте, доступ до такої пам'яті звичайно менш ефективний, чим до пам'яті сегмента даних.

Проте при використанні рядкових команд надлишковий сегмент винятково ефективний. Всі рядкові команди, що виконують запис у пам'ять, використовують для цього адресу *ES:DI*. Це означає, що *ES* дуже корисний як сегмент призначення при копіюванні блоків, порівнянні рядків, скануванні пам'яті й очищенні блоків пам'яті.

Регістр SS. Регістр *SS* указує на початок стекового сегмента, що являє собою блок пам'яті розміром 64Кб, де розташовується стек. Всі команди, що неявно використовують регістр *SP*, включаючи команди помещення на стек, зняття зі стека виклику і повернення – працюють із стековим сегментом, оскільки *SP* може адресувати пам'ять тільки стекового сегмента. Як було відзначено раніше, регістр *BP* також працює відносно стекового сегмента. Це дозволяє використовувати *BP* для адресації параметрів і змінних, що зберігаються на стеку.

Схема обробки запитів переривання. Мікропроцесор К1810ВМ86 може обробляти до 256 видів перериванньь. Можливі переривання трьох типів: зовнішні, внутрішні і програмні.

Зовнішні запити переривання надходять по двох зовнішніх виводах мікросхеми *INT* і *NMI*. По входу *INT* надходять масковані запити, що задовольняються після виконання поточної команди, якщо тригер дозволу переривання встановлений в одиницю (це означає, що ознака дозволу переривання *IF* = 1). Схема обробки запитів переривання (ОЗПР) не запам'ятовує маскований запит, тому необхідно підтримувати сигнал *INT* доти, поки не буде отримане підт-

вердження переривання. По вході *NMI* надходить немаскований запит переривання, що запам'ятовується і розпізнається поза залежністю від стана ознаки дозволу переривання. Проте немаскований запит так само, як і маскований, не розпізнається до завершення поточної команди.

До внутрішніх переривань відносяться переривання по переповнюванню (*INT0*), помилці ділення та покроковий режим. При надходженні внутрішніх запитів переривання схема ОЗПР виробляє загальний запит переривання. Внутрішні переривання не маскуються й обробляються після виконання останнього такту команди, як і апаратні переривання (*INT* і *NMI*).

Програмні переривання викликаються відразу після виконання спеціальної команди *INTn*. Тип переривання закодований у самій команді, отже, відпадає необхідність у виконанні циклів підтвердження переривання з метою одержання покажчика (типу переривання).

При системному скиданні зовнішні масковані переривання запитуються, тому що регістр ознак, а отже, і ознака дозволу переривання скидаються в нуль.

Схема керування доступом до магістралі. Мікропроцесор К1810ВМ86 має особливі зовнішні виводи 30 і 31. У мінімальному режимі (вивід $MN / \overline{MX}$ залучений до джерела живлення Ucc) ЦПП надає доступ до каналу по запиті захоплення (*HLD* / *HLDA*). Цей режим ЦПП призначений для використання в невеличких однопроцесорних системах. У мінімальному режимі ЦПП сам виробляє сигнали керування каналом ($OP / \overline{IP}, M / \overline{IO},\ STB, \overline{DE}, \overline{R}, \overline{W}, \overline{INTA}$).

Коли схема керування доступом до магістралі одержує по вході *HLD* від контролера каналу запит на доступ до каналу, вона видає по виході *HLDA* сигнал підтвердження захоплення в середині останнього такту поточного циклу каналу (такт Т4 або холостий такт Т5). Одночасно з видачею сигналу *HLDA* процесор переводить у високоімпедан-

сний стан канал *A/D* і керуючі виводи, тобто ЦПП логічно відключається від каналу, надаючи його в користування зовнішньому пристрою, що потребує його. Вхід *HLD* є асинхронним, тому він опитується по передньому фронті кожного тактового імпульсу (ТІ).

Як тільки сигнал на вході *HLD* перейде в стан низького рівня, тобто зовнішній пристрій, що володіє каналом, знімає запит, ЦПП в наступному такті припиняє видачу сигналу підтвердження захоплення *HLDA* і повертає собі керування каналом.

У максимальному режимі (вивід $MN/\overline{MX}$ підключений до виводу *GND*) виводи 30 і 31 перевизначаються ($HLD \rightarrow \overline{RQ}/\overline{E0}; HLDA \rightarrow \overline{RQ}/\overline{E1}$). У цьому режимі ЦПП передає функції керування каналом контролеру каналу, а саме перебудовується для роботи в умовах складної мультипроцесорної системи. Замість сигналів *HLD* і *HLDA* використовуються сигнали запит/дозвіл доступу до магістралі ($\overline{RQ}/\overline{E0}$ і $\overline{RQ}/\overline{E1}$), що забезпечують особливий механізм доступу до каналу в максимальному режимі. Відмінність цих шин складається в тому, що замість двох шин *HLD* і *HLDA* для функції захоплення використовується тільки одна $\overline{RQ}/\overline{E0}$ або $\overline{RQ}/\overline{E1}$.

При подачі запиту доступу до магістралі на один із входів $\overline{RQ}/\overline{E}$ (від іншого процесора системного каналу) ЦПП наприкінці поточного циклу каналу (такти Т4 або Т5) видає на ту ж шину сигнал підтвердження. ЦПП входить у стан "захоплення" у наступному такті (переводить у високоімпедансний стан системний канал). Керування каналом приймає процесор, що вимагав доступу. Цей процесор по закінченні своїх операцій із каналу видає імпульс знову на ту ж $\overline{RQ}/\overline{E}$ шину, повідомляючи про те, що готовий звільнити системний канал. І в наступному ж такті ЦПП повер-

тає собі керування каналом. Обмін сигналами в цьому процесі строго синхронізований.

Якщо запит надходить одночасно на обидві шини $\overline{RQ}/\overline{E}$, то першим задовольняється запит по вході $\overline{RQ}/\overline{E0}$.

Запит по вході *HLD* розпізнається негайно після системного скидання, а запит по вході $\overline{RQ}/\overline{E}$ – у першому або в другому такті після скидання.

Запит по входах $\overline{RQ}/\overline{E}$ (так само, як і по вході *HLD*) має більш високий пріоритет, чим переривання.

2.4 Опис режимів і принципів роботи МП

2.4.1 Керування МП

Запуск і скидання МП. Для запуску або установки МП у вихідний стан використовують вхід МП *CLR*.

Для правильного запуску МП одночасно з напругою живлення на вхід *CLR* подається сигнал *CLC* високого рівня. Сигнал *CLR* повинен залишатися в стані високого рівня не менше 50 мкс після досягнення напругою живлення номінального значення.

Для установки МП у вихідний стан у процесі роботи (перезапуск МП) необхідно подати на вхід *CLR* сигнал високого рівня тривалістю не менш 4 періодів тактової частоти.

При надходженні сигналу *CLR* МП припиняє виконання внутрішніх операцій, переводить виходи каналу адреси/даних у високоімпедансний стан, виходи каналу керування у високоімпедансний або визначений неактивний стан (табл.2.5) і залишається в цьому стані на весь час дії сигналу *CLR*.

Таблиця 2.5

Вихід	Стан
AD15-AD0	Третій
A19-A16/ST6-ST3	Третій
$\overline{BME}$ / *ST 7*	Перехід через "1" у третій стан
M / IO($\overline{ST\ 2}$*)*	Перехід через "1" у третій стан
OP / $\overline{IP}$*(*$\overline{ST\ 1}$*)*	Перехід через "1" у третій стан
$\overline{DE}$*(*$\overline{ST\ 0}$*)*	Перехід через "1" у третій стан
$\overline{W}$ *(*$\overline{LOCK}$ *)*	Перехід через "1" у третій стан
$\overline{R}$	Перехід через "1" у третій стан
$\overline{INTA}$	1
STB	0
HLDA	0
$\overline{RQ}$ / $\overline{E0}$	1
$\overline{RQ}$ / $\overline{E1}$	1
QS0	0
QS1	0

Оскільки МП забезпечує внутрішню синхронізацію зовнішнього сигналу *CLR* такими імпульсами, то внутрішній сигнал *CLR* подається на внутрішні вузли МП із затримкою на один такт відносно зовнішнього сигналу *CLR* (рис. 2.4). При надходженні сигналу *CLR* виходи каналу керування перед переходом у високоімпедансний стан встановлюються в неактивний стан на час одного такту, як показано на рис. 2.4 і табл. 2.5.

У мінімальному режимі МП вихідні процеси *STB* і *HLDA* під час дії сигналу *CLR* переводять не в третій (високоімпедансний) стан, а в неактивний стан (див. табл. 2.5).

У максимальному режимі сигнали $\overline{RQ}$ / $\overline{E0}$ і $\overline{RQ}$ / $\overline{E1}$ підтримуються в неактивному стані, а сигнали стана черги команд *QS0* і *QS1* указують на відсутність операцій. Якщо під час дії сигналу внутрішньої установки на входи МП надходить сигнал *NMI* або *HLD* (запиту доступу до магістралі

$\overline{RQ}/\overline{E}$ в максимальному режимі), то вони не сприймаються МП і не підтверджуються. Якщо сигнал *HLD* у мінімальному режимі МП або імпульси $\overline{RQ}/\overline{E0}$, $\overline{RQ}/\overline{E1}$ у максимальному режимі активні відразу після закінчення внутрішнього сигналу *CLR*, то вони будуть сприйняті МП перед вибором першої команди (із стана установки МП перейде в стан захоплення або дозволу доступу до магістралі).

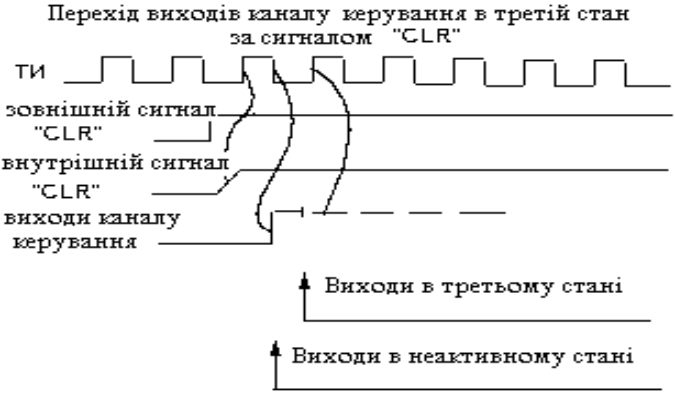


Рис. 2.4

Оскільки сигнал *CLR* скидає тригер дозволу переривання в регістрі ознак у стан "0", то сигнал *INT* не сприймається МП, навіть якщо цей сигнал знаходиться в стані високого рівня відразу після закінчення внутрішнього сигналу установки. Якщо на вхід *NMI* надійде сигнал високого рівня раніш, ніж через 9 тактів після закінчення сигналу *CLR*, то МП може виконати одну команду раніше, ніж перейде до обробки переривання.

Після переходу сигналу *CLR* у стан низького рівня МП починає виконувати внутрішні операції, у результаті яких вміст сегментних регістрів, регістрів ознак, регістра адреси команди і черга попередньо обраних команд встановлюються відповідно до табл. 2.6.

Таблиця 2.6

Внутрішній регістр	Зміст
Регістр ознак	Скинутий
Регістр *IP*	0000H
Регістр *CS*	FFFFH
Регістр *SS*	0000H
Регістр *ES*	0000H
Регістр *DS*	0000H
Попередні регістри команд	Порожньо

Мікропроцесору потрібно приблизно 8 тактів для виконання цих внутрішніх операцій. Тимчасова діаграма запуску Мікропроцесора після зняття сигналу CLR показана на рис. 2.5.

Рис. 2.5

Оскільки вміст регістра сегмента коду *CS* відповідно до табл. 2.6 дорівнює FFFFH, а вміст регістра адреси команди IP*(PC)* – 0000H,то мікропроцесор вибирає першу команду після установки з комірки пам'яті з адресою FFFF0H. Звичайно ця комірка містить команду прямого межсегментного переходу *JMP* на справжній початок системної програми.

Зупин мікропроцесора. При виконанні команди *HLT* (Зупин) мікропроцесор переходить у стан зупину. Цей стан можна інтерпретувати як припинення всіх дій до появи зовнішніх сигналів запиту переривання або сигналу *CLR*.

Під час зупину мікропроцесор не видає ніяких керуючих сигналів, а канал адреси/даних знаходиться у високоімпедансному стані. Якщо на вхід *HLD* мікропроцесора, що знаходиться в стані зупину, надійшов сигнал високого рівня (або імпульси на входи $\overline{RQ}/\overline{E0}$ $\overline{RQ}/\overline{E1}$, у максимальному режимі), то мікропроцесор підтверджує перехід у стан захоплення (стан дозволу доступу до магістралі) звичайним образом і повертається в стан Зупину після зняття сигналу *HLD* ($\overline{RQ}/\overline{E}$).

Стан зупину можна використовувати, якщо якась зовнішня подія перешкоджає правильному функціонуванню системи, наприклад, переривання по збою в живленні. Якщо виникла ситуація, що вимагає відключення живлення, то процесор може використовувати час, що залишився до відключення живлення, на пересилку вмісту внутрішніх регістрів і регістра ознак або яких-небудь важливих змінних із звичайної пам'яті (ОЗП) в область пам'яті, що забезпечує зберігання інформації при відключенні живлення, а потім перейти в стан зупину. Мікропроцесор залишається в стані зупину до появи зовнішнього сигналу *INT*, що свідчить про появу напруги живлення.

У залежності від обраного режиму роботи мікропроцесор ініціює перехід у стан зупину двома засобами. У мінімальному режимі процесор видає під час виконання команди *HLT* один сигнал *STB* без зміни стана керуючих сигналів. У максимальному режимі при виконанні команди *HLT* процесор видає комбінацію сигналів стана циклу каналу $\overline{ST0},\overline{ST1},\overline{ST2}$ (011), що вказує, що він знаходиться в стані зупину.

Сигнали стана мікропроцесора. У максимальному режимі мікропроцесор видає 8 сигналів стана, що можуть бути використані зовнішніми пристроями.

Сигнали $\overline{ST\ 0}, \overline{ST\ 1}, \overline{ST\ 2}$ визначають тип циклу обміну по каналу *A/D*, котрий процесор починає виконувати (табл.2.7). Ці сигнали повинні декодуватися спеціальною схемою системного контролера каналу.

Таблиця 2.7

Сигнали стана			Тип циклу каналу
$\overline{ST\ 2}$	$\overline{ST\ 1}$	$\overline{ST\ 0}$	
0	0	1	Підтвердження переривання
0	1	0	Читання ПВВ
0	1	1	Запис в ПВВ
1	0	0	Зупин
1	0	1	Вибірка команди
1	1	0	Запис у пам'ять
1	1	1	Пасивний

Сигнали стана *ST3* і *ST4* визначають, якийсь сегментний регістр використовується в поточному циклі обміну для обчислення фізичної адреси комірки пам'яті (табл. 2.8).

Таблиця 2.8

Сигнали стана		Сегментний регістр, що використовується
ST4	*ST3*	
0	0	*ES*
0	1	*SS*
1	0	*CS* або ніякий(при вводі-виводі або перериванні)
1	1	*DS*

Сигнал стана *ST5* видає стан тригера переривання регістра ознак.

Сигнал *ST6* завжди знаходиться в стані низького рівня, а сигнал *ST7* є, як би допоміжним сигналом і його стан у процесі роботи мікропроцесора не визначено.

Попередня черга команд. У той час, коли пристрій обробки мікропроцесора зайнятий виконанням потоку команд, пристрій сполучення каналу вільний і може зайнятися попередньою вибіркою команд із пам'яті. Для тимчасового

збереження цих команд мікропроцесор надає внутрішню область оперативної пам'яті, так звану попередню чергу команд довжиною 6 байт.

Коли пристрій обробки готовий до виконання команди, він витягає з черги байтів об'єктного коду команди і починає виконувати цю команду. Якщо ж черга порожня, то мікропроцесор очікує її заповнення. Розмір попередньої черги команд дозволяє постачати мікропроцесор заздалегідь обраними командами без монополізації доступу до каналу. Попередня вибірка наступного байта провадиться завжди, коли в черзі звільняється два байти і відсутній запит на доступ до каналу від пристрою обробки. За одне звертання звичайно витягається два байти об'єктного коду. У більшості випадків черга містить не менше одного байта, тому мікропроцесору майже не доводиться чекати, коли пристрій обробки вибере команду. При виконанні команди передачі керування черга скидається, а після переходу до нової команди (у точку передачі керування) починається нове заповнення черги. Якщо пристрою обробки необхідно звертання до каналу, то вибірка команд у попередню чергу припиняється.

Мікропроцесор у максимальному режимі видає по виходах *QS0* та *QS1* інформацію про стан попередньої черги команд. У табл. 2.9 подані чотири можливі комбінації стана цих сигналів. Сигнали *QS0* та *QS1* стану черги призначені для того, щоб зовнішні процесори могли приймати від ЦПП мікропроцесора К1810ВМ86 команди та (або) операнди за допомогою команди *ESC*. Зовнішній процесор може шляхом опитування каналу визначити момент вибірки і виконання в ЦПП команди *ESC*. Сигнали стана, отримані в поточному такті, описують дії (стан) попередньої черги в попередньому такті.

Таблиця 2.9

QS0	*QS1*	Стан черги
0	0	Немає операцій. В останньому такті з черги нічого не вибиралося.
0	1	Перший байт. Байт, обраний із черги, був першим байтом команди
1	0	Черга порожня. Черга була реініціалізована в результаті виконання команди переходу
1	1	Наступний байт. Байт, обраний із черги, був наступним байтом команди

2.4.2 Структура переривань

Мікропроцесор К1810ВМ86 має просту і гнучку структуру переривань. Кожному типу переривання привласнений окремий код типу. Мікропроцесор може опрацьовувати до 256 видів різноманітних переривань. Переривання можуть визиватися пристроями, зовнішніми стосовно ЦПП, а також командами переривання і самим ЦПП при деяких умовах (рис. 2.6). Переривання можуть бути програмними й апаратними.

Апаратні переривання можуть бути маскованими і немаскованими.

Зовнішні масковані переривання. Мікропроцесор має два зовнішніх виводи, по котрим запити переривань надходять від зовнішніх пристроїв. Вивід *INT* (запит переривання) підключається до виходу схеми контролера переривань, що програмується, який у свою чергу залучений до пристроїв, що потребують в обслуговуванні переривання. Контролер переривань служить для прийому запитів від пов'язаних із ним пристроїв, визначення пристрою, із найвищим пріоритетом і збудження шини *INT* мікропроцесора, якщо пристрій, що запитує, має більш високий пріоритет, чим поточний пристрій, що обслуговується.

Рис. 2.6

Запит переривання по вході *INT* може маскуватися внутрішньо командою *CLI*, що скидає тригер дозволу переривання *IF*.

Опрацювання маскованого переривання починається по закінченні останнього такту поточної команди, тому сигнал на вході *INT* повинен мати високий рівень протягом тактового імпульсу, що передує кінцю команди або операції передачі блока. Якщо ж під час опрацювання переривання по входу *INT* прийде наступний запит, то він буде ігноруватися доти, поки підпрограма обслуговування поточного переривання не відновить тригер дозволу переривання IF.

Дії ЦПП, таким чином, залежать від стана тригера дозволу *IF*. Проте, доти, поки не завершиться виконання поточної команди, ніяких дій по перериванню не провадиться. Якщо тригер дозволу *IF* скинений (тобто переривання по входу *INT* замасковане або заборонено), то ЦПП ігнорує запит і переходить до виконання наступної коман-

ди. Сигнал *INT* у ЦПП не запам'ятовується, тому він повинний підтримуватися, поки не буде отримана відповідь на запит або знятий запит. Якщо ж масковані переривання дозволені (*IF*="1"), то ЦПП розпізнає запит і обслуговує його. Запит переривання, що надходить по входу *INT*, може бути програмно дозволений по виконанні команди *STI* або заборонений командою *CLI*. Переривання можуть бути також вибірково замасковані (деякі типи дозволені, а інші заборонені) за допомогою команди, яка записана в контролер переривань. Зауважимо, що з метою зменшення можливості надмірного росту глибини стека, застосовують команди *STI* та *IRET*, що знову дозволяють переривання тільки після завершення поточної команди, а не в будь-якому такті виконання команди.

Якщо на вході *INT* є активний сигнал, а переривання дозволені (*IF*="1"), то мікропроцесор виконує так звану послідовність підтвердження переривання. Щоб гарантувати підтвердження переривання, необхідно сигнал на вході *INT* підтримувати доти, поки ЦПП не почне цикл підтвердження. Зауважимо, що підтвердження переривання виробляється тільки у відповідь на масковані запити (по вході *INT*). Якщо пристрій спряження каналу починає виконувати цикл каналу після того, як виявляє умову переривання (це відбувається, якщо пристрій спряження витягає наступну команду після попередньої команди), то запит переривання повинний підтримуватися на вході мікропроцесора протягом двох тактів перед початком такту Т4 поточного циклу. У цьому випадку наступним циклом каналу буде цикл підтвердження переривання. Якщо цих двох тактів не достатньо, то перед циклом підтвердження ЦПП виконає інший цикл каналу - цикл чекання. А якщо до того ж очікується і захоплення каналу (це відбувається, як тільки з'являться запити переривання і захоплення, під час виконання заблокованої

команди), то переривання буде обслуговане після того, як завершиться обслуговування захоплення.

На рис. 2.7 показана діаграма циклу підтвердження. Центральний процесорний пристрій підтверджує запит переривання по входу *INT*, виробляючи два послідовних цикли каналу (один за іншим), розділених двома холостими тактами Т5. У кожному з циклів підтвердження переривання подається сигнал $\overline{INTA}$ ("Підтвердження переривання"), але не сигнал читання $\overline{R}$. Протягом обох циклів підтвердження переривання ЦПП не видає адреси, але сигнали стана і сигнал дозволу старшого байта $\overline{BHE}$ дійсні. А оскільки виробляється строб адреси (*STB*), то зовнішні регістри адреси будуть завантажуватися невизначеною інформацією. Таке положення потребує, щоб системні пристрої не могли видавати інформацію на вихід, не отримавши команду читання ($\overline{R}$). Якщо протягом циклів підтвердження переривання надходить запит захоплення (по виводах *HLD* або $\overline{RQ}$ / $\overline{E}$), то він не розпізнається до завершення цих циклів.

Крім того, якщо мікропроцесор К1810ВМ86 функціонує в максимальному режимі, то він видає сигнал блокування каналу ($\overline{LOCK}$), протягом циклів підтвердження переривання, вказуючи тим самим іншим процесорам системи, що захоплення каналу для них заборонено. Перший цикл $\overline{INTA}$ вказує контролеру переривань, що запит прийнятий; у другому циклі $\overline{INTA}$ контролер видає на канал байт даних - тип переривання 0-255, пов'язаний із перериваючим пристроєм, що потребує обслуговування, ЦПП зчитує код типу переривання і використовує його для виклику підхожої підпрограми обслуговування переривання.

Основна відмінність циклу підтвердження переривання від циклу читання складається в тому, що замість сигналу читання виробляється сигнал підтвердження перери-

вання (*INTA*) і канал адреси знаходиться у високоімпедансному стані.

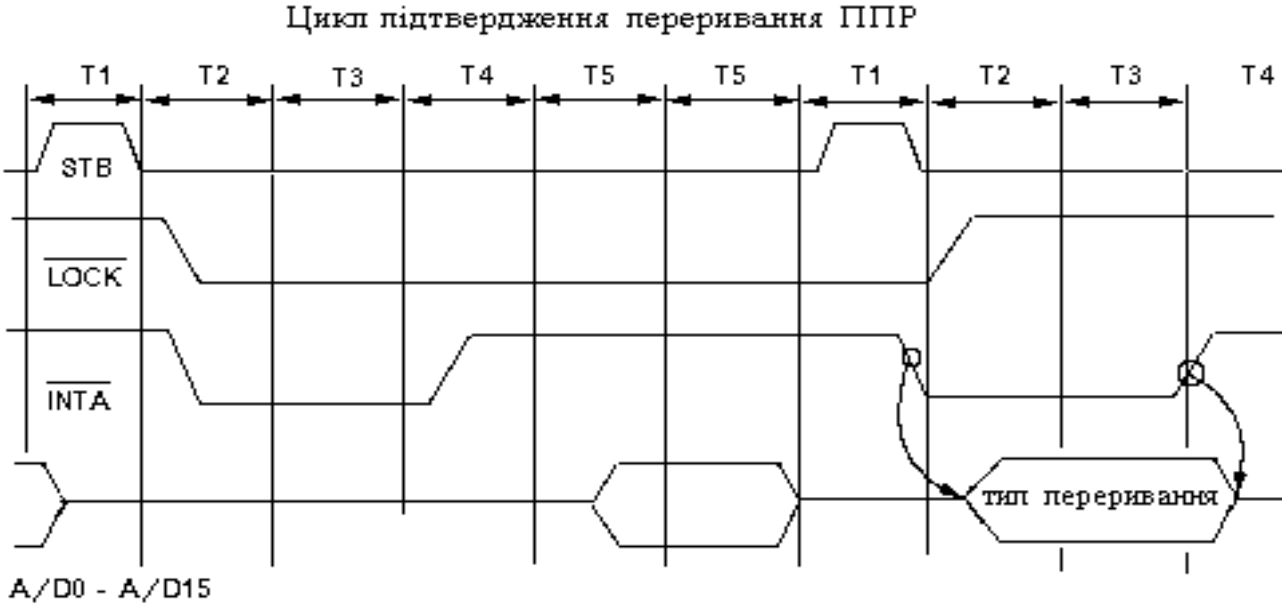

Цикл підтвердження переривання ППР
T1
T2
T3
T4
T5
T5
T1
T2
T3
T4
STB
LOCK
INTA
тип переривання
A/D0 - A/D15

Рис. 2.7

Зовнішні немасковані переривання. Зовнішні запити можуть надходити в мікропроцесор К1810ВМ86 не тільки по входу 18 (*INT*), але і по входу 17 (*NMI*). Немасковане переривання має найвищий пріоритет у порівнянні з маскованими і програмними перериваннями. Немасковані переривання використовують для повідомлення ЦПП про такі "катастрофічні" події, як погроза відключення живлення, виявлення помилки пам'яті та помилки парності каналу.

Реакція на *NMI* відбувається при позитивному перепаді напруги на вході немаскованого переривання і викликає безумовний перехід на підпрограму в пам'яті.

Немасковані запити не можуть бути заборонені. Сигнал на вході *NMI* повинен залишатися в стані високого рівня більш двох тактів, але в його синхронізації немає необхідності, тому що він внутрішньо фіксується в ЦПП й обслуговується наприкінці будь-якої команди мікропроцесора, за винятком випадку блокових передач, що можуть бути перервані.

Немаскованим перериванням заздалегідь привласнений тип 2 переривання, тому відпадає необхідність передачі в ЦПП коду типу для виклику процедури обслуговування *NMI*, а отже, не потрібно циклу підтвердження переривання.

Найбільша затримка немаскованого переривання відбувається у випадку виконання команд множення, ділення і зсуву на *n* розрядів. Причому розмір затримки залежить від числа тактів, що залишилися до кінця поточної команди.

Внутрішні переривання. Внутрішні переривання можна умовно розбити на три класи: передвстановлені переривання, що виникають при появі специфічних внутрішніх умов функціонування мікропроцесора, а також встановлюються користувачем апаратні і програмні переривання,шо слід зазначити, що переривання будь-якого типу можуть бути викликані користувачем апаратним і (або) програмним засобом.

Команда *INT* викликає переривання відразу по її виконанні. Коли виникає переривання, ЦПП одержує тип переривання, закодований у самій команді, у виді коду, пов'язаного зі специфічним видом переривання. Користувач повинний мати підпрограму обслуговування й ініціалізувати відповідну адресу в так званій таблиці покажчиків (векторів) переривань. А оскільки можна призначити будь-який код типу, то програмні переривання можна використовувати для перевірки підпрограм обслуговування переривання, записаних для обслуговування зовнішніх пристроїв.

Якщо встановлений тригер переповнювання (*OF*="1"), то по команді *INTO* виробляється переривання типу 4 відразу після її виконання.

Після виконання команд ділення *DIV* або *IDIV* ЦПП виробляє переривання типу 0, якщо результат операції має більшу розрядність, чим приймач.

Якщо встановлений тригер покрокового режиму (*IF*="1"), то ЦПП автоматично виробляє переривання типу 1 після кожної команди. Такий режим виконання програм називається покроковим і є зручним прийомом налагодження програм.

Всі внутрішні переривання (*INT, INTO*, помилка ділення, покроковий режим) мають наступні властивості:

– код типу переривання або утримується в команді, або заздалегідь визначений;

– цикли каналу ППР (підтвердження переривання) не виконуються;

– внутрішні переривання не можуть бути заборонені, за винятком покрокового переривання;

– будь-яке внутрішнє переривання (крім покрокового) має більш високий пріоритет, чим будь-яке зовнішнє (табл. 2.10).

Таблиця 2.10

Тип переривання	Пріоритет	Зростання

	переривання	пріоритету
Переривання по помилці ділення, *INTn, INT0*	Вищий	
Немасковане переривання *(NMI)*		
Масковане переривання *(INT)*		
Покроковий режим	Нижчий	

Переривання типу 0. Переривання типу 0 - помилка ділення - виникає всякий раз, коли намагаються виконати операцію ділення, результат якої перевищує максимальний припустимий розмір, наприклад при діленні на нуль.

Дане переривання немасковане і використовується при виконанні команди ділення. Якщо в підпрограмі обслуговування переривання не передбачена установка в "1" тригера дозволу переривання, то час виконання процедури обслуговування потрибно включати в тривалість виконання команди ділення в найгіршому випадку, наприклад, при оцінці максимальної тривалості виконання команди та її впливу на затримку обслуговування апаратних переривань.

Переривання типу 1. Переривання типу 1 - покроковий режим. Таке переривання виникає по виконанню однієї команди після того, як був установлений тригер покрокового режиму (*IF*="1"). Воно використовується для виконання послідовності команд у програмно-кроковому режимі. Такий режим ініціалізується шляхом запису в стек вмісту регістра ознак, установки в стеку біта *IF*=1 і вибірки зі стека вмісту регістра ознак.

Процедура обслуговування покрокового режиму зберігає ознаки і вміст регістра адреси команди(покажчика команд *IP* і регістра *CS*), а потім скидає тригер покрокового режиму *IF*, щоб покрокова процедура пройшла нормально. При поверненні з переривання відновляється вміст регістрів *CS, IP* і ознак, у тому числі встановлюється в "1" тригер *IF*.

Це дає можливість виконати ще одну команду з послідовності перед повторним входом у підпрограму обслуговування покрокового переривання.

Переривання покрокового режиму не маскується бітом дозволу переривання *IF*.

Мікропроцесор К1810ВМ86 не має команд безпосереднього очищення й установки тригера покрокового режиму *IF*. Але його стан можна змінити шляхом модифікації вмісту регістра ознак у стеку.

Переривання типу 2. Переривання типу 2 - немасковане переривання (*NMI*). Має самий високий пріоритет серед апаратних переривань.

Сигнал на вході *NMI* запам'ятовується в тригері, але не синхронізується з тактовими імпульсами процесора. Цей сигнал повинний залишатися активним протягом двох тактів, щоб гарантувати його розпізнавання мікропроцесором, тому що мінімальна тривалість вхідної напруги високого рівня, що забезпечує надійне запам'ятовування сигналу внутрішнім тригером, складає два такти. Сигнал немаскованого переривання *NMI* може бути знятий перед входом у підпрограму обслуговування. Помилкового переходу тут не виникне, тому що переривання виробляється по позитивному перепасу напруги на вході. Вхід *NMI* резервований для випадків "катастрофічних" збоїв (наприклад, збій живлення або переповнювання системного таймера-лічильника).

Переривання типу 3 і 4. Однобайтне переривання типу 3 називається спеціальною командою переривання довжиною в один байт і використовується, в основному, як переривання-призупин при налагодженні програм. Дане переривання не маскується.

Переривання типу 4 - це переривання по переповнюванню. Виникає воно, якщо тригер переповнювання *OF* встановлений у "1" і виконана команда *INTO*. По цій ко-

манді відбувається перехід на підпрограму обслуговування помилки переповнювання. Переривання не маскується.

Переривання типів 0 і 2 можуть виникнути без впливу програміста в той час, як виробка переривань типів 1, 3 і 4 потребує свідомих дій програміста. Всі типи переривань, крім типу 2, визиваються програмним засобом і прямо пов'язані з визначеною командою.

Програмні переривання. Програмні переривання встановлюються користувачем. Користувач може викликати переривання програмним засобом по двобайтній команді переривання *INTn*. Перший байт команди є кодом операцій *INT*, а другий байт (*n*) - кодом типу переривання, що варто виконати. Команда *INTn* не маскується ознакою дозволу переривання. Її можна використовувати для передачі керування підпрограмі, що динамічно переміщається, адреса в пам'яті якої не відомо основній програмі. Перед передачею керування команда *INTn* зберігає в стеку вміст регістра ознак програми, що викликає. Викликана підпрограма обслуговування переривання по закінченні обслуговування повинна повернути керування в точку переривання основної програми за допомогою команди повернення *IRET* шляхом передачі вмісту регістра ознаки зі стека і повного відновлення стана основної програми.

Всі програмні переривання не маскуються ознакою дозволу переривання *IF* та ініціюють передачу керування наприкінці тієї команди, під час виконання якої вони з'являються. Але ці переривання не ініціюють циклу підтвердження і забороняють наступні масковані переривання шляхом скидання тригерів *IP* та *IF* . Вектори переривання таких типів або припускаються, або включаються в поле команди. Оскільки немасковане переривання є для ЦПП асинхронною подією, то сутність розпізнавання його й ініціалізації передачі така ж, як для апаратних немаскованих переривань.

Таблиця покажчиків (векторів) переривань. Таблиця зв'язує код типу переривань та процедуру, призначену для опрацювання зазначеного типу переривання (рис. 2.8).

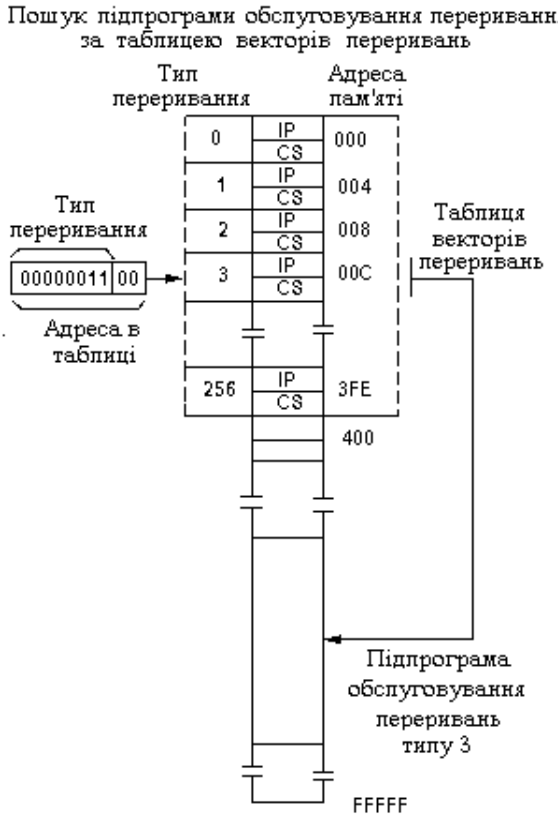


Рис. 2.8

Таблиця займає перший Кбайт області пам'яті, що адресується низько. В ній може бути до 256 входів: по одному на кожний тип переривання, можливий у системі. Кожний вхід у таблиці - це подвійне слово, що містить адресу процедури опрацювання переривання даного типу. У

старшому слові покажчика утримується адреса сегмента, в якому зберігається процедура, а в молодшому слові - зміщення процедури від початкової адреси сегмента. Оскільки кожний вхід має довжину чотири байти, то ЦПП може обчислити місцезнаходження відповідного входу в таблиці для даного типу переривання простим множенням на 4 (тип×4).

Мікропроцесор К1810ВМ86 починає опрацювання переривання в такий спосіб: записує вміст регістрів *CS* та *IP* у стеку, заміняє їхній вміст на друге і перше слова елемента таблиці з адресою (тип 4), передаючи тим самим керування підпрограмі обслуговування переривання.

Якщо в системі використовується менше 256 типів переривань, то старші комірки таблиці покажчиків, не зайняті під типи переривань, що не використовуються в даному застосуванні, можуть бути застосовані для інших цілей. Проте молодші комірки (0H – 7FH), зайняті покажчиками передвстановлених переривань (типи 0 – 4) і зарезервовані, не можна використовувати ні для яких цілей, тому що це порушить правильне функціонування системи.

Якщо в мікропроцесор надходить одночасно декілька запитів переривання, то він активізує їх обслуговування відповідно до пріоритету.

Процедури опрацювання переривань. Вміст регістрів ознак *CS* та *IP* засилається в стек, коли викликається процедура обслуговування переривання, а тригери дозволу переривання *IF* та покрокового режиму *TF* скидаються. Всередині самої процедури можна знову розв'язати зовнішнє переривання за допомогою команди *STI*. Цією дією процедура дозволяє перервати себе по сигналу *INT* (варто зауважити, що в дійсності переривання дозволяється лише після виконання наступної за *STI* команди).

Процедура переривання може бути перервана по запиту немаскованого переривання (*NMI*). Програмні та внутрішні переривання, що з'явилися усередині процедури, та-

кож перервуть процедуру. Необхідно стежити за тим, щоб усередині процедури, що обслуговує переривання визначеного типу, не з'явилося переривання того ж типу. Наприклад, при спробі ділення на нуль усередині процедури переривання типу 0 може відбутися безкінечне виконання цієї процедури. Обсяг стека повинний бути достатнім для максимальної глибини переривань, можливої в системі.

Процедури переривання повинні зберігати в стеку будь-які регістри, що будуть використовуватися основною програмою надалі, а також відновлювати їхній вміст перед закінченням опрацювання переривання. Всередині процедур корисно дозволяти зовнішні переривання.

Сказане відноситься до всіх фрагментів процедур, за винятком "критичних", тобто таких, що не можна переривати без ризику одержання помилкового результату. Якщо в якийсь процедурі зовнішні переривання будуть занадто довго заборонені, то масковані запити переривань по запровадженню *INT* можуть бути загублені.

Всі процедури переривань повинні закінчуватися командою *IRET* (повернення з переривання). При виконанні команди *IRET* передбачається, що стек знаходиться в тому ж стані, що і перед процедурою. По команді *IRET* три верхніх слова зі стека пересилаються в регістри *IP*, *CS* і регістр ознак, тобто керування передається тій команді, перед якою була активізована процедура переривання.

Якщо процедура обслуговує зовнішній пристрій, то вона повинна забезпечити цей пристрій командою про зняття запиту переривання. Після цього процедура повинна прийняти від пристрою інформацію про його стан, щоб визначити причину переривання і діяти відповідно цьому.

Процедури програмних переривань можна використовувати в якості підпрограм ("виклик супервізора") для інших програм у системі. У таких випадках процедура переривання активізується, коли програма, що обслуговуєть-

ся, потребує в більшій увазі, ніж зовнішній пристрій. Подібні ситуації виникають при необхідності переслати повідомлення в іншу програму, знайти файл для запису, при запиті на переміщення вільної пам'яті і т.д. Процедури програмних переривань зручно використовувати в системах із динамічним переміщенням програм під час виконання. Процедури можуть викликати одна одну по таблиці покажчиків переривань за допомогою команд програмних переривань, оскільки ця таблиця розташована у фіксованих комірках пам'яті. Процедури переривань і самі можуть переміщатися, тому що таблиця їхніх покажчиків може обновлятися, забезпечуючи зв'язок з програмою, що зв'язує, за допомогою коду типу переривання.

Існує принципова відмінність між процедурами, що викликаються внутрішніми перериваннями, і процедурами, що обслуговують зовнішні переривання. Процедури обслуговування внутрішніх переривань можуть, а часто повинні, бути повторюваними. З іншого боку, процедури обслуговування зовнішніх переривань повинні розглядатися як тимчасові задачі в тому смислі, що задача - це єдина послідовність виконання, і вона не повинна повторюватися. Реакція мікропроцесора на зовнішнє переривання може бути подана у виді послідовності таких дій:

– припинення виконання активної задачі;

– створення нової задачі - процедури обслуговування переривання, що стає виконуваною задачею;

– завершення і знищення задачі опрацювання переривання;

– активізація припиненої задачі і її виконання з того місця, де вона була зупинена.

Процедура опрацювання зовнішнього переривання повинна бути перерваною тільки по тим запитам, що активізують іншу процедуру переривання. Коли число пристроїв - джерел переривання не занадто велике, ця вимога автома-

тично задовольняється, якщо усім джерелам переривання присвоїти коди типу і процедури обслуговування. У системах із великим числом однотипних джерел (наприклад, 500 ліній зв'язку) можна використовувати метод, що ілюструється на рис. 2.9, для того щоб запобігти в системі повторення процедур обслуговування переривання і втрати деяких переривань, навіть якщо вони повторюються безперервно.

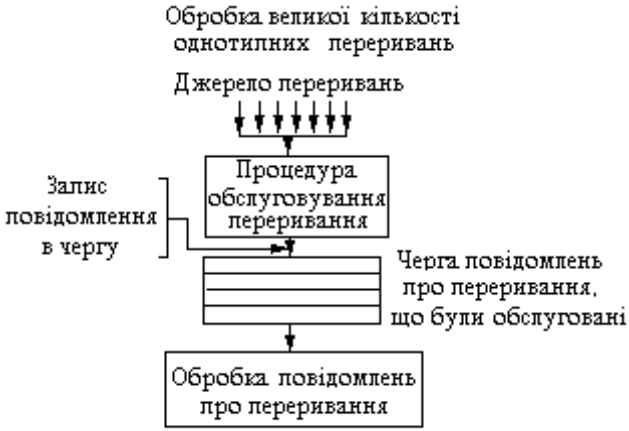


Рис. 2.9

Метод полягає в розділенні процедури переривання. Сама процедура обслуговування переривання пишеться як можна більш короткою, вона виконує абсолютний мінімум дій, необхідних для обслуговування всіх пристроїв. Ця процедура потім будує повідомлення, що містить достатню інформацію для рішення іншої задачі - опрацювання повідомлення про переривання, і для завершення обслуговування переривання. Процедура обслуговування переривання ставить чергове повідомлення про переривання в чергу повідомлень, що може бути організована у виді зв'язаного списку, і закінчується, так що черга виявляється доступною

для обслуговування наступного переривання. Програма опрацювання повідомлення про переривання, що не відновляється, одержує повідомлення з черги, закінчує наступне повідомлення (якщо таке є в черзі) і т.д. Коли зростає частота переривань, черга повідомлень буде збільшуватися, але переривання не будуть губитися доти, поки є достатній час між запитами для активізації і виконання процедури обслуговування переривання.

Цикл підтвердження переривання. Мікропроцесор К1810ВМ86 виробляє цикл підтвердження переривання тільки у відповідь на масковані запити переривання по входу 18 (*INT*).

На рис. 2.7 показана діаграма циклу підтвердження. Він складається з двох послідовних циклів каналу $\overline{INTA}$, розділених двома холостими тактами. Під час циклів $\overline{INTA}$ видається сигнал $\overline{INTA}$, а не сигнал читання $\overline{R}$. З плином обох циклів мікропроцесор К1810ВМ86 адреси не видає (сигнали стана та дозволу старшої половини каналу $\overline{BHE}$ дійсні), але продовжують виробляти строб адреси *STB*, по якому зовнішні регістри адреси будуть завантажуватися невизначеною інформацією. От чому системні пристрої, не отримавши сигналу читання $\overline{R}$, не можуть викликати інформацію на вихід. Як буде показано нижче, сигнал *STB* використовується в сполученні з пріоритетним контролером переривань.

Протягом циклів каналу $\overline{INTA}$ сигнали $OP/\overline{IP}$ й $\overline{DE}$ обумовлюють прийом у ЦПП однобайтного типу переривання із системи переривань. Перший цикл $\overline{INTA}$ свідчить про початок циклу підтвердження переривання і дозволяє системі підготуватися до появи байта типу переривання в наступному циклі $\overline{INTA}$. В першому циклі ЦПП не приймає з каналу ніякої інформації. Код типу переривання повинний передаватися в мікропроцесор по молодшій половині 16-

бітного каналу даних у другому циклі. Це значить, що пристрій, що забезпечує процесор кодом типу, повинен підключатися до молодшої половини каналу. Тимчасова діаграма циклів $\overline{INTA}$ схожа на діаграму циклу читання.

Відмінність послідовності підтвердження переривання мікропроцесорів К1810ВМ86 і КР580ИК80А складається в тому, що ЦПП мікропроцесора К1810ВМ86 не видає при підтвердженні ніяких команд. Для мікропроцесора КР580ИК80А в якості частини циклу $\overline{INTA}$ необхідна команда *RST* або *CALL*, щоб викликати передачу керування.

В мінімальному режимі циклу $\overline{INTA}$ сигнал $M / \overline{ID}$ буде низького рівня, свідчачи про ввід/вивід. З такту Т2 першого циклу до такту Т2 другого циклу буде вироблятися внутрішній сигнал блокування каналу $\overline{LOCK}$, перешкоджаючи захопленню каналу, запит якого відбувається по входу $\overline{RQ} / \overline{E}$, і запобігаючи передачу каналу (за допомогою логіки арбітра каналу) між послідовними циклами $\overline{INTA}$ в складній системі. Призначення сигналу готовності *RDY* таке ж, як і в циклах читання та запису.

Центральний процесорний пристрій мікропроцесора К1810ВМ86, прийнявши код типу переривання (із каналу – при апаратних перериваннях, із потоку команд – при програмних перериваннях або з передвстановленої умови), множить його на 4, у результаті одержує зміщення відповідного покажчика в таблиці векторів. Вектор складається з чотирьох байтів: молодший і старший байти сегментного регістра коду. Під час передачі керування ЦПП записує в стек ознаки і поточний вміст покажчика команд (*IP*) і сегментного регістра коду (*CS*). В регістри *CS* і *IP* завантажуються нові значення, а тригери покрокового режиму (*TF*) і дозволи переривання (*IF*) скидаються. Скидання тригера *IF* забороняє мікропроцесору реагувати на наступні апаратні переривання в плині виконання підпрограми обслуговуван-

ня поточного переривання доти, поки ця підпрограма сама знову не встановить тригер IF у "1". Значення покажчика команд *IP* і сегмента коду *CS* у циклах читання даних зчитуються з таблиці векторів. При звертанні до таблиці векторів під час переходу до підпрограми обслуговування ніякі сегментні регістри не використовуються. При формуванні 20-розрядної адреси зміщення вектора задаються за нуля, а сигнали станів *ST4, ST3* = 10 вказують на те, що ніякий сегментний регістр не вибирається.

Фактично дії каналу, пов'язані з підтвердженням апаратного переривання, полягають у наступному:

– два цикли підтвердження переривання;

– запис регістра ознак у стек;

– запис старого значення регістра *IP* у стек.

– запис старого значення регістра *CS* у стек;

– зчитування нового значення сегментного регістра коду *CS* із таблиці векторів;

– зчитування нового покажчика команд *IP* із таблиці векторів;

– вибір коду операції першої команди підпрограми обслуговування переривання.

Після запису в стек старого значення регістра *IP* пристрій спряження каналу відновлює свої звичайні дії по попередньому виробітку команд у чергу і по обслуговуванню запитів пристрою опрацювання на видачу операндів. Рівень сигналу стана *ST5* (стан тригера дозволу переривання) стає низьким у другому такті після зчитування нового значення регістра *CS*.

Кількість тактів від кінця команди, при виконанні якої з'явилося переривання, до початку виконання підпрограми обслуговування переривання складає 61 такт. Послідовність циклів каналу при програмних перериваннях відрізняється тільки відсутністю циклів підтвердження переривання, що призводить до зменшення затримки виконання

підпрограми до 51 такту для команд *INTn* і переривання покрокового режиму, до 52 тактів для команди *INT3*, до 53 тактів для команди *INT0*. При програмних перериваннях до роботи пристрою спряження пред'являються ті ж вимоги, що установлені для апаратних переривань. Якщо для зовнішніх запам'ятовуючих пристроїв, що постачають ЦПП типом переривання, вводяться стани чекання для узгодження швидкодії, то встановлена затримка виконання підпрограми збільшуються на відповідне число тактів чекання.

2.4.3 Організація та робота каналу мікропроцесора

Структура каналу мікропроцесора К1810ВМ86 в самому загальному вигляді показана на рис.2.10. Є канали двох типів: *системний* і *локальний* (місцевий). Канали обох типів можуть розділятися в часі між декількома процесорами. Мікропроцесори завжди підключають до локального каналу, а до системного каналу звичайно підключають ЗП й ПВВ. Локальний канал мікропроцесора К1810ВМ86 за допомогою інтерфейсів пов'язаний із системним.

Локальний канал. Локальний канал використовується мікропроцесором К1810ВМ86.

Через те, що стандартні прилади пам'яті й ПВВ не підключаються безпосередньо до локального каналу, інформацію можна мультиплексировати і кодувати для підвищення ефективності використання зовнішніх виводів ВІС мікропроцесора. Деякі виводи звільняються і їм можна передати функції по координації спільної роботи декількох процесорів, залучених до локального каналу.

Локальний канал можна розділяти в часі як між незалежними процесорами, так і між сопроцесорами. При цьому логікою арбітра визначається, якому з процесорів надати канал.

Процесори на локальному каналі спільно використовують ті самі інтерфейси каналу, тому локальна конфігурація мікропроцесора утворює компактну і недорогу систему.

Системний канал. Повний системний канал утворює п'ять груп виводів: канал адреси, канал даних, канал керування, шини переривання, арбітражні шини. Канал адреси, керування, даних припускають застосування в системі стандартних пристроїв пам'яті й ПВВ.

Канали адреси і даних демультексировані. У системному каналі передбачені звичайні сигнали керування, наприклад читання/запису ЗП/ПВВ і т.д. У залежності від конкретного застосування ці групи сигналів можуть входити або не входити до складу системного каналу, наприклад, арбітражні шини не потрібні в однопроцесорних системах або в мультипроцесорній системі, в якій поділ каналу в часі (арбітраж) виконано на рівні локального каналу.

Група інтерфейсів перетворить локальний канал у системний канал. Кількість інтерфейсів, необхідних для утворення системного каналу, залежить від розміру і складності системи.

Конфігурацію інтерфейсу визначають наступні основні параметри: розмір області адресації (кількість регістрів), розрядність каналу даних (кількість шинних формувачів) і необхідність поділу каналу в часі (наявність логіки арбітра каналу).

Узагальнена структура каналу мікропроцесора К1810ВМ86

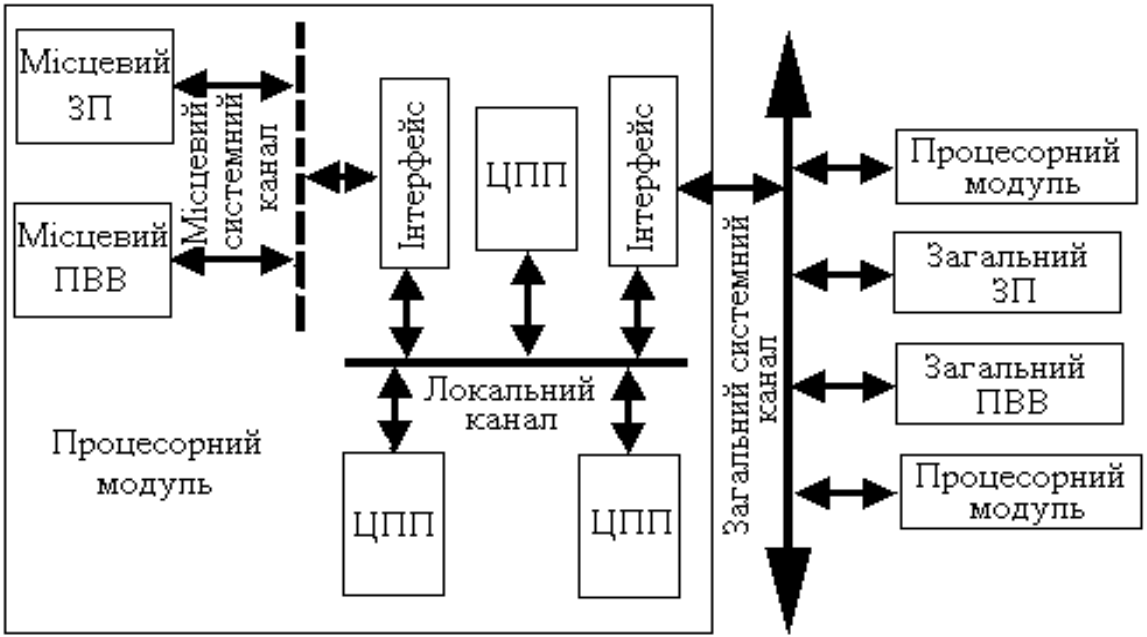


Рис. 2.10

Робота каналу. Роздивимося цикл каналу, який виконує пристрій спряження каналу мікропроцесора К1810ВМ86. Цикл каналу, по суті, є асинхронною подією, протягом якої по каналу видається адреса комірки ЗП або ПВВ, супроводжувана керуючим сигналом читання (прочитати дані з пристрою, що адресується) або запису, а також відповідні дані (передати або записати дані в пристрій, що адресується). Обраний пристрій (ЗП або ПВВ) приймає дані каналу протягом циклу запису або видає викликані дані протягом циклу читання. По закінченні циклу пристрій фіксує дані, що записуються в нього, або знімає зчитуваємі дані.

Кожний цикл каналу складається, як мінімум, із чотирьох машинних тактів: Т1, Т2, Т3 і Т4. У такті Т1 ЦПП поміщає на канал адреса ЗП або ПВВ. У циклі запису ЦПП видає дані на канал у тактах Т2 – Т4. А при виконанні циклу читання ЦПП приймає дані, присутні на каналі, у тактах Т3 та Т4, у такті Т2 мультиплексний канал адреси/даних знаходиться у високоімпедансному стані, що дозволяє ЦПП переключитися з режиму запису (вивід адреси) у режим читання (введення даних).

Як показано на тимчасових діаграмах (рис. 2.11, 2.12), цикли читання і цикли запису мають відмінності. Зауважимо, що тимчасові діаграми наведені для мінімального режиму.

ЦПП мікропроцесора К1810ВМ86 видає 20-розрядну адресу на мультиплексирований канал адреси/даних протягом такту Т1. У такті Т2 ЦПП знімає адресу з каналу й або перекладає у вискоімпедансний стан 16 молодших розрядів каналу *A/D* перед зчитуванням даних, або видає на ці шини дані для запису. На шини адреси/стани в цей ж час видаються сигнали стана циклу каналу. Протягом такту Т3 код стана циклу каналу зберігається на шинах адреси/стани, а на молодших 16 шинах адреси/даних підтримуються або дані,

що записуються, або опитивються зчитуваємі дані. Цикл каналу закінчується в такті Т4 (у цей час керуючі сигнали знімаються, а пристрій, що адресується, відключається від каналу).

Цикл читання мікропроцесору К1810ВМ86

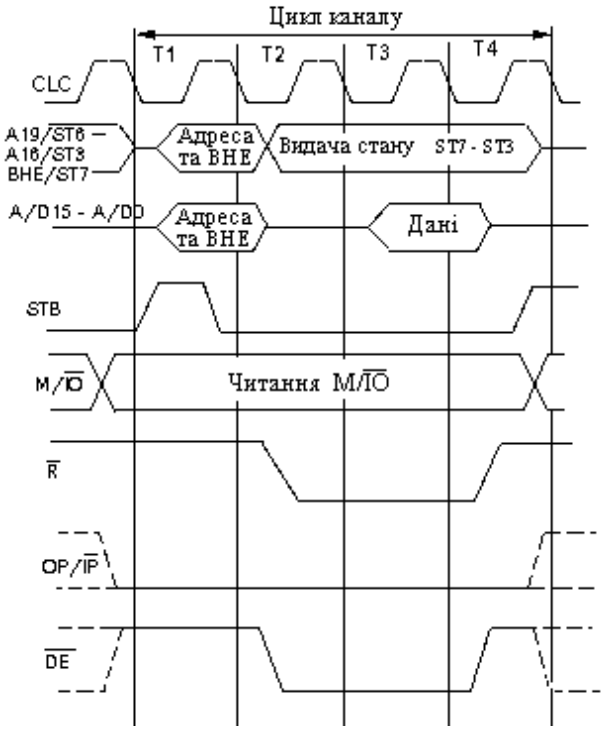


Рис. 2.11

Більшості типів ЗП й ПВВ потрібна постійна адреса протягом усього циклу каналу. У такті Т1 кожного циклу каналу видається керуючий сигнал *STB* стробуючий фіксацію адреси в зовнішніх регістрах (тобто адреса – це код, прийнятий по задньому фронті сигналу STB). Сигнал адреси видається в мінімальному режимі безпосередньо, тобто

самим мікропроцесором, а в максимальному – побічно, тобто через контролер каналу. Таке "демультеплексування" каналу адреси/даних може бути зроблене для кожного системного пристрою окремо, або тільки локально для ЦПП і поширено на всю систему у виді окремого каналу адреси, для досягнення оптимальної продуктивності системи і для сумісності з мультипроцесорними системами рекомендується застосовувати локально- демультиплексований канал адреси.

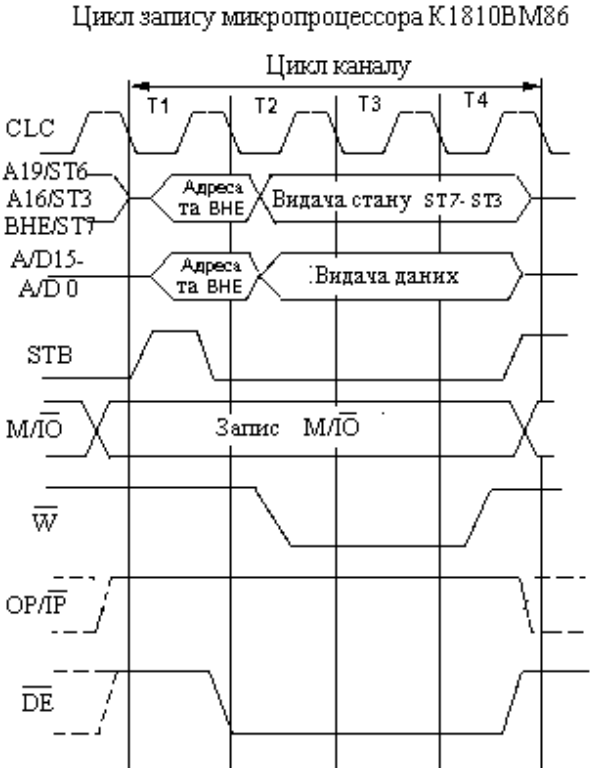


Рис. 2.12

Канал даних не можна демультиплексувати через розходження між циклами читання та запису, а також через

різниці у швидкодії різноманітних пристроїв пам'яті та ПВВ. От чому мультиплексированый канал даних можна лише буферизувати або використовувати безпосередньо. У циклі читання, коли ЗП або ПВВ залучено прямо до не буферизованого каналу, важливо, щоб адреса, що присутня на каналі в такті Т1, не спотворювалася пристроєм. З цією метою вихідні формувачи пристрою повинні дозволятися ззовні по сигналу читання з ЦПП. В мікропроцесорі К1810ВМ86 сигнал читання не буде дійсним, доки адреса не зафіксується по сигналу STB. Багато приладів вводу/виводу ПЗП/ЭППЗП та ОЗП мають керовані виходи, що дозволяє підключати ці прилади прямо до не буферизованого мультиплексного каналу адреси/даних. Навпроти, використання буферизованого каналу даних дозволяє знизити вимоги до периферійних пристроїв, що призводить до збільшення навантажувальної спроможності каналу і нечутливості до ємнісного навантаження.

Мінімальний і максимальний режими. Вибір режиму функціонування в мікропроцесорі К1810ВМ86 пропонує користувачу унікальну можливість вибору підмножини вихідних керуючих сигналів у відповідності зі ступенем складності проектованої мікропроцесорної системи. Системне "настроювання" можливо завдяки наявності спеціального виводу вибору режиму $MN/\overline{MX}$ (вхідний сигнал). Функціональне призначення виводів мікропроцесора К1810ВМ86 для мінімального і максимального режимів наведені в табл. 2.11.

В мінімальний режим ЦПП переводиться, якщо вивід $MN/\overline{MX}$ підключити до шини живлення Ucc. В мінімальному режимі, орієнтованому на малі обчислювальні системи (одно- і двох платні), ЦПП видає сигнали керування каналом, необхідні для функціонування в системі ЗП та ПВВ.

Таблиця 2.11

Номер виводу	Режим	
	Мінімальний	Максимальний
31	*HLD*	$\overline{RQ}$ / $\overline{E0}$
30	*HLDA*	$\overline{RQ}$ / $\overline{E1}$
29	$\overline{W}$	$\overline{LOCK}$
28	*M* / $\overline{IO}$	$\overline{ST\ 2}$
27	*OP*/ $\overline{IP}$	$\overline{ST\ 1}$
26	$\overline{DE}$	$\overline{ST\ 0}$
25	*STB*	*QS1*
24	$\overline{INTA}$	*QS0*

Якщо вивід *MN*/ $\overline{MX}$ залучений до корпуса, то ЦПП знаходиться в максимальному режимі. У такій конфігурації ЦПП кодує вихідні керуючі сигнали так, що вісім сигналів надаються закодованими в три сигнали стана циклу каналу. У системах максимальної конфігурації передбачається застосування конролеру каналу, що декодує сигнали стана $\overline{ST\ 0} - \overline{ST\ 2}$, які надходять із ЦПП, та видає розширений набір сигналів керування для іншої частини системи. А ЦПП використовує п'ять виводів, що звільнилися, для нових сигналів, необхідних для координації спільної роботи з іншими процесорами мультипроцесорної системи.

Мінімальний режим. Мінімальний режим (вивід *MN*/ $\overline{MX}$ залучений до шини *Ucc*) орієтований на використання ЦПП в малих однопроцесорних системах, що містять системний канал і невеличке число пристроїв. У мінімальному режимі ЦПП сам виробляє сигнали керування каналом (*OP*/ $\overline{IP}$, $\overline{DE}$, *STB*, *M* / $\overline{IO}$), сигнали керування циклами каналу ($\overline{R}$, $\overline{W}$ або $\overline{INTA}$), а також забезпечує доступ до каналу по запиту (*HLD*, *HLDA*) від контролера доступу до каналу.

Якщо контролеру каналу необхідний доступ до каналу, то він за допомогою своєї логіки запиту подає сигнал запиту на вхід *HLD* ЦПП. У відповідь на запит захоплення ЦПП видає сигнал *HLDA* як підтвердження пристрою, що запросив захоплення, і одночасно переводить у високоімпедансний стан системний канал і шини керування. Тому що запит каналу є асинхронною подією, то ЦПП опитує вхідний канал *HLD* по передньому фронту кожного тактового сигналу CLC (рис. 2.13) і видає сигнал *HLDA* наприкінці поточного циклу каналу (якщо виконується цикл каналу) або в холостому такті T5. Стан захоплення зберігається доти, поки пристрій - володар каналу не зніме запиту з входу HLD. ЦПП в цей момент повертає собі контроль над системним каналом. Слід зазначити, що ЦПП в стані захоплення буде продовжувати виконання команд, доки йому не буде потрібен цикл каналу.

Максимальний режим. В максимальному режимі (вивід $MN/\overline{MX}$ заземлений) функції керування каналом бере на себе контролер каналу (додавання ще й арбітра каналу дозволяє побудувати мікропроцесорну систему). Контролер каналу повинний забезпечити усі вихідні керуючі сигнали замість ЦПП. Максимальний режим розширює можливість архітектури системи, оскільки мультипроцесирования та застосування сопроцесорів спеціального призначення підвищує продуктивність системи.

Сигнали стана циклу каналу. Контролер каналу використовує код стана циклу каналу (вихідні сигнали ЦПП $\overline{ST\,0} - \overline{ST\,2}$) для генерування всіх вихідних сигналів керування каналом, необхідним для синхронізації циклу каналу. Сигнали $\overline{ST\,0} - \overline{ST\,2}$ ідентифікують тип циклу каналу, що починає виконувати ЦПП.

Часова діаграма циклу захват/ підтвердження захвату (HLD/HLDA)

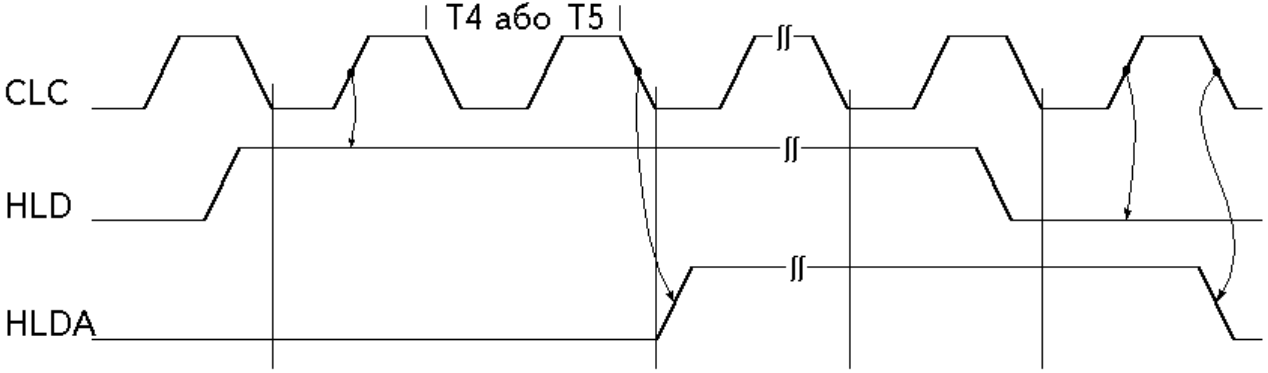


Рис. 2.13

Мікропроцесор К1810ВМ86 дає можливість стежити за ходом виконання в ньому послідовності команд шляхом опитування сигналів стана циклу каналу ($\overline{ST\,0}$, $\overline{ST\,1}$, $\overline{ST\,2}$) та сигналів стана внутрішньої черги команд (*QS0* і *QS1*). Сигнали стана черги команд показують, яка інформація витягається з черги, а при виконанні програмної передачі керування вказують, що черга очищена. Відстеження стана попередньої черги команд дозволяє обробляти розширення системи команд за допомогою сопроцесора. Код стана черги команд *QS0, QS1* указує на стан черги в такті, після виконання якого була виявлена активність черги. Такий механізм дозволяє сопроцесору розпізнати виконання команди *ESC*, яка вказує сопроцесору, яку задачу потрібно виконати.

Додаткові шини стана ($\overline{ST\,0}$, $\overline{ST\,1}$, $\overline{ST\,2}$) призначені для зв'язку зі схемами котролеру каналу й арбітра каналу. Сигнали стана вказують контролеру каналу, коли варто почати цикл каналу, якого типу команду видати і коли закінчити цикл каналу. Контролер каналу повинний опитувати ці сигнали на початку кожного такту ЦПП (*CLC*).

При запуску циклу каналу ЦПП переводить сигнали стана з пасивного стана ($\overline{ST\,0}$, $\overline{ST\,1}$, $\overline{ST\,2}$=111) в один із семи можливих станів (див. табл. 2.7) по передньому фронту тактового імпульсу (*CLS*) машинного такту Т4 попереднього циклу каналу або в холостому такті Т5. По задньому фронту кожного такту контролер каналу опитує сигнали стана циклу каналу і виявляє зміну стана. Після цього він повинний видати строб адреси (*STB*) і сигнал керування напрямком передачі інформації для буфера в наступному такті після зміни циклу каналу, тобто в такті Т1. У наступних тактах дозволяється робота шинних формувачів і встановлюються обрані сигнали керування. А коли стан циклу знову стає пасивним, контролер каналу закінчує керування каналом (рис. 2.14).

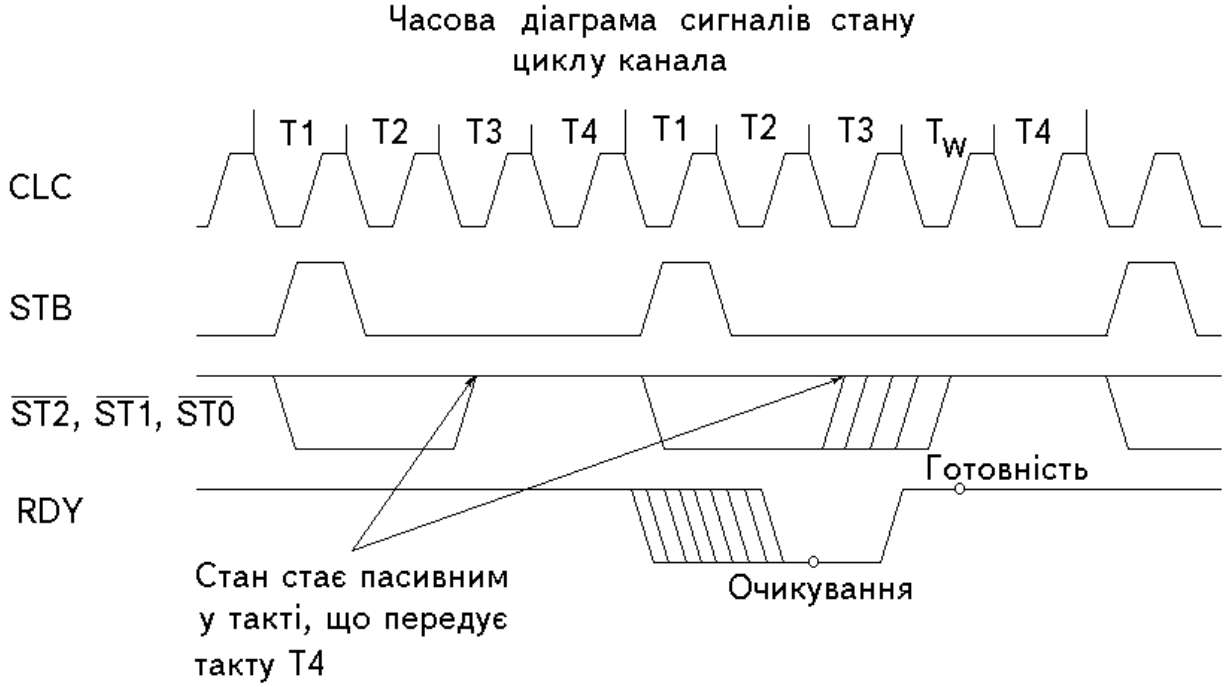

Часова діаграма сигналів стану
циклу канала
T1
T2
T3
T4
T1
T2
T3
Tw
T4
CLC
STB
ST2, ST1, ST0
RDY
Готовність
Стан стає пасивним
у такті, що передує
такту T4
Очикування

Рис. 2.14

Оскільки ЦПП не повертається в пасивний стан, доки не одержить підтвердження "готовності" від ЗП/ПВВ, контролер каналу буде підтримувати керування каналом ще протягом декількох тактів чекання.

Вихідний сигнал блокування $\overline{LOCK}$. В максимальному режимі мікропроцесор, видаючи сигнал блокування, забезпечує собі монопольний доступ до розділеного в часі каналу. Вихідний сигнал $\overline{LOCK}$ управляється програмним засобом шляхом використання префікса блокування *LOCK* перед командою, що потребує особливого доступу.

Сигнал блокування видається після машинного такту, в якому пристрій опрацювання мікропроцесора декодує префікс *LOCK*, та підтримується на виході до кінця першого такту після виконання заблокованої команди. Механізм блокування запобігає втраті керування каналом при виконанні критичних команд, дозволяючи виконувати багатократні цикли каналу без втручання і без можливого перекручування даних з боку інших ЦПП.

Сигнали $\overline{RQ}/\overline{E0}$ і $\overline{RQ}/\overline{E1}$ забезпечують для ЦПП особливий механізм доступу до каналу в максимальному режимі, орієнтований на мультипроцесорні застосування з використанням процесора вводу/виводу.

Особливість цих сигналів полягає в тому, що функцію запиту та дозволу доступу виконує лише одна шина ($\overline{RQ}/\overline{E0}$ або $\overline{RQ}/\overline{E1}$), тоді як у мінімальному режимі дві шини *HLD/HLDA*.

Процес запит/дозвіл складається з трьох фаз: запит, підтвердження та звільнення каналу.

Процес почнеться з того, що інший процесор системного каналу подає сигнали запиту на один із виводів $\overline{RQ}/\overline{E}$ (фаза запиту). У відповідь на запит ЦПП наприкінці поточного циклу каналу (T4) або холостого такту T5 видає сигнал дозволу на ту ж шину, оголошуючи, що він переклал

системний канал у високоімпедансний стан, а в наступному такті відключиться логічно від контролера каналу (фаза дозволу). Так ЦПП входить в особливий стан (захоплення).

У стані захоплення мікропроцесор продовжує виконувати команди зі своєї попередньої черги, поки черга не спорожніє або поки не знадобиться звертання за операндом.

У фазі звільнення процесор, що надає, знову видає ту ж шину $\overline{RQ} / \overline{E}$, попереджаючи ЦПП, що він може повернути собі керування каналом у наступному такті. ЦПП одержує керування через два такти після прийому від сопроцесора сигналу звільнення.

Запити по шині $\overline{RQ} / \overline{E0}$ мають більш високий пріоритет. При виконанні заблокованих команд запит $\overline{RQ} / \overline{E}$ запам'ятовується, але не задовольняється до кінця команди. Вихідний сигнал блокування $\overline{LOCK}$ на переривання не впливає. Команда *LOCK HALT*, наприклад, змушує ЦПП ігнорувати запити захоплення ($\overline{RQ} / \overline{E}$ або *HLD*), але дозволяє йому вийти з зупину по перериванню. Затримка між одержанням запиту захоплення і дозволом зростає, якщо на виході $\overline{LOCK}$ ЦПП видається активний сигнал $\overline{LOCK}$ (при виконанні команди з префіксом блокування або виконанні циклу підтвердження).

2.5 Організація пам'яті

Мікропроцесор може адресувати до 1048576 байт пам'яті й у максимальному, і в мінімальному режимах.

Пам'ять МП подана як масив 8-розрядних байтів. Коди команд, байти і слова даних можуть бути записані в будь-яке місце пам'яті, незалежно друг від друга, що дозволяє їм бути щільно упакованими в пам'яті. Слово даних із непарною адресою не використовує переваги МП, спроможного зчитувати або записувати 16 бітів за одне звертання

до пам'яті. Розміщення команд у пам'яті не впливає на роботу процесора.

Слова даних завжди записуються старшим байтом в комірку пам'яті з більшою адресою. У більшості випадків ця умова непомітна при роботі з МП, за винятком випадку, коли аналізується системна шина або читається вміст пам'яті. Мікропроцесор може також працювати з класом даних, що записуються як подвійні слова. Ці дані називаються покажчиками і використовуються для адресації даних і кодів команд, що знаходяться поза поточним адресним сегментом. Слово покажчика з молодшою адресою містить значення відносної адреси та слово зі старшою адресою містить базову адресу сегмента.

Сегментація. Пам'ять МП К1810ВМ86 функціонально розбита на сегменти. Сегменти є логічними пристроями пам'яті, що можуть мати довжину до 64 Кбайт.

Кожний сегмент виконаний як безупинна область пам'яті. Сегменти є незалежними та довільно адресуємими логічними пристроями. Кожному сегменту програмно призначається базова адреса, що є адресою першої комірки пам'яті в сегменті. Ці адреси кратні 16. Інших обмежень на розміщення сегмента немає. Сегменти можуть стикатися, не стикатися, перекриватися цілком або частково, не перекриватися. Фізична комірка пам'яті може належати одному або декільком сегментам.

Сегментні регістри містять базові адреси чотирьох поточних сегментів. Програма одержує доступ до кодів команд і даним, що зберігаються в інших сегментах за допомогою заміни вмісту сегментних регістрів на адреси нових сегментів.

Сегментна структура пам'яті сприяє модульності програмування при створенні монолітних програм великих розмірів.

Генерація фізичної адреси. У системі, що засновується на МП К1810ВМ86, зручно розглядати два види адрес кожної комірки пам'яті: фізичний і логічний. Фізична адреса є 20-бітним значенням, що однозначно вказує кожний байт пам'яті. Двійковий код фізичної адреси видається на зовнішні шини адреси даних МП по сигналу *STB*. Фізична адреса може змінятися від *0H* до *FFFFFH*. Всі передачі інформації між процесором і пам'яттю використовують фізичну адресу.

Логічна адреса складається із сегментного значення (**сегмент**) і значення відносної адреси (**зміщення**). Для будь-якої комірки пам'яті базове значення сегмента вказує перший байт поточного сегмента, а відносна адреса указує відстань у байтах від поточної комірки пам'яті до початку сегмента. Базова адреса сегмента і значення відносної адреси є 16-бітними беззнаковими величинами. Генерація фізичної адреси здійснюється зсувом значення базового сегмента на чотири біти і додаванням його зі значенням відносної адреси (рис. 2.15).

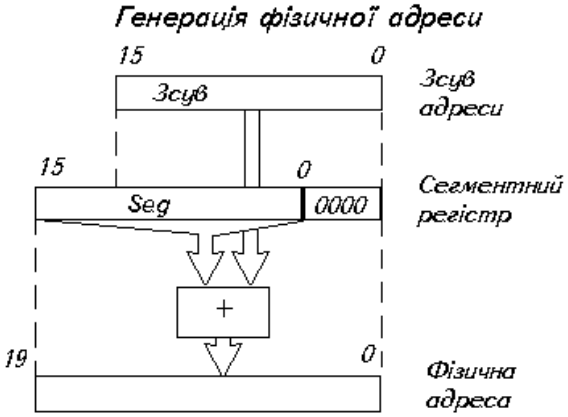


Рис. 2.15.

Інтерфейсний пристрій одержує логічну адресу комірки пам'яті з різноманітних джерел, що залежать від типу

інформації, що одержується або видається (табл. 2.12). Коди команд завжди зчитуються із поточного сегмента коду. Регістр *IP* містить значення відносної адреси команди, що зчитувється відносно початку сегмента. Команди роботи зі стеком завжди оперують у поточному сегменті стека. Регістр *SP* містить відносну адресу вершини стека. Більшість оброблюваних операндів розташовано в поточному сегменті даних. За допомогою програми можна змінювати вміст сегментних регістрів при адресації сегментів.

Зміщення для доступу до змінних пам'яті обчислюється за допомогою операційного пристрою. Ці обчислення відбуваються на основі типу адресації, зазначеного в команді. Результат обчислення називається виконавчою (ефективною) адресою операнда (*EA*).

Таблиця 2.12.

Тип звертання до пам'яті	Сегментний регістр,який використовується за умовчанням	Сегментні регістри,які використовуються з префіксом заміни сегмента	Зміщення
Зчитування кодів команд	*CS*	немає	*IP*
Операції зі стеком	*SS*	немає	*SP*
Змінні (за винятком описаних нижче)	*DS*	*CS, ES, SS*	Виконавча адреса
Рядок даних джерела	*DS*	*CS, ES, SS*	*SI*
Рядок даних приймача	*ES*	немає	*DI*
Адресація, у якій *BP* використовується як базовий регістр	*SS*	*CS, DS, ES*	Виконавча адреса

Рядки даних адресуються інакше, ніж інші змінні. Операнд рядка даних джерела розташований в поточному

сегменті даних. Зміщення операнда джерела береться з регістра *SI*. Операнд приймача рядка даних розташовується завжди в поточному сегменті, додатковому сегменті і його зміщення береться з регістра *DI*. Команди опрацювання рядків автоматично коректують вміст регістрів *SI* та *DI* в залежності від того, чи опрацьовується за одне звертання до пам'яті слово або байт.

Коли регістр *BP* призначається в команді як базовий регістр, змінні розташовуються в поточному сегменті стека. Регістр *BP* забезпечує за умовою адресацію даних у стеку. Регістр *BP* може бути використаний для доступу даних у будь який сегмент.

У більшості випадків призначення сегментів визначається за умовчанням. Але в програмістів є можливість визначати поточний сегмент стека змінної, що вибирається (за винятком операнда призначення в командах опрацювання рядків, що завжди знаходиться в додатковому сегменті) шляхом установки перед командою префікса заміни сегмента. Ця однобайтова машинна команда вказує МП, який сегментний регістр буде використовуватися для доступу до змінної, описаної в наступній за ним команді.

Програми, що переміщаються динамічно. Структура пам'яті МП К1810ВМ86 дає можливість створювати позиційно-незалежні програми або програми, що переміщаються динамічно. Динамічне переміщення дозволяє системам із мультипрограмуванням або поділом часу одержувати в ряді випадків ефект від використання пам'яті. Програма, що не управляє мікропроцесором, може бути записана на диску, і в область, що вона займала раніш, записується інша програма. Якщо програма, що знаходиться на диску, буде потрібна надалі, вона може бути зчитана з диска у вільну область пам'яті і знову запущена на виконання.

Для того, щоб бути програмою, що переміщається динамічно, вона не повинна змінювати вміст сегментних

регістрів, тобто всі зміщення в програмі повинні бути відносними щодо фіксованих значень, що утримуються в сегментних регістрах. Це дозволяє пересувати програму в будь-яке місце пам'яті, змінюючи значення сегментних регістрів і встановлюючи тим самим покажчики нових базових адрес.

Реалізація стека. Стек у системі, заснованій на МП К1810ВМ86, розміщається в пам'яті і визначається за допомогою регістра *SS* та регістра покажчика стека.

Система має практично необмежену довжину стека. Стек може мати ємність до 64 Кбайт.

У кожний момент часу тільки один стек безпосередньо адресується і є поточним стеком.

Регістр *SS* містить адресу поточного сегмента стека, регістр *SP* указує вершину стека, тобто містить зміщення відносно базової адреси сегмента стека.

Команди які працюють із стеком, записують у стек і витягають із нього по одному слову за одне звертання до пам'яті. Вміст регістра *SP* автоматично змінюється при цьому на 2 і вказує нову вершину стека. Вершина стека змінюється тільки в результаті змін умісту покажчика стека *SP*.

Зарезервовані комірки пам'яті. Області пам'яті від *0H* до *7FH* (128 байт) і від *FFFF0H* до *FFFFFH* (16 байт) використовуються для організації переривань і системного скидання і не повинні використовуватися в прикладних програмах користувача в інших цілях.

Адресація зовнішньої пам'яті. Мікропроцесор К1810ВМ86 має 20-розрядний канал адреси і може адресувати область пам'яті обсягом до 1 Мбайта. Два будь-яких байти можуть бути обрані як одне слово. Область пам'яті, що адресується, поділена фізично на дві частини, так називані банки розміром по 512 Кбайт (рис. 2.16).

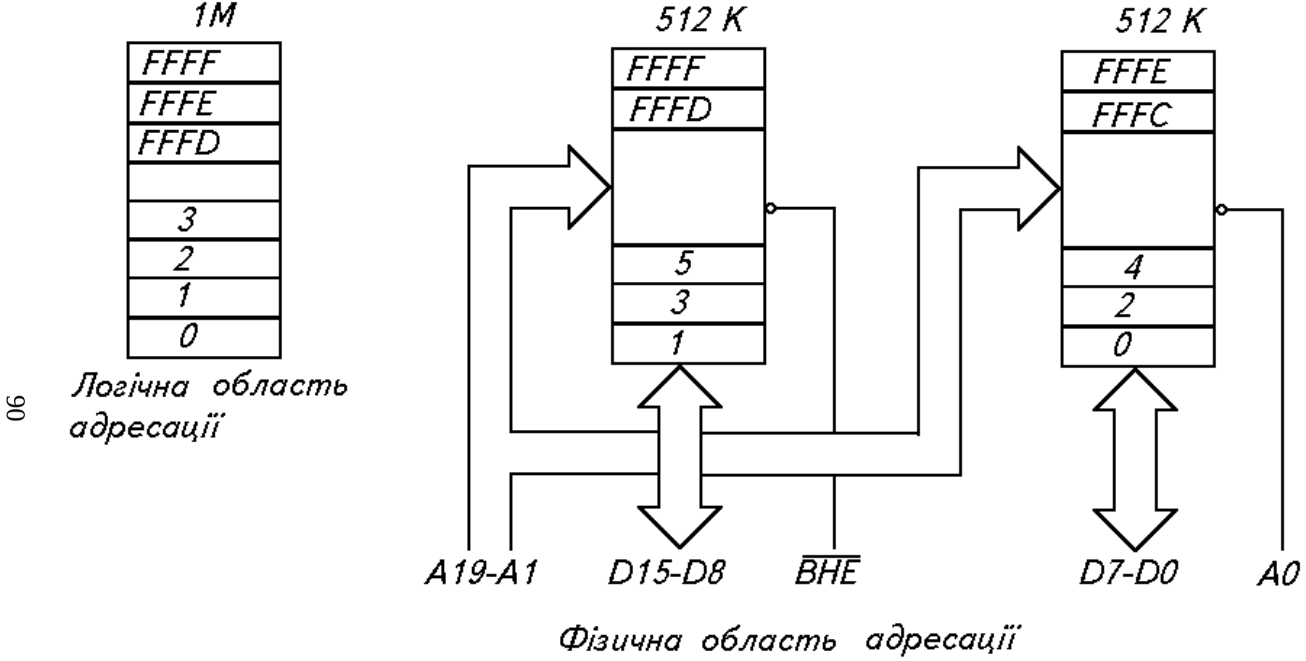

1M
FFFF
FFFE
FFFD
3
2
1
0
Логічна область адресації
512 K
FFFF
FFFD
5
3
1
512 K
FFFE
FFFC
4
2
0
A19-A1
D15-D8
BHE
D7-D0
A0
Фізична область адресації

Рис. 2.16

Один банк пов'язаний із молодшою половиною 16-розрядного каналу даних ЦПП (*D7–D0*), а інший – із старшою (*D15–D8*). Адреса (*A19–A1*) використовується для того, щоб адресувати комірки з байтами даних як у старшому, так і в молодшому банках. Молодший розряд адреси *A0* при адресації пам'яті не використовується, але використовується для вибору банку пам'яті. Молодший банк містить байти, що парноадресуються вибирається коли *A0=0.* Старший банк, у якому зберігаються байти, що непарноадресуємі (*A0=1)*, вибирається по окремому сигналі дозволу старшої половини каналу ($\overline{BHE} = 0$). Механізм вибору банку по сигналах $\overline{BHE}$ і *A0* розшифровується в табл. 2.13.

Таблиця 2.13

$\overline{BHE}$	*A0*	Байт, що пересилається
0	0	Обидва байти
0	1	Старший байт у/із непарної адреси
1	0	Молодший байт у/із парної адреси
1	1	Нічого

При звертанні до байта даних, розташованого по парній адресі (рис. 2.17), цей байт пересилається в молодший банк або передається з нього по молодшій половині каналу даних (*D7–D0*).

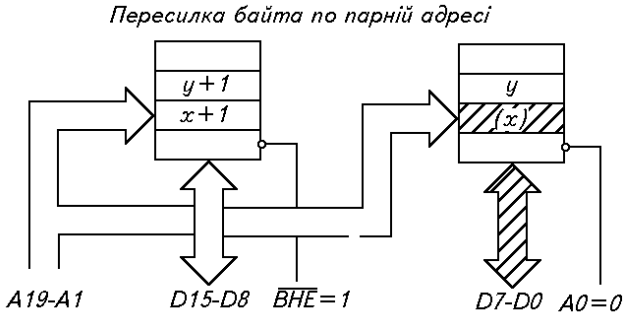


Рис. 2.17

В цьому випадку розряд адреси *A0=0* дозволяє вибірку байта, які адресується, з молодшого банку $\overline{BHE} = 1$, а сигнал забороняє вибірку байта, що адресується, зі старшого банку. І навпаки, при звертанні до байта, розташованого по непарній адресі, цей байт пересилається в старший банк або передається з нього по старшій половині каналу даних (*D15–D8*). Активний рівень сигналу $\overline{BHE} = 0$ дозволяє звертання до старшого банку, а активний рівень сигналу *A0 (A0=1)* забороняє звертання до молодшого банку, як показано на рис. 2.18.

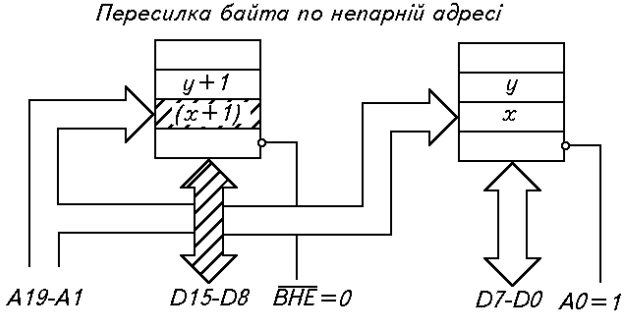


Рис. 2.18

З даних, наведених у табл. 2.13, випливає, що МП К1810ВМ86 може вибирати байти одночасно зі старшого і молодшого банку у формі 16-розрядного слова даних. Коли молодший байт слова розташований у комірці з парною адресою, тобто в молодшому банку, говорять, що слово вирівняне на парну межу, і вибірка його здійснюється за один цикл. При вибірці слова, вирівняного на парну межу, адреса *A19–A1* використовується для звертання до відповідних байтів усередині кожного банку, а сигнали *A0* і $\overline{BHE}$ – для одночасного дозволу звертання до обох банків (*A0 = 0,* $\overline{BHE} = 0$). Як проходить вибірка, показано на рис. 2.19.

Якщо відбувається звертання до слова, не вирівняного на парну межу (молодший байт розташований по непар-

ній адресі, тобто знаходиться в старшому банку), то виконується два послідовних цикли каналу. У першому циклі пересилається молодший байт із старшого банку по старшій половині каналу (рис. 2.20). Сигнали $\overline{BHE}$ й *A0* активні (*A0 = 1,* $\overline{BHE} = 0$). В другому циклі каналу адреса слова *A19 - A0* інкременується, і виходить, розряд *A0* стає рівним нулю (вибирається молодший банк і вже адресує таку фізичну комірку пам'яті).

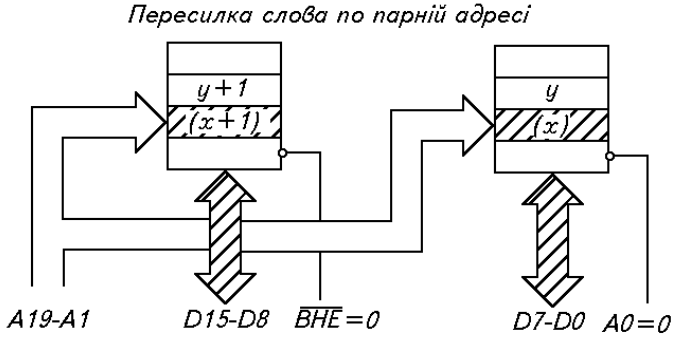


Рис. 2.19

Отже, у другому циклі пересилається старший значущий байт слова даних, обраний по парній адресі з молодшого банку. Мікропроцесор ініціалізує таку послідовність всякий раз, коли виконується вибірка слова по непарній адресі.

Пересилки молодшого і старшого байтів внутрішніх 16-розрядних регістрів ЦПП на відповідні половини каналу даних провадяться автоматично.

Сигнал дозволу старшої половини каналу $\overline{BHE}$ – мультиплексований сигнал, що по часових характеристиках ідентичний адресним сигналам *A19–A16*. Так само, як і адреса сигнал $\overline{BHE}$ фіксується по сигналу *STB*. У тактах Т2 - Т4 вихідний сигнал $\overline{BHE}$ мультиплексується з еквівалентним сигналом стана *ST7*. При читанні байта ЦПП перекла-

дає 16-розрядний канал даних у високоімпедансний стан, хоча байт даних може з'явитися як на старшій, так і на молодшій половині каналу даних. Такий підхід спрощує вимоги процесу вибору пристрою (ПЗП).

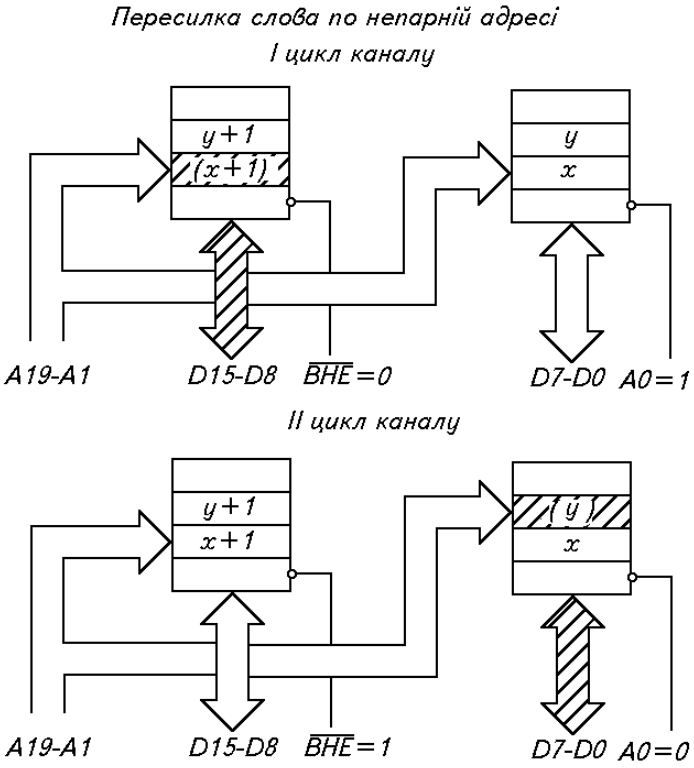


Рис. 2.20

При виконанні запису байта ЦПП формує дані на своїх виходах по всім 16 розрядах каналу даних. Інформація на тій частині каналу, по котрій не потрібно виконувати передачу даних, є невизначеною. Мікропроцесор К1810ВМ86 забезпечує адресацію 64К 8-розрядних або 32К 16-розрядних пристроїв вводу/виводу (регістрів). Область вводу/виводу не сегментується на відміну від області пам'я-

ті. Команди вводу/виводу *IN* і *OUT* виконують передачу даних між акумулятором (*AL* – для передачі байта, *AX* – для передачі слова) і регістрами області вводу/виводу. При звертанні до регістра вводу/виводу МП видає адресу регістра (0 – 64К) на канал адреси *A0* - *A15*. Команди вводу-виводу, у яких адреса вводу/виводу задається постійним значенням у полі команди, можуть адресувати тільки 256 пристроїв вводу/виводу. Всю область адресації (64К) використовують команди, у яких адреса вводу/виводу зазначена у регістрі *DX*.

Принцип адресації пристрою вводу/виводу аналогічний принципу адресації зовнішньої пам'яті.

Адресація вводу/виводу. Восьми розрядні ПВВ можуть бути залучені до старшої або молодшої половини каналу даних. Для рівномірного розподілу навантаження необхідно до обох половин каналу підключати рівне число ПВВ. Якщо пристрій підключений до старшої половини каналу даних (*D15–D8)*, то усі адреси вводу/виводу повинні бути непарними (*A0=1)*. Якщо пристрій підключений до старшої половини каналу даних (*D7–D0)*, то усі адреси вводу/виводу повинні бути парними (*A0=0)*. При такому розподілі адресного простору забезпечується просте керування напрямком передачі 8-розрядних даних по старшій (що адресується непарно) або по молодшій (що адресується парно) половинам 16-розрядного каналу даних. А оскільки для кожного окремого ПВВ розряд адреси *A0* завжди повинний дорівнювати 0 або 1, то розряд адреси *A0* можна не використовувати в якості розряду адреси при виборі регістра в окремому пристрої. Якщо ПВВ на старшій й ПВВ на молодшій половинах каналу даних привласнені адреси, що відрізняються тільки в розряді адреси *A0* (сусідні непарні і парні адреси), то сигнали $\overline{BHE}$ й *A0* можуть бути використані для формування умови вибору пристрою, що дозволяє спрямувати помилковий запис в інший пристрій.

При використанні 16-розрядних ПВВ із метою підвищення їхньої ефективності і спрощення процесу вибору пристрою необхідно привласнювати їм парні адреси. Щоб забезпечити вибір ПВВ тільки для 16-розрядних пересилок, варто використовувати сигнали *A0* і $\overline{BHE}$ в коді вибору пристрою.

2.6 Режими адресації

Питання адресації пам'яті мікропроцесора К1810ВМ86 має два аспекти.

1. Як формуються адреси пам'яті.
2. Які є режими адресації.

Будь-яка адреса пам'яті обчислюється шляхом підсумовування вмісту сегментного регістра і виконавчої адреси. Виконавча адреса обчислюється різноманітними засобами в залежності від режиму адресації, як у більшості інших мікропроцесорів. Вміст належного сегментного регістра зсовує вліво на чотири біта й докладається до виконавчої адреси. Результат є фізичною адресою пам'яті.

Так формується 20-бітова адреса пам'яті, що дає можливість безпосередньо адресувати 1048576 байт зовнішньої пам'яті.

Отже, адреса в процесорі К1810ВМ86 складається з двох різноманітних адрес: умісту сегментного регістра та виконавчої адреси, що називається також зміщенням.

Сегментні регістри мікропроцесора К1810ВМ86 не схожі на інші регістри. Вони є базовими регістрами, за допомогою яких може бути зазначена будь-яка комірка пам'яті, що лежить на адресній межі, тобто на межі 16 байтів.

Кожний сегментний регістр указує на початок блока пам'яті, довжиною 65436 байтів. Оскільки в мікропроцесорі К1810ВМ86 є чотири сегментних регістри, то одночасно в пам'яті може розміститися чотири сегменти розміром 65536 байтів. Для формування фізичної адреси завжди вибирає-

ся належний сегментний регістр. На вміст сегментного регістра не накладається ніяких обмежень. Таким чином, пам'ять процесора К1810ВМ86 не розбивається на фізичні сторінки розміром 65536 байтів і сегменти можуть перекриватися. Кожний сегментний регістр указує на початок сегмента пам'яті, розміром 65536 байтів, що може розташовуватися в будь-якому місці фізичної пам'яті. Цей сегмент може перекриватися або не перекриватися з іншими сегментами.

Режими адресації мікропроцесора 1810ВМ86 підрозділяються на два типи:

1. Режим адресації пам'яті програми.
2. Режим адресації пам'яті даних.

2.6.1 Режими адресації пам'яті програми

Щораз, коли здійснюється вибірка команди, попередньо обчислюється адреса комірки пам'яті, у якій вона зберігається. Ця адреса визначається як сума лічильника програми (він називається ще і регістром *IP*) і адреси сегмента, одержуваного із сегментного регістра *CS*. Як правило, уміст регістра *IP* збільшується на 1 при виконанні команди. Проте, за допомогою команд *JMP* і *CALL* можна перевизначати уміст цього регістра.

Адресувати пам'ять програм можна трьома способами.

Відносна адресація. 8- або 16- бітове зміщення вибирається безпосередньо з команди і добавляється до вмісту регістра *IP* відповідно до правил додавання чисел із знаком. Ця операція не впливає на вміст сегментного регістра *CS*, тому її називають внутрисегментною.

Пряма адресація. Нова адреса утримується в команді у формі безпосередніх даних і завантажується в лічильник програми і регістр *CS*. Ця операція називається межсегментною.

Непряма адресація. Будь-які опції адресації пам'яті даних (про які мова йтиме нижче) можна використовувати для читання даних із пам'яті. При цьому дані, що зберігаються у відповідних комірках пам'яті або регістрах інтерпретуються командами *JMP* і *CALL* як адреси пам'яті. Є два варіанти непрямої адресації. Зчитується одне 16-бітове слово. Воно завантажується в лічильник програми і команда *JMP* або *CALL* передає керування на цю адресу усередині поточного сегмента. В іншому випадку зчитуються два 16-бітових слова даних, що завантажуються в лічильник програми і сегментного регістра. В результаті можна звертатися до будь-якої комірці, що лежить в іншому сегменті за допомогою команд *JMP* і *CALL*.

2.6.2 Режими адресації пам'яті даних

У мікропроцесорі К1810ВМ86 є велика розмаїтість засобів адресації пам'яті даних. Ми роздивимося 6 основних:

1. Безпосередню адресацію.
2. Пряму адресацію.
3. Пряму адресацію з індексуванням.
4. Непряму адресацію.
5. Адресацію по базі.
6. Стекову адресацію.

Безпосередня адресація. У цьому режимі адресації один з операндів записується в байт (байти), розташовані безпосередньо за кодом операції. Якщо байти, на які вказує другий операнд, розташовані безпосередньо за кодом операції, то безпосередні дані розташовуються в слід за ними. Наприклад,

ADD AX, 3064h

формує команду додавання, що додає *3064H* до вміста регістра *AX*. Схема формування адреси показана на рис. 2.21.

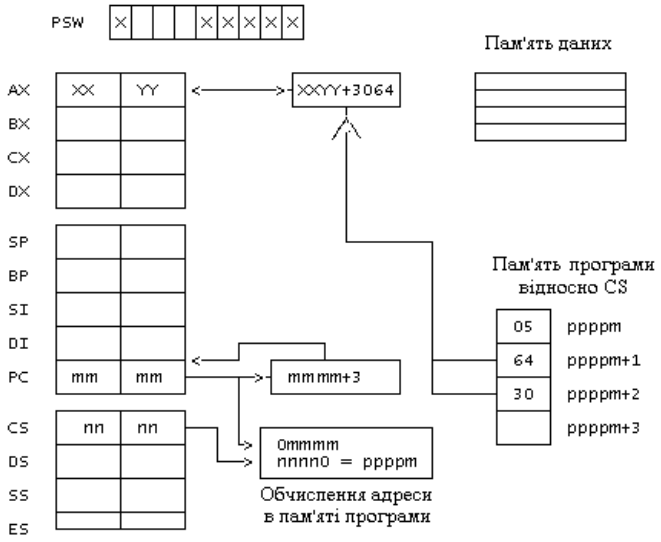


Рис. 2.21

Слід зазначити, що безпосередній 16-бітовий операнд зберігається в пам'яті програми так, що молодший байт передує старшому. Це цілком відповідає способу збереження безпосередніх даних у пам'яті програми. Крім того, аналогічно зберігаються операнди й у пам'яті даних. Коли 16-бітове слово потрапляє в пам'ять, молодші 8 бітів даних зберігаються в молодшому байті пам'яті, а старші 8 бітів - у наступному байті.

У наведеному вище прикладі 2 байта, наступні безпосередньо за кодом команди додавання, додаються до вмісту регістру AX.

Пряма адресація пам'яті. У мікропроцесорі К1810ВМ86 є режим прямої або абсолютної адресації. У цьому режимі вміст двох байтів об'єктного коду команди додається до вміста регістра *DS* і в такий спосіб формується адреса фізичної пам'яті рис. 2.22.

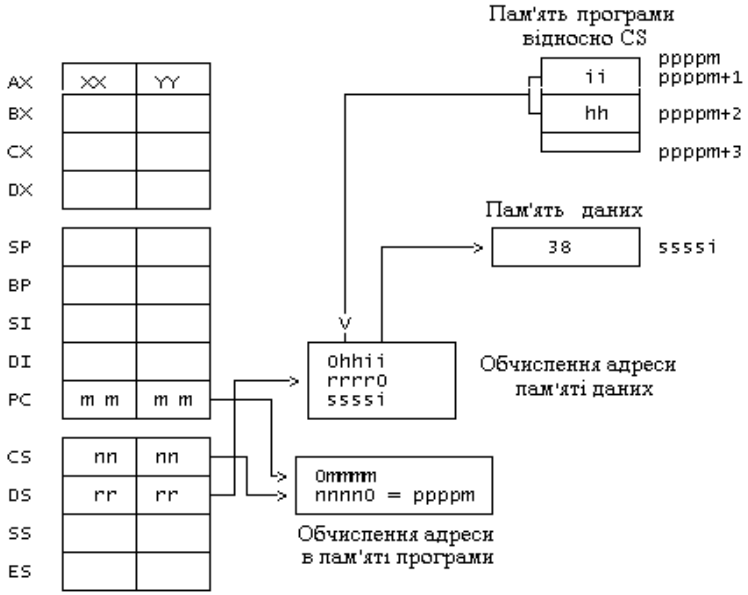


Рис. 2.22

Слід зазначити, що в 16-бітовому адресному зміщенні, що зберігається в оперативній пам'яті, молодший байт передує старшому. Це відповідає способу збереження адрес у пам'яті програм мікропроцесора 1810ВМ86.

У регістрі *DS* повинна бути базова адреса сегмента пам'яті даних (дивися рис. 2.22).

Пряма адресація з індексуванням. Пряма адресація з індексуванням здійснюється коли в якості індексного регістра вказується регістр *SI* або *DI*. При цьому до вмісту обраного індексного регістра додається або 8, або 16 бітів зміщення й у результаті формується виконавча адреса.

16-бітове зміщення зберігається в двох байтах об'єктного коду команди. Молодший байт зміщення в пам'яті розташований раніш старшого байта, аналогічно тому, як це було в режимі прямої адресації. Якщо використовується 8-бітове зміщення, то старший біт молодшого байта розмно-

жується в старший байт, формуючи, таким чином, 16-бітове зміщення. Ця процедура показана нижче:

Зміщення	10110101	01101011
Розмноження знакового біта	11111111 10110101	00000000 01101011

Схема формування прямої адреси з індексуванням показана на рис. 2.23.

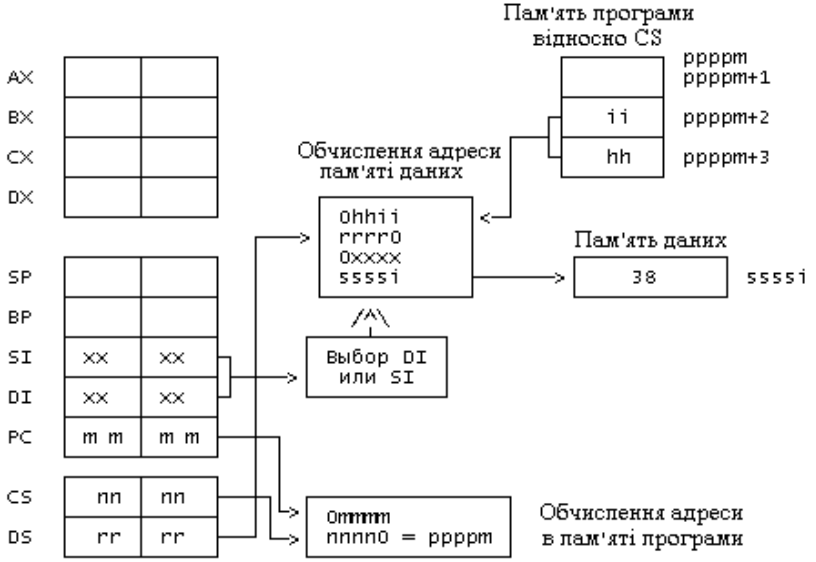


Рис. 2.23

HHII є 16-бітовим або 8-бітовим зміщенням, що витягається з пам'яті програми. *XXXX* є індексом, що витягається або з регістра *SI* або з регістра *DI*.

Непряма адресація. Непряма адресація пам'яті реалізується в мікропроцесорі К1810ВМ86 як засіб, похідний від прямої адресації з індексуванням. Якщо не зазначене зміщення при використанні прямої адресації з індексуванням, то насправді буде мати місце непряма адресація за допомогою регістрів *SI* або *DI* рис. 2.24.

(Можна замість *CS*, *SS* або *ES* використовувати *DS*, виконавши додаткову однобайтову команду).

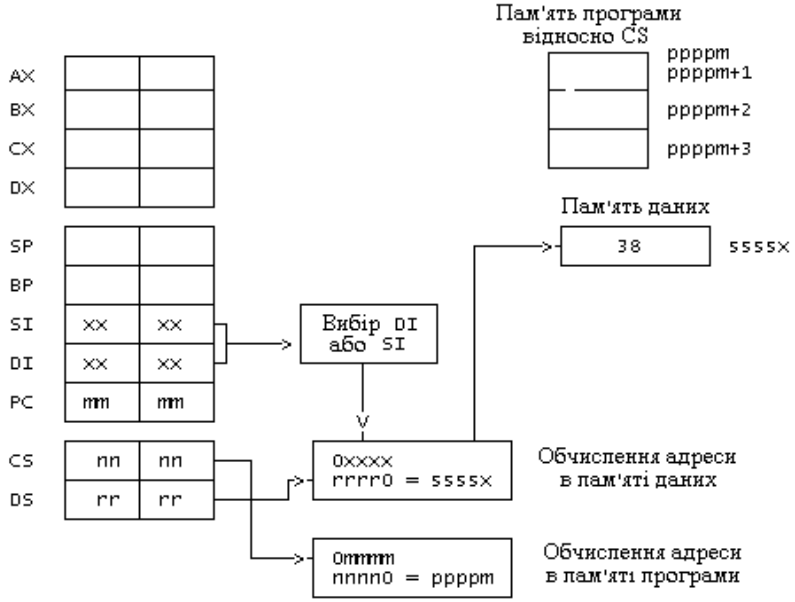


Рис. 2.24

Адресація по базі. У мікропроцесорі 1810ВМ86 приймаються два способи адресації по базі:

1. Адресація по базі пам'яті даних, що має місце всередині сегмента *DS* (пам'яті даних).

2. Адресація в стеку, що має місце в сегменті *SS* (стеку).

Якщо використовується адресація по базі пам'яті даних, то основою для формування виконавчої адреси є регістр *BX*. Всі вже названі режими адресації (крім безпосереднього режиму) можуть використовуватися в сполученні з базуванням пам'яті даних. При використанні базування вміст регістра *BX* додається до виконавчї адреси, яка була би отримана без базування. Схема формування адреси показана на рис. 2.25.

(Можна замість *CS*, *ES* або *SS* використовувати *DS*, виконавши додаткову однобайтну команду).

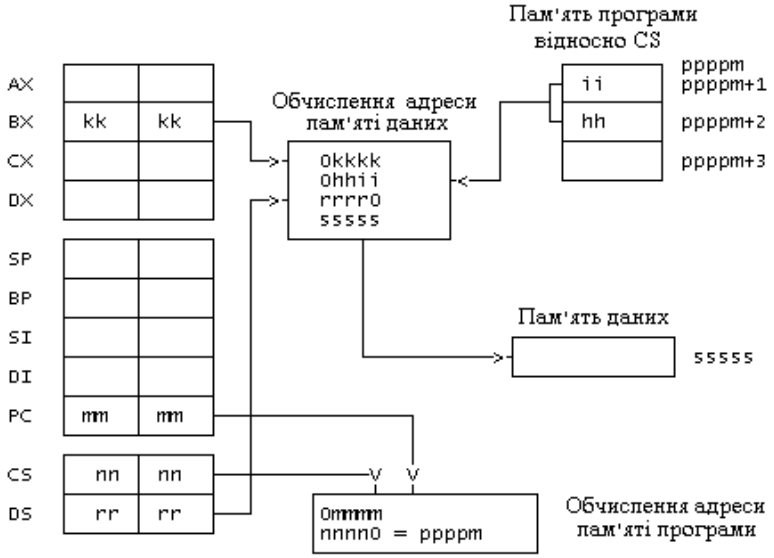


Рис. 2.25

Якщо використовується звичайна пряма адресація, то завжди формується 16-бітове зміщення. У режимі адресації по базі припускається 8- або 16-бітове зміщення (на схемі *hhii*), причому знаковий розряд поширюється на старший байт. У цьому режимі зміщення може навіть зовсім бути відсутнім.

У режимі індексної адресації з базуванням для формування виконавчої адреси вміст регістра *BX* додається до вмісту відповідного індексного регістра. Ця процедура показана на рис. 2.26.

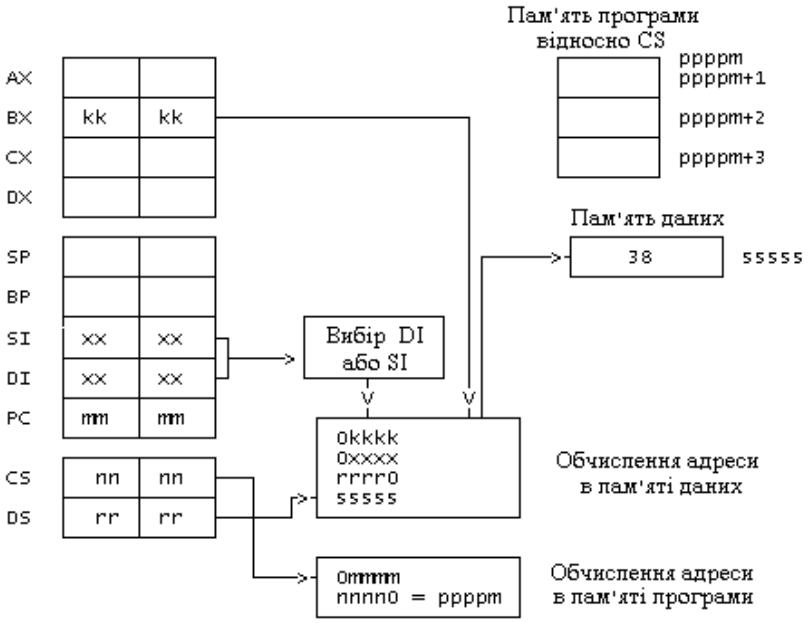


Рис. 2.26

Може показатися, що пряма індексна адресація з базуванням є занадто складною, хоча насправді це не так. Ми просто додаємо вміст регістра *BX* до виконавчої адреси, що обчислюється відповідно до правил прямої адресації з індексуванням. Схема формування прямої адреси з індексуванням і базуванням для пам'яті даних показана на рис. 2.27.

Індекс *xxxx* у наведеному вище прикладі не є обов'язковим. Є режим прямої адресації з базуванням. У цьому випадку вміст регістрів *SI* і *DI* не впливає на процес обчислення адреси і *0xxxx* можна прибрати зі схеми.

Стекова адресація. У мікропроцесорі К1810ВМ86 є стековий варіант адресації пам'яті даних із базуванням. У цьому випадку в якості базового регістра використовується регістр *BP*. На рис. 2.28 наводиться схема прямої стекової адресації

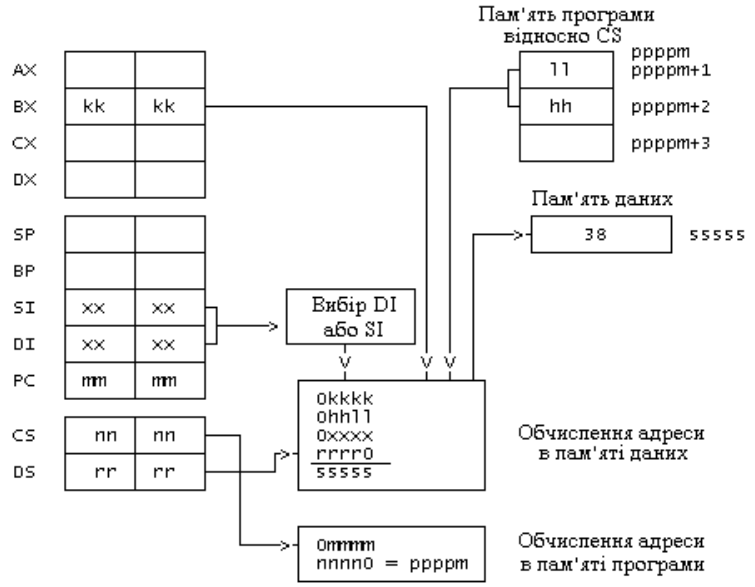


Рис. 2.27

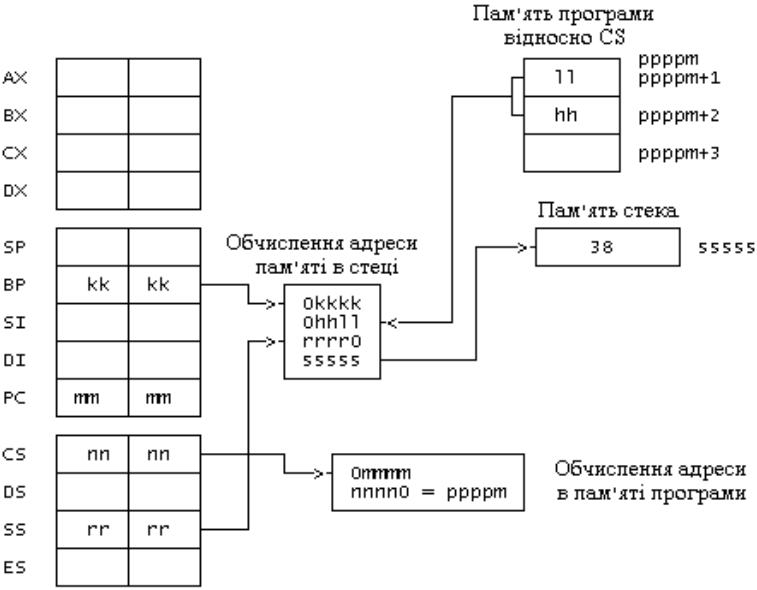


Рис. 2.28

На малюнку зміщення *hhll* є або 16-бітовим, або 8-бітовим із розмноженням знака. Якщо використовується адресація стекової пам'яті з базуванням, то зміщення повинне задаватися, навіть якщо воно дорівнює нулю.

2.6.3 Байт режиму адресації

У мікропроцесорі К1810ВМ86 є велика розмаїтість засобів адресації. Тепер самий час порушити питання, як реалізується зворотній режим адресації в об'єктному коді? Більшість режимів адресації в процесорі К1810ВМ86 задається за допомогою 1 байта об'єктного коду, що називається байтом режиму адресації. З байтом режиму адресації може бути пов'язано декілька додаткових байтів, що визначають зміщення. Байт режиму адресації завжди є другим байтом команди. Винятком є ситуація, коли використовується префікс.

Байт режиму адресації має таку структуру (рис. 2.29):

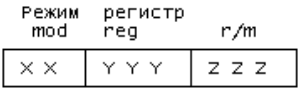


Рис. 2.29

Поле *XX* складається з двох бітів. Це поле призначене для вказівки об'єкта, який адресується, тобто регістра або пам'яті. У випадку, коли адресується пам'ять, це поле визначає скільки байтів, наступних за байтом режиму адресації приділяється для запису зміщення.

Поле *YYY* складається з трьох бітів. Це поле використовується для вказівки регістра, що застосовується в операції. У деяких випадках ці три біти використовуються для уточнення команди.

Поле *ZZZ* складається з трьох бітів. Це поле використовується разом із полем *mod* для визначення режиму адресації.

Поле *mod*. Це поле може мати таку конфігурацію бітів:

– 00 – адресується пам'ять. Поле *mod* цілком визначає режим адресації. Байти зміщення відсутні.

– 01 – адресується пам'ять. Поле *mod* цілком визначає режим адресації. Є один байт зміщення. Зміщення як число зі знаком із діапазону від -128 до 127. При обчисленні виконавчої адреси відбувається поширення знакового біта аж до 16 розряду. Для обчислення адреси використовуються такі байти:

mod	reg	r/m	зміщення

Тут *mod* = 01, а зміщення є 8-бітовим числом із знаком.

– 10 – адресується пам'ять. Поле r/m задає режим адресації. Є два байти зміщення. Перший байт зміщення – це молодші 8 бітів зміщення, а другий байт зміщення – старші 8 бітів. Коли ці байти беруть участь в адресній арифметиці, то їхній вміст трактується як 16-бітове число без знака. Для обчислення адреси використовуються такі байти (рис. 2.30):

mod reg r/m	молодші біти зміщення	старші біти зміщення

Рис. 2.30

Тут *mod* = 10, а в наступних 2 байтах зберігається зміщення.

– 11 – адресується регістр. Поле *r/m* визначає цей регістр. Для вказівки того, який 8- або 16-бітовий регістр використовується служить біт *W* у коді операції.

Поле *reg*. Це поле використовується разом із додатковим бітом, тобто бітом *W*, що довизначає регістр. Біт *W* є частиною коду операції і служить для вибору розрядності

операції, тобто чи працює команда з 8- або 16-розрядними даними (табл. 2.14).

Таблиця 2.14

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*

Поле *r/m.* Поле *r/m* разом із полем *mod* визначає режим адресації (табл. 2.15). Тут *disp* позначає зміщення.

Таблиця 2.15 .

r/m	*mod=00*	*mod=01*	*mod=10*	*mod=11*	
				W=0	*W=1*
000	*BX+SI*	*BX+SI+disp*	*BX+SI+disp*	*AL*	*AX*
001	*BX+DI*	*BX+DI+disp*	*BX+DI+disp*	*CL*	*CX*
010	*BP+SI*	*BP+SI+disp*	*BP+SI+disp*	*DL*	*DX*
011	*BP+DI*	*BP+DI+disp*	*BP+DI+disp*	*BL*	*BX*
100	*SI*	*SI+disp*	*SI+disp*	*AH*	*SP*
101	*DI*	*DI+disp*	*DI+disp*	*CH*	*BP*
110		*BP+disp*	*BP+disp*	*DH*	*SI*
111	*BX*	*BX+disp*	*BX+disp*	*BH*	*DI*

Таблиця 2.15. не потребує пояснень за винятком випадку прямої адресації. Коли поле *mod* містить *00*, а поле *r/m* містить 110, адреса зміщення вибирається безпосередньо з двох байтів, що розташовуються слідом за байтом режиму адресації. Це показано на рис. 2.31.

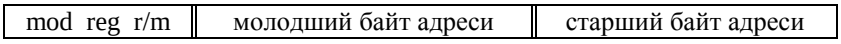


Рис. 2.31

Тут *mod = 00, r/m = 110,* молодший байт адреси містить 8 молодших бітів зміщення, а старший байт адреси - 8 старших бітів.

Перепризначення сегментів. Кожному режиму адресації відповідає за умовчанням деякий сегментний регістр. У більшості випадків можна користуватися й ін. сегментним регістром. Для цього є байт префікса. Щоб скористатися цим байтом його потрібно помістити перед командою, для котрої потрібно скасувати правило вибору сегментного регістра, що діє за умовчанням.

Рис. 2.32

Два бити – *rr* задають сегментний регістр, що використовується в наступній за префіксом команді.

Значення *rr* задає наступні сегменти.

Rr = 00 для регістра *ES*.

Rr = 01 для регістра *CS*.

Rr= 10 для регістра *SS*.

Rr= 11 для регістра *DS*.

Перепризначенням сегментних регістрів не можна користуватися в наступних 3 випадках:

1. Командах звертання до стека (наприклад, у командах *PUSH* і *CALL*) у яких для обчислення виконавчої адреси застосовується регістр *SP* (покажчик стека) необхідно завжди користуватися сегментним регістром *SS*.

2. У командах роботи з рядками, коли використовується регістр *DI*, завжди варто користуватися сегментним регістром *ES*. Якщо в команді опрацювання рядків використовуються обидва регістри *SI* і *DI*, то байт префікса впливає на сегментний регістр для адреси, що обчислюється на підставі *регістра* SI.

3. Для зміни сегментного регістра, за допомогою якого здійснюється вибірка команд. Вибірка команд завжди виконується відносно регістра *CS*.

Зведена таблиця адресації. У таблиці 2.16 зібрана інформація про режими адресації та байті режиму адресації.

Таблиця 2.16

r/m	*mod=00*	*mod=01*	*mod=10*
000	Індексування і базування *BX+SI*	Пряма адресація з індексуванням і базуванням *BX+SI+disp*	Пряма адресація з індексуванням і базуванням *BX+SI+disp*
001	Індексування і базування *BX+DI*	Пряма адресація з індексуванням і базуванням *BX+DI+disp*	Пряма адресація з індексуванням і базуванням *BX+DI+disp*
010	Стекова адресація з індексуванням і базуванням *BP+SI*	Стекова пряма адресація з індексуванням і базуванням *BP+SI+disp*	Стекова пряма адресація з індексуванням і базуванням *BP+SI+disp*
011	Стекова адресація з індексуванням і базуванням *BP+DI*	Стекова пряма адресація з індексуванням і базуванням *BP+DI+disp*	Стекова пряма адресація з індексированням і базуванням *BP+DI+disp*
100	Непряма *SI*	Пряма індексна *SI+disp*	Пряма індексна *SI+disp*
101	Непряма *DI*	Пряма індексна *DI+disp*	Пряма індексна *DI+disp*
110	Пряма	Пряма стекова адресація з базуванням *BP+disp*	Пряма стекова адресація з базуванням *BP+disp*
111	Базова *BX*	Пряма адресація з базуванням *BX+disp*	Пряма адресація з базуванням *BX+disp*

Слід зазначити, що дуже часто один з операндів команди вибирається з пам'яті, а другий – із регістра центрального процесора. Не менш часто використовуються операції, у яких обидва операнди вибираються з регістрів. Мікропроцесор К1810ВМ86 не має у своєму розпорядженні команд, у яких обидва операнди вибираються з пам'яті. Виняток складають декілька команд, призначених для опрацю-

вання рядкових даних. Таким чином, можливі наступні сполучення:

Джерело	Призначення	Результат
Регістр ЦП	Регістр ЦП	Регістр ЦП
Комірка пам'яті	Регістр ЦП	Регістр ЦП
Регістр ЦП	Комірка пам'яті	Комірка пам'яті

2.7 Система команд мікропроцесора К1810ВМ86

2.7.1 Формати команд

Формати команд у значній мірі залежать від типу команд, використовуваного методу адресації, розміру безпосередніх даних і можуть складатися з одного, двох, трьох, чотирьох, п'ятьох і шести байтів (рис. 2.33). В усіх командах перший байт містить код виконуваної операції.

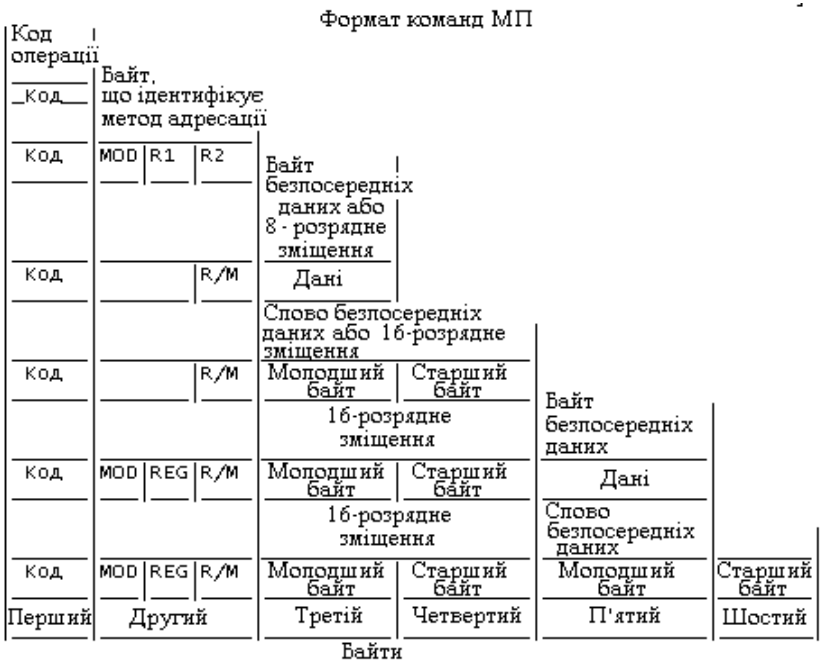


Рис. 2.33

Однобайтні команди задають операцію, яка виконується над вмістом якогось регістра.

Двобайтні команди є командами груп регістр-регістр і вводу-виводу. Другий байт визначає адреси регістра-джерела, регістра-приймача і задає засіб адресації, який використовуваний.

Трибайтні команди містять у третьому байті або безпосередні дані, або, у командах звертання до пам'яті, 8-розрядне зміщення. Другий байт задає місце розташування операнда (регістр або пам'ять).

Чотирибайтні команди містять у третьому і четвертому байтах або слово безпосередніх даних, або, у командах звертання до пам'яті, 16-розрядне зміщення. Другий байт команди задає місце розташування операнда.

П'ятибайтні команди містять у третьому й у четвертому байтах 16-розрядне зміщення, а в п'ятому - безпосередні дані. Другий байт визначає метод адресації у команді, який використовується.

Шестибайтні команди, на відміну від пятибайтних, містять у п'ятому і шостому байтах слово безпосередніх даних. У іншому шестибайтний формат аналогічний п'ятибайтному.

2.7.2 Команди мікропроцесора К1810ВМ86/88

AAA – корекція коду ASCII при додаванні

Якщо *((AL) and 0FH) >9* або *(AF)=1*, то
(AL) ← *(AL)+6*
(AH) ← *(AH)+1*
(CF) ← *1*
(AF) ← *1*
(AL) ← *(AL) and 0FH*

По цій команді вміст регістра *AL* перетвориться в число в двійково-десятковому неупакованому форматі з нульовими старшими чотирма бітами. Вміст регістра *AL* являє

собою результат додавання двох чисел у двійково-десятковому неупакованому вигляді. Команда впливає на прапори *AF, CF*. Стани прапорів *OF, SF, ZF, PF* не визначені.

AAD – корекція коду ASCII при діленні

(AL) ← частка від ділення *(AL) : 10*
(AH) ← залишок від ділення *(AL) : 10*

Вміст регістра *AX* розглядається як двійково-десяткове число. По команді *AAD* це число перед діленням на неупакований дільник перетвориться в нормальне беззнакове ціле число для одержання частки в правильному неупакованому двійково-десятковому форматі. Для одержання правильного результату в наступній команді *DIV* регістр *AH* повинний завжди містити нуль. Команда впливає на прапори *PF, SF, ZF*. Стан прапорів *AF, CF, OF* не визначено.

AAM – корекція коду ASCII при множенні

(AL) ← *(AH) × 10 + (AL)*
(AH) ← *0*

Команда працює з неупакованими двійково-десятковими числами. Вміст регістра *AX* перетвориться в правильне двійково-десяткове число. Вміст регістра *AX* розглядається як результат множення двох неупакованих двійково-десяткових чисел. Для того щоб корекція результату провадилася правильно, старші полубайти операндів, що перемножуються, повинні бути рівні нулю. Команда впливає на прапори *PF, SF, ZF*. Стан прапорів *AF, CF, OF* не визначено.

AAS – корекція коду ASCII при відніманні

Якщо *((AL) and 0FH) >9* або *(AF)=1*, то
(AL) ← *(AL) - 6*
(AH) ← *(AH) - 1*
(CF) ← *1*
(AF) ← *1*
(AL) ← *(AL) and 0FH*

Операндами команди є неупаковані двійково-десяткові числа. По команді вміст регістра *AL* перетвориться в неупаковане двійково-десяткове число з нульовими старшими чотирма бітами. Вміст регістра *AL* розглядається як результат віднімання двох неупакованих двійково-десяткових чисел. Команда впливає на прапори *AF, CF*. Стани прапорів *OF, SF, ZF, PF* не визначені.

ADC – додавання двох операндів із переносом

(1-й операнд) ← (1-й операнд)+ (2-й операнд)+ *CF*
ADC reg8/mem8,reg8
ADC reg16/mem16,reg16
ADC reg8,reg8/mem8
ADC reg16,reg16,mem16
ADC reg8/mem8,data8
ADC reg16/mem16,data16

По команді здійснюється додавання першого і другого операндів і до результату додається значення прапора переносу *CF* (0 або 1). Результат записується в комірки збереження першого операнда, а *початкове* значення першого операнда губиться. Другий операнд залишається незмінним.

Перший операнд може зберігатися в регістрі або комірці пам'яті. Другий операнд може бути заданий у регістрі, комірці пам'яті або безпосередньо числовим значенням. Не дозволяється задавати операнди в регістрах сегментів, а також збереження (запис) двох операндів одночасно в комірках пам'яті. Операнди можуть бути байтами або словами, представляти числа зі знаком або без знака. Оскільки команда *ADC* використовує прапор переносу *CF* , то вона може застосовуватися для додавання чисел, довжина яких перевищує 16 біт.

Приклади запису команди на мові Асемблера:

```
ADC     [BX],BL
ADC     DH,DL
ADC     [SI+12],DX
ADC     DX,SI
ADC     CL,VAR_BYTE
```

```
ADC     CX,[DI]
ADC     AL,03
ADC     AX,-1
ADC     DH,10
ADC     BYTE PTR [DI+0F23H],1
ADC     BP,2211H
ADC     WORD PTR [SI],2301H
ADC     CH,10
ADC     BX,01
ADC     WORD PTR VAR2,03
```

ADD – додавання двох операндів

(1-й операнд) ← (1-й операнд)+ (2-й операнд)

```
ADD reg8/mem8,reg8
ADD reg16/mem16,reg16
ADD reg8,reg8/mem8
ADD reg16,reg16,mem16
ADD reg8/mem8,data8
ADD reg16/mem16,data16
```

По команді здійснюється додавання першого і другого операндів. Результат записується в комірки збереження першого операнда, а *початкове* значення першого операнда губиться. Другий операнд залишається незмінним.

Перший операнд може зберігатися в регістрі або комірці пам'яті. Другий операнд може бути заданий у регістрі, комірці пам'яті або безпосередньо числовим значенням. Не дозволяється задавати операнди в регістрах сегментів, а також збереження (запис) двох операндів одночасно в комірках пам'яті. Операнди можуть бути байтами або словами, представляти числа зі знаком або без знака.

Приклади запису команди на мові Асемблера:

```
ADD     [BX],BL
ADD     DH,DL
ADD     [SI+12],DX
ADD     DX,SI
ADD     CL,VAR_BYTE
ADD     CX,[DI]
ADD     AL,03
ADD     AX,-1
```

```
ADD     DH,10
ADD     BYTE PTR [DI+0F23H],1
ADD     BP,2211H
ADD     WORD PTR [SI],2301H
ADD     CH,10
ADD     BX,01
ADD     WORD PTR VAR2,03
```

AND – логічне множення двох операндів

(1-й операнд) ← (1-й операнд) *and* (2-й операнд)

(CF) ← 0

(OF) ← 0

```
AND reg8/mem8,reg8
AND reg16/mem16,reg16
AND reg8,reg8/mem8
AND reg16,reg16,mem16
AND reg8/mem8,data8
AND reg16/mem16,data16
```

По команді здійснюється логічне множення першого і другого операндів. Результат записується в комірки збереження першого операнда, а *початкове* значення першого операнда губиться. Другий операнд залишається незмінним.

Перший операнд може зберігатися в регістрі або комірці пам'яті. Другий операнд може бути заданий у регістрі, комірці пам'яті або безпосередньо числовим значенням. Не дозволяється задавати операнди в регістрах сегментів, а також збереження (запис) двох операндів одночасно в комірках пам'яті. Операнди можуть бути байтами або словами. Після виконання цієї команди прапори *CF, OF* завжди рівні нулю

Приклади запису команди на мові Асемблера:

```
AND     [BX],BL
AND     DH,DL
AND     [SI+12],DX
AND     DX,SI
AND     CL,VAR_BYTE
AND     CX,[DI]
AND     AL,03
```

```
AND     AX,-1
AND     DH,10
AND     BYTE PTR [DI+0F23H],1
AND     BP,2211H
AND     WORD PTR [SI],2301H
AND     CH,10
AND     BX,01
AND     WORD PTR VAR2,03
```

CALL – виклик процедури (підпрограми)

Межсегментний виклик:

(SP) ← (SP) - 2
(Вершина стека) ← (CS)
(SP) ← (SP) - 2
(Вершина стека) ← (IP)
(CS) ← Сегмент процедури
(IP) ← Зміщення процедури

Внутрисегментний виклик:

(SP) ← (SP) - 2
(Вершина стека) ← (IP)
(IP) ← Зміщення процедури

```
CALL    FAR LABEL  ; межсегментний виклик
CALL    NEAR LABEL ;усередині сегментний виклик
CALL    reg16/mem16 ;усередині сегментний
CALL    mem16 ;межсегментний виклик
```

По цій команді в стандартному вигляді здійснюється безумовна передача керування процедурі (підпрограмі). Адреса повернення записується в стек (на його вершину), що дає можливість команді процедур *RET* (повернення з процедури) повернути керування наступній за *CALL* команді.

Команда CALL має чотири різноманітні модифікації:

– внутрисегментна передача керування з прямою адресацією;

– внутрисегментна передача керування з непрямою адресацією;

– межсегментна передача керування з прямою адресацією;

– межсегментна передача керування з непрямою адресацією.

Внутрисегментна команда *CALL* здійснює передачу керування тільки усередині поточного 64-килобайтового командного сегмента. Іншими словами, адреса процедури, що викликається, повинна знаходитися усередині діапазону від +32787 до -32768 байт від *адреси* наступної за CALL команди. Для внутрисегментних команд у стеку зберігається вміст тільки регістра *IP*. Перед записом МП автоматично змінює регістр покажчика команд *IP* так, щоб він вказував адресу наступної за *CALL* команди.

У випадку внутрисегментних команд із прямою адресацією для одержання адреси процедури, що викликається, значення 16-бітового зміщення додається до вмісту регістра-покажчика команд *IP*. У цих командах регістр загального призначення (*AX, BX, CX, DX, SI, DI, BP, SP*) або комірка пам'яті можуть бути використані для збереження операнда, що задає адресу процедури, яка викликається .

Межсегментна команда *CALL* може передавати керування будь-якій процедурі в межах усього мегабайтового адресного простору мікропроцесора. При виконанні межсегментних команд *CALL* у стеку зберігається вміст регістрів *CS* і *IP*.

Прямі межсегментні команди *CALL* містять значення кодового сегмента і зміщення процедури, що викликається, в самій команді, і в процесі виконання їх поміщають у регістри *CS* і *IP*. У межсегментних командах передачи керування з непрямою адресацією для запису операндів можуть використовуватися тільки комірки пам'яті. У цьому випадку при виконанні команди вміст подвійного слова пам'яті записується в регістрах *CS* і *IP*.

Приклади запису команди на мові Асемблера:

```
CALL    FAR PTR TICK
CALL    NEXT_PROC
CALL    SI
```

CALL WORD PTR [BX+12]
CALL DWORD PTR [DI]

CBW – поширення знака регістра AL на регістр AH

Якщо *(AL) < 80H*, то *(AH) ← 0*
Якщо *(AL) ≥ 80H*, то *(AH) ← FFH*

Якщо старший біт регістра *AL* дорівнює 1, то вміст регістра *AH* стає рівним *FFH*, у противному випадку воно дорівнює 0.

CLC – очищення прапора переносу

(CF) ← 0

Ця команда встановлює біт переносу рівним 0. Інші прапои не змінюються.

CLD – очищення прапора напрямку

(DF) ← 0

Команда встановлює біт *DF* у нуль. Ця команда застосовується для операції опрацювання рядків з автозбільшенням.

CLI – очищення прапора переривання

(IF) ← 0

Прапор переривання одержує нульове значення. У результаті блокуються всі переривання за винятком "немаскованих" переривань, що надходять по лінії *NMI*.

CMC – інвертування прапора переносу

Якщо *(CF) = 0*, то *(CF) ← 1*
Якщо *(CF) = 1*, то *(CF) ← 0*

Значення прапора переносу інвертується. Інші прапори не змінюються.

CMP – порівняння двох операндів

(1-й операнд) – (2-й операнд)
CMP reg8/mem8,reg8
CMP reg16/mem16,reg16
CMP reg8,reg8/mem8
CMP reg16,reg16/mem16

CMP reg8/mem8,data8
CMP reg16/mem16,data16

Вміст другого операнда при цій команди відраховується з вмісту першого операнда. Результат операції змінює прапори *AF, CF, OF, PF, SF, ZF.* Самі операнди не змінюються. Перший операнд може бути записаний у регістрі або в комірці пам'яті. Другий операнд може бути заданий у регістрі, комірці пам'яті або безпосереднім операндом. Не припускається використовувати для запису операндів сегментний регістр або записувати обидва операнди одночасно в комірки пам'яті. Команда здійснює операції як із словами, так і з байтами.

Приклади запису команди на мові Асемблера:

```
CMP     [BX],BL
CMP     DH,DL
CMP     [SI+12],DX
CMP     DX,SI
CMP     CL,VAR_BYTE
CMP     CX,[DI]
CMP     AL,03
CMP     AX,-1
CMP     DH,10
CMP     BYTE PTR [DI+0F23H],1
CMP     BP,2211H
CMP     WORD PTR [SI],2301H
CMP     CH,10
CMP     BX,01
CMP     WORD PTR VAR2,03
```

CMPS – порівняння рядків

(операнд за адресою в регістрі SI) – (операнд за адресою в регістрі DI).

Якщо *(DF) = 0*, то *(SI)* ← *(SI)+ 1* (байт)
(DI) ← *(DI)+ 1* (байт),
або
(SI) ← *(SI)+ 2* (слово)
(DI) ← *(DI)+ 2* (слово)

Якщо *(DF) = 1*, то *(SI)* ← *(SI) - 1* (байт)
(DI) ← *(DI) - 1* (байт),

або

(SI) ← *(SI) - 2* (слово)

(DI) ← *(DI) - 2* (слово)

Вміст комірки пам'яті, адреса якого знаходиться в регістрі *DI*, відраховується з вмісту комірки пам'яті, адреса якої знаходиться в регістрі *SI*. З результатом віднімання змінюється вміст регістра прапорів процесора, але при цьому не змінюється інформація в комірках пам'яті, що адресуються регістрами *SI* і *DI*. В мові Асемблера команда *CMPS* записується як *CMPSB* при роботі з однобайтовими розмірами і як *CMPSW* в операціях із словами. Якщо прапор *DF* дорівнює нулю, вміст регістрів *SI* і *DI* збільшується на 1 після виконання команди *CMPSB* і на 2 після виконання команди *CMPSW*. Якщо прапор *DF* дорівнює одиниці, уміст регістрів *SI* і *DI* зменшується на 1 або 2 після виконання команд *CMPSB* і *CMPSW* відповідно.

Якщо команда *CMPS* містить префіксну частину *REP*/*REPE*/*REPZ*, то операція повторюється доти, поки вміст регістра *CX* не стане рівним нулю й елементи послідовностей не стануть рівними (у цьому випадку прапор ZF дорівнює *1*). Якщо префіксом є вираження *REPNE*/*REPNZ*, то повторення здійснюється доти, поки вміст регістра *CX* не стане рівним нулю, а елементи рядків стануть нерівними (прапор ZF дорівнює *0*).

CWD – поширення знака з регістра AX у регістр DX

Якщо *(AX) < 8000H*, то *(DX)* ← *0*

Якщо *(AX) ≥ 8000H*, то *(DX)* ← *FFFFH*

По команді *CWD* значення самого старшого (знакового) біта слова в регістрі AX переноситься в регістр *DX*. Якщо знаковий біт регістра *AX* дорівнює 0, то в регістр *DX* записуються нулі, а вміст регістрів стає 32-бітовим позитивним числом. Якщо знаковий біт регістра AX дорівнює 1, то в регістр *DX* записується шістнадцяткове число *FFFFH* і вміст пари регістрів стає негативним 32-розрядним числом. Команда *CWD* не робить впливу ні на які прапори і не має

операндів. Вона може використовуватися для перетворення двобайтового діленого в ділене подвійної довжини при діленні на слово.

DAA – десяткова корекція акумулятора при додаванні

Якщо *((AL) and 0FH) > 9* або *(AF) = 1*, то

(AL) ← (AL) + 6, (AF) ← 1.

Якщо *(AL) > 9FH* або *(CF) = 1*, то

(AL) ← (AL) + 60, (CF) ← 1.

По цій команді виконується операція над упакованим двійково-десятковими числами. Вона впливає на прапори *AF, PF, ZF, SF* і *CF*. Стан прапора *OF* не визначено.

DAS – десяткова корекція при відніманні

Якщо *((AL) and 0FH) > 9* або *(AF) = 1*, то

(AL) ← (AL) - 6, (AF) ← 1.

Якщо *(AL) > 9FH* або *(CF) = 1*, то

(AL) ← (AL) - 60, (CF) ← 1.

По команді *DAS* уміст регістра *AL* заміняється правильним двійково-десятковим числом в упакованому форматі. Команда розглядає вміст регістра *AL* як результат віднімання двох упакованих двійково-десяткових чисел і впливає на прапори *AF, PF, ZF, SF* і *CF*. Стан прапора *OF* не визначено.

DEC – зменшення вмісту регістра або комірки пам'яті

(Операнд) ← (Операнд) -1

DEC reg8/mem8

DEC reg16/mem16

DEC reg16

З умісту зазначеного регістра або комірки пам'яті віднімається одиниця. Операція може бути 8- або 16-бітової.

Команда не впливає на прапор *CF*.

Приклади запису на мові Асемблера:

DEC AX

DEC CH
DEC BYTE PTR TESTB
DEC WORD PTR [SI+23]

DIV – ділення двох операндів

DIV reg8/mem8
DIV reg16/mem16

Для однобайтових операцій виконується ділення вмісту регістра *AX* на операнд. При цьому, якщо *((AX): (Операнд) >FFH)*, генерує переривання типу 0.

(*AL*) ← частка

(*AH*) ← залишок.

Для двухбайтовых операцій виконується ділення *(DX,AX) : (Операнд)*. Якщо *((DX,AX) : (Операнд) > FFFFH)*, генерує переривання типу 0.

(AX) ← частка

(DX) ← залишок.

По команді *DIV* виконується беззнакове ділення вмісту регістра-акумулятора на заданий операнд. При однобайтовому діленні вміст регістра *AX* розглядається як беззнакове 16-розрядне ділене. При двобайтовом діленні вміст пари регістрів *DX*, *AX* розглядається як беззнакове 32-розрядне ділене, де регістр *DX* містить старші 16 розрядів. Операнд може бути заданий у регістрі загального призначення або комірці пам'яті, при цьому операнд розглядається як беззнаковий 8- або 16-бітовий дільник відповідно при однобайтовому і двобайтовому діленні.

При однобайтовому діленні частка записується в регістр *AL*, а залишок – у регістр *AH*. При двобайтовому діленні регістри *AX* і *DX* використовуються для збереження 16-розрядних частки і залишку відповідно, поданих як беззнакові числа.

Після виконання команди *DIV* стани розрядів регістра прапорів процесора не визначені. Якщо значення частки

перевищує розрядність акумулятора (десяткове число 255 при однобайтовому діленні і 65535 при двобайтовому) або починається спроба ділення на нуль, то автоматично генерує переривання типу 0. Якщо відбулося переповнювання, то частка і залишок не визначені. У комірках пам'яті з абсолютними адресами *0000H* і *0003H* утримується подвійне слово, що визначає командний сегмент і адресу, що переміщається, процедури опрацювання переривання нульового типу.

Приклади запису програми на мові Асемблера:

DIV CH
DIV BYTE PTR [SI+09]
DIV CX
DIV WORD PTR XNUM

ESC – видача

ESC opcode,reg8/mem8
ESC opcode,reg16/mem16

Якщо *Mod ≠ 11B*, то шина даних ← *(EA)*

Команда *ESC* забезпечує одержання команд мікропроцесора 1810ВМ86 зовнішніми процесорами, такими, як процесор опрацювання цифрової інформації 1810ВМ87, а також використання ними засобів адресації пам'яті 1810ВМ86. Мікропроцесор виконує операцію, еквівалентну дії команди *NOP* (відсутність операції) у тому випадку, коли операнд команди *ESC* зазначений у регістрі, і операцію псевдочитання, що реалізує установку на інформаційних шинах даних із пам'яті у випадку, коли операнд зберігається в комірці пам'яті.

Приклади запису на мові Асемблера:

ESC 33H,CL
ESC 33H,CX
ESC BYTE PTR 1DH,[SI+234]
ESC WORD PTR 1DH,[BX]

HLT – зупин

Після виконання команди *HLT* процесор зупиняється і залишається в цьому стані до приходу сигналу *RESET* по керуючій шині.

При встановленому в 0 прапорі дозволу переривань *IF* МП відновляє роботу тільки по сигналах немаскованих переривань, що приходять на його вхід *NMI*. Якщо прапор *IF* встановлений у 1, роботу мікропроцесора відновляють як масковані, так і не масковані переривання, що надходять на вхід *INT*. Команда *HLT* не впливає на прапори і не має операндів.

IDIV – цілочислове ділення із знаком

IDIV reg8/mem8

IDIV reg16/mem16

Однобайтове ділення *(AX) : (Операнд).*

Якщо *((AX):(Операнд) > + 127)*, то виробляється переривання типу 0.

Якщо *((AX):(Операнд) < - 127)*, то виробляється переривання типу 0.

(*AL*) ← частка.

(*AH*) ← залишок.

Двухбайтове ділення (*DX,AX):(Операнд).*

Якщо (*DX,AX):(Операнд*) > 32767), то виробляється переривання типу 0.

Якщо ((DX,AX):(Операнд) < -32767), то виробляється переривання типу 0.

(*AX*) ← частка.

(*DX*) ← залишок .

По цій команді виконується ділення вмісту акумулятора на операнд. При однобайтовому діленні вміст регістра AX інтепретується як 16-розрядне ділене зі знаком. При двобайтовом діленні 32-розрядне ділене зі знаком знаходиться в парі регістрів *DX, AX*, причому старші шістнадцять бітів містить регістр *DX*. Операнд може бути записаний у

регістр загального призначення або комірку пам'яті, вміст яких у залежності від типу операції інтепретується як 8- або 16-бітовий дільник із знаком.

У випадку однобайтового ділення частка записується в регістр *AL*, а залишок – у регістр *AH*. При двобайтовому діленні регістри *AX* і *DX* використовуються для збереження частки і залишку відповідно. Обидва результати розглядаються як числа зі знаком.

Після виконання команди *IDIV* стан регістра прапорів процесора не визначено. Якщо значення частки виходить за діапазон [-127, +127] при однобайтовому діленні та [-32767, +32767] при двобайтовом або при спробі ділення на нуль, то автоматично генерується переривання типу 0. У цьому випадку частка і залишок не визначені. У комірках пам'яті з абсолютними адресами 00000H і 00003H утримується інформація, що визначає командний сегмент і адреса процедури, що переміщається, опрацювання переривання типу 0. Дробове число зменшується до цілого числа. Залишок має той же знак, що і ділене.

Приклади запису на мові Асемблера:

IDIV CH
IDIV BYTE PTR [SI+09]
IDIV CX
IDIV WORD PTR XNUM

IMUL – цілочислове множення зі знаком

IMUL reg8/mem8
IMUL reg16/mem16

Однобайтова операція.

(*AX*) ← (*AL*)*(Операнд)

Якщо (*AH*) = знакове розширення (*AL*), то (*CF*) ← 0,
(*OF*) ← 0.

Якщо (*AH*) ≠ знакове розширення (*AL*), то (*CF*) ← 1,
(*OF*) ← 1.

Операція зі словами.

(DX,AX) ← *(AX)*(Операнд)*

Якщо (*DX*) = знакове розширення (*AX*), то (*CF*) ← 0,
(*OF*) ← 0.

Якщо (*DX*) ≠ знакове розширення (*AX*), то (*CF*) ← 1,
(*OF*) ← 1.

По команді *IMUL* виконується знакове множення вмісту акумулятора на операнд. Якщо операнд поданий байтом, то він множеться на вміст регістра *AL* і результат (16 біт) записується в регістр *AX*. Якщо операнд є словом, він множеться на вміст регістра *AX* і результат (32 біта) записується в пару регістрів *DX*, *AX*, причому старші 16 біт записуються в регістр *DX*. Операнд може бути записаний у регістр загального призначення або комірку пам'яті й інтепретується як число зі знаком.

Якщо розрядність вмісту регістра *AH* при однобайтовому множенні не збігається з розрядністю регістра *AL* або розрядність вмісту регістра *DX* не збігається з розрядністю регістра *AX* при двобайтовому множенні, то прапори переносу *CF* і переповнювання *OF* встановлюються в 0. Стан прапорів *AF, PF, SF* і *ZF* після виконання множення не визначені. Установка *CF* і *OF* у 1 означає, що регістри *AH* або *DX* містять значущі розряди результату.

Приклади запису на мові Асемблера:

```
IMUL  CH
IMUL  BYTE PTR [SI+09]
IMUL  CX
IMUL  WORD PTR XNUM
```

IN – ввід із порту

(Акумулятор) ← (Порт вводу-виводу).

```
IN    AL,port8
IN    AX,port8
IN    AL,DX
IN    AX,DX
```

По команді *IN* здійснюється передача байта або слова з порту вводу-виводу в регістр *AL* або *AX* відповідно. Адреса порту (від 1 до 255) може бути зазначена у вигляді константи. Якщо для завдання адреси порту використовується регістр *DX*, його вміст дозволяє адресуватися до 65536 різноманітних портів. Команда *IN* не впливає на вміст регістра прапорів.

INC – інкремент

(Операнд) ← (Операнд) + 1

INC reg8/mem8
INC reg16/mem16
INC reg16

По цій команді вихідний операнд збільшується на 1. Операнд може бути заданий у регістрі загального призначення або в комірці пам'яті. Сегментні регістри не можуть бути використані для збереження операндів. Команда *INC* може використовуватися як з однобайтовими, так і двобайтовими операндами. Операнди інтепретуються як числа без знака. Команда впливає на прапори *AF, OF, PF, SF* і *ZF*.

Приклади запису на мові Асемблера:

INC CH
INC BYTE PTR [SI]
INC BP
INC WORD PTR [BX+01]

INT – програмне переривання

(*SP*) ← (*SP*) - 2
(Вершина стека) ← (Регістр прапорів)
(*SP*) ← (*SP*) - 2
(Вершина стека) ← (*CS*)
(*SP*) ← (*SP*) - 2
(Вершина стека) ← (IP)
(*IF*) ← 0
(*TF*) ← 0

(*CS*) ← (Тип переривання * 4 + 2)
(*IP*) ← (Тип переривання * 4)
INT type8

По команді *INT* виконується передача керування програмі опрацювання переривань, адреса якої генерується відповідно типу переривання. Поточний стан мікропроцесора, як-то вміст регістра прапорів, кодового регістра CS *і* покажчика команд IP, записується в стек. Це дозволяє здійснювати вихід із процедури опрацювання переривання за допомогою команди *IRET* (вихід із переривання) із наступною передачею керування черговій виконуваній команді. Прапори *IF* і *TF* скидаються в 0, що запобігає можливість проходження маскованих і покрокових переривань.

Абсолютне значення адреси вектора переривання, що відповідає типу переривання, який визначається операндом команди *INT*, обчислюється множенням коду типу переривання на чотири. Вміст двійкового слова в цій області пам'яті передається відповідно в регістри *IP* і *CS*.

INTO – переривання по переповнюванню

(*SP*) ← (*SP*) - 2
(Вершина стека) ← (Регістр прапорів)
(*SP*) ← (*SP*) - 2
(Вершина стека) ← (*CS*)
(*SP*) ← (*SP*) - 2
(Вершина стека) ← (IP)
(*IF*) ← 0
(*TF*) ← 0
(*CS*) ← (*12H*)
(*IP*) ← (*10H*)

По команді *INTO* та вектору переривання, що знаходиться в комірці пам'яті з абсолютною адресою 00010H, у тому випадку, якщо прапор переповнювання OF встановлений у 1, здійснюється передача керування процедурі опра-

цювання переривання (тип 4). Поточний стан регістрів мікропроцесора *CS*, *IP* і регістра прапор записуються в стек. Така організація дозволяє здійснити за допомогою команди *IRET* (вихід із переривання) вихід із процедури опрацювання та передачу керування наступній за *INTO* команді. Прапор дозволу переривань *IF* і прапор *TF* скидаються в нуль, що дозволяє виключити прихід маскованих і покрокових переривань.

IRET – повернення з переривання

(*IP*) ← (Вершина стека)
(*SP*) ← (*SP*) + 2
(*CS*) ← (Вершина стека)
(*SP*) ← (*SP*) + 2
(Регістр прапорів) ← (Вершина стека)
(*SP*) ← (*SP*) + 2

Команда *IRET* здійснює обернену передачу керування в місце переривання, відновлюючи за інформацією зі стека вміст регістрів *IP*, *CS* і регістра прапорів. Будь-яка процедура опрацювання переривань повинна закінчуватися командою *IRET*.

JA/JNBE – перехід, якщо більше/перехід, якщо не менше або дорівнює

Якщо ((*CF*) = 0 і (*ZF*) = 0), то (*IP*) ← (*IP*) + *disp* 8.
JA short-label
JNBE short-label

Команди здійснюють передачу керування за адресою, що задається цільовим операндом, при виконанні умови одночасної рівності нулю прапорів *CF* і *ZF*. У противному випадку виконується наступна за чергою команда. Адреса цільового операнда генерується додаванням байта зміщення до адреси наступної команди. Тому що байт зміщення являє собою 8-бітове ціле число зі знаком, команди можуть вказувати адресу переходу в діапазоні + 127 байт - 128

байт від останньої адреси команди. Команди зручно використовувати при порівнянні чисел без знака.

Приклади запису на мові Асемблера:

JA UPPER_CASE

JNBE @@32

JAE/JNB/JNC – перехід, якщо більше або дорівнює/перехід, якщо не менше/перехід, якщо немає переносу

Якщо (*CF*) = 0, то (*IP*) ← (*IP*) + *Disp8*

При виконанні умови рівності нулю прапора *CF* команди передають керування за адресою, що задається цільовим операндом. У противному випадку виконується наступна команда. Адреса цільового операнда визначається додаванням байта зміщення до адреси наступної команди. Тому що байт зміщення є 8-бітове число зі знаком, то команди *JAE/JNB/JNC* можуть передавати керування командам з адресою пам'яті з діапазону +127 байт -128 байт. Команди використовуються при порівнянні чисел без знака.

JB/JNAE/JC – перехід, якщо менше/перехід, якщо не більше або дорівнює/перехід, якщо є перенос

Якщо (*CF*) = 1, то (*IP*) ← (*IP*) + *Disp8*.

Команди передають керування за адресою, що задається цільовим операндом, за умови, що прапор переносу *CF* встановлений у 1. У противному випадку виконується наступна по черзі команда. Тому що байт зміщення є 8-бітове ціле число зі знаком, то команда *JB/JNAF/JC* може передавати керування командам з адресами з діапазону +127 байт -128 байт від останньої адреси команди переходу.

JBE/JNA – перехід, якщо менше або дорівнює/перехід, якщо не більше

Якщо ((*CF*) = 1 або (*ZF*) = 1), то (*IP*) ← (*IP*) + *Disp8*.

Команди передають керування за адресою, що задається цільовим операндом, за умови, що прапор переносу

CF або прапор нуля *ZF* встановлені в 1. У противному випадку виконується наступна по черзі команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Тому що зміщення є 8-бітове ціле число зі знаком, то команди *JBE/JNA* можуть передавати керування командам з адресами з діапазону +127 байт - 128 байт від останньої адреси команди переходу. Команди *JBE/JNA* корисні при порівнянні цілих чисел із знаками.

JCXZ – перехід, якщо вміст регістра CX дорівнює нулю

Якщо (*CX*) = 0, то (*IP*) ← (*IP*) +*Disp8*.

Команда передає керування за адресою, що задається цільовим операндом, за умови, що вміст регістра *CX* дорівнює 0. У противному випадку виконується наступна за *JCXZ* команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Тому що зміщення є 8-бітове ціле число зі знаком, то команда може вказувати адресу переходу в діапазоні + 127 байт - 128 байт від кінця команди.

JE/JZ – перехід, якщо дорівнює/перехід по нулі

Якщо (*ZF*) = 1, то (*IP*) ← (*IP*) + *Disp8*.

Команди передають керування за адресою, що задається цільовим операндом за умови рівності одиниці прапора *ZF*. У противному випадку виконується наступна команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Тому що зміщення є 8-бітове ціле число зі знаком, то команди можуть вказувати адресу переходу в діапазоні + 127 байт -128 байт від кінця команди.

JG/JNLE – перехід, якщо більше ніж/перехід, якщо не менше ніж або дорівнює

Якщо ((*SF*) = (*OF*) *and* (*ZF*) = 0), то (*IP*) ← (*IP*) + *Disp8*.

Команди передають керування за адресою, що задається цільовим операндом, за умови одночасної рівності значень прапорів *SF і OF* і рівності нулю прапора *ZF*. У противному випадку виконується наступна по черзі команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Тому що зміщення є 8-бітове ціле число зі знаком, то команда *JG/JNLE* може вказувати адресу переходу в діапазоні +127 байт -128 байт від кінця команди. Команди корисні при порівнянні чисел із знаком.

JGE/JNL – перехід, якщо більше або дорівнює/перехід, якщо не менше ніж

Якщо (*SF*) = (*OF*), то (*IP*) ← (*IP*) + *Disp8*.

Команди здійснюють передачу керування за адресою, що задається цільовим операндом, за умови, що значення *прапорів* SF *і* OF рівні. У противному випадку виконується наступна по черзі команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Тому що зміщення є 8-бітове ціле число зі знаком, то команди *JGE/JNL* можуть вказувати адресу переходу в діапазоні +127 байт -128 байт від кінця команди. Ці команди корисно використовувати при порівнянні чисел із знаком.

JL/JNGE – перехід, якщо менше/перехід, якщо не більше або дорівнює

Якщо (*SF*) ≠ (*OF*), то (*IP*) ← (*IP*) +*Disp8*.

Команди передають керування за адресою, що задається цільовим операндом, за умови, що прапори *SF і OF* не рівні. У противному випадку виконується наступна за чергою команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Тому що зміщення є 8-бітове ціле число зі знаком, то команди *JL/JNGE* можуть вказувати адресу переходу в діапазоні

+127 байт -128 байт від кінця команди. Ці команди корисно використовувати при порівнянні чисел із знаком.

JLE/JNG – перехід, якщо менше або дорівнює/перехід, якщо більше

Якщо (*SF*) ≠ (*OF*) або (*ZF*) = 1, то (*IP*) ← (*IP*) + *Disp8*.

Команди передають керування за адресою, що задається цільовим операндом, за умови, що прапори *SF і OF* не рівні або якщо прапор *ZF* встановлений у 1. У противному випадку виконуються наступна по черзі команда. Адреса цільового операнда обчислюється додаванням байта зміщення з адресою наступної команди. Тому що зміщення являє собою 8-бітове ціле число зі знаком, то команди *JNE/JNG* можуть передавати керування за адресою в діапазоні +127 байт -128 байт від кінця команди. Ці команди корисно використовувати при порівнянні чисел із знаком.

JMP – безумовний перехід

Межсегментний перехід:

(*CS*) ← сегмент цільового операнда,

(*IP*) ← адреса цільового операнда, що переміщається

При внутрисегментном переході:

(*IP*) ← адреса цільового операнда, що переміщається.

Команда здійснює безумовну передачу керування в однім із п'ятьох різноманітних форматів: внутрисегментном прямому, внутрисегментном прямому укороченому, внутрисегментном непрямому, межсегментном прямому і межсегментном непрямому.

Внутрисегментна команда виконує передачу керування за адресою усередині поточного 64-кілобайтового сегмента. Іншими словами, цільова адреса повинна бути розташована в діапазоні +32767 байт -32768 байт від команди *JMP*.

Межсегментна команда з прямою адресацією може передавати керування по будь-якій адресі усередині всього мегабайтового адресного простору мікропроцесора.

При внутрисегментній команді з прямою адресацією адреса цільового операнда утворюється додаванням вмісту регістра IP до розміра 16-бітового зміщення, що задається в команді. У тому випадку, коли цільовий операнд знаходиться в діапазоні +127 байт -128 байт від команди *JMP*, можна використовувати варіант команди *JMP* укороченого формату з прямою адресацією. У цьому форматі до вмісту регістра *IP* додається значення 8-бітового зміщення, яке задається в команді.

В межсегментних командах JMP із непрямою адресацією в якості операндів, вміст яких задає адреса цільового операнда, використовуються 16-бітові регістри загального призначення (*AX, BX, CX, DX, SI, DI, BP, SP*) або двухбайтові комірки пам'яті.

Межсегментні команди *JMP* містять адресу командного сегмента і значення адреси цільового операнда, що переміщається, як частина команди, яка при виконанні команди *JMP* переписується в регістри CS і IP. Попередній вміст цих регістрів губиться. У межсегментних командах *JMP* із непрямою адресацією для запису операнда може бути використана комірка пам'яті. У противному випадку при виконанні команди *JMP* у регістри CS і IP переписується вміст подвійного слова пам'яті.

Приклади запису на мові Асемблера:

JMP FAR PTR TICK
JMP SHORT NEXT
JMP WORD PTR [SI+4]
JMP DWORD PTR [BP+6]

JNE/JNZ – перехід по нерівності/перехід, якщо не нуль

Якщо (*ZF*) = 0, то (*IP*) ← (*IP*) + *Disp8*

Команди здійснюють передачу керування за адресою, що задається цільовим операндом, за умови, що прапор ZF дорівнює 0. У противному випадку виконуються наступна команда. Адреса цільового операнда визначається додаванням байта зміщення до адреси наступної команди. Оскільки байт зміщення являє собою 8-бітове ціле число зі знаком, ці команди можуть виконувати передачу керування по адресах у діапазоні +127 байт -128 байт від кінця команди.

JNO – перехід, якщо немає переповнювання

Якщо (*OF*) = 0, то (*IP*) ← (*IP*) + *Disp8*.

Команди передають керування за адресою, що задається цільовим операндом, за умови рівності нулю значення прапора *OF*. У противному випадку виконується наступна команда. Адреса цільового операнда визначається додаванням байта зміщення до адреси наступної команди. Тому що байт зміщення являє собою 8-бітове ціле число зі знаком, то команда *JNO* може здійснювати передачу керування по адресах у діапазоні +127 байт -128 байт від кінця команди.

JNP/JPO – перехід при відсутності парності

Якщо (*PF*) = 0, то (*IP*) ← (*IP*) + *Disp8*.

Команди здійснюють передачу керування за адресою, що задається цільовим операндом, за умови, що прапор *PF* дорівнює 0. У противному випадку виконується наступна команда. Адреса цільового операнда визначається додаванням байта зміщення з адресою наступної команди. Оскільки байт зміщення являє собою 8-бітове ціле число зі знаком, то команди *JNP/JPO* можуть передавати керування командам з адресами в діапазоні +127 -128 байт від кінця команди переходу.

JNS – перехід, якщо немає знака

Якщо (*SF*) = 0, то (*IP*) ← (*IP*) + *Disp8*.

Команди передають керування за адресою, що задається цільовим операндом, за умови, що прапор знака *SF* встановлений у нуль. У противному випадку виконується наступна команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Оскільки зміщення є 8-бітове ціле число зі знаком, то команда *JNS* може здійснювати передачу керування по адресах у діапазоні +127 байт -128 байт від кінця команди.

JO – перехід по переповнюванню

Якщо (*OF*) = 1, то (*IP*) ← (*IP*) + *Disp8*.

Команда здійснює передачу керування за адресою, що задається цільовим операндом, за умови, що прапор переповнювання *OF* встановлений в одиницю. У противному випадку виконується наступна команда. Адреса цільового операнда визначається додаванням байта зміщення до адреси наступної команди. Тому що зміщення являє собою 8-бітове ціле число зі знаком, то команда може передавати керування за адресою в діапазоні +127 байт -128 байт від кінця команди.

JP/JPE – перехід по парності

Якщо (*PF*) = 1, то (*IP*) ← (*IP*) + *Disp8*.

Команди здійснюють передачу керування за адресою, що задається цільовим операндом, за умови, що прапор паритету *PF* дорівнює 1. У противному випадку виконується наступна команда. Адреса цільового операнда визначається додаванням байта зміщення до адреси наступної команди. Тому що зміщення являє собою 8-бітове ціле число зі знаком, то команда може передавати керування по адресах у діапазоні +127 байт -128 байт від кінця команди.

JS – перехід за знаком

Якщо (*SF*) = 1, то (*IP*) ← (*IP*) + *Disp8*.

Команда здійснює передачу керування за адресою, яка задається цільовим операндом, за умови, що прапор *SF*

дорівнює 1. У противному випадку виконується наступна команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Тому що зміщення являє собою 8-бітове ціле число зі знаком, то команда *JS* може передавати керування по адресах у діапазоні +127 байт -128 байт від кінця команди.

LAHF – завантаження регістра AH із регістра прапорів

(*AH*) ← (Молодший байт регістра прапорів)

Молодший байт регістра прапорів процесора при виконанні цієї команди записується в регістр *AH*. Цей байт містить прапори знака *SF*, нуля *ZF*, допоміжного переносу *AF*, паритету *PF* і переносу *CF*. Вміст регістра прапорів при цьому не змінюється. Команда *LAHF* використовується в основному для забезпечення сумісності мікропроцесора К1810ВМ86 і його попередників мікропроцесорів 580.

LDS – завантаження покажчика в регістр DS

(1-й Операнд) ← (Виконавча адреса),

(Регістр *DS*) ← (Виконавча адреса + 2).

LDS reg16,mem16.

Молодше слово подвійного слова пам'яті ЕОМ, що задається другим операндом, передається в 16-розрядний регістр загального призначення, який задається першим операндом. Старше слово цього подвійного слова передається в регістр *DS*. Перший операнд команди записується в 16-розрядний регістр загального призначення, а другий операнд - у комірку пам'яті. Сегментні регістри для цього не використовуються. Команда *LDS* не впливає на вміст регістра прапорів. Вона дуже зручна для ініціалізації покажчика пам'яті, який використовується в рядкових операціях, таких, як *LODS* (Load String - завантажити рядок), *MOVS* (Move String – передати рядок), *CMPS* (Compare String - порівняти рядок).

Приклади запису на мові Асемблера:

LDS BX,[BX]
LDS CX,VAR2
LDS BP,[DI+8FH]

LEA – завантаження виконавчої адреси

(1-й Операнд) ← Виконавча адреса 2-го операнда.

LEA reg16,mem16.

По цій команді 16-бітова адреса (зміщення), що переміщається, другого операнда передається в регістр першого операнда. Перший операнд зберігається в 16-бітовому регістрі загального призначення, а другий операнд - у комірці пам'яті. Регістри сегментів для збереження операндів не використовуються. Команда *LEA* на прапори не впливає.

Приклади запису на мові Асемблера:

LEA AX,[BX]
LEA DX,VAR2_MEM
LEA BP,[DI+8FH]

LES – завантаження покажчика з використанням регістра ES

(1-й Операнд) ← (Виконавча адреса 2-го операнда), (Регістр ES) ← (Виконавча адреса 2-го операнда + 2).

LES reg16,mem16.

Команда *LES* передає молодші 16 біт подвійного слова другого операнда з пам'яті в 16-бітовий регістр загального призначення. Старші 16 біт подвійного слова передаються в регістр ES. Перший операнд команди повинний зберігатися в 16-розрядному регістрі загального призначення. Не можна використовувати для збереження операндів регістри сегментів. Другий операнд повинний задаватися в комірці пам'яті. Команда *LES* не впливає на прапори. Ця команда виявляється зручною для ініціалізації покажчика пам'яті в тих випадках, коли використовуються команди рядкового типу, такі, як *MOVS* (Move String), *SCAS* (Scan String), *CMPS* (Compare String), *STOS* (Store String).

Приклади запису на мові Асемблера:

LES BX,[BX]
LES CX,VAR2
LES BP,[DI+8FH]

LOCK - захоплення шини

Команда є однобайтовим префіксом, по якому мікропроцесор у многопроцесорній системі виробляє сигнал *LOCK* доти, доки виконується наступна команда. Це дозволяє забезпечити мікропроцесору доступ до пам'яті без ризику втручання інших процесорів системи. Команда *LOCK* не має операндів і не впливає на прапори.

Приклад запису на мові Асемблера:

LOCK MOV BYTE PTR MEM,0

LODS - завантаження рядка

(Акумулятор) ← (Операнд за адресою в *SI*)

Якщо (*DF*) = 0, то (*SI*) ← (*SI*) + 1 (байт)

або (*SI*) ← (*SI*) + 2 (слово).

Якщо (*DF*) = 1, то (*SI*) ← (*SI*) - 1 (байт)

або (SI) ← (SI) - 2 (слово) .

По цій команді вміст пам'яті за адресою, що знаходиться в регістрі *SI*, передається в регістр *AL* або *AX* у залежності від характеру виконуваної операції: над байтами або словами. У мові Асемблера команда *LODS* записується як *LODSB* при операціях над байтами і *LODSW* при операціях над словами. вміст регістра *SI* при виконанні цих команд збільшується на 1 або 2 у тому випадку, коли прапор напрямку *DF* дорівнює нулю, і зменшується на 1 або 2, коли прапор *DF* встановлений у 1. Команда не має операндів і не впливає на прапори.

LOOP – цикл

(*CX*) ← (*CX*) - 1

Якщо (*CX*) ≠ 0, то (*IP*) ← (*IP*) + *Disp8*.

LOOP short label.

При виконанні цієї команди вихідний вміст регістра *CX* зменшується на 1. Якщо вміст регістра *CX* не дорівнює нулю, керування передається за адресою, що задається цільовим операндом. У противному випадку виконується наступна команда. Адреса цільового операнда обчислюється додаванням байта зміщення до адреси наступної команди. Тому що зміщення являє собою однобайтове ціле число зі знаком, то команда *LOOP* може передавати керування по адресах у діапазоні +127 байт -128 байт від кінця команди. Оскільки вміст регістра *CX* інтерпретується як ціле число без знака, то команда може бути використана для повторення виконуваної послідовності кодів до 65536 разів, причому регістр *CX* використовується як лічильник ітерацій.

Приклад запису на мові Асемблера:

LOOP @NEXT_LBL

LOOPE/LOOPZ – цикл якщо дорівнює/цикл якщо нуль

(*CX*) ← (*CX*) - 1

Якщо ((*CX*) ≠ 0 *and* (*ZF*) = 1), то (IP) ← (*IP*) + *Disp8*.

При виконанні команди вихідний вміст регістра *CX* зменшується на 1. Якщо вміст регістра *CX* не дорівнює 0 і прапор *ZF* встановлений у 1, здійснюється передача керування за адресою, що задається цільовим операндом. У противному випадку виконується наступна команда. Тому що байт зміщення являє собою однобайтове ціле число зі знаком, то команди *LOOPE, LOOPZ* можуть передавати керування по адресах у діапазоні +127 байт -128 байт від кінця команди. Оскільки вміст регістра *CX* інтепретується як ціле число без знака, ці команди можуть використовуватися для повторення виконуваної послідовності кодів до 65536

разів, причому регістр *CX* використовується як лічильник ітерацій.

LOOPNE/LOOPNZ – цикл якщо не дорівнює/цикл якщо не нуль

(*CX*) ← (*CX*) - 1

Якщо ((*CX*) ≠ AND (*ZF*) = 0), то (*IP*) ← (*IP*) + *Disp8*.

При виконанні команди вихідний уміст регістра *CX* зменшується на 1. Якщо вміст регістра *CX* не дорівнює 0 і прапор *ZF* встановлений у 0, то здійснюється передача керування за адресою, що задається цільовим операндом. У противному випадку виконується наступна команда. Адреса цільового операнда визначається додаванням байта зміщення до адреси наступної команди. Тому що зміщення являє собою однобайтове ціле число зі знаком, то команди *LOOPNE, LOOPNZ* можуть передавати керування по адресах у діапазоні +127 байт -128 байт від кінця команди. Оскільки вміст регістра *CX* інтепретується як ціле число без знака, ці команди можуть використовуватися для повторення виконуваної послідовності кодів до 65536 разів, причому регістр *CX* використовується як лічильник ітерацій.

MOV - пересилка

(1-й Операнд) ← (2-й Операнд)

MOV reg8/mem8,reg8
MOV reg16/mem16,reg16
MOV reg8,reg8/mem8
MOV reg16,reg16/mem16
MOV Segreg,reg16/mem16
MOV reg16/mem16,Segreg
MOV reg8/mem8,data8
MOV reg16/mem16,data16

Другий операнд при виконанні команди *MOV* займає місце збереження першого операнда, при цьому перший операнд губиться.

Перший операнд може бути заданий у регістрі загального призначення, регістрі сегмента (крім регістра *CS*) або комірки пам'яті. Другий операнд, крім того, може бути ще і безпосереднім операндом. Команда *MOV* працює як з однобайтовими, так і з двобайтовими словами.

Приклади запису на мові Асемблера:

```
MOV   [BX],DL
MOV   DL,CH
MOV   [SI+12],AX
MOV   CX,SP
MOV   BL,TEST_BYTE
MOV   SI,[DI]
MOV   DS,AX
MOV   ES,NEW
MOV   AX,CS
MOV   OLDSEG,DS
MOV   AL, TESTB
MOV   AX, TESTW
MOV   TESTB,AL
MOV   CX,-5
MOV   BYTE PTR [BP+0F23H],65
MOV   WORD PTR [SI],1000H
```

MOVS пересилка рядка

(Операнд за адресою в регістрі DI) ← (Операнд за адресою в регістрі SI).

Якщо *(DF) = 0,* то *(SI)* ← *(SI)+ 1* (байт)
(DI) ← *(DI)+ 1* (байт),
або
(SI) ← *(SI)+ 2* (слово)
(DI) ← *(DI)+ 2* (слово)

Якщо *(DF) = 1,* то *(SI)* ← *(SI) - 1* (байт)
(DI) ← *(DI) - 1* (байт),
або

(SI) ← *(SI)* - *2* (слово)

(DI) ← *(DI)-2*(слово)

У залежності від оброблюваних даних (однобайтових або двухбайтових) при виконанні команди *MOVS* уміст комірок пам'яті, що адресуються регістром *SI* (щодо сегмента даних), передається в область пам'яті, що адресується регістром *DI* (щодо додаткового сегмента). Вміст області пам'яті, що адресується регістром *SI*, не змінюється. У мові Асемблера команда *MOVS* записується як *MOVSB* для операцій із байтами і *MOVSW* – для операцій із словами. При зкинутом у 0 прапорі *DF* вміст регістрів *SI* і *DI* збільшується на 1 або 2 при виконанні команди *MOVSB* і *MOVSW* відповідно, а при одиничному прапорі *DF* вміст цих регістрів зменшується автоматично на ту ж саму величину. Команда *MOVS* не має операндів і не впливає на прапори.

MUL – множення двох операндів

MUL reg8/mem8

MUL reg16/mem16

Однобайтова операція.

(*AX*) ← (*AL*)*(Операнд)

Якщо (*AH*) = 0, то (*CF*) ← 0, (*OF*) ← 0.

Якщо (*AH*) ≠ 0, то (*CF*) ← 1, (*OF*) ← 1.

Операція зі словами.

(DX,AX) ← (AX)(Операнд)*

Якщо (*DX*) = 0, то (*CF*) ← 0, (*OF*) ← 0.

Якщо (*DX*) ≠ 0, то (*CF*) ← 1, (*OF*) ← 1.

По команді *MUL* виконується беззнакове множення вмісту акумулятора й операнда-джерела. Якщо операнд є байтом, то він домножається на вміст регістра *AL* і результат подвійної довжини (слово) записується в регістр *AX*. Якщо операнд – слово, то він домножається на вміст регістра *AX* і результат (подвійне слово) записується в пару регістрів *DX* і *AX*, причому *DX* містить старші розряди результа-

ту. Операнд може бути заданий у регістрі загального призначення або комірці пам'яті й інтепретується як число без знака.

Якщо вміст регістра *AH* після однобайтового множення або вміст регістра *DX* після двобайтового множення не рівні нулю, прапори *CF* і *OF* встановлюються в 1. У противному випадку вони скидаються в 0. Стан прапорів *AF, PF, SF* і *ZF* після виконання команди *MUL* не визначено.

NEG – заперечення

(Операнд) ← 0- (Операнд)

NEG reg8/mem8

NEG reg16/mem16

По цій команді заданий операнд відраховується з нуля, а результат записується за адресою операнда, що стирається. Операнд може бути заданий у регістрі загального призначення або комірці пам'яті у форматі байта або слова. Команда *NEG* впливає на прапори *AF, CF, OF, PF, SF* і *ZF*.

NOP – відсутність операції

По команді *NOP* мікропроцесор не робить ніяких дій. Команда не впливає на прапори і не має операндів. Вона використовується для організації циклів керованої затримки в тих випадках, коли центральний процесор повинний на визначений час перейти в стан чекання перед виконанням деяких визначених команд. Машинний код команди *NOP* цілком збігається з кодом команди *XCHG AX,AX* по котрому також не провадиться ніяких операцій.

NOT – логічне заперечення

(Операнд) ← Інверсія всіх розрядів (Операнд)

По цій команді виконується інверсія всіх розрядів операнда. Операнди можуть бути задані в регістрах загального призначення або одно- або двобайтових комірках пам'яті. Команда *NOT* не впливає на прапори.

OR – логічне АБО

(1-й Операнд) ← (1-й Операнд) АБО (2-й Операнд).
(CF) ← 0
(OF) ← 0

```
OR    reg8/mem8,reg8
OR    reg16/mem16,reg16
OR    reg8,reg8/mem8
OR    reg16,reg16/mem16
OR    reg8/mem8,data8
OR    reg16/mem16,data16
```

По цій команді виконується операція логічного додавання обох операндів. Результат записується замість першого операнда, попередній стан якого стирається. Другий операнд може бути заданий у регістрі або комірці пам'яті. Другий операнд може бути також і безпосереднім операндом. Не припускається використовувати для запису операндів регістри сегментів, а також задавати обидва операнди одночасно в комірках пам'яті. Операнди можуть бути задані одно- або двобайтовими числами. Команда *OR* завжди скидає в 0 прапори *CF* і *OF*.

OUT – вивід у порт

(Порт вводу-виводу) ← (Акумулятор).

```
OUT   port8,AL
OUT   port8,AX
OUT   DX,AL
OUT   DX,AX
```

По цій команді байт або слово передається в порт вводу-виводу з регістра *AL* або *AX* відповідно. Адреса порту може бути зазначена або в самій команді у вигляді байтової константи, що визначає порти з адресами від 0 до 255, або за допомогою адрес, розміщених у регістрі *DX*, що дозволяють звертатися до 65536 різноманітних портів перезаписом вмісту регістра *DX*, команда не впливає на прапори.

PUSHF – запис у стек змісту регістра прапорів

(SP) ← (SP) - 2

(Вершина стека) ← (Регістр прапорів)

По цій команді вміст регістра покажчика стека SP зменшується на 2, після чого на вершину стека записується поточний стан регістра прапорів. Команда не впливає на прапори і може використовуватися для зберігання поточного стана центрального процесора.

POPF – читання зі стека змісту регістра прапорів

(Регістр прапорів) ← (Вершина стека)

(SP) ← (SP) + 2

При виконанні команди *POPF* число з поточної вершини стека передається в регістр прапорів процесора. Вміст регістра *SP* зростає на 2. Таким чином, команда впливає на всі прапори, включаючи прапор *TF*. Вона може використовуватися для відновлення попереднього стана центрального процесора при поверненні в процедуру, що викликає.

PUSH - запис у стек

(SP) ← (SP) - 2

(Вершина стека) ← (Операнд).

PUSH reg16/mem16

PUSH Segreg

Операнд команди *PUSH* може бути заданий у 16-бітовому регістрі або комірці пам'яті, припускається також використання будь-яких регістрів загального призначення або сегментних регістрів. При виконанні цієї команди вміст регістра зменшується на 2. Потім операнди записуються в стек на його вершину. Команда не впливає на прапори і часто використовується для зберігання в стеку проміжних даних.

POP – читання зі стека

(Операнд) ← (Вершина стека)
(SP) ← (SP) + 2
POP reg16/mem16
POP Segreg

Операнд команди *POP* може бути заданий у 16- бітовому регістрі або комірці пам'яті, а також у будь-якому регістрі загального призначення або сегментного регістра (крім регістра *CS*). По цій команді уміст вершини стека передається в операнд, а вміст регістра SP збільшується на 2. Команда не впливає на прапори і часто використовується для читання зі стека проміжних даних, попередньо записаних у стек по команді *PUSH*.

RCL – циклічний зсув уліво з переносом

(Тимчасовий біт) ← (*CF*)
(*CF*) ← (Старший біт операнда)
(Операнд) ← (Операнд)*2 + (Тимчасовий біт).
RCL reg8/mem8,1
RCL reg16/mem16,1
RCL reg8/mem8,CL
RCL reg16/mem16,CL

По цій команді виконується циклічний зсув розрядів у байті або слові операнда. При використанні формату *Reg/Mem,1* провадиться зсув тільки на 1 розряд. Якщо використовується формат *Reg/Mem,CL*, то число зсувів задається вмістом регістра *CL*. Прапор *CF* також бере участь в операціях зсувів, його значення заноситься в молодший значущий біт операнда-приймача, а на його місце заноситься значення самого старшого значущого біта цього операнда. У режимі зсуву на один біт (без регістра *CL*) прапор *OF* встановлюється в 1 у тому випадку, якщо значення прапора *CF* відрізняється від самого старшого значення біта операнда-приймача. Стан прапора *OF* не визначено при зсуві на число розрядів, більше одиниці. Операнди в цій команді мо-

жуть бути задані в регістрах загального призначення або комірках пам'яті.

RCR – циклічний зсув управо з переносом

(Тимчасовий біт) ← *CF*
(*CF*) ← (Молодший біт операнда)
(Операнд) ← (Операнд)/2
(Старший біт операнда) ← (Тимчасовий біт)
RCR reg8/mem8,1
RCR reg16/mem16,1
RCR reg8/mem8,CL
RCR reg16/mem16,CL

По цій команді виконується циклічний зсув розрядів у байті або слові операнда. При використанні формату *Reg/Mem,1* виконується зсув тільки на 1 розряд. Якщо використовується формат *Reg/Mem,CL*, то число зсувів задає вміст регістра *CL*. Прапор *CF* також використовується в операціях зсуву, його значення передається в комірки збереження старшого біта операнда-приймача, а на його місце заноситься значення молодшого значущого біта вихідного значення цього операнда. При зсуві на 1 біт (без регістра *CL*) прапор *OF* встановлюється в 1 у тому випадку, якщо значення двох старших бітів результуючого операнда не рівні. При зсуві на число розрядів, більше 1, стан прапора *OF* не визначено. Операнди можуть бути однобайтовими або двобайтовими і задаватися в регістрах загального призначення або комірках пам'яті.

REP/REPE/REPZ – повторення/повторення, якщо дорівнює/повторення, якщо нуль

Команда *REP* звичайно використовується як префіксний байт для рядкових команд, таких, як *MOVS, STOS*. Вміст регістра *CX* задає число повторень рядкових операцій. Таким чином, префікс *REP*, пов'язаний із рядковими командами, можна інтепретувати як "повторення рядкової

операції доти, доки вміст регістра *CX* не стане рівним нулю".

Для полегшення програмування передбачені альтернативні команди *REPE* і *REPZ*. Вони, як правило, використовуються в якості префіксних байтів у таких рядкових командах, як *CMPS* і *SCAS*. Перед їхнім повторенням необхідно встановити в 1 прапор нуля *ZF*. Повторення послідовності закінчується по ознаці рівності нулю прапора *ZF*, навіть якщо вміст регістра *CX* не дорівнює нулю. Префікси повторення можуть бути перервані по ознаках переривань, тому їх не варто використовувати ні з якими іншими командами, крім стандартних рядкових команд.

RET – повернення з процедури

(IP) ← (Вершина стека),
(SP) ← (SP) + 2
Межсегментна процедура
(CP) ← (Вершина стека),
(SP) ← (SP) + 2
При наявності зміщення
(SP) ← (SP) + зміщення.
RET
RET disp16

Команда *RET* повертає керування від процедури наступній за *CALL* команді. Макроасемблер генерує внутрисегментну команду *RET* у тому випадку, якщо процедура визначена як *NEAR*, і межсегментну *RET*, коли вона визначена як *FAR*. При внутрисегментній операції повернення з процедури команда здійснює читання зі стека (що задається регістром *SP*) слова і запис його в регістр *IP*. Вміст регістра *SP* при цьому збільшується на 2. По межсегментній команді слово зі стека записується в регістр *CS*, причому вміст регістра *SP* збільшується на 2.

У команді може бути використане 16-бітове зміщення, що додається до вмісту регістра *SP* для обходу парамет-

рів і проміжних даних, приміщених у стек після виконання команди *CALL*.

ROL – циклічний зсув вліво

(CF) ← (Старший біт операнда)
(Операнд) ← (Операнд)*2 +(*CF*)
ROL reg8/mem8,1
ROL reg16/mem16,1
ROL reg8/mem8,CL
ROL reg16/mem16,CL

По цій команді здійснюється циклічний зсув розрядів у байті або слові операнда. При використанні формату *Reg/Mem,1* виконується зсув тільки на один розряд. Якщо використовується формат *Reg/Mem,CL*, то число зсувів задається вмістом регістра CL. Старший біт вихідного операнда зсовується в комірку молодшого біта операнда-приймача. При зсуві на 1 біт (без регістра *CL*) прапор *OF* встановлюється в 1 у тому випадку, коли значення прапора CF при зсуві на довільне число розрядів, більше 1, не визначено. Операнд команди *ROL* може бути заданий у регістрі загального призначення або комірці пам'яті. Використовуються як однобайтові, так і двухбайтові операнди.

ROR – циклічний зсув вправо

(CF) ← (Молодший біт операнда)
(Операнд) ← (Операнд)/2
(Старший біт операнда) ← (CF)
ROR reg8/mem8,1
ROR reg16/mem16,1
ROR reg8/mem8,CL
ROR reg16/mem16,CL

По команді *ROR* виконується операція циклічного зсуву розрядів у байті або слові операнда. При використанні формату *Reg/Mem,1* виконується зсув тільки на один розряд. Якщо використовується формат *Reg/Mem,CL*, то число зсу-

вів задається вмістом регістра *CL*. При цьому молодший біт вихідного операнда зсовується в комірку старшого бита операнда-приймача. При зсуві на 1 розряд (без регістра *CL*) прапор *OF* встановлюється в 1 за умови, що значення прапора переносу *CF* не збігається зі значенням старшого розряду операнда-приймача. Стан прапора *OF* при операціях зсуву на довільне число розрядів, більше 1, не визначено. Операнди команди *ROR* можуть бути задані в регістрах або комірках пам'яті.

REPNE/REPNZ – повторення, якщо дорівнює/повторення, якщо нуль

Команди використовуються з командами рядкових операцій, такими, як *CMPS* і *SCAS*. Вміст регістра *CX* задає число повторень виконання команди. Перед початком наступного повторення прапор нуля *ZF* повинний бути скинен у 0. Повторення припиняється, якщо прапор *ZF* встановлений у 1, навіть якщо при цьому вміст регістра *CX* відмінно від нуля. Оскільки виконання префіксних команд може перериватися, їх не варто використовувати з іншими командами, крім стандартних рядкових команд.

SAHF – запис вмісту регістра AH у регістр прапорів

(Молодший байт регістра прапорів) ← (*AH*).

Вміст регістра *AH* при виконанні цієї команди переписується в молодший байт регістра прапорів, що містить у собі прапори *SF, ZF, AF, PF* і *CF*. На інші прапори команда не впливає. Команда *SAHF* використовується в основному для забезпечення сумісності мікропроцесорів К1810ВМ86 і 580.

SAL/SHL – арифметичний зсув вліво/логічний зсув вліво

(CF) ← (Старший біт операнда),

(Операнд) ←(Операнд)*2.

SAL reg8/mem8,1

SAL reg16/mem16,1
SAL reg8/mem8,CL
SAL reg16/mem16,CL

По цих командах виконуються зсуви в байті або слові операнда. При використанні формату *Reg/Mem,1* виконується зсув тільки на 1 розряд. Якщо використовується формат *Reg/Mem,CL*, то число зсувів задається вмістом регістра *CL*. Молодші розряди байта або слова при зсуві заповнюються нулями. Якщо знаковий біт результуючого операнда не змінює свого значення, прапор переповнювання *OF* скидається в 0. Стан прапора *OF* після зсуву на довільне число розрядів, більше 1, не визначено. Операнди команд *SAL* і *SHL* можуть бути задані в регістрах загального призначення або комірках пам'яті. Формат операндів – байт або слово. Команди впливають на прапори *CF, OF, PF, SF* і *ZF*. Стан прапора *AF* не визначено.

SAR – арифметичний зсув вправо

(SF) ← (Молодший біт операнда),
(Операнд) ← (Операнд)/2.
SAR reg8/mem8,1
SAR reg16/mem16,1
SAR reg8/mem8,CL
SAR reg16/mem16,CL

По команді виконується зсув у байті або слові операнда. Старший значущий біт при цьому зберігає своє значення. При використанні формату *Reg/Mem,1* виконує зсув на 1 розряд, а у форматі *Reg/Mem,CL* – на довільне число розрядів, що визначається вмістом регістра *CL*. Якщо після зсуву значення двох старших бітів результату збігаються, то прапор *OF* скидається в 0. Прапор *OF* скидається в 0 при довільному числі зсувів, більше 1. Операнди команди *SAR* можуть бути задані в регістрах загального призначення або комірках пам'яті. Формат операндів – байт або слово. Ко-

манда впливає на прапори *CF*, *OF*, *PF*, *SF* і *ZF*. Стан прапора *AF* не визначено.

SBB – віднімання з позикою

(1-й операнд) ← (1-й операнд) - (2-й операнд) – (прапор переносу *CF*)

SBB reg8/mem8,reg8
SBB reg16/mem16,reg16
SBB reg8,reg8/mem8
SBB reg16,reg16,mem16
SBB reg8/mem8,data8
SBB reg16/mem16,data16

По команді *SBB* виконується віднімання значення прапора *CF* і другого операнда з першого операнда. Результат міститься на місце першого операнда, попереднє значення якого губиться. Вміст другого операнда не змінюється.

Перший операнд може бути заданий у регістрі або комірці пам'яті. другий операнд, крім того може бути заданий безпосередньо. Не припускається використовувати для запису операндів сегментні регістри або задавати обидва операндів у комірках пам'яті. Операнди можуть бути однобайтовими або двухбайтовими числами зі знаком або без знака. Можливість позики дозволяє використовувати команду *SBB* для організації віднімання чисел із розрядністю, що перевищує 16 біт.

SCAS – сканування рядка

(Акумулятор) – (Операнд за адресою в DI)

Якщо *(DF) = 0,* то *(DI)← (DI)+ 1* (байт),
або
(DI) ← (DI)+ 2 (слово)

Якщо *(DF) = 1,* то *(DI) ← (DI) - 1* (байт),
або
(DI)← (DI) - 2 (слово)

По цій команді вміст комірки пам'яті, адреса якого знаходиться в регістрі *DI* у залежності від розміру операндів: одно- або двухбайтових, відраховується з умісту регістра *AL* або *AX* відповідно. Результат операції впливає на регістр прапорів, але не змінює вмісту акумулятора. У мові Асемблера для операцій із байтами команда записується як *SCASB*, а для операцій із словами – як *SCASW*. Якщо прапор *DF* скинений у 0, то вміст регістра *DI* збільшується на 1 або 1 при виконанні команд *SCASB* і *SCASW* відповідно. Якщо прапор *DF* встановлений у 1, то при виконанні цих команд уміст регістра *DI* зменшується на 1 і 2 відповідно. Команда *SCAS* використовується разом із префіксами *REPE/REPZ* для пошуку в рядку елемента, відмінного від заданого значення, або з префіксами *REPNE/REPNZ* для пошуку заданих значень.

SHR – логічний зсув вправо

(CF) ← (Молодший біт операнда),
(Операнд) ← (Операнд)/2

SHR reg8/mem8,1
SHR reg16/mem16,1
SHR reg8/mem8,CL
SHR reg16/mem16,CL

По команді *SHR* виконуються зсуви в байті або слові операнда. При форматі *Reg/Mem,1* виконується зсув тільки на 1 розряд. Якщо заданий формат *Reg/Mem,CL*, то число розрядів зсуву задається вмістом регістра *CL*. Старші розряди результату заповнюються нулями. Якщо після зсуву знаковий біт результату зберігає своє початкове значення, то прапор *OF* скидається в 0. При довільному числі зсувів (більше 1) стан прапора *OF* не визначено. Операнди команди *SHR* можуть бути задані в регістрах загального призначення або комірках пам'яті. Команда змінює прапори *CF, OF, PF, SF* і *ZF*. Стан прапора *AF* не визначено.

STC – встановити прапор переносу

(*CF*) ← 1

Команда встановлює в одиничний стан прапор *CF* і не впливає на інші прапори і регістри. Операнди в команді відсутні.

STD – встановити прапор напрямку

(*DF*) ← 1

Команда встановлює в одиничний стан прапор *DF* і не впливає на інші прапори і регістри. Команда операндів не має. При використанні рядкових команд (*CMPS, LODS, MOVS, SCAS, STOS*) уміст регістрів *SI* і *DI* зменшується на 1 або 2 для однобайтових і двухбайтових чисел відповідно.

STI – встановити прапор переривання

(*IF*) ← 1

Команда встановлює в одиничний стан прапор *IF* і не впливає на інші прапори і регістри. Команда операндів не має. При одиничному прапорі *IF* мікропроцесор дозволяє програмне переривання і будь-які масковані або немасковані зовнішні переривання, що приходять на входи *INTR* і *NMI*.

STOS – запам'ятати рядок

(Операнд за адресою в регістрі DI) ← (Акумулятор).

Якщо *(DF) = 0*, то ← *(DI)+ 1* (байт),

або *(DI)* ← *(DI)+ 2* (слово)

Якщо *(DF) = 1*, то *(DI)* ← *(DI) - 1* (байт),

або *(DI)*← *(DI) - 2* (слово)

По цій команді вміст регістрів *AL* або *AX* переписується в комірку пам'яті, відповідно однобайтову або двухбайтовую, адреса якої знаходиться в регістрі *DI*. У мові Асемблера для операцій із байтами ця команда записується як *STOSB*, а для операцій із словами як *STOSW*. Якщо прапор напрямку скинен у 0, то вміст регістра *DI* збільшується

на 1 при виконанні команди *STOSB* і на 2 при виконанні команди *STOSW*. Якщо прапор *DF* встановлений у 1, при виконанні цих команд вміст регістра *DI* зменшується на 1 або 2 відповідно. Команда не має операндів і не впливає на прапори. Адреса, що переміщається, яка вказується регістром *DI* у цій команді, завжди визначається відносно базової адреси додаткового сегмента, що знаходиться в регістрі *ES*. Використання префікса перепризначення сегмента з командою *STOS* неприпустимо.

SUB – віднімання

(1-й операнд) ← (1-й операнд) - (2-й операнд)

SUB reg8/mem8,reg8
SUB reg16/mem16,reg16
SUB reg8,reg8/mem8
SUB reg16,reg16,mem16
SUB reg8/mem8,data8
SUB reg16/mem16,data16

Вміст другого операнда при виконанні команди відраховується із вмісту першого операнда. Перший операнд при цьому губиться, а другий зберігається незмінним.

Перший операнд може бути заданий у регістрі або комірці пам'яті, другий операнд може бути заданий у регістрі, комірці пам'яті або безпосередньо. Не припускається використання сегментного регістра або одночасного запису обох операндів у комірках пам'яті. Операнди можуть бути як знаковими, так і беззнаковими числами. Можливо виконання 8- і 16- розрядних операцій.

TEST – тест

(Перший операнд) *and* (Другий операнд).

(*CF*) ← 0
(*OF*) ← 0.

TEST reg8/mem8,reg8
TEST reg16/mem16,reg16

TEST reg8/mem8,data8

TEST reg16/mem16,data16

По цій команді здійснюється логічне множення першого і другого операндів. Результат змінює стан прапорів, але не змінює самих операндів.

Перший операнд може бути заданий у регістрі або комірці пам'яті. Другий операнд може бути заданий у регістрі, комірці пам'яті або безпосередньо. Неприпустимо для будь-якого з операндів використовувати сегментний регістр або одночасно задавати їх у комірках пам'яті. Операнди можуть бути знаковими і беззнаковими числами. Можливо представлення 8- і 16-розрядних команд. При виконанні операції змінюється стан прапорів переносу *CF*, переповнювання *OF*, паритету *PF*, знака *SF* і нуля *ZF*. Стан прапора допоміжного переносу *AF* виявляється невизначеним. Команда *TEST* дуже зручна при тестуванні визначених розрядів байта або слова.

WAIT – чекання

Команда переводить мікропроцесор у стан чекання, у якому він знаходиться доти, доки не буде активізована лінія TEST. Команда *WAIT* не впливає ні на які прапори і використовується для забезпечення синхронізації мікропроцесорів 1810ВМ86 і 1810ВМ87.

XCHG – заміна

(Тимчасове зберігання) ← (Перший операнд)

(Перший операнд) ← (Другий операнд)

(Другий операнд) ← (Тимчасове зберігання).

XCHG reg8,reg8/mem8

XCHG reg16,reg16/mem16

По цій команді перший і другий операнди обмінюються місцями. Операнди можуть бути задані тільки в регістрах або комірках пам'яті. Команда не впливає на прапори.

XLAT – передача з таблиці

(AL) ← ((BX) + (AL)).

Команда використовує вміст регістра *AL* як зміщення щодо початку 256-байтової таблиці, адреса початку якої задається регістром BX. Байт таблиці, визначений цим зміщенням, займає вихідний вміст регістра *AL*. Зміщення 0 визначає початковий байт таблиці. Команда не використовує операнди і не впливає на прапори. Вона дуже зручна для вибірки даних із таблиці для інших команд.

XOR – АБО, що виключає

(1-й операнд) ← (1-й операнд) *xor* (2-й операнд).

(*CF*) ← 0

(*OF*) ← 0

XOR reg8/mem8,reg8
XOR reg16/mem16,reg16
XOR reg8,reg8/mem8
XOR reg16,reg16,mem16
XOR reg8/mem8,data8
XOR reg16/mem16,data16

По цій команді виконується логічна операція АБО, що виключає, з першим і другим операндами. Результат переміщається на місце збереження першого операнда, і він губиться. Другий операнд зберігається. Перший операнд може бути заданий у регістрі або комірці пам'яті. Другий операнд може бути заданий у регістрі, комірці пам'яті або безпосередньо. Неприпустимо використовувати для запису любого з операндів сегментні регістри або записувати їх одночасно в комірки пам'яті. Операція провадиться як над 8-, так і над 16-розрядними числами. Прапори переносу *CF* і переповнювання *OF* скидаються в 0.

3. МІКРОКОНТРОЛЕР КМ1816ВЕ51

Мікроконтролер виконаний на основі високорівневої *n-МОП* технології і випускається в корпусі, що має 40 зовнішніх виводів. Цоколівка корпуса МК51 і найменування виводів показані на рис. 3.1. Для роботи МК51 потрібно одне джерело електроживлення +5 вольт. Через чотири порти вводу/виводу, що програмуються, МК51 взаємодіє із середовищем у стандарті ТТЛ-схем із трьома станами виходу.

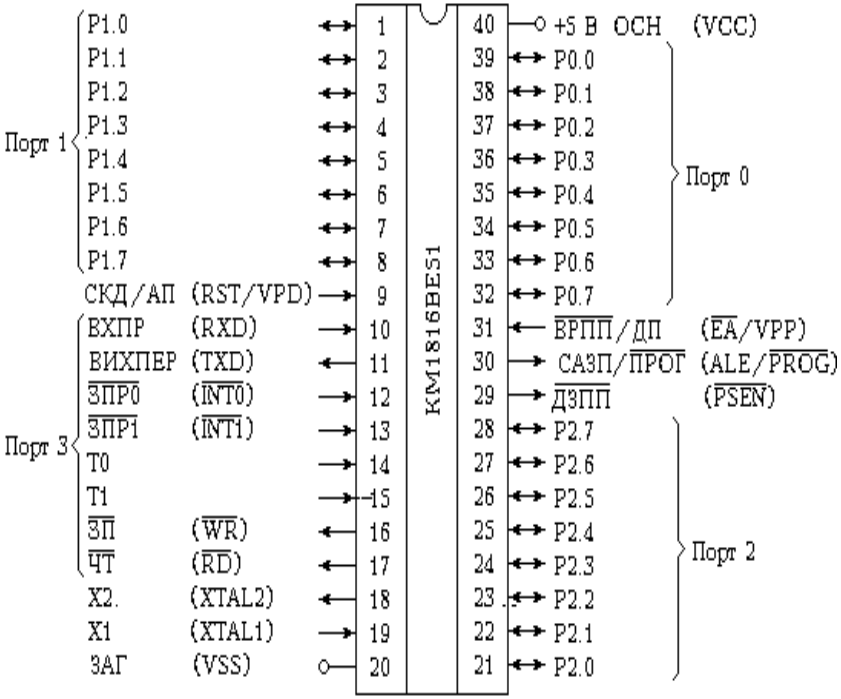


Рис. 3.1

Корпус МК51 має два виводи для підключення кварцового резонатора, чотири виводи для сигналів, що управляють режимом роботи МК51, і вісім ліній порту 3 що можуть бути запрограмовані користувачем на виконання спеціалізованих (альтернативних) функцій обміну інформацією із середовищем.

3.1 Структурна схема МК51

Основу структурної схеми МК51 (рис. 3.2) утворює внутрішня двунаправлена 8-бітна шина, що зв'язує між собою всі основні вузли й пристрої: резидентну пам'ять, АЛП, блок регістрів спеціальних функцій, устрій керування і порти вводу/виводу.

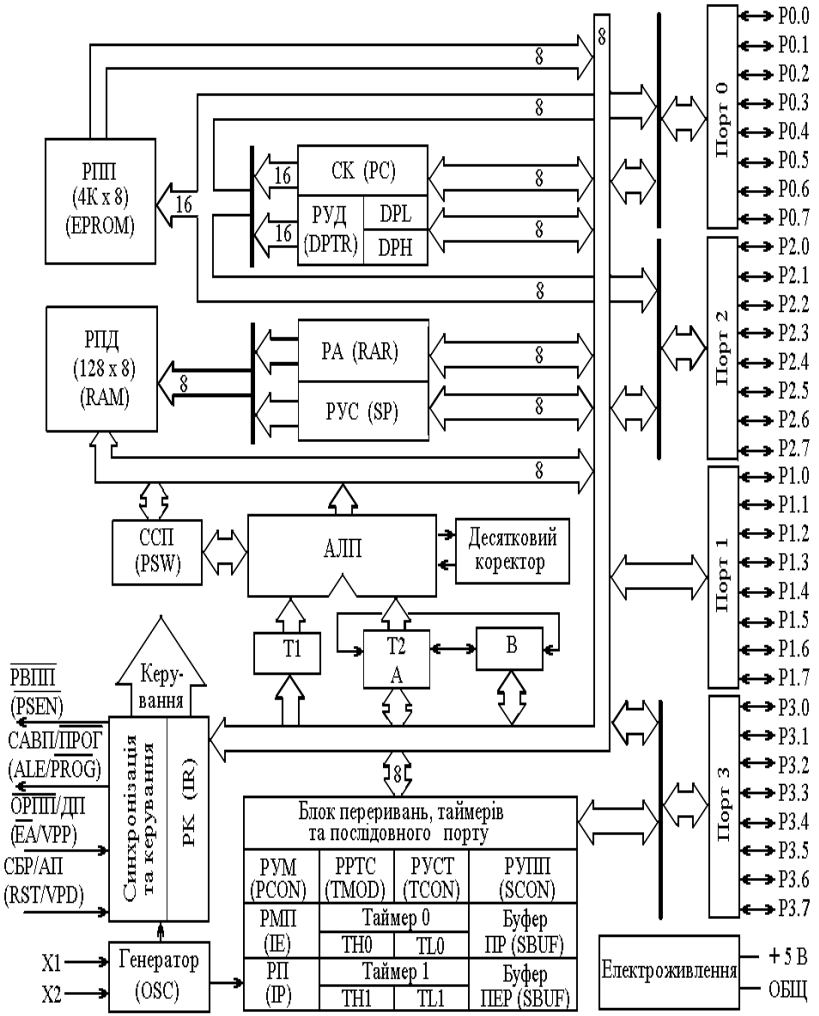


Рис. 3.2

Роздивимося основні елементи структури й особливості організації обчислювального процесу в МК51.

3.1.1 Арифметико-логічний пристрій

8-бітне АЛП може виконувати арифметичні операції додавання, віднімання, множення і ділення; логічні операції І, АБО, АБО, що виключає, а також операції циклічного зсуву, скидання, інвертування і т.п. В АЛП є програмно недоступні регістри *T1* і *T2*, призначені для тимчасового зберігання операндів, схема десяткової корекції і схема формування ознак.

Важливою особливістю АЛП є його спроможність оперувати не тільки байтами, але і бітами. Окремі програмно-доступні біти можуть бути встановлені, скинені, інвертовані, передані, перевірені і використані в логічних операціях. Ця спроможність АЛП оперувати бітами настільки важлива, що в багатьох описах МК51 говориться про наявність в ньому "булевського процесора". Для керування об'єктами часто застосовуються алгоритми, що містять, операції над вхідними і вихідними булевськими змінними (істина/хибність), реалізація яких засобами звичайних мікропроцесорів пов'язана із певними труднощами.

Таким чином, АЛП може оперувати чотирма типами інформаційних об'єктів: булевськими (1 біт), цифровими (4 біти), байтними (8 біт) і адресними (16 біт). В АЛП виконується 51 різноманітна операція пересилки або перетворення цих даних. Тому що використовується 11 режимів адресації (7 для даних і 4 для адрес), то шляхом комбінування "операція/ режим адресації" базове число команд 111 розширюється до 255 із 256 можливих при однобайтнім коді операції.

3.1.2 Резидентна пам'ять

Пам'ять програм і пам'ять даних, розміщені на кристалі МК51, фізично і логічно розділені, мають різноманітні механізми адресації, працюють під керуванням різноманітних сигналів і виконують різні функції.

Пам'ять програм (ПЗП або СППЗП) має ємність 4 Кбайта і призначена для збереження команд, констант, управляючих слів ініціалізації, таблиць перекодування вхідних і вихідних перемінних. Резидентна пам'ять програм (РПП) має 16-бітну шину адреси, через яки забезпечується доступ із лічильника команд або з регістра-покажчика даних. Останній виконує функції базового регістра при непрямих переходах по програмі або використовується в командах, що оперують із таблицями.

Пам'ять даних (ОЗП) призначена для збереження змінних у процесі виконання прикладної програми, адресується одним байтом і має ємність 128 байт. Крім того, до адресного простору резидентної пам'яті даних (РПД) примикають адреси регістрів спеціальних функцій (РСФ), що перераховані в табл. 3.1.

Таблиця 3.1

Символ	Найменування	Адреса
*ACC**	Акумулятор	*0E0H*
*B**	Регістр-розширювач акумулятора	*0F0H*
*PSW**	Слово стана програми	*0D0H*
SP	Регістр-покажчик стека	*81H*
DPTR	Регістр-покажчик даних (*DPH*), (*DPL*)	*83H, 82H*
*P0**	Порт 0	*80H*
*P1**	Порт 1	*90H*
*P2**	Порт 2	*0A0H*
*P3**	Порт 3	*0B0H*
*IP**	Регістр пріоритетів	*0B8H*
*IE**	Регістр маски переривань	*0A8H*
TMOD	Регістр режиму таймера/ лічильника	*89H*
*TCON**	Регістр керування/ статусу таймера	*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 прийомопередавачем	*98H*
SBUF	Буфер прийомопередавача	*99H*
PCON	Регістр керування потужністю	*87H*
Примітка. Регістри, імена яких відзначені знаком (*), припускають адресацію окремих біт.		

Пам'ять програм, так само як і пам'ять даних, може бути розширена до 64 Кбайт шляхом підключення зовнішніх ВІС.

Акумулятор і слово стана програми (ССП). Акумулятор є джерелом операнда і місцем фіксації результату при виконанні арифметичних, логічних операцій і ряду операцій передачі даних. Крім того, тільки з використанням акумулятора можуть бути виконані операції зсувів, перевірка на нуль, формування прапора паритету і т.п.

При виконанні багатьох команд в АЛП формується ряд ознак операцій (прапорів), що фіксуються в регістрі ССП. В табл. 3.2 наводиться перелік прапорів ССП, даються їх символічні імена й описуються умови їх формування.

Найбільш “активним” прапором ССП є прапор переносу, що бере участь і модифікується в процесі виконання безлічі операцій, включаючи додавання, віднімання і зсуви. Крім того, прапор переносу (*C*) виконує функції “булевого акумулятора” у командах, що маніпулюють із бітами. Прапор переповнювання (*OV*) фіксує арифметичне переповнювання при операціях над цілими числами зі знаком і уможливлює використання арифметики в додаткових кодах. Арифметико-логічний пристрій не управляє прапорами селекції банку регістрів (*RS0, RS1*), і їх значення цілком визначається прикладною програмою і використовується для вибору одного з чотирьох реєстрових банків.

Широке поширення одержало уявлення про те, що в мікропроцесорах, архітектура яких спирається на акумулятор, більшість команд працюють із ними, використовуючи адресацію “по умовчанню” (неявну). У МК51 справа інша.

Хоча процесор у МК51 має у своїй основі акумулятор, проте, він може виконувати безліч команд і без участі акумулятора. Наприклад, дані можуть бути передані з будь-якої комірки РПД у будь-який регістр, будь-який регістр може бути завантажений безпосереднім операндом і т.д. Багато логічних операцій можуть бути виконані без участі акумулятора.

Таблиця 3.2

Символ	Позиція	Ім'я і призначення
C	*PSW. 7*	Прапор переносу. Встановлюється і скидається апаратними засобами або програмою при виконанні арифметичних і логічних операцій.
AC	*PSW. 6*	Прапор допоміжного переносу. Встановлюється і скидається тільки апаратними засобами при виконанні команд додавання і віднімання і сигналізує про перенос або позику в біті 3.
F0	*PSW. 5*	Прапор 0. Може бути встановлений, скинен або перевірений програмою як прапор, що специфіцирується користувачем.
RS1 *RS0*	*PSW. 4* *PSW. 3*	Вибір банку регістрів встановлюється і скидається програмою для вибору робочого банку регістрів (див. примітку).
OV	*PSW. 2*	Прапор переповнювання. Встановлюється і скидається апаратно при виконанні арифметичних операцій.
-	*PSW. 1*	Не використовується.

Символ	Позиція	Ім'я і призначення
P	*PSW. 0*	Прапор паритету. Встановлюється і скидається апаратно в кожному циклі команди і фіксує непарне/ парне число одиничних біт в акумуляторі, тобто виконує контроль по парності.

Примітка. Вибір робочого банку регістрів

RS1	*RS0*	Банк	Межа адрес
0	0	0	*00H - 07H*
0	1	1	*08H - 0FH*
1	0	2	*10H - 17H*
1	1	3	*18H - 1FH*

Крім того, змінні можуть бути інкрементовані, декрементовані і перевірені (*test*) без використання акумулятора. Прапори і керуючі біти можуть бути перевірені і замінені аналогічно.

Регістри-покажчики. 8-бітний регістр покажчик стека (РПС) може адресувати будь-яку область РПД. Його вміст інкрементується перед тим, як дані будуть запом'ятовані в стеку в ході виконання команд *PUSH* і *CALL*. Вміст РПС декрементирується після виконання команд *POP* і *RET*. Подібний засіб адресації елементів стека називають передінкременним/постдекрементним. У процесі ініціалізації МК51 після сигналу "Скидання" в РПС автоматично завантажується код 07Н. Це значить, що якщо прикладна програма не перевизначає стек, то перший елемент даних у стеку буде розташовуватися в комірці РПД з адресою 08Н.

Двобайтний регістр-покажчик даних (РПД) звичайно використовується для фіксації 16-бітної адреси в операціях з звертанням до зовнішньої пам'яті. Командами МК51 регістр-покажчик даних може бути використаний або як 16-бітний регістр, або як два незалежні 8-бітних регістри (*DPH* і *DPL*).

Таймер/лічильник. У складі засобів МК51 є реєстрові пари із символічними іменами *TH0, TL0* і *TH1, TL1*, на основі яких функціонують два незалежних програмно-керованих 16-бітних таймери/лічильники подій.

Буфер послідовного порту. Регістр із символічним ім'ям *SBUF* являє собою два незалежних регістри – буфер приймача і буфер передавача. Завантаження байта в *SBUF* негайно викликає початок процесу передачі через послідовний порт. Коли байт зчитує з *SBUF*, це значить, що його джерелом є приймач послідовного порту.

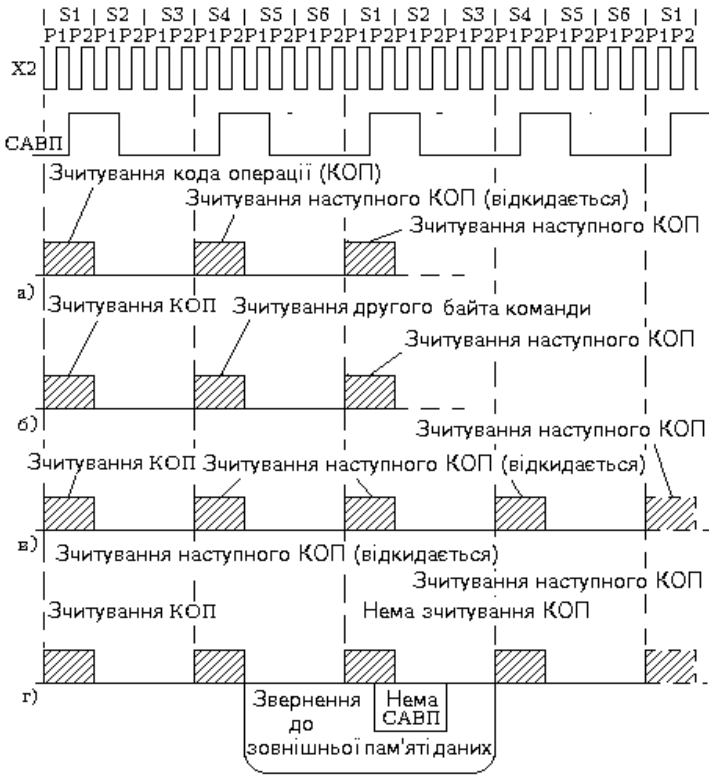
Регістри спеціальних функцій. Регістри із символічними іменами *IP, IE, TMOD, TCON, SCON* і *PCON* використовуються для фіксації і програмної зміни керуючих біт і біт стану схеми переривання, таймера/лічильника, прийомопередавача послідовного порту і для керування потужністю електроживлення МК51. Їхня організація буде описана нижче при розгляді особливостей роботи МК51 у різноманітних режимах.

3.1.3 Пристрій керування і синхронізації

Кварцовий резонатор, що підключається до зовнішніх виводів *X1* і *X2* корпуса МК51, управляє роботою внутрішнього генератора, що у свою чергу формує сигнали синхронізації.

Пристрій керування МК51 на основі сигналів синхронізації формує машинний цикл фіксованої тривалості, рівної 12 періодам резонатора або шести станам первинного керуючого автомата (*S1–S6*). Кожний стан керуючого автомата містить дві фази (*P1, P2)* сигналів резонатора. У фазі *P1,* як правило, виконується операція в АЛП, а у фазі *P2* здійснюється межрегистрова передача. Весь машинний цикл складається з 12 фаз, починаючи з фази *S1P1* закінчуючи фазою *S6P2,* як показано на рис. 3.3. Ця тимчасова діаграма ілюструє роботу пристрою керування МК51 при вибірці і виконанні команд різного ступеня складності. Всі заштриховані сигнали є внутрішніми і недоступні користувачу МК51 для контролю. Зовнішніми сигналами, що спостерігаються, є тільки сигнали резонатора і строба адреси зовнішньої пам'яті. Як очевидно з тимчасової діаграми, сигнал *ALE* (САЗП) формується двічі за один машинний цикл (*S1P2 – S2P1* і *S4P2 – S5P1*) і використовується для керування процесом звернення до зовнішньої пам'яті.

Більшість команд МК51 виконується за один машинний цикл. Деякі команди, що оперують із 2-байтними словами або зв'язані зі зверненням до зовнішньої пам'яті, виконуються за два машинних цикли. Тільки команди ділення і множення потребують чотирьох машинних циклів. На основі цих особливостей роботи пристрою керування МК51 провадиться розрахунок часу виконання прикладних програм.



а - команда 1 байт/1 цикл, наприклад, INC A ;

б - команда 2 байти/1 цикл, наприклад, ADD A,#d ;

в - команда 1 байт/2 цикли, наприклад, INC DPTR;

г - команда 1 байт/2 цикли, наприклад, MOVX.

Рис. 3.3

3.2 Порти вводу/виводу інформації

Всі чотири порти МК51 призначені для вводу або виводу інформації побайтно. Кожний порт містить керовані регістр-засувку, вхідний буфер і вихідний драйвер.

Вхідні драйвери портів 0 і 2, а також вхідний буфер порту 0 використовується при звертанні до зовнішньої пам'яті (ЗП). При цьому через порт 0 у режимі тимчасового мультиплексированія спочатку ви-

водиться молодший байт даних. Через порт 2 виводиться старший байт адреси в тих випадках, коли розрядність адреси дорівнює 16 біт.

Всі виводи порту 3 можуть бути використані для реалізації альтернативних функцій, перерахованих у табл. 3.3. Альтернативні функції можуть бути задіяні шляхом запису 1 у відповідні біти регистра- засувки (*P3. 0 - P3. 7*) порт 3.

Таблиця 3.3

Символ	Позиція	Ім'я і призначення
$\overline{RD}$	*P3.7*	Читання. Активний сигнал низького рівня формується апаратно при звертанні до зовнішньої пам'яті даних (ЗПД).
$\overline{WR}$	*P3.6*	Запис. Активний сигнал низького рівня формується апаратно при звертанні до ЗПД.
T1	*P3.5*	Вхід таймера/лічильника 1 або тест-вхід.
T0	*P3.4*	Вхід таймера/лічильника 0 або тест-вхід.
$\overline{INT}1$	*P3.3*	Вхід запиту переривання 1. Сприймається сигнал низького рівня або зріз.
$\overline{INT}0$	*P3.2*	Вхід запиту переривання 0. Сприймається сигнал низького рівня або зріз.
TXD	*P3.1*	Вихід передавача послідовного порту. Вихід синхронізації в режимі регістра, що зсовує.
RXD	*P3.0*	Вхід приймача послідовного порту. Ввід/вивід даних у режимі регістра, що зсовує.

Порт 0 є двонаправленим, а порти 1, 2 і 3 - квазі двонаправленими. Кожна лінія портів може бути використана незалежно для вводу або виводу інформації. Для того щоб деяка лінія порту використовувалася для вводу, у *D*- тригер регістра-засувки порту повинна бути записана 1, що закриває *n-МОП-транзистор* вихідного ланцюга. По сигналу "Скидання" у регістри-засувки всіх портів автоматично записуються одиниці, що настроюють їх тим самим на режим вводу.

Всі порти можуть бути використані для організації вводу/виводу інформації по двонаправленим лініям передачі. Проте порти 0 і 2 не можуть бути використані для цієї цілі у випадку, якщо МК-система має зовнішню пам'ять, зв'язок із якою організується через загальну розділяєму шину адреси/даних, працюючу в режимі тимчасового мультиплексування.

Особливості роботи портів. Звертання до портів вводу/виводу можливо з використанням команд, що оперують із байтом, окремим бітом і довільною комбінацією бітів. При цьому в тих випадках, коли порт є одночасно операндом і місцем призначення результату, пристрій

керування автоматично реалізує спеціальний режим, що називається "читання-модифікація-запис". Цей режим звертання припускає ввід сигналів не з зовнішніх виводів порту, а з його регістра-засувки, що дозволяє виключити неправильне зчитування раніше виведеної інформації. Подібний механізм звертання до портів реалізований у таких командах:

ANL- логічне І, наприклад, *ANL P1, A*;

ORL- логічне АБО, наприклад, *ORL P2, A*;

XRL- АБО, що виключає, наприклад, *XRL P3, A*;

JBC- перехід, якщо в адресуємому біті одиниця, і наступне скидання біта, наприклад, *JBC P1. 1, LABEL*;

CPL- інверсія бита, наприклад, *CPL P3. 3*;

INC- інкремент порту, наприклад, *INC* P2;

DEC- декремент порту, наприклад, *DEC* P2;

DJNZ- декремент порту і перехід, якщо його вміст не дорівнює нулю, наприклад, *DJNZ P3,* LABEL;

MOV PX Y, C- передача біту переносу в біт *Y* порту *X*;

SET PX. Y- установка біта *Y* порту *X*;

CLR PX. Y- скидання біта *Y* порту *X*.

Зовсім не очевидно, що останні три команди в приведеному списку є командами "читання-модифікація-запис". Проте це саме так. По цих командах спочатку зчитується байт із порту, а потім записується новий байт у регістр-засувку.

3.3 Доступ до зовнішньої пам'яті

У микроконтролерних системах, побудованих на основі МК51, можливо використання двох типів зовнішньої пам'яті: постійної пам'яті програм (ВПП) і оперативної пам'яті даних (ВПД). Доступ до ВПП здійснюється за допомогою керуючого сигналу $\overline{PSEN}$, що виконує функцію строб-сигналу читання. Доступ до ВПД забезпечується керуючими сигналами $\overline{RD}$ і $\overline{WR}$, що формуються в лініях Р3.7 і Р3.6 при виконанні портом 3 альтернативних функцій (див. табл. 3.3).

При звертанні до ВПП завжди використовується 16-бітна адреса. Доступ до ВПД можливий із використанням 16-бітної адреси (*MOVX A, @ DPTR*) або 8-бітної адреси (*MOVX A,@ Ri*).

У будь-яких випадках використання 16-бітної адреси старший байт адреси фіксується (і зберігається незмінним протягом одного циклу запису або читання) у регістрі-засувці порту 2.

Якщо черговий цикл зовнішньої пам'яті (*MOVX A,@DPTR*) випливає не відразу ж за попереднім циклом зовнішньої пам'яті, то не-

змінний вміст регістра-засувки порту 2 відновляється в наступному циклі. Якщо використовується 8-бітна адреса (*MOVX A,@ Ri*), то вміст регістра-засувки порту 2 залишається незмінним на його зовнішніх виводах протягом усього циклу зовнішньої пам'яті.

Через порт 0 у режимі тимчасового мультиплексування здійснюється видача молодшого байта адреси і передача байта даних. Сигнал *ALE* повинний бути використаний для запису байта адреси в зовнішній регістр. Потім у циклі запису виведений байт даних з'являється на зовнішніх виводах порту 0 тільки перед появою сигналу $\overline{WR}$. В циклі читання вводимий байт даних приймається в порт 0 по фронті сигналу, $\overline{RD}$ що стробує.

При будь-якому звертанні до зовнішньої пам'яті пристрій керування МК51 завантажує в регістр-засувку порту 0 код *0FFH*, стираючи тим самим інформацію, що могла в ньому зберігатися.

Доступ до ВПП можливий при виконанні двох умов: або на вхід відключення резидентної пам'яті програм $\overline{EA}$ подається активний сигнал, або вміст лічильника команд перевищує значення *0FFFH*. Наявність сигналу $\overline{EA}$ необхідно для забезпечення доступу до молодших 4К адресам адресного простору ВПП при використанні МК31 (мікроконтролера без резидентної пам'яті програм).

Часові діаграми на рис. 3.4 ілюструють процес генерації керуючих сигналів *ALE* (САВП) і $\overline{EA}$ (РВПП) при звертанні до зовнішньої пам'яті.

Основна функція сигналу *ALE* – забезпечити тимчасове узгодження передачі з порту 0 на зовнішній регістр молодшого байта адреси в циклі читання з ВПП. Сигнал *ALE* набуває значення 1 двічі в кожному машинному циклі. Це відбувається навіть тоді, коли в циклі вибірки немає звертання до ВПП. Доступ до ВПП можливий тільки в тому випадку, якщо *ALE* відсутній. Перший сигнал *ALE* у другому машинному циклі команди *MOVX* блокується. Отже в будь-якій МК-системі, що не використовує ВПП, сигнал *ALE* генерується з постійною частотою, рівною 1/16 частоти резонатора, і може бути використаний для синхронізації зовнішніх пристроїв або для реалізації різноманітних часових функцій.

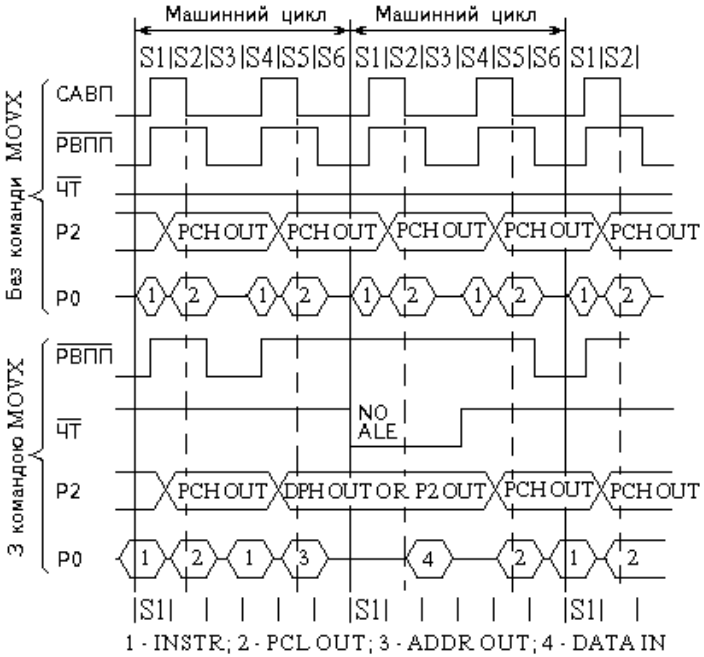


Рис. 3.4

Тимчасові діаграми на рис. 3.5 і 3.6 ілюструють процес вибірки команди з ВПП і роботу з ВПД у режимах читання і запису відповідно.

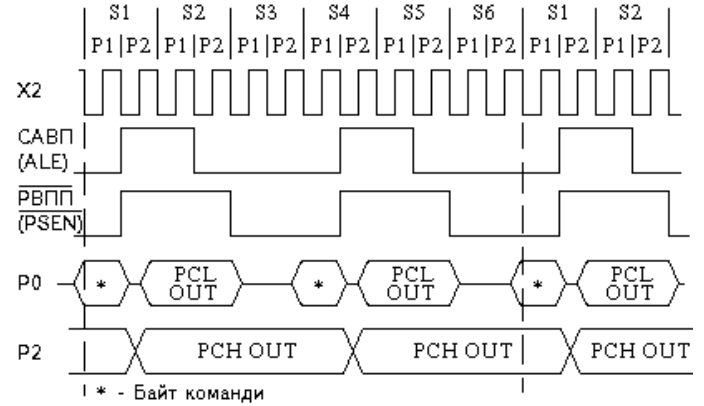


Рис. 3.5

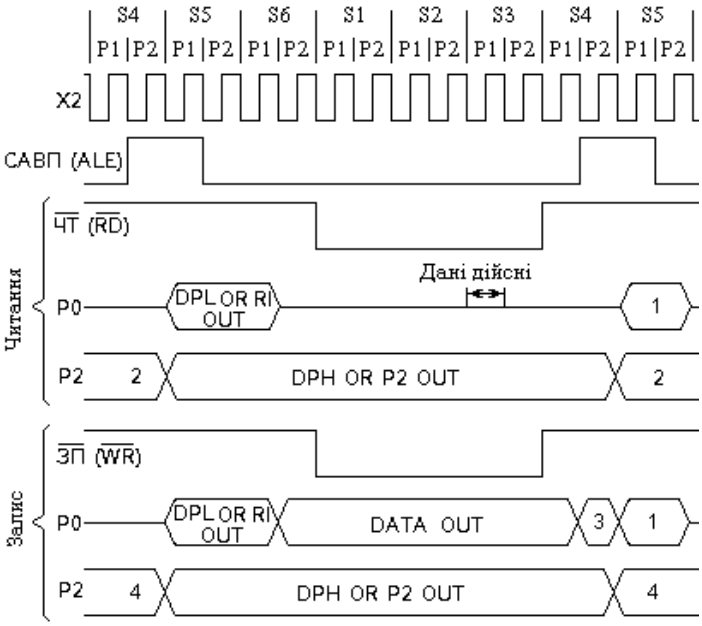


Рис. 3.6

При звертанні до РПП сигнал $\overline{EA}$ не генерується, а при звертанні до ВПП він виконує функцію строб-сигналу читання. Повний цикл читання ВПД, включаючи установку і зняття сигналу $\overline{RD}$, займає 12 періодів резонатора.

3.4 Таймер/лічильник

Два 16-бітних таймера/лічильника (Т/Л0 і Т/Л1), що програмуються, можуть бути використані в якості таймеров або лічильників зовнішніх подій. При роботі в якості таймера вміст Т/Л інкрементирується в кожному машинному циклі, тобто через кожні 12 періодів резонатора. При роботі в якості лічильника вміст Т/Л інкрементирується під впливом переходу з 1 у 0 зовнішнього вхідного сигналу, який подається на відповідний (*T0, T1*) вивід МК51. Вміст лічильника буде збільшено на 1 у тому випадку, якщо в попередньому циклі був зчитаний вхідний сигнал високого рівня (1), а в наступному – сигнал низького рівня (0). Нове

(інкрементироване) значення лічильника буде сформовано в циклі, що випливає за тим , у якому був виявлений перехід сигналу з 1 у 0. Тому що на розпізнавання переходу потрібно два машинних цикли, то максимальна частота підрахунку вхідних сигналів дорівнює 1/24 частоти резонатора.

На тривалість періоду вхідних сигналів обмежень зверху немає. Для гарантованого читання вхідного сигналу, що вводиться, він повинний утримувати значення 1 як мінімум протягом одного машинного циклу МК51.

Для керування режимами роботи Т/С и для організації взаємодії таймеров із системою переривання використовуються два регістри спеціальних функцій (*TMOD* і *TCON*), опис яких наводиться в табл. 3.4 і 3.5 відповідно. Як випливає з опису керуючих бітів *TMOD*, для обох Т/С режими роботи 0,1 і 2 однакові. Режими для Т/С0 і Т/С1 різні. Роздивимося коротко роботу Т/С у всіх чотирьох режимах.

Таблиця 3.4

Символ	Позиція	Ім'я і призначення
GATE	*TMOD*. 7 для Т/С1 і *TMOD*. 3 для Т/С0	Керування блокуванням. Якщо біт установлений, то таймер/лічильник "*x*" дозволений доти, доки на вході "*INTx*" високий рівень і біт керування "*TRx*" установлений. Якщо біт скинен, то Т/С дозволяється, як тільки біт керування "*TRx*" установлюється
$C/\overline{T}$	*TMOD*. 6 для Т/С1 і *TMOD*. 2 для Т/С0	Біт вибору режиму таймера або лічильника подій. Якщо біт скинен, то працює таймер від внутрішнього джерела сигналів синхронізації. Якщо біт установлений, то працює лічильник від зовнішніх сигналів на вході "*Tx*"
M1	*TMOD*. 5 для Т/С1 і *TMOD*. 1 для Т/С0	Режим роботи (див. примітка)
M0	*TMOD*. 4 для Т/С1 і *TMOD*. 0 для Т/С0	Режим роботи (див. примітку)
Примітка.		
M1	*M0*	Режим роботи
0	0	Таймер. "*TLx*" працює як 5-бітний преддільник
0	1	16-бітний таймер/лічильник. "*THx*" і "*TLx*"

		включені послідовно
1	0	8-бітний таймер/лічильник, який автоперезагружається. "*THx*" зберегає значення, що повинно бути перезагружено в "*TLx*" щораз по переповнюванню
1	1	Таймер/ лічильник 1 зупиняється. Таймер/ лічильник 0: *TL0* працює як 8-бітний таймер/ лічильник, і його режим визначається бітами таймера, що встановлюють, 0. *TH0* працює тільки як 8-бітний таймер, і його режим визначається керуючими бітами таймера 1.

Таблиця 3.5

Символ	Позиція	Ім'я і призначення
TF1	*TCON. 7*	Прапор переповнювання таймера 1. Встановлюється апаратно при переповнюванні таймера/лічильника. Скидається при обслуговуванні перериваній апаратно.
TR1	*TCON. 6*	Біт керування таймера 1. Встановлюється/скидається програмою для пуску/зупину
TF0	*TCON. 5*	Прапор переповнювання таймера 0. Встановлюється апаратно. Скидається при обслуговуванні переривання
TR0	*TCON. 4*	Біт керування таймера 0. Встановлюється/скидається програмою для пуску/зупину таймера/лічильника
IE1	*TCON. 3*	Прапор фронту переривання 1. Встановлюється апаратно, коли детектирується зріз зовнішнього сигналу *INT1*. Скидається при обслуговуванні переривання
IT1	*TCON. 2*	Біт керування типів переривання 1. Встановлюється/скидається програмно для специфікації запиту *INT1* (зріз/низький рівень)
IE0	*TCON. 1*	Прапор фронту переривання 0. Встановлюється по зрізі сигналу *INT0*. Скидається при обслуговуванні переривання
IT0	*TCON. 0*	Біт керування типом переривання 0. Встановлюється/скидається програмно для специфікації запиту *INT0* (зріз/низький рівень)

Режим 0. Переведення будь-якого Т/С в режим 0 робить його 8-бітним лічильником, до входу якого підключений 5-бітний преддільник частоти на 32. У цьому режимі таймерний регістр має розрядність 13 біт.

При переході зі стана "всі одиниці" у стан "усі нулі" установлюється прапор переривання від таймера *TF1*.

Режим 1. Робота будь-якого Т/С в режимі 1 така ж, як і в режимі 0, за винятком того, що таймерний регістр має розрядність 16 біт.

Режим 2. У режимі 2 робота організована таким чином, що переповнювання (перехід із стана "всі одиниці" у стан "усі нулі") 8-бітного лічильника *TL1* призводить не тільки до установки прапора *TF1*, але й автоматично перевантажує в *TL1* уміст старшого байта (*TH1*) таймерного регістра, що попередньо був заданий програмним шляхом. Перевантаження лишає вміст *TH1* незмінним. У режимі 2 Т/С0 і Т/С1 працюють цілком однаково.

Режим 3. У режимі 3 Т/С0 і Т/С1 працюють по-різному. Т/С1 зберігає незмінним свій поточний уміст. Іншими словами, ефект такий же, як і при скиданні керуючого біта *TR1* у нуль.

У режимі 3 *TL0* і *TH0* функціонують як два незалежні 8-бітних лічильники. Роботу *TL0* визначають керуючі біти Т/С0 ($C/\overline{T}$, *GATE*, *TR0*), вхідний сигнал *INT0* і прапор переповнювання *TF0*. Роботу *TH0*, що може виконувати тільки функції таймера (підрахунок машинних циклів МК), визначає керуючий біт *TR1*. При цьому *TH0* використовує прапор переповнювання *TF1*.

Режим 3 використовується в тих випадках застосування МК51, коли потрібно наявність додаткового 8-бітного таймера або лічильника подій. Можна вважати, що в режимі 3 МК51 має у своєму складі три таймера/лічильники. У тому випадку, якщо Т/С0 використовується в режимі 3, Т/С1 може бути або включений, або виключений, або переведений у свій власний режим 3, або може бути використаний послідовним портом у якості генератора частоти передачі, або, нарешті, може бути використаний у будь-якому застосуванні, що не потребує переривання.

3.5 Послідовний інтерфейс

Через універсальний асинхронний прийомопередавач (УАПП) здійснюється прийом і передача інформації, поданої послідовним кодом (молодшими бітами вперед), у повному дуплексному режимі обміну. До складу УАПП, який часто називають послідовним портом, входять зсовуючи регістри, що приймають і передають, а також спеціальний буфе-

рний регістр (*SBUF*) прийомопередавача. Запис байта в буфер призводить до автоматичного перепису байта в регістр передавача, що зсовує, і ініціює початок передачі байта. Наявність буферного регістра приймача дозволяє суміщати операцію читання раніше прийнятого байта з прийомом чергового байта. Якщо до моменту закінчення прийому байта попередній байт не був зчитаний із *SBUF*, то він буде загублений.

Послідовний порт МК51 може працювати в чотирьох різноманітних режимах.

Режим 0. У цьому режимі інформація і передається і приймається через зовнішній вивід входу приймача (*RXD*). Приймаються або передаються 8 біт даних. Через зовнішній вивід виходу передавача (*TXD*) видаються імпульси зсуву, що супроводжують кожний біт. Частота передачі біта інформації дорівнює 1/12 частоти резонатора.

Режим 1. У цьому режимі передаються через *TXD* або приймаються з *RXD* 10 біт інформації: старт-біт (0), 8 біт даних і стоп-біт (1). Швидкість прийому/передачи – величина змінна і задається таймером.

Режим 2. У цьому режимі через *TXD* передаються або з RXD приймаються 11 біт інформації: старт-біт, 8 біт даних, дев'ятий біт, що програмується, і стоп-біт. При передачі дев'ятий біт даних може приймати значення 0 і 1, або, наприклад, для підвищення достовірності передачі шляхом контролю по парності в нього може бути приміщене значення ознаки паритету зі слова стана програми (*PSW. 0)*. Частота прийому/передачи вибирається програмою і може дорівнювати або 1/32, або 1/64 частоти резонатора в залежності від керуючого біта SMOD.

Режим3. Режим 3 збігається з режимом 2 у всіх деталях, за винятком частоти прийому/передачи, що є величиною перемінною і задається таймером.

Регістр керування/статусу УАПП. Керування режимом роботи УАПП здійснюється через спеціальний регістр із символічним ім'ям *SCON*. Цей регістр містить не тільки керуючі біти, що визначають режим роботи послідовного порту, але і дев'ятий біт прийнятих або переданих даних *(*RB8 і TB8) і біти переривання прийомопередавача *(*R1 *і* T1).

Функціональне призначення бітів регістра керування/статусу УАПП наводиться в табл. 3.6.

Прикладна програма, шляхом завантаження в старші біти спецрегістра *SCON* 2-бітного коду, визначає режим роботи УАПП. В усіх чотирьох режимах роботи передача з УАПП ініціюється будь-якою командою, в якій буферний регістр *SBUF* зазначений як одержувач байта. Прийом до УАПП у режимі 0 здійснюється за умови, що *R1*=0 і *REN*=1.

У режимах 1, 2, 3 прийом починається з приходом старт-біта, якщо *REN*=1.

Таблиця 3.6

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

Примітка.

SM0	*SM1*	Режим роботи УАПП
0	0	Регістр розширення, що зсовує, вводу/виводу
0	1	УАПП-8 біт. Змінювана швидкість передачі
1	0	УАПП-9 біт. Фіксована швидкість передачі
1	1	УАПП-9 біт. Змінювана швидкість передачі

В біті *TB8* програмно встановлюється значення дев'ятого біта даних, що буде переданий у режимі 2 або 3. У біті *RB8* фіксується, у

режимах 2 і 3, дев'ятий прийнятий біт даних. У режимі 1, якщо *SM2*=0, у біт *RB8* заноситься стоп-біт. У режимі 0 біт *RB8* не використовується.

Прапор переривання передавача *TI* встановлюється апаратно наприкінці періоду передачі восьмого біта даних у режимі 0 і на початку періоду передачі стоп-біта в режимі 1, 2 і 3. Відповідна підпрограма обслуговування переривання повинна скидати біт *TI*.

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

3.6 Система переривань

Спрощена схема переривань МК51 показана на рис. 3.7.

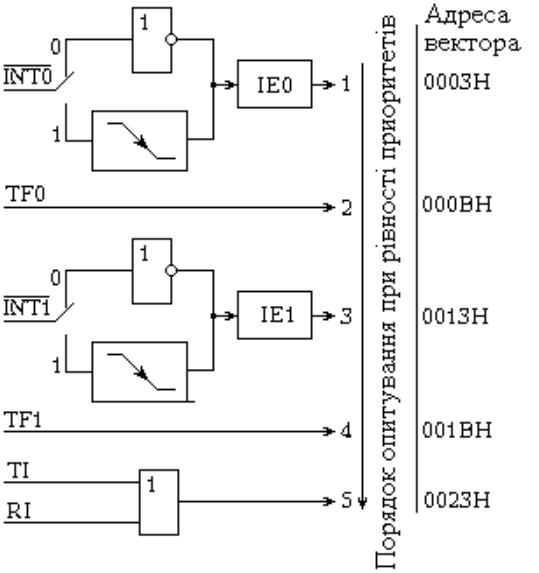


Рис. 3.7

Зовнішні переривання $\overline{INT\ 0}$ і $\overline{INT\ 1}$ можуть бути викликані або рівнем, або переходом сигналу з 1 у 0 на входах МК51 в залежності від значень керуючих біт *IT0* і *IT1* у регістрі *TCON*. Від зовнішніх переривань встановлюються прапори *IE0* і *IE1* у регістрі *TCON*, що ініціюють виклик відповідної підпрограми обслуговування переривання. Скидання цих прапорів виконується апаратно тільки в тому випадку,

якщо переривання було викликано по переході (зрізу) сигналу. Якщо ж переривання викликане рівнем вхідного сигналу , то скиданням прапора *IE* керує відповідна підпрограма обслуговування переривання, шляхом впливу на джерело переривання з метою зняття їм запиту.

Прапори запитів переривання від таймеров *TF0* і *TF1* скидаються автоматично при передачі керування підпрограмі обслуговування. Прапори запитів переривання *RI* і *TI* установлюються блоком керування УАПП апаратно, але скидатися повинні програмою.

Переривання можуть бути викликані або скасовані програмою, тому що всі перераховані прапори програмно-доступні і можуть бути встановлені/скинені програмою з тим же результатом, як якби вони були встановлені/скинені апаратурними засобами.

У блоці регістрів спеціальних функцій є два регістри, призначених для керування режимами переривань і рівнями пріоритету. Формати цих регістрів, що мають символічні імена *IE* і *IP*, описані в табл. 3.7 і 3.8 відповідно. Можливість програмної установки/ скидання будь-якого керуючого бита в цих двох регістрах робить систему переривань МК51 винятково гнучкою.

Прапори переривань опитуються в момент *S5P2* кожного машинного циклу. Ранжування переривань за рівнем пріоритету виконується протягом наступного машинного циклу. Система переривань сформує апаратно виклик (*LCALL*) відповідної підпрограми обслуговування, якщо вона не заблокована однією з наступних умов:

1) в даний момент обслуговується запит переривання рівного або більш високого рівня пріоритету;

2) поточний машинний цикл – не останній в циклі виконуваної команди;

3) виконується команда *RETI* або будь-яка команда, пов'язана з звертанням до регістрів *IE* або *IP*.

Відзначимо, що якщо прапор переривання був встановлений, але по одній з перерахованих вище умов не одержав обслуговування і до моменту закінчення блокування вже був скинен, то запит переривання губиться і ніде не запам'ятовується.

По апаратно сформованому коді *LCALL* система переривання поміщає в стек тільки вміст лічильника команд (*PC*) і завантажує в лічильник команд адреси вектора відповідної підпрограми обслуговування.

Таблиця 3.7

Символ	Позиція	Ім'я і призначення
EA	*IE. 7*	Зняття блокування переривань. Скидається програмно для заборони всіх переривань незалежно

		від стана *IE. 4- IE.0*
- -	*IE. 6* *IE. 5*	Не використовуються
ES	*IE. 4*	Біт дозволу переривання від УАПП. Установка/скидання програмою для дозволу/заборони переривань від прапорів *TI* або *RI*
ET1	*IE. 3*	Біт дозволу переривання від таймера 1. Установка/скидання програмою для дозволу/заборони переривань від таймера 1
EX1	*IE. 2*	Біт дозволу зовнішнього переривання 1. Установка/скидання програмою для дозволу/заборони переривань
ET0	*IE. 1*	Біт дозволу переривання від таймера 0. Працює аналогічно *IE.* 3
EX0	*IE. 0*	Біт дозволу зовнішнього переривання 0. Працює аналогічно *IE.* 2

Таблиця 3.8

Символ	Позиція	Ім'я і призначення
-	*IP. 7-IP. 5*	Не використовуються
PS	*IP. 4*	Біт пріоритету УАПП. Установка/скидання програмою для присвоювання перериванню від УАПП вищого/нижчого пріоритету
PT1	*IP. 3*	Біт пріоритету таймера 1. Установка/скидання програмою для присвоювання перериванню від таймера 1 вищого/нижчого пріоритету
PX1	*IP. 2*	Біт пріоритету зовнішнього переривання 1. Установка/скидання програмою для присвоювання вищого/нижчого пріоритету зовнішньому перериванню *INT1*
PT0	*IP. 1*	Біт пріоритету таймера 0. Працює аналогічно *IP. 3*
PX0	*IP. 0*	Біт пріоритету зовнішнього переривання 0. Працює аналогічно *IP. 2*

За адресою вектора повинна бути розкладена команда безумовної передачі керування (*JMP*) до початкової адреси підпрограми обслуговування переривання. Підпрограма обслуговування в разі потреби повинна починатися командами запису в стек (*PUSH*) слова стана програми (*PSW*), акумулятора, розширювача, покажчика даних і т.д. і за-

кінчується командами відновлення зі стека (*POP*). Підпрограми обслуговування переривання обов'язково завершуються командою *RETI*, по якій у лічильник команд перевантажується зі стека збережена адреса повернення в основну програму. Команда *RET* також повертає керування перерваній основній програмі, але при цьому не знімає блокування переривань, що призводить до необхідності мати програмний механізм аналізу закінчення процедури обслуговування даного переривання.

3.7 Система команд мікроконтролера К1816ВЕ51

3.7.1 Загальні відомості

Система команд МК51 містить 111 базових команд, що зручно розділити по функціональній ознаці на п'ять груп: команди передачі даних, арифметичних операцій, логічних операцій, передачі керування й операцій із бітами.

Більшість команд (94) мають формат один або два байти і виконуються за один або два машинних цикли. При тактовій частоті 12 МГц тривалість машинного циклу складає 1 мкс.

На рис. 3.8 показані 13 типів команд МК51. Перший байт команди будь-якого типу і формату завжди містить код операції (КОП). Другий і третій байти містять або адреси операндів, або безпосередні операнди.

Типи операндів. Операнди в МК51 бувають чотирьох типів: біти, 4-бітні цифри, байти і 16-бітні слова.

Мікроконтролер має 128 програмно-керованих прапори користувача. Є можливість адресації окремих біт блока регістрів спеціальних функцій і портів. Для адресації біт використовується пряма 8-бітна адреса (*bit*). Непряма адресація біта неможлива. Карти адрес окремих біт подані на рис. 3.9 і 3.10.

Чотирибітні операнди використовуються тільки при операціях обміну (команди *SWAP* і *XCHD*). Восьмибітний операнд може бути коміркою пам'яті програм або даних (резидентною або зовнішньою), константою (безпосередній операнд), регістром спеціальних функцій (РСФ), а також портами вводу/виводу. Порти і РСФ адресуються тільки прямим способом. Байти пам'яті можуть адресуватися також і непрямим образом через адресні регістри (*R0, R1, DPTR* і *PC*). Двобайтові операнди — це константи і прямі адреси, для представлення яких використовуються другий і третій байт команди.

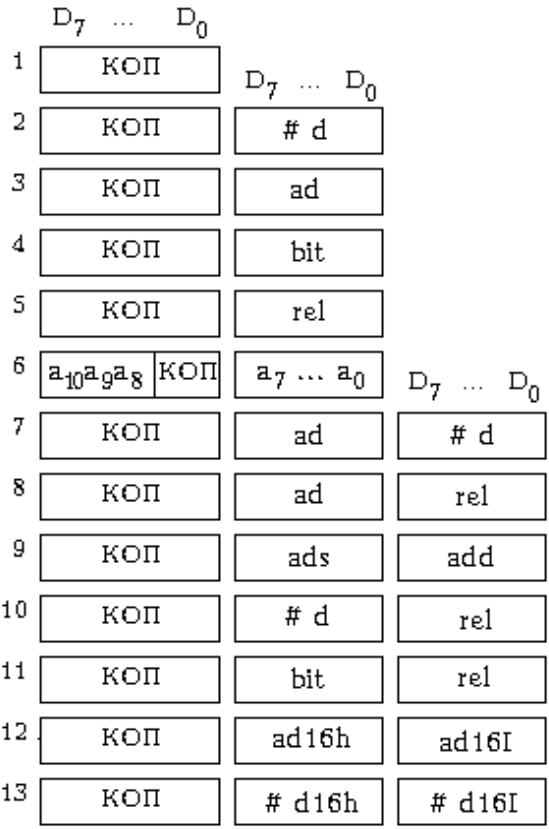


Рис. 3.8

Способи адресації даних. В МК51 використовуються наступні види адресації: пряма, безпосередня, непряма і неявна. Слід зазначити, що при непрямої адресації використовуються тільки регістри *R0* і *R1*.

Прапори результату. Слово стана програми (*PSW*) містить у собі чотири прапори: *C* — перенос, *AC* — допоміжний перенос, *OV* — переповнювання і *P* — паритет.

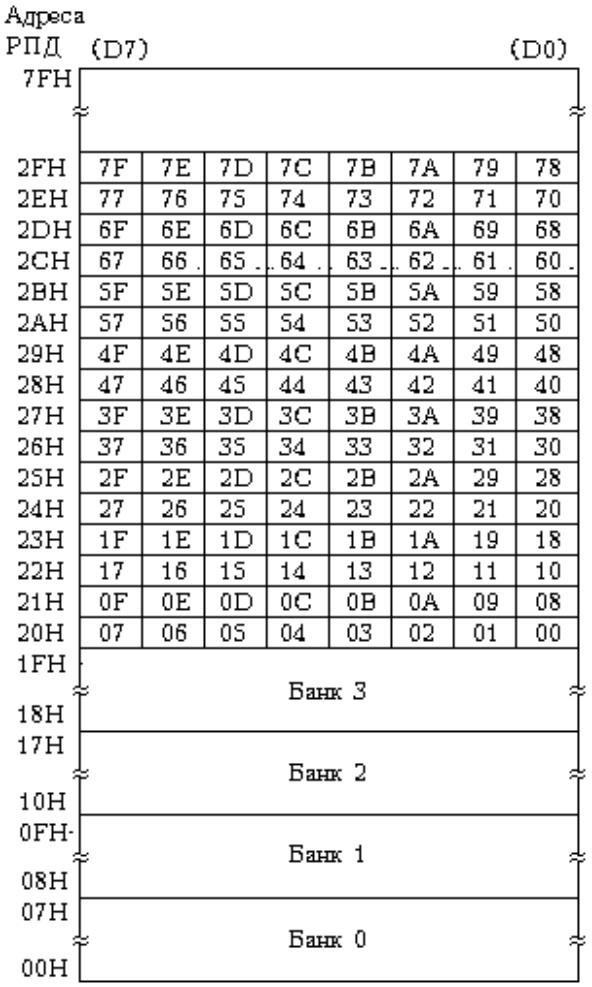


Рис. 3.9

У табл. 3.9 перераховуються команди, при виконанні яких модифікуються прапори результату.

Таблиця 3.9

Команди	Прапори	Команди	Прапори
ADD	*C,OV,AC*	*CLR C*	*C=0*
ADDC	*C,OV,AC*	*CPL C*	$C=\overline{C}$
SUBB	*C,OV,AC*	*ANL C,b*	*C*

Команди	Прапори	Команди	Прапори
MUL	*C=0, OV*	*ANL C,/b*	*C*
DIV	*C=0, OV*	*ORL C,b*	*C*
DA	*C*	*ORL C,/b*	*C*
RRC	*C*	*MOV C,b*	*C*
RLC	*C*	*CJNE*	*C*
SETB C	*C=1*		

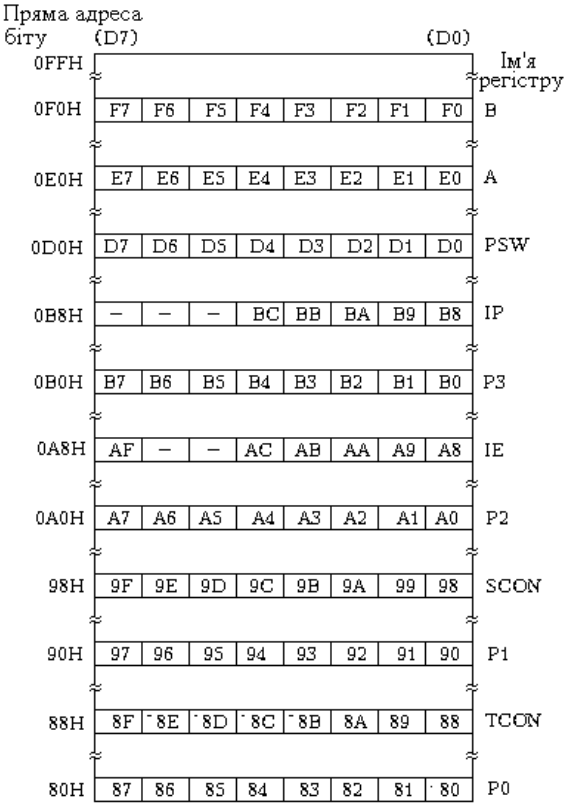


Рис. 3.10

В таблиці відсутній прапор паритету, тому що його значення змінюється всіма командами, що змінюють вміст акумулятора. Крім команд, приведених в таблиці, прапори модифікуються командами, в яких місцем призначення результату визначені *PSW* або його окремі біти, а також командами операцій над бітами.

Символічна адресація. При використанні Асемблера МК51 (*ASM51*) для одержання об'єктних кодів програм припускається застосування в програмах символічних імен регістрів спеціальних функцій, портів і їх окремих біт (рис. 3.10).

Для адресації окремих біт регістрів спеціальних функцій і портів можна використовувати символічне ім'я біта наступної структури:

<ім'я РСФ або порту>. <номер біта>

Наприклад, символічне ім'я п'ятого біта акумулятора буде наступним: *ACC. 5*. Символічні імена РСФ, портів і їх біт є зарезервованими словами для *ASM51*, і їх не треба визначати за допомогою директив Асемблера.

Структура інформаційних зв'язків. В залежності від способу адресації і місця розташування операнда можна виділити дев'ять типів операндів, між якими можливий інформаційний обмін. Граф можливих операцій передачі даних показаний на рис. 3.11.

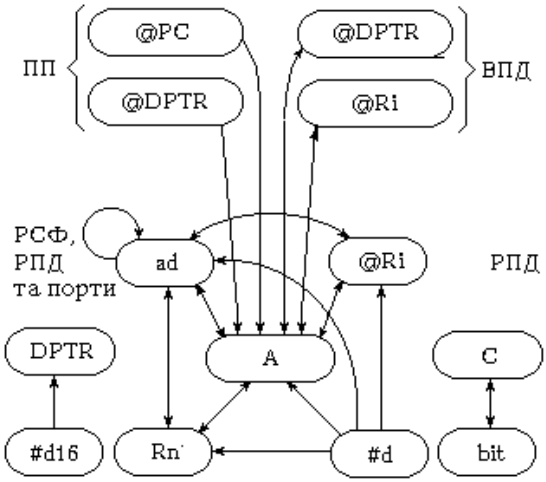


Рис. 3.11

Акумулятор (*A*) поданий на цьому графі окремою вершиною, тому що багато команд використовують неявну адресацію.

Акумулятор. Звертання до акумулятора, в МК51, може бути виконане з використанням неявної і прямої адресації. В залежності від способу адресації акумулятора застосовується одне із символічних імен: *A* або *ACC* (пряма адреса). При прямій адресації звертання до акумулятора провадиться як до одного з РСФ, і його адреса вказується в другому байті команди.

Використанню неявної адресації акумулятора надається перевага, проте це не завжди можливо, наприклад, при звертанні до окремих бітів акумулятора.

Звертання до зовнішньої пам'яті даних. При використанні команд *MOVX @Ri* забезпечується доступ до 256 байтів зовнішньої пам'яті даних.

Існує також режим звертання до розширеного ЗПД, коли для доступу використовується 16-бітна адреса, що зберігається в регістрі-покажчику даних *(DPTR).* Команди *MOVX @DPTR* забезпечують доступ до 65536 байтів ВПД.

3.7.2 Команди мікроконтролера К1816ВЕ51

Опис кожної машинної команди складається зі словесного опису операції, яка виконється, і представлення машинної команди у вигляді таблиці.

Таблиця, що подає машинну команду, складається з чотирьох граф: "код", "інструкція", "алгоритм" і "час".

В графі "код" наводиться двійковий код машинної команди, згрупований у байти.

При записі коду команди використовуються наступні позначення:

- *R* – біт коду робочого регістра-покажчика *R0, R1*. Встановлюється наступна відповідність між кодом робочого регістра-покажчика і його ім'ям:

код: 0 1
ім'я: *R0 R1*

- *RRR* – три бита коду робочого регістра *R0 - R7*. Встановлюється наступна відповідність між кодом робочого регістра і його ім'ям:

код: 000 001 010 011 100 101 110 111
ім'я: *R0 R1 R2 R3 R4 R5 R6 R7*

- *DDDDDDDD* – байт безпосередніх даних;

- *BBBBBBBB* або *BSBSBSBS* або *BDBDBDBD* – байт прямої адреси регістра спеціальної функції або комірки внутрішньої пам'яті даних;

- *IIIIIII* – байт прямої адреси тригера спеціальної функції або розряду комірки внутрішньої побітово адресуємої пам'яті даних;

- *DHDHDHDH DLDLDLDL* – слово безпосередніх даних (перший байт старший);

- *AAA ALALALAL* – пряма 11-розрядна адреса пам'яті програм (*AAA* - старші 3 біти);

- *AHAHAHAH ALALALAL* – пряма 16-розрядна адреса пам'яті програм (перший байт - старший);

- *SSSSSSSS* – байт позиційно незалежного зміщення.

У графі "інструкція" наводиться форма запису інструкції мови Асемблера, що подає відповідну машинну команду. У формі запису інструкції використовуються такі позначення:

- <*R*> – ім'я робочого регістра-покажчика *R0, R1*;

- <*RRR*> – ім'я робочого регістра *R0-R7*;

- <*D*> – вираз для визначення байта безпосередніх даних;

- <*B*> або <*BS*> або <*BD*> – вираз для визначення байта прямої адреси регістра спеціальної функції або комірки внутрішньої пам'яті даних;

- <*I*> – вираз для визначення байта прямої адреси тригера спеціальної функції або розряду комірки внутрішньої побітово адресуємої пам'яті даних;

- <*D16*> – вираз для визначення слова безпосередніх даних;

- <*A11*> – вираз для визначення прямої 11-бітної адреси;

- <*A16*> – вираз для визначення прямої 16-бітної адреси;

- <*S*> – вираз для визначення байта позиційно -незалежного зміщення.

У графі "алгоритм" наводиться алгоритм виконання машинної команди МК51. При записі алгоритму використовуються такі позначення:

- (*X*) – вміст устрою X;

- ((*X*)) – вміст за адресою, що зберігається в пристрої *X*;

- *X[M]* – розряд *M* пристрою *X*;

- *X[M1-MN]* – група розрядів *M1-MN* пристрою *X*;

- *X:Y* – об'єднання(конкатенація) розрядів пристроїв *X* (старша частина) і *Y*;

-TMP – допоміжний регістр.

У графі "час" наведена кількість циклів МК51, необхідних для виконання команди. Час виконання одного машинного циклу визначається по наступній формулі:

$$T = 12 / F ,$$

де *T* - час циклу, мкс; *F* - частота генератора тактових сигналів, МГц.

Перелік команд, що встановлюють тригери ознак результату - *C, AC,* P *i* OV наведений у табл. 3.10.

Тригери ознак результату можуть бути встановлені командами звертання по прямій адресі до регістра спеціальної функції *PSW* (адреса *0D0H*) або до тригера спеціальної функції *C* (адреса *0D7H*), *AC* (адреса *0D6H*), *OV* (адреса *0D2H*) і *P* (адреса *0D0H*).

Арифметичні команди *ADD, ADDC* і *SUBB* установлюють тригери ознак результату *C, AC* і *OV* у такий спосіб.

Якщо при виконанні операції додавання був перенос із 7-го розряду або при виконанні операції віднімання була позика в 7-й розряд, то тригер ознаки переносу *C* встановлюється в стан "1", у противному випадку – у стан "0".

Таблиця 3.10

Команда	Ознаки результату			
	C	*AC*	*OV*	*P*
ADD	+	+	+	+
MUL	0		+	+
ADDC	+	+	+	+
SUBB	+	+	+	+
DIV	0		+	+
DA	+			+
RRC	+			+
RLC	+			+
SETB C	1			
CLR C	0			
CPLC	+			
ANL C, <I>	+			
ANL C, <I>	+			
ORL C, <I>	+			
ORL C,/ <I>	+			
MOL C, <I>	+			
CJNZ	+			

Якщо при виконанні операції додавання був перенос із 3-го розряду або при виконанні операції віднімання була позика в 3-й розряд, то тригер ознаки допоміжного переносу *AC* встановлюється в стан "1", у противному випадку в стан "0".

Якщо при виконанні операції був перенос із 6-го розряду і не було переносу з 7-го розряду або не було переносу з 6-го розряду і був перенос із 7-го розряду, то тригер ознаки переповнювання *OV* встановлюється в стан "1", у противному випадку - у стан "0".

Тригер ознаки парності *P* встановлюється при виконанні тільки операцій, для яких операнд-приймач розміщається в акумуляторі. Якщо при виконанні такої операції сума зі модулем 2 вмісту всіх розрядів акумулятора дорівнює 1, то тригер ознаки парності встановлюється в стан "1", у противному випадку, у стан "0".

Операнд-приймач операції МК51, в інструкції, що містить два аргументи, визначається першим аргументом, в інструкції, що містить три аргументи – другим аргументом.

Значення операнда-джерела, що знаходиться в пристрої паралельного вводу-виводу (порт *P0-P3*) по прямій адресі *BBBBBBBB* або в розряді цього пристрою по прямій адресі *IIIIIII*, у процесі виконання команди зчитується з виводів МК51 значення аналогічного операнда-приймача зчитується з фіксатора відповідного пристрою. Результат в обох випадках записується у фіксатор пристрою.

ACALL

По команді *ACALL* (табл.3.11) здійснюється виклик підпрограми, розташованої в поточному банку пам'яті програм ємністю 2048 байт, у такий спосіб:

– вміст лічильника команд *PC* (тобто повна адреса наступної команди) записується у верхівку стека (молодший байт адреси записується першим);

– вміст покажчика стека *SP* збільшується на два;

– в розряди 0–10 лічильника команд записується довга (11-бітна) адреса підпрограми, що зберігається в коді команди. Вміст розрядів 11–15 лічильника команд не змінюється.

Таким чином, точка призначення команди *ACALL* повинна розміщатися в тій ж сторінці пам'яті програм, що і її перший байт.

Таблиця 3.11

Код	Інструкція	Алгоритм	Час
AAA10001 *ALALALAL*	*ACFLL <A11>*	*(SP):= (SP)+1* *((SP)):= (PC[7-0])* *(SP):= (SP)+1* *((SP)):= (PC[15-8])* *(PC[10-0]):= AAA:* *AL...AL*	2

Приклад виконання команди:

```
        ;(PC)=0035H,(SP)=26H,((SP)+1)=00H,
        ;((SP)+2)=00H
        ;proc = 0444H
@@1: ACALL proc
        ;(PC)<=0444H,(SP)<=28H,((SP)):((SP)-1)<=0035H+ +2
        ORG 444H
proc:  PUSH  ACC
        RET
```

ADD

По команді ADD (табл.3.12):

– операнд-джерело додається до вмісту акумулятора;

– результат записується в акумулятор;

– тригери ознак результату *(C), (AC), (OV)* і (*P*) встановлюються відповідно до табл. 3.10.

Таблиця 3.12

Код	Інструкція	Алгоритм	Час
00101RRR	*ADD A,<R>*	*(A):=(A)+(RRR)* *(C),(AC),(OV),(P):=0\1*	1
0010011R	*ADD A,@<R>*	*(A):=(A)+((R))* *(C),(AC),(OV),(P):=0\1*	1
00100100 *DDDDDDDD*	*ADD A,#<D>*	*(A):=(A)+D...D* *(C),(AC),(OV),(P):=0\1*	1
00100101 *BBBBBBBB*	*ADD A,<B>*	*(A):=(A)+(B...B)* *(C),(AC),(OV),(P):=0\1*	1

Приклад виконання команди:

;(A) = 0C3H, (R0) = 0AAH

@@1: ADD A,R0

;(A) ← 0C3H+ 0AAH = 6DH,

;(C) ← 1, (AC) ← 1, (OV) ← 1, (P) ← 0

ADDC

По команді *ADDC* (табл.3.13):

– додають операнд-джерело, біт ознаки переносу (*C*) і вміст акумулятора;

– результат записується в акумулятор;

– тригери ознак результату *(C), (AC), (OV)* і *(P)* встановлюються відповідно до табл. 3.10.

Таблиця 3.13

Код	Інструкція	Алгоритм	Час
00111RRR	*ADDC A,<R>*	*(A):=(A)+(RRR)+(C)* *(C),(AC),(OV),(P):=0\1*	1
0011011R	*ADDC A,@<R>*	*(A):=(A)+((R))+(C)* *(C),(AC),(OV),(P):=0\1*	1
00110100 *DDDDDDDD*	*ADDC ,#<D>*	*(A):=(A)+D... D+(C)* *(C),(AC),(OV),(P):=0\1*	1
00110101 *BBBBBBBB*	*ADDC A,<B>*	*(A):=(A)+(B... B)+(C)* *(C),(AC),(OV),(P):=0\1*	1

Приклад виконання команди:

;(A) = 0C3H, (R0) = 0AAH, (C) = 0

@@1: ADDC A,R0

;(A) ← 0C3H + 0AAH + 0 = 6DH

;(C) ← 1, (AC) ← 1, (OV) ← 1, (P) ← 0

AJMP

По команді *AJMP* (табл.3.14) здійснюється перехід у точку призначення, розташовану в поточному банку пам'яті програм ємністю 2048 байт, у такий спосіб:

В розряди 0-10 лічильника команд записується довга (11-бітна) адреса точки призначення, подана 7-5 розрядами першого байта *AAA* (старші три розряди) і другим (молодші вісім розрядів) байтом *AL... AL* коду команди. Вміст розрядів 11−15 лічильника команд не змінюється.

Таким чином, точка призначення команди *AJMP* повинна розміщатися на тій ж сторінці пам'яті програм, що і її перший байт.

Таблиця 3.14

Код	Інструкція	Алгоритм	Час
AAA00001 ALALALAL	*AJMP <A11>*	*(PC[10-0]):=AAA:AL... AL*	2

Приклад виконання команди:

loop: MOV A,R1

;loop = 0E80FH

...

;(PC) = 0EADCH

@@1: AJMP loop

;(PC) ← 0E80FH

ANL

По команді *ANL* (табл. 3.15):

- операнд-джерело порозрядно логічно умножається на операнд-приймач;

- результат записується на місце операнда-приемача.

Якщо в якості операнда-приймача використовується вміст акумулятора, то тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.15

Код	Інструкція	Алгоритм	Час
01011RRR	*ANL A, <R>*	*(A):= (A) AND (RRR)* *(P):= 0\1*	1
0101011R	*ANL A,@<R>*	*(A):= (A) AND ((R))*	1

		(P):= 0\1	
01010100 DDDDDDDD	*ANL A, #<D>*	*(A):= (A) AND D... D (P):= 0\1*	1
01010101 BBBBBBBB	*ANL A, <B>*	*(A):= (A) AND (B... B) (P):= 0\1*	1

01010010 BBBBBBBB	*ANL <B>, A*	*(B... B):= (B... B)AND(A)*	2
01010011 BBBBBBBB DDDDDDDD	*ANL <B>, #D*	*(B... B):= (B... B) AND D... D*	2
10000010 I I I I I I I I	*ANL C, <BIT>*	*(C):= (C) AND (I... I)*	2
10110000 I I I I I I I I	*ANL C,/<BIT>*	*(C):= (C) AND (NOT(I... I))*	2

Приклади виконання команди:

```
                ;(P1) = 01010101H
@@11: ANL P1,# 00001111H
                ;(P1) ← 00000101H
                ;установити біт переносу в одиницю, якщо
                ;виконується наступна умова: (P1. 0)=1, ;(ACC. 7)=1
                i ; (OV)=0
@@12: MOV C,P1. 0
          ANL C,ACC. 7
          ANL C,/OV
```

CALL

По команді *CALL* (табл. 3.16) здійснюється виклик підпрограми (подібно командам *ACALL* і *LCALL*).

Таблиця 3.16

Код	Інструкція	Алгоритм	Час
AAA10001 ALALALAL	*CALL* <вир> ;вел (выр)<=2047	см. інструкцію *ACALL*	2
00010010 AHAHAHAH ALALALAL	*CALL* <вир> ;вел (вир)<=2048	см. інструкцію *LCALL*	2

CJNE

По команді *CJNE* (табл. 3.17) здійснюється розгалуження в програмі в такий спосіб.

Якщо операнд-приймач (прм) не дорівнює операнду-джерелу (джр), то керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, подане третім байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

Якщо операнд-приймач менше операнда-джерела, то тригер ознаки переносу встановлюється в стан "1", у противному випадку - у стан "0".

Таблиця 3.17

Код	Інструкції	Алгоритм	Час
10110101 *BBBBBBBB* *SSSSSSSS*	*CJNE A,<B>, <S>* *;(A) - прм* *;(B... B) - ист*	*(PC):= (PC)+3* *IF прм <> ист THEN* *(PC):= (PC)+S... S* *END* *IF прм < ист THEN* *(C):=1* *ELSE (C):=0* *END*	2
10110100 *DDDDDDDD* *SSSSSSSS*	*CJNE A,#<D>, <S>* *;(A) - прм* *;D... D - ист*		2
10111RRR *DDDDDDDD* *SSSSSSSS*	*CJNE R,#<D>, <S>* *;(RRR) - прм* *;D... D - ист*		2
1011011R *DDDDDDDD* *SSSSSSSS*	*CJNE @R,#<D>, <S>* *;((R)) - прм* *;D... D - ист*		2

Приклад виконання команди:

```
    ORG   0100H
          ;(A)= 0, (P1) = 34H
       ;S... S = loop-($+2) = 0125H - (0100H+2) = 23H
          ;(PC) = 0100H
@@1:  CJNE  A, P1, loop
          ;(PC) ← (PC) + 2 = 0100H + 2 = 0102H
          ;(PC) ← (PC) + S... S = 0102H + 23H = 0125H
      ...
loop:  MOV   A,R1
          ;loop = 0125H
```

CLR

По команді *CLR* (табл. 3.18) операнд-приймач встановлюється в стан "нуль".

Якщо в якості операнда-приймача використовується вміст акумулятора, то тригер ознаки парності (*P*) встановлюється в стан "1".

Таблиця 3.18

Код	Інструкція	Алгоритм	Час
11100100	*CLR A*	*(A):=0* *(P):=1*	1
11000011	*CLR C*	*(C):=0*	1
11000010 *I I I I I I I I*	*CLR <BIT>*	*(I... I):=0*	2

Приклади виконання команди:

```
                ;(A) = 5CH
@@11: CLR   A
                ;(A) ← 00H
                ;(P1) = 01011101B
@@12: CLR   P1. 2
                ;(P1) ← 01011001B
```

CPL

По команді *CPL* (табл. 3.19) операнд-приймач інвертується.

Таблиця 3.19

Код	Інструкція	Алгоритм	Час
11110100	*CPL A*	*(A):= NOT (A)*	1
10110011	*CPL C*	*(C):= NOT (C)*	1
10110010 *I I I I I I I I*	*CPL <BIT>*	*(I... I):= NOT (I... I)*	2

Приклади виконання команди:

```
                ;(A) = 10101010B
@@11: CPL   A
                ;(A) ← 01010101B
                ;(P1) = 01011101B
@@12: CPL   P1. 1
                ;(P1) ← 01011011B
```

DA

По команді DA (табл. 3.20) результат двійкового додавання упакованих двійково-десяткових чисел в акумуляторі (додавання вико-

нується командою *ADD* або *ADDC*) перетвориться в упаковане двійково-десяткове число в такий спосіб.

Якщо число в молодших чотирьох розрядах акумулятора більше дев'ятьох (тобто XXXX1010B-XXXX1111B) або біт ознаки допоміжного переносу *AC* дорівнює одиниці, то до вмісту акумулятора додається число 6. При переносі з 7-го розряду акумулятора тригер ознаки переносу *C* встановлюється в стан "1".

Таблиця 3.20

Код	Інструкція	Алгоритм	Час
11010100	*DA A*	*IF (A[3-0])> 9 OR (AC)=1* *THEN IF ((A)+6)> OFFH* *THEN (C):=1* *END* *(A):= (A)+6* *END* *IF (A[7-4])> 9 OR (C)=1* *THEN IF((A)+6OH)> OFFH* *THEN (C):=1* *END* *(A):= (A)+60H* *END* *(P):=0\1*	1

Якщо число в старших чотирьох розрядах акумулятора більше дев'ятьох (тобто 1010XXXXB-1111XXXXB) або *біт* ознаки переносу дорівнює одиниці, то до вмісту акумулятора додається число 60H. При переносі з 7-го розряду акумулятора тригер ознаки переносу встановлюється в стан "1".

Тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Приклад виконання команди:

```
                ;(A) = 56H (тобто двійково-десяткове число 56)
        ;(R3) = 67H тобто двійково-десяткове число 67)
        ;(C) = 1
@@1:  ADDC  A,R3
        ;(A) ← 0BEH, (C) ←0, (AC) ← 0
          DA   A
        ;(C) ← 1, (A) ← 24H
        ;( тобто двійково-десяткове число 124)
```

DEC

По команді *DEC* (табл. 3.21) операнд-приймач зменшується на одиницю.

Якщо в якості операнда-приймача використовується вміст акумулятора, то тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

З стана *00H* операнд-приймач переходить у стан *0FFH*.

Таблиця 3.21

Код	Інструкція	Алгоритм	Час
00010100	*DEC A*	*(A):= (A)-1* *(P):= 0\1*	1
00011RRR	*DEC <R>*	*(RRR):= (RRR)-1*	1
0001011R	*DEC @<R>*	*((R)):= ((R))-1*	1
00010101 *BBBBBBBB*	*DEC <B>*	*(B... B):=(B... B)-1*	1

Приклади виконання команди:

```
                ;(R0) = 7FH, ((R0)-1) = 00H, ((R0)) = 40H
@@1:  DEC   @R0
      DEC   R0
      DEC   @R0
            ;(R0) ← 7FH- 1 = 7EH
            ;((R0)) ← 0FFH, ((R)+1) ← 41H
```

DIV

По команді *DIV* (табл. 3.22):

– вміст акумулятора (ділене) ділиться на вміст регістра *B* (дільник);

– ціла частина результату (частка) записується в акумулятор, залишок від розподілу - у регістр *B*;

– тригер ознаки переносу *(C)* встановлюється в стан "0"; тригер ознаки переповнювання (*OV*) встановлюється в стан "1", якщо дільник у регістрі B дорівнює нулю, у противному випадку -в стан "0"; тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Операнди команди розглядаються, як порядкове число. Якщо дільник дорівнює нулю, то результат виконання команди не визначений.

Таблиця 3.22

Код	Інструкція	Алгоритм	Час
10000100	*DIV AB*	*IF (B)=0* *THEN (OV):=1* *ELSE (OV):=0*	4

		END *(TMP):=(A)/(B)* *(B) :=(A) MOD (B)* *(A) :=(TMP)* *(C) :=0* *(P) :=0\1*	

Приклади виконання команди:

```
        ;(A)= 0FBH,(B) = 12H
@@1:  DIV   AB
        ;(A) ←0DH і (B) ←11H, тому що
        ;0FBH=0DH*12H+11H
        ;(C) ← 0, (OV) ← 0, (P) ← 0
```

DJNZ

По команді *DJNZ* (табл. 3.23) здійснюється розгалуження в програмі в такий спосіб.

Операнд-приймач зменшується на одиницю. Потім, якщо операнд-примач не дорівнює нулю, то керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, поданий третім байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

При зменшенні операнд-приймач із стана *00H* переходить у стан *0FFH*.

Таблиця 3.23

Код	Інструкція	Алгоритм	Час
11011RRR *SSSSSSSS*	*DJNZ <R>,<S>*	*(PC):=(PC)+2* *(RRR):=(RRR)-1* *IF (RRR)<>0 THEN* *(PC):=(PC)+S... S* *END*	2
11010101 *BBBBBBBB* *SSSSSSSS*	*DJNZ <B>,<S>*	*(PC):=(PC)+3* *(B... B):=(B... B)-1* *IF (B... B)<>0 THEN* *(PC):=(PC)+S... S* *END*	2

Приклад виконання команди:

```
    ORG   0100H
        ;(P1) = 34H
        ;S...S = loop - ($+2) = 0125H - (0100H+2)=23H
        ;(PC) = 0100H
```

```
@@1:  DJNZ  P1, loop
                ;(P1) ← 33H
                ;(PC) ← (PC) + 2 = 0100H + 2 = 0102H
                ;(PC) ← (PC) + S...S = 0102H + 23H = 0125H
          ...
loop:   MOV   A,R1
                ;loop = 0125H
```

INC

По команді *INC* (табл. 3.24) операнд-приймач збільшується на одиницю.

Якщо в якості операнда-приймача використовується вміст акумулятора, то тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

З стана *0FFH* операнд-приймач переходить у стан *00H*.

Таблиця 3.24

Код	Інструкція	Алгоритм	Час
00000100	*INC A*	*(A):=(A)+1* *(P):=0\1*	1
00001RRR	*INC <R>*	*(RRR):=(RRR+1)*	1
0000011R	*INC @<R>*	*((R)):=((R))+1*	1
00000101 *BBBBBBBB*	*INC <B>*	*(B... B):=(B... B+1)*	1
10100011	*INC DPTR*	*(DPTR):=(DPTR)+1*	2

Приклади виконання команди:

```
        ;(R0) = 7EH, ((R0)) = 0FFH, ((R0)+1)=40H
@@11: INC   @R0
      INC   R0
      INC   @R0
              ;(R0) ← 7EH+ 1 = 7FH
              ;((R0)) ← 00H, ((R)+1) ← 41H
              ;(DPH) = 12H, (DPL) = 0FEH
@@12: INC   DPTR
      INC   DPTR
      INC   DPTR
              ;(DPH) ← 13H, (DPL) ← 01H
```

JB

По команді *JB* (табл. 3.25) здійснюється розгалуження в програмі в такий спосіб.

Якщо біт комірки внутрішньої пам'яті даних, що прямо адресується, або регістра спеціальної функції дорівнює одиниці, то керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, подане третім байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

Пряма адреса аналізованого біту даних представляється другим байтом коду команди *I... I*.

Таблиця 3.25

Код	Інструкція	Алгоритм	Час
00100000 *I I I I I I I I* *SSSSSSSS*	*JB <S>*	*(PC):=(PC)+3* *IF (I... I):=1 THEN* *(PC):=(PC)+S... S* *END*	2

Приклад виконання команди*:*

```
loop:   MOV   A,R1
                    ;loop = 0125H
          ...
        ORG   0150H
          ;(P1) = 00000100B
          ;S... S = loop - ($+2) = 0125H - (0150H+2)=-2DH
          ; (тобто 0D3H)
@@1:  JB        P1. 2, loop
          ;(PC) ← (PC) + 2 = 0150H + 2 = 0152H
          ;(PC) ← (PC) + S... S = 0152H + (-2DH) = 0125H
```

JBC

По команді *JBC* (табл. 3.26) здійснюється розгалуження в програмі в такий спосіб.

Якщо біт комірки внутрішньої пам'яті даних, що прямо адресується, або регістра спеціальної функції дорівнює одиниці, то цей біт встановлюється в стан "нуль", а керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, поданий третім байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

Пряма адреса аналізованого біта даних представляється другим байтом коду команди *I... I*.

Таблиця 3.26

Код	Інструкція	Алгоритм	Час
00100000 *I I I I I I I I*	*JBC<BIT>,<S>*	*(PC):=(PC)+3* *IF (I... I):=1 THEN*	2

SSSSSSSS		*(PC):=(PC)+S... S* *END*	

Приклад виконання команди*:*

```
loop:   MOV   A,R1
        ;loop = 0125H
           ...
           ORG   0150H
        ;(P1) = 00000100B
        ;S... S= loop-($+2)=0125H-(0150H+2)=-2DH
        ;(тобто 0D3H)
@@1:  JBC   P1. 2, loop
        ;(P1) ← 00000000B
        ;(PC) ← (PC)+ 2 = 0150H + 2 = 0152H
        ;(PC) ← (PC)+ S... S = 0152H+ (-2DH) = 0125H
```

JC

По команді *JC* (табл. 3.27) здійснюється розгалуження в програмі в такий спосіб.

Якщо біт ознаки переносу (*C*) дорівнює одиниці, то керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, подане другим байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

Таблиця 3.27

Код	Інструкція	Алгоритм	Час
01000000 *SSSSSSSS*	*JB <S>*	*(PC):=(PC)+2* *IF (C):=1 THEN* *(PC):=(PC)+S... S* *END*	2

Приклад виконання команди:

```
loop:   MOV   A,R1
                ;loop = 0125H
           ...
           ORG   0150H
                ;(C)= 1
                ;S... S=loop-($+2)=0125H-(0150H+2)=-2DH
@@1:  JC        loop    ;(тобто 0D3H)
                ;(PC) ← (PC) + 2 = 0150H + 2 = 0152H
                ;(PC) ← (PC)+S... S = 0152H+ (-2DH) = 0125H
```

JMP

По команді *JMP* (табл. 3.28) здійснюється безумовний прямий (подібно командам *SJMP*, *AJMP* і *LJMP*) або непрямий перехід у точку призначення.

У випадку непрямого переходу вміст лічильника команд установлюється рівним сумі вмісту акумулятора і вмісту регістра покажчика даних *DPTR*. Сума обчислюється по модулі 2**16. Вміст акумулятора інтепретується, як ціле число без знака в межах від нуля до 255.

Таблиця 3.28

Код	Інструкція	Алгоритм	Час
10000000 SSSSSSS	*JMP <выр> ;вів(вир) ←255*	*см. інструкцію SJMP*	2
AAA00001 ALALALAL	*JMP <вир> ;вів(вир) ←256 ;вів(вир) ←2047*	*см. інструкцію AJMP*	2
00000010 AHAHAHAH ALALALAL	*JMP <вир> ;вів(вир)←2048*	*см. інструкцію LJMP*	2
01110011	*JMP @A+DPTR*	*(PC):=(A)+(PTR)*	2

Приклад виконання команди:

```
    MOV   DPTR,#JMP_TBL
          ;(A)= 04H, (DPTR) =    1000H
    JMP   @A+DPTR
          ;(PC) ← 1004H
JMP_TBL:
    JMP   LABEL0        ;$=1000H
    JMP   LABEL1        ;$=1002H
    JMP   LABEL2        ;$=1004H
  ;перехід після виконання команди JMP @A+DPTR
    JMP   LABEL3        ;$=1006H
```

JNB

По команді *JNB* (табл. 3.29) здійснюється розгалуження в програмі в такий спосіб.

Якщо біт комірки внутрішньої пам'яті даних, що прямо адресується, або регістра спеціальної функції дорівнює нулю, то керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, подане третім байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

Пряма адреса аналізованого біта даних представляється другим байтом коду команди *I... I*.

Таблиця 3.29

Код	Інструкція	Алгоритм	Час
00110000 *I I I I I I I I* *SSSSSSSS*	*JNB <S>*	*(PC):=(PC)+3* *IF (I... I):=0 THEN* *(PC):=(PC)+S... S* *END*	2

Приклад виконання команди:

```
    ORG   0100H
          ;(ACC) = 00001000B
          ;S... S = loop - ($+2) = 0125H - (0100H+2)=23H
          ;(PC) = 0100H
@@1:  JNB   ACC. 3, loop
          ;(PC) ← (PC) + 2 = 0100H + 2 = 0102H
          ;(PC) ← (PC) + S... S = 0102H + 23H = 0125H
      ...
loop:  MOV   A,R1
          ;loop = 0125H
```

JNC

По команді *JNC* (табл. 3.30) здійснюється розгалуження в програмі в такий спосіб.

Якщо біт ознаки переносу (*C*) дорівнює нулю, то керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, подане другим байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

Таблиця 3.30

Код	Інструкція	Алгоритм	Час
01010000 *SSSSSSSS*	*JNC <S>*	*(PC):=(PC)+2* *IF (C):=0 THEN* *(PC):=(PC)+S... S* *END*	2

Приклад виконання команди:

```
    ORG   0100H
          ;(C)= 0
          ;S... S=loop - ($+2) = 0125H - (0100H+2)= 23H
          ;(PC) = 0100H
@@1:  JNC   loop
```

```
                ;(PC) ← (PC) + 2 = 0100H + 2 = 0102H
                ;(PC) ← (PC) + S... S = 0102H + 23H = 0125H
          ...
loop:   MOV   A,R1
                ;loop = 0125H
```

JNZ

По команді *JNZ* (табл. 3.31) здійснюється розгалуження в програмі в такий спосіб.

Якщо вміст акумулятора не дорівнює нулю, то керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, подане другим байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

Таблиця 3.31

Код	Інструкція	Алгоритм	Час
01110000 *SSSSSSSS*	*JNZ <S>*	*(PC):=(PC)+2* *IF (A):=1 THEN* *(PC):=(PC)+S... S* *END*	2

Приклад виконання команди:

```
          ORG   0100H
                ;(A)<> 0
                ;S... S=loop - ($+2) = 0125H - (0100H+2) = 23H
                ;(PC) = 0100H
@@1:    JNZ   loop
                ;(PC) ← (PC) + 2 = 0100H + 2 = 0102H
                ;(PC) ← (PC) + S... S = 0102H + 23H = 0125H
          ...
loop:   MOV   A,R1
                ;loop = 0125H
```

JZ

По команді *JZ* (табл. 3.32) здійснюється розгалуження в програмі в такий спосіб.

Якщо вміст акумулятора дорівнює нулю, то керування передається в точку призначення (до поточного вмісту лічильника команд додається позиційно-незалежне зміщення, подане другим байтом коду команди *S... S*). У противному випадку керування передається наступній команді.

Таблиця 3.32

Код	Інструкція	Алгоритм	Час
01100000 *SSSSSSSS*	*JZ <S>*	*(PC):=(PC)+2* *IF (A):=0 THEN* *(PC):=(PC)+S... S* *END*	2

Приклад виконання команди:

```
loop:   MOV   A,R1
                ;loop = 0125H
          ...
          ORG   0150H
                ;(A)= 0
                ;S... S=loop-($+2)=0125H-(0150H+2)=-2DH
@@1:  JZ        loop    ;(тобто 0D3H)
                ;(PC) ← (PC) + 2 = 0150H + 2 = 0152H
                ;(PC) ← (PC)+S... S = 0152H + (-2DH) = 0125H
```

LCALL

По команді *LCALL* (табл. 3.33) здійснюється виклик підпрограми, розташованої в будь-якій точці пам'яті і програм, у такий спосіб:

- уміст лічильника команд *PC* (тобто повна адреса наступної команди) записується у верхівку стека (молодший байт адреси записується першим);
- уміст покажчика стека *SP* збільшується на два;
- у лічильник команд записується повна (16-бітний) адреса підпрограми, подана другим (старшим) *AH... AH* і третім (молодшим) *AL... AL* байтами коду команди.

Таблиця 3.33

Код	Інструкція	Алгоритм	Час
00010010 *AHAHAHAH* *ALALALAL*	*LCALL <A16>*	*(SP):=(SP)+1* *((SP)):=(PC[7-0])* *(SP):=(SP)+1* *((SP)):=(PC[15-8])* *(PC):=AH... AH:AL... AL*	2

Приклад виконання команди:

```
                ;(PC)=0035H,(SP)=26H, ((SP)+1)=00H,((SP)+2)=00H
                ;proc= 8000H
@@1: LCALL proc
;(PC) ←8000H,(SP) ←28H,((SP)):((SP)-1) ←0035H+2
          ...
```

```
        ORG   8000H
proc:  PUSH  ACC
        ...
        RET
```

LJMP

По команді *LJMP* (табл. 3.34) здійснюється безумовний перехід у точку призначення, розташовану в будь-якому місці пам'яті програм, у такий спосіб.

У лічильник команд записується повна (16-бітний) адреса точки призначення, подана другим (старшим) і третім (молодшим) байтами коду команди.

Таблиця 3.34

Код	Інструкція	Алгоритм	Час
00010010 AHAHAHAH ALALALAL	*LJMP <A16>*	*(PC):=AH... AH:AL... AL*	2

Приклад виконання команди:

```
loop:  MOV   A,R1
                ;loop  = 0080FH
        ...
                ;(PC)  = 0EADCH
@@1:  LJMP  loop
                ;(PC) ← 0080FH
```

MOV

По команді *MOV* (табл. 3.35) операнд-джерело пересилається на місце операнда-приймача.

Якщо в якості операнда-приймача використовується вміст акумулятора, то тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.35

Код	Інструкція	Алгоритм	Час
11101RRR	*MOV A, <R>*	*(A):=(RRR)* *(P):=0\1*	1
1110011R	*MOV A,@<R>*	*(A):=((R))* *(P):=0\1*	1
01110100 DDDDDDDD	*MOV A,#<D>*	*(A):=D...D* *(P):=0\1*	1
11100101	*MOV A,<B>*	*(A):=(B...B)*	1

BBBBBBBB		*(P):=0\1*	
11111RRR	*MOV <R>,A*	*(RRR):=(A)*	1
01111RRR DDDDDDDD	*MOV <R>,#<D>*	*(RRR):=D...D*	1
10101RRR BBBBBBB	*MOV <R>,<B>*	*((RRR)):=(B...B)*	1
1111011R	*MOV @<R>,A*	*((R)):=(A)*	1
0111011R DDDDDDD	*MOV @<R>,#<D>*	*((R)):=D... D*	1
1010011R BBBBBBB	*MOV @<R>,<B>*	*((R)):=(B... B)*	1
11110101 BBBBBBBB	*MOV <B>,A*	*(B... B):=(A)*	2
10001REG BBBBBBBB	*MOV <B>,<REG>*	*(B... B):=(REG)*	1
1000011R BBBBBBBB	*MOV <B>,@<R>*	*(B... B):=((R))*	1
01110101 BBBBBBBB DDDDDDDD	*MOV<B>,#<DATA>*	*(B... B):=DATA*	2
10000101 BSBSBSBS BDBDBDBD	*MOV <BD>,<BS>*	*(BD... BD):= (BS... BS)*	2
10010000 DHDHDHDH DLDLDLDL	*MOV DPTR,#<D16>*	*(DPTR):= DH... DH:DL... DL*	2
10100010 I I I I I I I	*MOV C,<BIT>*	*(C):=(I... I)*	1
10010010 I I I I I I I I	*MOV <BIT>,C*	*(I... I):=(C)*	2

Приклади виконання команди:

```
@@11: MOV   R0,#30H
            ;(R0) ← 30H
            ;((R0)) = 40H
      MOV   A,@R0
            ;(A) ← 40H
      MOV   R1,A
            ;(R1) ← 40H
            ;((R1)) = 10H
      MOV   B,@R1
```

```
                ;(B) ← 10H
@@12: MOV   DPTR,#1234H
                ;(DPH) ← 12H,  (DPL) ← 34H
                ;(C)= 1, (P3) =  00000000B
@@13: MOV   P3. 2, C
                ;(P3) ← 00000100B
```

MOVC

По команді *MOVC* (табл. 3.36) операнд-джерело (байт коду команди або 8-бітної константи) із пам'яті програм пересилається в акумулятор.

Є два засоби формування адреси операнда-джерела:

1. як сума вмісту акумулятора і регістра покажчика даних *DPTR*;

2. як сума вмісту акумулятора і регістра лічильника команд *PC*.

Після вибірки поточної команди лічильник команд збільшується на число, рівне кількості байтів, що займаються цією командою (для команди *MOVC* - на одиницю).

Старший байт адреси передається через порт #2, молодший - через порт #0.

Тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.36

Код	Інструкція	Алгоритм	Час
10010011	*MOVC A,@A+DPTR*	*(A):=((A)+(DPTR))* *(P):=0\1*	2
10000011	*MOVC A,@A+PC*	*(A):=((A)+(PC))* *(P):=0\1*	2

Приклади виконання команди:

```
                ;(A)= 1
субр:   INC   A
                ;(A) ← 2
        MOVC  A,@A+PC
                ;(A) ← (2 +$+1) = 77H
        RET
        DB      66H
        DB      77H
        DB      88H
        DB      99H
;$- адреса команди MOVC
```

MOVX

Є два типи команди *MOVX* (табл. 3.37).

По команді першого типу операнд-джерело із зовнішньої пам'яті даних пересилається в акумулятор.

По команді другого типу вміст акумулятора пересилається в зовнішню пам'ять даних.

У якості адреси комірки пам'яті даних може використовуватися:

1. Вміст робочого регістра з кодом *R* (тобто R0 і R1*)*. У цьому випадку адреса і дані мультиплексно передаються через порт #0. Розмір області даних, що адресується – 256 байт. З метою збільшення розміру області даних, що адресується, до 64 Кбайт, рекомендується використовувати порт #2 для попередньої передачі старшого байта адреси;

2. Вміст регістра покажчика даних *DPTR*. Молодший байт адреси мультиплексно передається через порт #0, старший байт - через порт #2. У фіксаторі порту #2 (тобто P2) відновляється його попередньо збережене значення. У буфер порту #2 записуються дані, що пересилаються. Розмір адресуємої області даних - 64 кбайт.

Якщо в якості операнда-приймача використовується вміст акумулятора, то тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.37

Код	Інструкція	Алгоритм	Час
1110001R	*MOVX A,@ <R>*	*(A):=((R))* *(P):=0\1*	2
11100000	*MOVX A,@DPTR*	*(A):=((DPTR))* *(P):=0\1*	2
1111001R	*MOVX @<R>,*	*((R)):=(A)*	2
11110000	*MOVX @DPTR,A*	*((DPTR)):=(A)*	2

Приклади виконання команди:

```
                ;(R0) = 12H, (R1) = 34H, ((R1)) = 56H
@@1:  MOVX  A,@R1
                ;(A) ← 56H
      MOVX  @R0,A
                ;((R0)) ← 56H
```

MUL

По команді MUL (табл. 3.38):

- вміст акумулятора умножається на вміст регістра *B*;

- результат записується в реєстрову пару *AB* (молодший байт - у регістр *A*, старший - у регістр *B*).

Тригер ознаки переносу *(C)* встановлюється в стан "0"; тригер ознаки переповнювання (*OV*) - у стан "0", якщо старший байт результату дорівнює нулю, або в стан "1", у противному випадку; тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Операнди команди розглядаються, як порядков числа.

Таблиця 3.38

Код	Інструкція	Алгоритм	Час
10100100	*MUL AB*	*(A):(B):=(A)*(B)* *(C):=0* *IF (AB)>255 THEN* *(OV):=1* *ELSE (OV):=0* *END* *(P):=0\1*	4

Приклади виконання команди:

```
                ;(A)= 50H, (B) = 0A0H
@@1:  MUL   AB
                ;(B):(A) ← 3200H
                ;(C) ← 0, (OV) ← 0, (P) ←1
```

NOP

По команді *NOP* (табл. 3.39) виконується холоста операція МК51 (уміст лічильника команд збільшується на одиницю).

Таблиця 3.39

Код	Інструкція	Алгоритм	Час
00000000	*NOP*	*(PC):=(PC)+1*	1

Приклад виконання команди:

```
    CLR   P2. 7
                ;(PC) = 0100H
    NOP
    NOP
    NOP
    NOP
                ;(PC) ← 0104H
    SETB  P2. 7
```

ORL

По команді *ORL* (табл. 3.40):

- операнд-джерело порозрядно логічно складається з операндом-приймачем;

- результат записується на місце операнда-приймача.

Якщо в якості операнда-приймача використовується вміст акумулятора, то тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.40

Код	Інструкція	Алгоритм	Час
01001RRR	*ORL A,<R>*	*(A):=(A) OR (RRR)* *(P):=0\1*	1
0100011R	*ORL A,@<R>*	*(A):=(A) OR ((R))* *(P):=0\1*	1
01000100 *DDDDDDDD*	*ORL A,#<D>*	*(A):=(A) OR D... D* *(P):=0\1*	1
01000101 *BBBBBBBB*	*ORL A,<B>*	*(A):=(A) OR (B... B)* *(P):=0\1*	1
01000010 *BBBBBBBB*	*ORL <B>,A*	*(B... B):=(B... B) OR (A)*	2
01000011 *BBBBBBB* *DDDDDDDD*	*ORL* *<B>,#<D>*	*(B... B):=(B... B)OR D...* *D*	2
01110010 *I I I I I I I I*	*ORL C,<BIT>*	*(C):=(C) OR (I... I)*	2
01010000 *I I I I I I I I*	*ORL C,/<BIT>*	*(C):=(C) OR (NOT(I...* *I))*	2

Приклади виконання команди:

```
                ;(P1)  = 01010101H
@@11: ORL   P1, # 00001111H
                ;(P1) ← 01011111H
              ; установити біт переносу в одиницю, якщо
              ; виконується наступна умова: (P1.  0)=1
              ;або (ACC.  7)=1 або (OV)=0
@@12: MOV   C,P1. 0
        ORL   C,ACC. 7
        ORL   C,/OV
```

POP

По команді *POP* (табл. 3.41):

- уміст стека за адресою, що знаходиться в регістрі покажчика стека *SP*, пересилається в комірку внутрішньої пам'яті даних або в регістр спеціальної функції, прямо адресуємого другим байтом коду команди *B... B*;

- уміст регістра покажчика стека *SP* зменшується на одиницю.

Таблиця 3.41

Код	Інструкція	Алгоритм	Час
11010000 BBBBBBBB	*POP <B>*	*(B... B):=((SP)) (SP):= (SP)-1*	2

Приклад виконання команди:

```
;(SP)=99H,((SP))=12H,((SP)-1)=34H,((SP)-2)=04H
@@1:  POP   DPL
      POP   DPH
     ;(DPTR) ← 1234H, (SP) ← 97H
      POP   SP
     ;(SP) ← 04H
```

PUSH

По команді *PUSH* (табл. 3.42):

– вміст регістра покажчика стека *SP* збільшується на одиницю;

– вміст комірки внутрішньої пам'яті даних або вміст регістра спеціальної функції, прямо адресуємої другим байтом коду команди *B... B*, записується в стек за адресою, що знаходиться в регістрі покажчика стека *SP*.

Таблиця 3.42

Код	Інструкція	Алгоритм	Час
11000000 BBBBBBBB	*PUSH <B>*	*(SP):=(SP)+1 ((SP)):=(B... B)*	2

Приклад виконання команди:

```
   ;(SP) = 09H, (DPTR) = 1234H
@@1:  PUSH  DPL
      PUSH  DPH
     ;(SP) ← 0BH,  ((SP)) ← 12H,  ((SP)-1) ← 34H
```

RET

По команді *RET* (табл. 3.43) здійснюється повернення з підпрограми, що була викликана командою *CALL*, *ACALL* або *LCALL*, у такий спосіб:

– вміст верхівки стека (тобто повна адреса повернення з підпрограми) пересилається в лічильник команд PC;

– вміст покажчика стека *SP* зменшується на два.

Таблиця 3.43

Код	Інструкція	Алгоритм	Час
00100010	*RET*	*(PC[15-8]):=((SP))*	2

		(SP):=(SP)-1 *(PC[7-0]):=((SP))* *(SP):=(SP)-1*	

Приклад виконання команди:

```
    LCALL proc
    ...
    ORG   8000H
proc:  PUSH  ACC
    ...
;(PC) = 8051H,(SP) =28H,((SP)):((SP)-1)=0037H
@@1:  RET
;(PC) ← 0037H, (SP) ← 26H
```

RETI

По команді *RETI* (табл. 3.44) здійснюється повернення з підпрограми обслуговування переривання в такий спосіб:

– уміст верхівки стека (тобто повна адреса повернення в точку переривання) пересилається в лічильник команд PC. Точкою переривання є команда, що випливає за командою, під час виконання якої надійшов запит на переривання;

– уміст покажчика стека *SP* зменшується на два;

– дозволяє переривання рівного або меншого рівня.

Якщо під час виконання команди *RETI* прийшов запит на переривання рівного або меншого рівня, то його опрацювання починається тільки після виконання наступної команди.

Таблиця 3.44

Код	Інструкція	Алгоритм	Час
00110010	*RET1*	*(PC[15-8]):=((SP))* *(SP):=(SP)-1* *(PC[7-0]):=((SP))* *(SP):=(SP)-1*	2

Приклад виконання команди:

```
    ORG   0003H
    JMP   INT_0
    ...
    ORG   8000H
INT_0:  PUSH  ACC
    ...
;(PC) = 8051H,(SP) =28H,((SP)):((SP)-1)=0037H
```

@@1: RETI

;(PC) ← 0037H, (SP) ← 26H

RL

По команді *RL* (табл. 3.45) вміст акумулятора циклічно зсовується вліво на один двійковий розряд.

Тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.45

Код	Інструкція	Алгоритм	Час
00100011	*RL A*	*FOR N:=6 TO 0 DO* *(A[N+1]):=(A[X])* *END* *(A[0]):=(A[7])* *(P):=0\1*	1

Приклади виконання команди:

;(A) = 11110000B

@@1: RL A

;(A) ← 11100001B

RLC

По команді *RLC* (табл. 3.46) вміст акумулятора і біт ознаки переносу циклічно зсовуються уліво на один двійковий розряд.

Тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.46

Код	Інструкція	Алгоритм	Час
00110011	*RLC A*	*(TMP):=(C)* *(C):=(A[7])* *FOR N:=6 TO 0 DO* *(A[N+1]):=(A[N])* *END* *(A[0]):=(TMP)* *(P):=0\1*	1

Приклади виконання команди:

;(C) = 0, (A) = 11111111B

@@1: RLC A

;(C) ← 1, (A) ← 11111110B

RR

По команді *RR* (табл. 3.47) вміст акумулятора циклічно зсовується вправо на один двійковий розряд.

Тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.47

Код	Інструкція	Алгоритм	Час
000000011	*RR A*	*FOR N:=6 TO 0 DO* *(A[N]):=(A[N+1])* *END* *(A[7]):=(A[0])* *(P):=0\1*	1

Приклади виконання команди:

```
                ;(A) = 11110000B
@@1:  RR        A
                ;(A) ← 01111000B
```

RRC

По команді *RRC* (табл. 3.48) вміст акумулятора і біт ознаки переносу циклічно зсовуються управо на один двійковий розряд.

Тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.48

Код	Інструкція	Алгоритм	Час
00010011	*RRC A*	*(TMP):=(C)* *(C):=(A[0])* *FOR N:=0 TO 6 DO* *(A[N]):=(A[N+1])* *END* *(A[7]):=(TMP)* *(P):=0\1*	1

Приклади виконання команди:

```
                ;(A) = 11111111B, (C)  = 0,
@@1:  RRC   A
                ;(A)← 01111111B, (C) ← 1,
```

SETB

По команді *SETB* (табл. 3.49) операнд-приймач встановлюється в стан "1".

Якщо в якості операнда-приймача використовується біт регістра спеціальної функції *PSW*. 0, *PSW*. 2, *PSW*. 6 або *PSW*. 7, то відповід-

ний тригер ознаки результату (*P), (OV), (AC*) або (*C*) встановлюється в стан "1".

Таблиця 3.49

Код	Інструкція	Алгоритм	Час
11010011	*SETB C*	*(C):=1*	1
11010010 *I I I I I I I I*	*SETB<BIT>*	*(I... I):=1*	1

Приклади виконання команди:

```
                ;(P1)  = 00000000B
@@1:  SETB  P1. 6
                ;(P1) ← 01000000B
```

SJMP

По команді *SJMP* (табл. 3.50) здійснюється безумовний прямий перехід у точку призначення, розташовану в межах від 128 байт, що передують команді, до 127 байт, що слідують за командою, шляхом додавання поточного вмісту лічильника команд *PC* і позиційно-незалежного зміщення, поданого другим байтом коду команди *S... S*.

Байт позиційно незалежного зміщення інтепретується, як ціле число зі знаком у межах ±127.

Таблиця 3.50

Код	Інструкція	Алгоритм	Час
10000000 *SSSSSSSS*	*SJMP <S>*	*(PC):=(PC)+2* *(PC):=(PC)+S...S*	2

Приклад виконання команди:

```
loop:   MOV   A,R1
          ;loop = 0125H
            ...
            ORG   0150H
          ;S... S=loop-($+2)=0125H-(0150H+2)=-2DH
@@1:  SJMP  loop;     (тобто 0D3H)
         ;(PC) ← (PC) + 2 = 0150H + 2 = 0152H
        ;(PC) ← (PC)+S... S= 0152H + (-2DH) = 0125H
```

SUBB

По команді *SUBB* (табл. 3.51):

– операнд-джерело і біт ознаки переносу відраховуються з вмісту акумулятора;

– результат записується в акумулятор.

Тригери ознак результату *(C), (AC), (OV)* і (*P*) установлюються відповідно до табл. 3.10.

Таблиця 3.51

Код	Інструкція	Алгоритм	Час
10011RRR	*SUBB A,<R>*	*(A):=(A)-(C)-(RRR)* *(C),(AC),(OV),(P):=0\1*	1
1001011R	*SUBB A,@<R>*	*(A):=(A)-(C)-((R))* *(C),(AC),(OV),(P):=0\1*	1
10010100 *DDDDDDDD*	*SUBB A,#<D>*	*(A):=(A)-(C)-D... D* *(C),(AC),(OV),(P):=0\1*	1
10010101 *BBBBBBBB*	*SUBB A,<B>*	*(A):=(A)-(C)-(B... B)* *(C),(AC),(OV),(P):=0\1*	1

Приклад виконання команди:

;(A)= 0C9H, R2 = 54H, (C) = 1

@@1: SUBB A,R2

;(A) ← 0C9H- 1 - 54H = 74H

;(C) ← 0, (AC) ← 0, (OV) ← 1, (P) ← 1

SWAP

По команді *SWAP* (табл. 3.52) розряди 0-3 і 4-7 акумулятора обмінюються вмістом, тобто вміст акумулятора циклічно зсовується на чотири двійкових розряди.

Тригер ознаки переносу (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.52

Код	Інструкція	Алгоритм	Час
11000100	*SWAP A*	*(TMP):=(A[3-0])* *(A[3-0]):=(A[7-4])* *(A[7-4]):=(TMP)* *(P):=0\1*	1

Приклад виконання команди:

;(A) = 00001111B

@@1: SWAP A

;(A) ← 11110000B

XCH

По команді *XCH* (табл. 3.53) в акумулятор пересилається операнд-джерело і одночасно на місце операнда-джерела пересилається початковий вміст акумулятора.

Тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.53

Код	Інструкція	Алгоритм	Час
11001RRR	*XCH A,<R>*	*(TMP):=(A)* *(A):=(RRR)* *(RRR):=(TMP)* *(P):=0\1*	1
1100011R	*XCH A,@<R>*	*(TMP):=(A)* *(A):=((R))* *((R)):=(TMP)* *(P):=0\1*	1
11000101 *BBBBBBBB*	*XCH A,<B>*	*(TMP):=(A)* *(A):=(B... B)* *(B... B):=(TMP)* *(P):=0\1*	1

Приклад виконання команди:

```
                ;(R0) = 20H,(A) = 3FH, ((R0)) = 75H
@@1:  XCH   A,@R0
                ;(A) ← 75H, ((R0)) ← 3FH
                ;(P) ← 0
```

XCHD

По команді *XCHD* (табл. 3.54) розряди 0-3 комірки внутрішньої пам'яті даних, адреса якої знаходиться в робочому регістрі R0 або R1, і акумулятора обмінюються вмістом (уміст 4-7 розрядів комірки пам'яті й акумулятора не змінюється).

Тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.54

Код	Інструкція	Алгоритм	Час
1101011R	*XCHD A,@<R>*	*(TMP):=(A[3-0])* *(A[3-0]):=((R[3-0]))* *((R[3-0])):=(TMP)* *(P):=0\1*	1

Приклад виконання команди:

```
                ;(A)= 55H, (R0) = 20H, ((R0)) = 0FFH
@@1:  XCHD  A,@R0
                ;(A) ← 5FH, ((R0)) ← 0F5H
```

XRL

По команді *XRL* (табл. 3.55):

– операнд-джерело складається по модулі 2 з операндом-приймачем;

– результат записується на місце операнда-приймача.

Якщо в якості операнда-приймача використовується вміст акумулятора, то тригер ознаки парності (*P*) установлюється відповідно до табл. 3.10.

Таблиця 3.55

Код	Інструкція	Алгоритм	Час
01101RRR	*XRL A,<R>*	*(A):=(A) AND NOT (RRR) OR NOT (A) AND (RRR) (P):=0\1*	1
0110011R	*XRL A,@<R>*	*(A):=(A) AND NOT ((R)) OR NOT (A) AND ((R)) (P):=0\1*	1
01100100 DDDDDDDD	*XRL A,#<D>*	*(A):=(A) AND NOT(D... D) OR NOT (A) AND(D... D) (P):=0\1*	1
01100101 BBBBBBBB	*XRL A,<B>*	*(A):=(A) AND NOT(B... B) OR NOT (A) AND(B... B) (P):=0\1*	1
01100010 BBBBBBBB	*XRL <B>,A*	*(B... B):=(B... B) AND NOT (A) OR NOT (B... B) AND (A)*	2
01100011 BBBBBBBB DDDDDDDD	*XRL <B>,#<D>*	*(B... B):=(B... B) AND NOT (D... D) OR NOT (B... B) AND (D... D)*	2

Приклади виконання команди:

```
                ;(P1)  = 01010101H
@@1:  XRL   P1, # 00001111H
                ;(P1) ← 01011010H
```

СПИСОК ЛІТЕРАТУРИ

1. **МИКРОПРОЦЕССОРЫ** и микропроцессорные комплекты интегральных микросхем: Справочник: В 2т./Н.Н Аверьянов, А.И. Березенко, Ю.И. Борщенко; Под ред. В.А. Шахнова. - М.: Радио и связь, 1988. - Т.1 - 386 с., Т.2 - 386 с.

2. **МИКРОПРОЦЕССОРНЫЙ** комплект К1810: Структура, программирование, применение: Справочная книга/Ю.М. Казаринов, В.Н. Номоконов, Г.С. Подклетнов, Ф.В. Филиппов; Под ред. Ю.М. Казаринова. - М.: Высщ. шк., 1990. - 269 с.

3. **ПРОЕКТИРОВАНИЕ** цифровых устройств на однокристальных микроконтроллерах/В.В. Сташин, А.В. Урусов, О.Ф. Мологонцева. М.: Энергоатомиздат, - 224 с.

4. **МИКРОПРОЦЕССОРЫ** и микроЭВМ в системах автоматического управления: Справочник/С.Т Хвощ, Н.Н Варлинский, Е.А Попов; Под ред. С.Т. Хвоща. - Л.: Машиностроение, 1987 - 640 с.

ЗМІСТ

Навчально-методичне видання

ШВЕЦЬ Валеріян Анатолійович,
МЕЛЕШКО Тетяна Вікторівна
ШЕСТАКОВА Віолета Володимірівна

“МІКРОПРОЦЕСОРИ В СИСТЕМАХ ТЕХНІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ”
(Архітектура і система команд)

Частина 1

Навчльний посібник