

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
ПРИДНІПРОВСЬКА ДЕРЖАВНА АКАДЕМІЯ  
БУДІВНИЦТВА ТА АРХІТЕКТУРИ

Кафедра автоматики та електротехніки

Методичні вказівки  
до виконання лабораторних робіт

# *Мікропроцесорна техніка*

для студентів спеціальності 6.050202  
“Автоматизоване управління технологічними процесами ”

Дніпропетровськ - 2009

# Заняття 1

## Принципи побудови пристроїв пам'яті

### Тема заняття

- Класифікація і структурні особливості пристроїв пам'яті ЕОМ.
- Характеристики і принципи функціонування.

### Цілі заняття

- Понять і засвоїти відмінності між типами пристроїв (ЗУ) ЕОМ, що запам'ятовують, їх класифікацією.
- Вивчити основні визначення і призначення параметрів оцінки ЗУ.
- Зрозуміти і засвоїти відмінності між способами адресації до пам'яті.
- Вивчити методи побудови ЗУ з адресним, асоціативним і стековим способом організації.
- Засвоїти послідовність дій при зверненні до ЗУ з різними способами адресації.

### Загальні відомості і класифікація пристроїв пам'яті

Пам'яттю ЕОМ називається сукупність пристроїв, що слугують для запам'ятовування, зберігання і видачі інформації. Окремі пристрої, що входять в цю сукупність, називають пристроями, що запам'ятовують, або пам'яттями того або іншого типу.

Обидва ці терміну в даний час стали майже синонімами. Проте, термін «пристрій» (ЗУ), що запам'ятовує, зазвичай вживають, коли мова йде про принципи побудови деякого пристрою пам'яті (наприклад, напівпровідникові ЗУ, ЗУ на магнітних дисках і так далі), а термін «пам'ять» - коли хочуть підкреслити виконувану пристроєм пам'яті логічну функцію або місце розташування в складі устаткування ЕОМ (наприклад, оперативна пам'ять, зовнішня пам'ять і так далі).

Продуктивність і обчислювальні можливості ЕОМ в значній мірі визначаються складом і характеристиками її ЗУ. У складі ЕОМ використовується одночасно декілька типів ЗУ (декілька типів пам'ятей), що відрізняються принципом дії, характеристиками і призначенням.

Основними операціями в пам'яті в загальному випадку є занесення інформації в пам'ять - запис і вибірка інформації з пам'яті - зчитування. Обидві ці операції називаються зверненням до пам'яті, або, докладніше, зверненням при зчитуванні і зверненням при записі.

При зверненні до пам'яті проводиться зчитування або запис деякої одиниці даних - різною для пристроїв різного типу.

Такою одиницею може бути, наприклад, байт, машинне слово або блок даних.

Найважливішими характеристиками окремих пристроїв пам'яті (пристроїв, що запам'ятовують) є ємність пам'яті, питома ємність, швидкодія.

**Ємність пам'яті** визначається максимальною кількістю даних, які можуть в ній зберігатися. Ємність вимірюється в двійкових одиницях (бітах),

машинних словах, але переважно в байтах (1 байт = 8 біт), при цьому часто ємність пам'яті виражають через число Кбіт (кілобіт), Кслів (кілослів) або Кбайт (кілобайт), при цьому 1024 Кбайт позначають як 1 Мбайт (мегабайт).

**Питома ємність** є відношення ємності ЗУ до його фізичного об'єму.

**Швидкодія пам'яті** визначається тривалістю операції звернення, тобто часом, що витрачається на пошук потрібної одиниці інформації в пам'яті і на її зчитування (час звернення при зчитуванні), або часом на пошук місця в пам'яті, що призначається для зберігання даної одиниці інформації, і на її запис в пам'ять (час звернення при записі).

Тривалість звернення до пам'яті (час циклу пам'яті) при зчитуванні:

$$t_{обр}^{счит} = t_{дост}^{счит} + t_{счит}$$

де  $t_{дост}^{счит}$  - час доступу, що визначається проміжком часу між моментом почала операції звернення при зчитуванні до моменту, коли стає можливим доступ до даної одиниці інформації;  $t_{счит}$  -продолжительность самого фізичного процесу зчитування, тобто процесу виявлення і фіксації станів відповідних елементів, що запам'ятовують, або ділянок поверхні носія інформації.

У деяких пристроях пам'яті зчитування інформації супроводжується її руйнуванням (стиранням). У такому разі цикл звернення повинен містити операцію відновлення (регенерації) ліченої інформації на колишньому місці в пам'яті.

Тривалість звернення (час циклу) при записі

$$t_{обр}^{зан} = t_{дост}^{зан} + t_{зан}$$

$t_{дост}^{зан}$  - час доступу при записі, тобто час від моменту початку звернення при записі до моменту, коли стає можливим доступ до елементів (або ділянкам поверхні носія), що запам'ятовують, в які проводиться запис;  $t_{дост}^{зан}$  - час підготовки, що витрачається на приведення в початковий стан елементів, що запам'ятовують, або ділянок поверхні носія інформації для запису певної одиниці інформації (наприклад, байта або слова);  $t_{зан}$  - час занесення інформації, тобто зміни стану елементів, що запам'ятовують (ділянок поверхні носія). Переважно:

$$t_{обр}^{счит} = t_{дост}^{зан} = t_{дост}$$

Як тривалість циклу звернення до пам'яті приймається величина:

Залежно від операцій, що реалізуються в пам'яті, звернення розрізняють:

$$t_{обр} = \max(t_{дост}^{счит}, t_{обр}^{зан})$$

а) пам'ять з довільним зверненням (можливі зчитування і запис даних в пам'ять);

б) пам'ять тільки для зчитування інформації («постійна» або «одностороння»).

Запис інформації в постійну пам'ять проводиться в процесі її виготовлення або настройки.

За способом організації доступу розрізняють пристрої пам'яті з безпосереднім (довільним), з прямим (циклічним) і послідовним доступами.

У пам'яті з безпосереднім (довільним) доступом час доступу, а тому і цикл звернення не залежать від місця розташування ділянки пам'яті, з якої проводиться зчитування, або в яку записується інформація. В більшості випадків безпосередній доступ реалізується за допомогою електронних (напівпровідникових) ЗУ. У подібних пам'ятях цикл звернення, зазвичай складає 10-50 нс. Число розрядів, що прочитуються або записуваних в пам'яті з безпосереднім доступом паралельно в часі за одну операцію звернення, називається шириною **вибірки**.

Ці типи пам'яті відповідають термінам RAM (random - access memory - пам'ять з довільним зверненням) і ROM (read-only memory - пам'ять тільки для зчитування).

У двох інших типах пам'яті використовуються повільніші електромеханічні процеси. У пристроях пам'яті з прямим доступом, до яких відносяться пристрої з магнітними дисками, завдяки безперервному обертанню носія інформації можливість звернення до деякої ділянки носія для зчитування або запису циклічно повторюється, В такій пам'яті час доступу складає декілька мілісекунд.

У пам'яті з послідовним доступом проводиться послідовний перегляд ділянок носія інформації, поки потрібну ділянку носія не займе деяке початкове положення. Характерним прикладом є ЗУ на магнітних стрічках. Час доступу може в несприятливих випадках розташування інформації досягти декількох хвилин.

Пристрої, що запам'ятовують, розрізняються також по виконуваних в ЕОМ функціях, залежних зокрема, від місця розташування ЗУ в структурі ЕОМ.

Вимоги до ємності і швидкодії пам'яті є суперечливими. Чим більше швидкодія, тим технічно важче досягається і дорожче обходиться збільшення ємності пам'яті. Вартість пам'яті складає значну частину загальної вартості ЕОМ. Тому пам'ять ЕОМ організовується у вигляді ієрархічної структури пристроїв, що запам'ятовують, володіють різною швидкістю і ємністю. У загальному випадку ЕОМ містить надоперативну пам'ять (СОП) або місцеву пам'ять, оперативну або основну пам'ять (ОП), пам'ять з прямим доступом на магнітних дисках, пам'ять з послідовним доступом на магнітних стрічках. Порядок перерахування пристроїв відповідає убуванню їх швидкодії і зростанню ємності. Кожен рівень ієрархії може містити декілька екземплярів (модулів) відповідних пристроїв для отримання потрібної ємності даного рівня пам'яті. Ієрархічна структура пам'яті дозволяє економічно ефективно поєднувати зберігання великих об'ємів інформації з швидким доступом до інформації в процесі обробки.

Оперативною або основною пам'яттю (ОП) називають пристрій, який служить для зберігання інформації (даних програм, проміжних і кінцевих результатів обробки), безпосередньо використовуваної в процесі виконання операцій в арифметико-логічному пристрої і пристрої управління процесора.

В процесі обробки інформації здійснюється тісна взаємодія процесора і ОП. З ОП в процесор поступають команди програми і операнди, над якими проводяться передбачені командою операції, а з процесора в ОП прямують для зберігання проміжні і кінцеві результати обробки.

Характеристики ОП безпосередньо впливають на основні показники ЕОМ і в першу чергу на швидкість її роботи. Оперативна пам'ять високопродуктивних ЕОМ має ємність декілька десятків мільйонів байт і цикл звернення близько 10 -50 нс (і менш). Пристрої, що запам'ятовують, ОП в даний час виготовляються на інтегральних мікросхемах з великим ступенем інтеграції (напівпровідникові ЗУ).

У ряді випадків швидкодія ОП виявляється недостатнім, і до складу машини доводиться включати СОП (буферну або кеш-пам'ять на декілька сотень тисяч машинних слів з циклом звернення, складовим декілька наносекунд. Такі СОП виконуються на швидкодіючих інтегральних мікросхемах. Швидкодія СОП повинна відповідати швидкості роботи арифметико-логічних пристроїв процесора, що управляють. Надоперативна (буферна) пам'ять використовується для проміжного зберігання зчитуваних процесором з ОП ділянок програми і груп даних, як робочі осередки програми, індексні реєстри, для зберігання службової інформації, використовуваної при управлінні обчислювальним процесом. Вона виконує роль ланки, що погоджує, між швидкодіючими логічними пристроями процесора і повільнішою ОП.

У якості ОП і СОП використовуються швидкодіючі ЗУ з довільним зверненням і безпосереднім доступом.

Поняття оперативної пам'яті вище визначалося, виходячи з виконуваних нею функцій. Оскільки завжди в якості ОП (як, втім, і СОП) використовуються ЗУ з довільним зверненням і безпосереднім доступом, часто останні незалежно від виконуваних функцій називають оперативною пам'яттю (оперативні ЗП).

Зазвичай ємність ОП виявляється недостатньою для зберігання всіх необхідних даних в ЕОМ. Тому ЕОМ містить в своєму складі декілька ЗУ з прямим доступом на дисках (ємність одного ЗУ на дисках до 100 Гбайт) і декілька ЗУ з послідовним доступом на магнітних стрічках (ємність одного ЗУ до 200 Гбайт).

Оперативна пам'ять разом з СОП і деякими іншими спеціалізованими пам'яттями процесора утворюють внутрішню пам'ять ЕОМ. Електромеханічні ЗУ утворюють зовнішню пам'ять ЕОМ, а самі вони, тому називаються зовнішніми пристроями, що запам'ятовують (ВЗУ).

Сучасні ЕОМ містять ряд спеціалізованих швидкодіючих пам'ятей: пам'яті каналів, ключів захисту пам'яті, окремих типів терміналів (дисплеїв і ін.), різні буферні пам'яті для проміжного зберігання інформації при обміні

нею між пристроями машини, що працюють, з різними швидкостями. Разом з цим використовуються також постійні пам'яті, в основному для зберігання мікропрограм, а в спеціалізованих ЕОМ - і для зберігання основних програм.

Пристрій будь-якого типу, що запам'ятовує, складається з масиву, що запам'ятовує, зберігає інформацію, і блоків, службовців для пошуку в масиві, записи і зчитування (а у ряді випадків і для регенерації) інформації.

### Адресна організації пам'яті

Пристрій, що запам'ятовує, з довільним зверненням, як правило, містить безліч однакових елементів, що запам'ятовують, створюючи масив, що запам'ятовує (ЗМ). Масив роздільний на окремі осередки; кожна з них призначена для зберігання двійкової коди, число розрядів в якому визначається шириною вибірки пам'яті (зокрема, це може бути одне, половина або декілька машинних слів). Спосіб організації пам'яті залежить від методів розміщення і пошуку інформації в масиві, що запам'ятовує. За цією ознакою розрізняють адресну, асоціативну і стекову пам'яті.

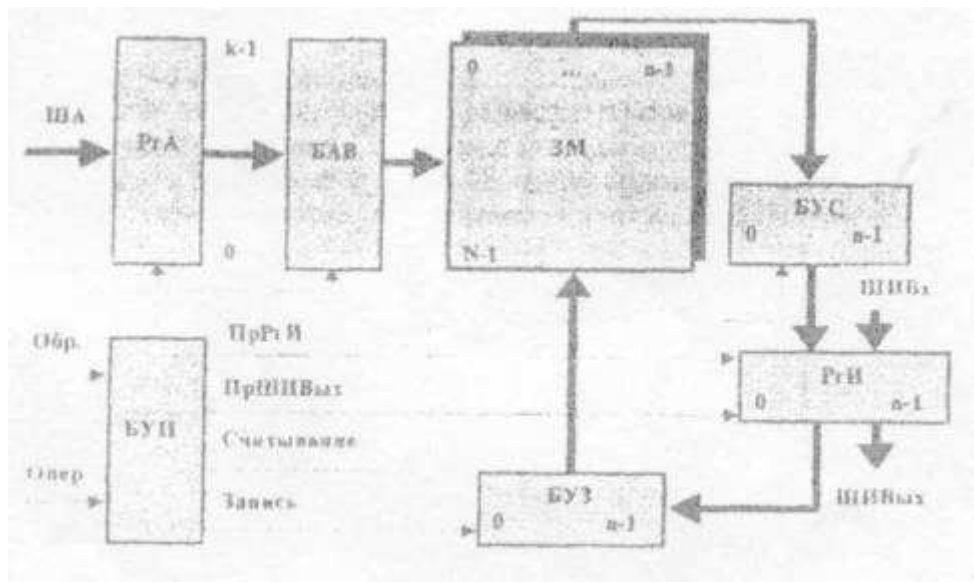
Адресна пам'ять. У пам'яті з адресною організацією розміщення і пошук інформації в ЗМ засновані на використанні адреси зберігання слова (числа, команди і т. п.). Адресою служить номер осередку ЗМ, в якому це слово розміщується. При записі (або зчитуванні) слова в ЗМ що ініціює цю операцію команда повинна указувати адресу (номер осередку), по якій проводиться запис (зчитування). Типова структура адресної пам'яті, показана на мал. 1.1.

Адресна пам'ять з довільною вибіркою містить масив з (N) n-розрядних осередків, що запам'ятовує, і його апаратне обрамлення, що включає:

- реєстр адреси **РГА**, що має до  $(k = \log_2 N)$  розрядів;
- інформаційний реєстр **РГИ**;
- блок адресної вибірки **Б АВ**;
- блок підсилювачів зчитування **БУС**;
- блок розрядних підсилювачів формувачів сигналів запису **БУЗ** і блок управління пам'яттю **БУП**.

За кодом адреси в РГА Б АВ формує у відповідному елементі пам'яті сигнали, що дозволяють провести в осередку зчитування або запис слова.

Цикл звернення до пам'яті ініціюється надходженням в БУП ззовні сигналу «**Звернення**» Загальна частина циклу звернення включає прийом в РГА з шини адреси (ША) адреси звернення і прийом в БУП і розшифровку сигналу, що управляє, «**Операція**», вказуючого вид запрошеної операції (зчитування або запис).



Мал. 1.1. Пам'ять з довільною вибіркою

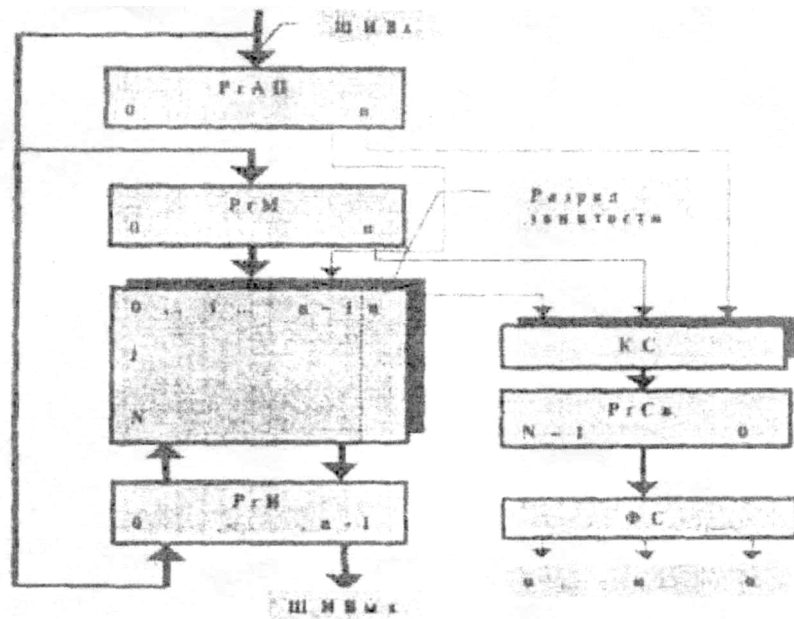
Далі при зчитуванні БАР дешифрує адресу, посилає сигнали зчитування в заданій адресою осередок ЗМ, при цьому код записаного в осередку слова прочитується підсилювачами зчитування БУС і передається в РГІ. Операція зчитування завершується видачею слова з РГІ на вихідну інформаційну шину Шивх.

При записі крім виконання вказаної вище загальної частини циклу звернення проводиться прийом записуваного слова з вхідної інформаційної шини Шивх в РГІ. Потім у вибраній осередку БАР записується слово з РГІ. Блок управління БУП генерує необхідні послідовності сигналів, що управляють, ініціюють роботу окремих вузлів пам'яті.

### Асоціативна організації пам'яті

У пам'яті цього типу пошук потрібної інформації проводиться не за адресою, а по її змісту (за асоціативною ознакою). При цьому пошук за асоціативною ознакою (або послідовно по окремих розрядах цієї ознаки) відбувається паралельно в часі для всіх осередків масиву, що запам'ятовує. У багатьох випадках асоціативний пошук дозволяє істотно спростити і прискорити обробку даних. Це досягається за рахунок того, що в пам'яті цього типу операція зчитування інформації суміщена з виконанням ряду логічних операцій. Типова структура асоціативної пам'яті представлена на мал. 1.2. Масив, що запам'ятовує, містить  $N$  ( $n + 1$ ) розрядних осередків. Для вказівки зайнятості осередку використовується службовий  $n$ -й розряд (0 - осередок вільний, 1 - в осередку записано слово).

По вхідній інформаційній шині Шивх в регістр асоціативної ознаки РГАП в розряди 0 -  $n - 1$  поступає  $n$ -розрядний асоціативний запит, а в регістр маски РГМ - код маски пошуку, при цьому  $n$ -й розряд РГМ встановлюється в 0. Асоціативний пошук проводиться лише для сукупності розрядів РГАП, яким відповідають 1 в РГМ (незамасковані розряди РГАП). Для слів, в яких цифри в розрядах співпали з незамаскованими розрядами РГАП, комбінаційна схема КС встановлює 1 у відповідні розряди регістра збігу Ргсв і 0 в решту розрядів.



Мал. 1.2 Асоціативна організації пам'яті

Комбінаційна схема формування результату асоціативного звернення ФС формує із слова, що утворилося в РГСв, сигнали  $(a_0, a_1, a_2)$ , відповідні випадкам відсутності слів в ЗМ, що задовольняють асоціативній ознаці, і наявності однієї (і більш) такого слова.

Формування вмісту РГСв і сигналів  $(a_0, a_1, a_2)$  по вмісту РГАП, РГМ і ЗМ називається операцією контролю асоціації. Ця операція є складовою частиною операцій зчитування і запису, хоча вона має і самостійне значення.

При зчитуванні спочатку проводиться контроль асоціації за асоціативною ознакою в РГАП. Потім при  $a_0 = 1$  зчитування відміняється через відсутність шуканої інформації, при  $a_2 = 1$  прочитується в РГИ знайдене слово, при  $a_2 = 1$  в РГИ прочитується слово з осередку, що має найменший номер серед осередків, відмічених 1 у РГСв. З РГИ лічене слово видається на Шивих.

За допомогою операції контролю асоціації можна, не прочитуючи слів з пам'яті, визначити по вмісту РГСв, скільки в пам'яті слів, що задовольняють асоціативній ознаці, наприклад реалізувати запити типу скільки студентів в групі мають відмінну оцінку по даній дисципліні. При використанні відповідних комбінаційних схем в асоціативній пам'яті можуть виконуватися достатньо складні логічні операції, такі, як пошук більшого (меншого) числа, пошук слів в певних межах, пошук максимального (мінімального) числа і ін.

Відзначимо, що для асоціативної пам'яті необхідні елементи, що запам'ятовують, допускають зчитування без руйнування записаної в них інформації. Це пов'язано з тим, що при асоціативному пошуку зчитування проводиться по всьому ЗМ для всіх незамаскованих розрядів і ніде зберігати тимчасово руйновану зчитуванням інформацію.

## Стекова пам'ять

Так само як і асоціативна, стекова пам'ять є безадресною. Стекову пам'ять можна розглядати як сукупність осередків, створюючи одновимірний масив, в якому сусідні осередки зв'язані один з одним розрядними ланцюгами передачі слів. Запис нового слова проводиться у верхній осередок (осередок 0), при цьому всі раніше записані слова (включаючи слово, що знаходилося в осередку 0), зрушуються вниз, в сусідні осередки з великими на 1 номерами. Зчитування можливе тільки з верхнього (нульовий) елемента пам'яті, при цьому, якщо проводиться зчитування з видаленням, решта всіх слів в пам'яті зрушується вгору, в сусідні осередки з великими номерами. У цій пам'яті порядок зчитування слів відповідає правилу: останнім поступив - першим обслуговується. У ряді пристроїв даного типу передбачається також операція простого зчитування слова з нульового осередку (без його видалення і зрушення слова в пам'яті). Іноді стекова пам'ять забезпечується лічильником стека Счст, що показує кількість занесених в пам'ять слів. Сигнал  $Счст = 0$  відповідає порожньому стеку,  $Счст = N - 1$  -заповненому стечу.

Зазвичай стекову пам'ять організують, використовуючи адресну пам'ять. В цьому випадку лічильник стека, як правило, відсутній, оскільки кількість слів в пам'яті можна виявити по покажчику стека. Широке застосування стекова пам'ять знаходить при обробці вкладених структур даних.

## Заняття 2

### Загальні принципи побудови процесорів

#### Цілі заняття

- Засвоїти терміни, що відносяться до поняття архітектура процесора.
- Вивчити призначення основних функціональних блоків процесора.
- Отримати уявлення про адресні структури основних типів пам'яті ЕОМ.
- Зрозуміти і засвоїти відмінності між структурними особливостями команд процесора.
- Вивчити способи кодування командної інформації.

#### Призначення і структура процесора

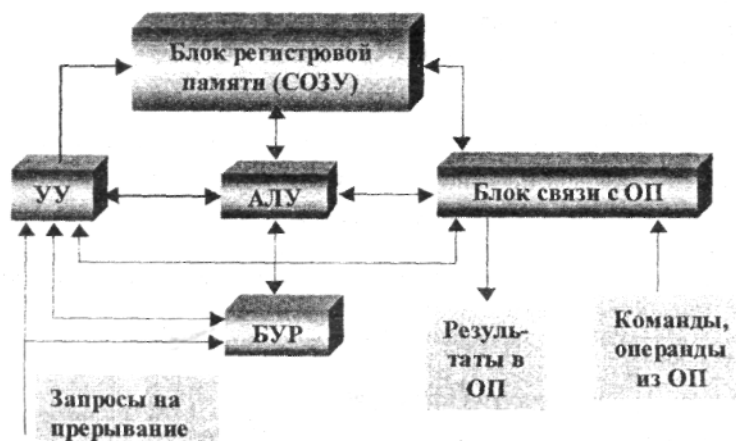
Процесором називається пристрій, що безпосередньо здійснює процес обробки даних і програмне управління цим процесом. Процесор дешифрує і виконує команди програми, організовує звернення до оперативної пам'яті, в потрібних випадках ініціює роботу периферійних пристроїв, сприймає і обробляє запити, що поступають з пристроїв машини і із зовнішнього середовища («запити переривання»).

Процесор займає центральне місце в структурі ЕОМ, оскільки він здійснює розшифровку і виконання команд, а також управління взаємодією всіх пристроїв, що входять до складу ЕОМ.

Виконання команди (машинній операції) розділене на дрібніші етапи - мікрооперації (мікрокоманди), під час яких виконуються певні елементарні дії. Конкретний склад мікрооперацій визначається системою команд і логічною структурою даної ЕОМ. Послідовність мікрооперацій (мікрокоманд), що реалізують дану операцію (команду), утворює мікропрограму операції.

Для визначення тимчасових співвідношень між різними етапами операції використовується поняття машинного такту. Машинний такт визначає інтервал часу, протягом якого виконується одна або одночасно декілька мікрооперацій.

Таким чином, може бути встановлена наступна ієрархія етапів виконання програм в процесорі: програма, команда (мікропрограма), мікрооперація (мікрокоманда). Спрощена структурна схема процесора показана на мал. 2.1.



Мал. 2.1. Спрощена структурна схема процесора.

На схемі зображені тільки основні функціональні блоки процесора:

- арифметико-логічний пристрій (АЛУ);
- керуючий пристрій (керуючий автомат) (КП);
- блок керуючих регістрів (БКР);
- блок регістрової пам'яті (місцева пам'ять);
- блок зв'язку з оперативною пам'яттю (ОП) і деяким іншим, зокрема зовнішнім по відношенню до ЕОМ, устаткуванням.

До складу процесора можуть також входити і інші блоки, що беруть участь в організації обчислювального процесу (блок переривання, блок захисту пам'яті, блок контролю правильності роботи і діагностики процесора і ін.).

Арифметико-логічний пристрій процесора виконує логічні і арифметичні операції над даними (операндами). У загальному випадку, в АЛУ виконуються:

- логічні перетворення над логічними кодами фіксованої і змінної довжини (над окремими бітами, групами біт, байтами і їх послідовностями);
- арифметичні операції над числами з фіксованою і плаваючою крапками, над десятковими числами;
- обробка алфавітно-цифрових слів змінної довжини і ін.

Керуючий пристрій (керуючий автомат) виробляє послідовність керуючих сигналів, ініціюючи виконання відповідної послідовності мікрооперацій, що забезпечує реалізацію поточної команди.

У процесорах ЕОМ застосовують цифрові керуючі автомати з зберігаємою в пам'яті логікою (мікропрограмні керуючі пристрої) і з «жорсткою» логікою. Блок керуючих регістрів призначений для тимчасового зберігання керуючої інформації. Він містить регістри і лічильники, що беруть участь в управлінні обчислювальним процесом: регістри, що зберігають інформацію про стан процесора, регістр-лічильник адреси команди — лічильник команд лічильники тактів, регістр запитів переривання і ін.

До блоку керуючих регістрів слід також віднести керуючі тригери фіксуєчі режими роботи процесора.

Для підвищення швидкодії і логічних можливостей процесора в його склад включають блок регістрової пам'яті (місцеву пам'ять) невеликої ємності, але вищого, ніж основна пам'ять (ОП), швидкодії. Регістри цього блоку (або елементи місцевої пам'яті) указуються в командах програми шляхом укороченої регістрової адресації і служать для зберігання операндів, як акумулятори (регістрів результату операцій), базові і індексні регістри, покажчик стека.

У деяких процесорах базові і індексні регістри входять до складу блоку керуючих регістрів.

Блок зв'язку (інтерфейс процесора) організовує обмін інформацією процесора з оперативною пам'яттю і захист ділянок ОП від недозволених даних програмі звернень, а також зв'язок процесора з периферійними пристроями і зовнішнім по відношенню до ЕОМ устаткуванням (іншими ЕОМ і так далі).

Блок контролю і діагностики (на мал. 2.1 не показаний) служить для виявлення збоїв і відмов в апаратурі процесора, відновлення роботи програми після збоїв і пошуку місця несправності при відмовах.

### **Адресні структури основних пам'ятей**

Основна (оперативна) пам'ять ЕОМ зазвичай є адресною. Це означає, що кожній одиниці інформації (байту, слову), що зберігається в ОП, ставиться у відповідність спеціальне число - адреса, що визначає місце її зберігання в пам'яті.

У сучасних ЕОМ різних типів, як правило, мінімальною одиницею інформації, що адресується в пам'яті, є байт, тобто 8-розрядний код (часто з додатковим дев'ятим контрольним розрядом). Крупніші одиниці інформації - основна машинна одиниця інформації - слово і похідні - півслово, подвійне слово і т. п.- утворюються з цілого числа байт. Звичайне слово відповідає формату даних, що найчастіше зустрічаються в даній машині як операнд. Часто, але необов'язково, формат слова відповідає ширині вибірки (розрядності шини даних) з основної пам'яті.

В більшості випадків адресою слова і інших одиниць інформації служить адреса байта з найменшим номером (молодшою адресою). Число байт в одиниці інформації, що адресується, задається в неявній формі кодом операції команди, або безпосередньо вказується у коді команди.

### **Структури і формати команд, кодування команд**

Всі можливі перетворення дискретної інформації можуть бути зведені до чотирьох основних видів:

- 1) передача інформації в просторі (наприклад, з одного блоку ЕОМ в іншій);
- 2) передача інформації в часі (зберігання);
- 3) логічні (порозрядні) операції;
- 4) арифметичні операції.

Обробка інформації (вирішення завдань) в ЕОМ здійснюється автоматично шляхом програмного управління. Програма є алгоритмом обробки інформації (наприклад, рішення математичної задачі), записаним у вигляді послідовності команд, які повинні бути виконані машиною, для отримання рішення задачі.

Команда є кодом, що визначає операцію обчислювальної машини і дані, що беруть участь в операції. Команда містить також в явній або неявній формі інформацію про адресу, по якій поміщається результат операції, і про адресу наступної команди. По характеру виконуваних операцій розрізняють наступні основні групи команд:

- а) команди арифметичних операцій для чисел з фіксованою і плаваючою крапками;
- б) команди десяткової арифметики;
- в) команди логічних (порозрядних) операцій (І, АБО і ін.);
- г) команди передачі кодів;
- д) команди операцій введення-виводу;
- е) команди управління порядком виконання команд;
- ж) команди завдання режиму роботи комп'ютера і ін.
- д) безадресна команда

## Способи адресації

Слід розрізнити поняття адресний код в команді Ак і виконавська адреса Аї. Адресний код - це інформація про адресу операнда, що міститься в команді. Виконавська адреса - це номер елемента пам'яті, до якої проводиться фактичне звернення. У сучасних ЕОМ, адресний код, як правило, не співпадає з виконавською адресою.

Вибір способів адресації, формування виконавської адреси і перетворення адрес є одним з найважливіших питань розробки ЕОМ. Розглянемо способи адресації, використовувані в сучасних ЕОМ.

**Операнд, що мається на увазі.** У команді не міститься явні вказівки про адресу операнда; операнд мається на увазі і фактично задається кодом операції, команди. Даний спосіб використовується не часто, проте є декілька важливих випадків його застосування. Як приклад можна привести команди підрахунку, в яких деякому числу (вмісту лічильника) додається фіксований приріст, частіше одиниця молодшого розряду. Один з операндів - число в лічильнику - зазвичай адресується явним методом, другий операнд - приріст - не адресується, в пам'яті машина не міститься і є таким, що мається на увазі.

**Адреса, що мається на увазі.** У команді не міститься явні вказівки про адресу операнда, що бере участь в операції, або адреси, по якій поміщається результат операції, але ця адреса мається на увазі. Наприклад, команда може містити адреси обох операндів що беруть участь в операції, при цьому мається на увазі, що результат операції поміщається за адресою одного з операндів, або команда вказує тільки адресу одного операнда, а адреса другого, яким є вміст спеціального регістра (званого регістром результату або акумулятором), мається на увазі.

**Безпосередня адресація.** У команді міститься не адреса операнда, а безпосередньо сам операнд. При безпосередній адресації не вимагається звернення до пам'яті для вибірки операнда і осередку для його зберігання. Це сприяє зменшенню часу виконання програми і займаного нею об'єму пам'яті. Безпосередня адресація зручна для зберігання різного роду констант; проте слід мати на увазі, що при цьому способі адресації довжина операнда коротша за коду команди, оскільки частина розрядів команди зайнята під код операції.

**Пряма адресація.** Виконавська адреса співпадає з адресною частиною команди. Цей спосіб адресації був загальноприйнятим по-перше обчислювальних машинах і продовжує застосовуватися в даний час в комбінації з іншими способами. У вказаній формі безпосередня адресація реалізується в ЕОМ з порівняно довгим машинним словом (32 розряди і більш).

**Відносна адресація або базування.** Виконавська адреса визначається сумою адресної коди команди Ак і деякого числа Аб, званого базовою адресою:  $A_i = A_k + A_b$ .

Для зберігання базових адрес в машині можуть бути передбачені регістри або спеціально виділені для цієї мети елементи пам'яті (базові регістри). У команді виділяється поле В для вказівки номера базового регістра.

Відносна адресація дозволяє при меншій довжині адресної коди команди забезпечити доступ до будь-якого елемента пам'яті. Для цього число розрядів в базовій адресі вибирають таким, щоб можна було адресувати будь-який осередок ОП, а адресний код Ак самої команди використовують для уявлення лише порівняно короткого «зсуву» (позначають буквою D). Зсув D визначає положення операнда відносно початку масиву, що задається базовою адресою Аб.

Відносна адресація забезпечує так звану переміщуваність програм, тобто можливість пересування програм в пам'яті без змін усередині самої програми.

**Укорочена адресація.** Для зменшення довжини коди команди часто застосовується так звана укорочена адресація. Суть її зводиться до того, що в команді задаються тільки молодші розряди адрес, старші розряди при цьому мають на увазі нульовими. Така адресація дозволяє використовувати тільки невелику групу фіксованих осередків з початковими (короткими) адресами і тому може застосовуватися лише спільно з іншими способами адресації.

**Регістрова адресація** - є окремий випадок укороченим, коли як фіксовані осередки з короткими адресами використовуються регістри (елементи надоперативної або місцевої пам'яті) процесора. Наприклад, якщо таких регістрів 16, то для адреси досить чотири двійкові розряди. Регістрова адресація разом з скороченням довжини адрес операндів дозволяє збільшити швидкість виконання операцій, оскільки зменшується число звернень до ОП.

**Непряма адресація.** Адресний код команди вказує адресу елемента пам'яті, в якій знаходиться адреса операнда або команди. Таким чином, непряма адресація може бути інакше визначена як «адресація адреси».

На непряму адресацію вказує код операції команди, а в деяких ЕОМ в команді відводиться спеціальний розряд (показчик адресації - УА), і цифра 0 або 1 в ній вказує, є адресна частина команди прямою адресою або непрямою.

У деяких ЕОМ використовується багатоступінчата непряма адресація. В цьому випадку елементи пам'яті містять також розряд-показчик непрямої адресації (УА). Якщо цей розряд вказує на продовження непрямої адресації, то машина послідовно вибирає з пам'яті адреси до тих пір, поки не буде знайдений осередок, в якому розряд-показчик визначить пряму адресацію. Адреса з цього останнього осередку і є шуканою виконавською адресою.

**Автоінкрементна і автодекрементна адресації.** Оскільки регістрова непряма адресація вимагає попереднього завантаження регістра з ОП непрямою адресою, що пов'язане з втратою часу, такий тип адресації особливо ефективний при обробці масиву даних, якщо є механізм автоматичного приросту або зменшення вмісту регістра при кожному зверненні до нього, званий відповідно автоінкрементною і автодекрементною адресацією. В цьому випадку достатньо 1 раз завантажити в регістр адресу першого оброблюваного елемента масиву, а потім при кожному зверненні до регістра в ній в результаті інкрементної (декрементом) процедури формується адреса наступного елемента масиву.

При автоінкрементній адресації по вмісту регістра спочатку вміст регістра використовується як адреса операнда, а потім отримує приріст, рівний числу

байт в елементі масиву. При автодекрементній адресації спочатку вміст вказаного в команді регістра зменшується на число, рівне числу байт в елементі масиву, а потім використовується як адреса операнда.

Автоінкрементна і автодекрементна адресації можуть розглядатися як спрощений варіант індексації - вельми важливого механізму перетворення адресних частин команд і організації обчислювальних циклів, тому їх часто називають автоіндексацією.

**Адресація слів змінної довжини.** Ефективність обчислювальних систем, призначених для обробки даних (економічних, планових і ін.), підвищується, якщо є можливість виконувати операції із словами змінної довжини. В цьому випадку в машині повинна бути передбачена адресація слів змінної довжини, яка зазвичай реалізується шляхом вказівки в команді місцеположення в пам'яті почала слова і його довжини.

Зазвичай в ЕОМ одночасно використовується декілька типів адресації. Тип адресації вказується або неявно кодом операції, або в явній формі із спеціальному полі адресної частини команди

### **Стекова адресація**

Стекова пам'ять, що реалізовує безадресне завдання операндів, є ефективним елементом сучасної архітектури ЕОМ, особливо широко використовуваним в мікропроцесорах. Стек є групою послідовно пронумерованих регістрів (апаратурний стек) або елементів пам'яті, забезпечених покажчиком стека (зазвичай регістром покажчиком стека (ВУС)), в якому автоматично при записі і зчитуванні встановлюється номер (адреса) останнього зайнятого осередку стека (вершини стека). При операції запису, слово, що заноситься в стек, поміщається в наступний по порядку вільний осередок стека, а при зчитуванні із стека витягується останнє слово, що поступило в нього. Таким чином, в стеку реалізується правило «останній прийшов - перший пішов».

Вказане правило при зверненні до стека реалізується автоматично, і тому при операціях із стеком можливе безадресне завдання операнда. Команда не містить адреси осередку стека, але містить адресу (або він має на увазі) елементу пам'яті або регістра, звідки слово передається в стек або куди поміщається із стека.

Безадресні команди на основі стекової адресації гранично скорочують формат команд, економлять пам'ять і сприяють підвищенню продуктивності ЕОМ.

### **Класифікація процесорів**

Універсальний процесор - містить суматор і не менше трьох регістрів з системою прийому, передачі і перетворення код операндів і команд. Може виконувати всі логічні і арифметичні операції.

Функціональний процесор - містить декілька операційних блоків або спеціалізованих процесорів. Застосування спеціалізованих процесорів, орієнтованих на певний тип виконуваних команд, дозволяє спростити їх структуру і отримати максимальну швидкодію для використовуваної

елементно-технологічної бази (спрощуються ланцюги, що управляють, зменшується кількість логічних рівнів логічного перетворення інформаційних сигналів, а також знижується величина затримок сигналів у вузлах процесорних блоків).

Стекові процесори - відрізняються від універсальних процесорів тим, що їх регістровий надоперативний запам'ятовуючий пристрій (СОЗУ) реалізований у вигляді стека. Стекові безадресні процесори призначені для виконання програм, написаних на проблемно-орієнтованих мовах програмування високого рівня. Короткий формат безадресних команд дозволяє ефективно використовувати інформаційний об'єм ОЗУ для зберігання програм. Застосування стека значно збільшує продуктивність процесора і зменшує кількість звернень до ОЗУ за операндами і проміжними результатами обчислень.

Недоліком стекового процесора є складність організації стекової регістрової пам'яті великої глибини. Значні затримки передачі управління при виконанні команд умовного і безумовного переходів.

Конвеєрні (магістральні) процесори. Для підвищення продуктивності процесора можна розбити всі мікрооперації виконуваних команд (т) фаз і кожен фазу виконувати в спеціалізованому блоці. Пари операндів в цьому випадку приймаються з ОЗУ і передаються послідовно в перший, другий і подальші операційні блоки. У кожному такті передачі виконуються деякі елементарні перетворення. Остаточний результат для першої пари операндів виходить через (т) тактів. Після заповнення «конвеєра» операційних блоків парами операндів результати виконання команд виходитимуть кожен такт. Отже, ніж більше фаз виділяється для виконання команд, тим вище продуктивність конвеєрного процесора. Як спеціалізовані операційні блоки можуть бути використані блоки порівняння порядків чисел, складання (множення) мантис і тому подібне. Конвеєрний процесор ефективно обробляє безперервні потоки операндів.

Векторні процесори - орієнтовані на обробку даних, представлених у вигляді векторів. Цей процесор має в своєму складі достатньо апаратних засобів для обробки паралельно одного рядка або стовпця матриці. За принципом роботи можуть бути конвеєрними.

Матричні процесори - за принципом організації схожі на векторних. Одночасно можуть обробляти паралельно елементи матриці. Для цього в єдину структуру під загальним управлінням об'єднують велику кількість спеціалізованих процесорів. Для завантаження таких процесорів можуть використовуватися універсальні процесори.

Асоціативні процесори - є матричні процесори з асоціативним ОЗУ. Ефективно реалізують операції пошуку на рівність операндів, впорядкування вибірки операндів, вибору найбільшого (найменшого) числа, вибірки чисел із заданого діапазону і тому подібне

## Заняття № 3

### Принципи організації арифметико-логічних пристроїв

#### Цілі заняття

- Отримати уявлення про призначення і перелік операцій, виконуваних АЛУ.
- Познайомитися з основними типами АЛУ.
- Зрозуміти і засвоїти архітектурні особливості побудови арифметико-логічних пристроїв ЕОМ.
  - Засвоїти послідовність дій, що виконуються функціональними елементами АЛУ при виконанні різних операцій над кодами операндів.

#### Загальні відомості

Арифметико-логічні пристрої (АЛУ) служать для виконання арифметичних і логічних перетворень над словами, званими в цьому випадку операндами.

Виконувані в АЛУ операції можна розділити на наступні групи:

- операції двійкової арифметики для чисел з фіксованою крапкою;
- операції двійкової (або шістнадцятиричної) арифметики для чисел з плаваючою крапкою;
  - операції десяткової арифметики;
  - операції індексної арифметики (при модифікації адрес команд);
  - операції спеціальної арифметики;
  - операції над логічними кодами (логічні операції);
  - операції над алфавітно-цифровими полями.

До арифметичних операцій відносяться складання, віднімання, віднімання модулів («короткі операції») і множення і ділення («довгі операції»). Групу логічних операцій складають операції диз'юнкція (логічне АБО) і кон'юнкція (логічне І) над багато розрядними двійковими словами, порівняння код на рівність. Спеціальні арифметичні операції включають нормалізацію, арифметичне зрушення (зрушуються тільки цифрові розряди, знаковий розряд залишається на місці), логічне зрушення (знаковий розряд зрушується разом з цифровими розрядами). Обширна група операцій редагування алфавітно-цифрової інформації.

Можна привести наступну класифікацію АЛУ.

За способом дії над операндами АЛУ діляться: на послідовних і паралельних. У послідовних АЛУ операнди представляються в послідовному коді, а операції проводяться послідовно в часі над їх окремими розрядами. У паралельних АЛУ операнди представляються паралельним кодом і операції здійснюються паралельно в часі над всіма розрядами операндів.

За способом представлення чисел розрізняють АЛУ: 1) для чисел з фіксованою крапкою; 2) для чисел з плаваючою крапкою; 3) для десяткових чисел.

По характеру використання елементів і вузлів АЛУ діляться: на блокових і багатофункціональних. У блоковому АЛУ операції над числами з фіксованою і плаваючою крапкою, десятковими числами і алфавітно-цифровими полями виконуються в окремих блоках, при цьому підвищується швидкість роботи, оскільки блоки можуть паралельно виконувати відповідні операції, але значно зростають витрати устаткування. У багатофункціональних АЛУ операції для всіх форм представлення чисел виконуються одними і тими ж схемами, які комутуються потрібним чином залежно від необхідного режиму роботи.

По своїх функціях АЛУ є операційним блоком, що виконує мікрооперації, що забезпечують прийом з інших пристроїв (наприклад, пам'яті) операндів, їх перетворення і видачу результатів перетворення в інші пристрої. Процес перетворення операндів в АЛУ управляється блоком, що управляє, генерує сигнали, що управляють, ініціюють виконання в АЛУ певних мікрооперацій. Що генерується блоком, що управляє, послідовність сигналів визначається кодом операції команди і оповіщаючими сигналами.

### **Структура АЛУ для складання і віднімання чисел з фіксованою крапкою**

Зазвичай в АЛУ операція складання алгебри зводиться до арифметичного складання код чисел шляхом застосування інверсних кодів - додаткового або зворотного для представлення чисел.

Алгоритми виконання в АЛУ арифметичних операцій залежать від того, в якому вигляді зберігаються в пам'яті ЕОМ числа - в прямому або додатковому. У останньому випадку скорочується час виконання операції за рахунок виключення операції перетворення отриманого в АЛУ додаткового коду негативного результату в прямий код, хоча при цьому декілька ускладнюється операція множення. Передбачається, що числа зберігаються в пам'яті в додатковому коді. До складу АЛУ входять:

- n-розрядний паралельний комбінаційний суматор **См**;
- регістр суматора **Ргсм**;
- вхідні регістри суматора **РГВ** і **РГА**;
- вхідний регістр АЛУ **Рг1**.

### **Пристрою для виконання логічних операцій**

До складу операцій, ЕОМ, що реалізуються, входять наступні:

- порозрядні логічні операції;
- підсумовування по модулю 2;
- логічне множення І;
- логічне складання АБО.

## **Заняття 4**

### **Організація однокристальних 8-розрядних мікропроцесорів K580.**

#### **Цілі заняття**

- Зрозуміти і засвоїти тенденції і шляхи вдосконалення мікропроцесорної техніки.
- Зрозуміти і засвоїти відмінності між технологіями виготовлення мікропроцесорів.
- Вивчити архітектуру мікропроцесора K580, призначення і характеристики блоків і вузлів.
- Представляти, дії, що виконуються мікропроцесором при зверненні до пам'яті і інших пристроїв обчислювальної системи.
- Вивчити функціональне призначення вхідних і вихідних сигналів мікропроцесора.

#### **Загальні відомості**

Розвиток інтегральної технології і схемотехніки привів до появи цифрових електронних схем з великим і надвеликим ступенями інтеграції (БІС і СБІС), що містять на одному кристалі (у одному корпусі) декілька десятків тисяч елементарних транзисторів. На основі таких схем останніми роками вдалося створити мікропроцесори — функціонально закінчені, з програмою пристрою, що зберігається в пам'яті, обробки цифрової інформації, виконані у вигляді однієї або декількох БІС або СБІС.

Мікропроцесори (МП) відрізняються у край малими габаритними розмірами, великою швидкістю, високою надійністю і дешевизною. Дуже важливою особливістю мікропроцесорів є їх універсальність, тобто можливість найрізноманітнішого застосування завдяки їх програмоване на виконання конкретних функцій.

Малі габаритні розміри, універсальність, здатність реалізувати складні функції обробки даних і управління служать основою для побудови персональних комп'ютерів, мікропроцесорних пристроїв і систем управління, що вбудовуються в різні апарати, машини, прилади і системи.

Мікропроцесорні засоби проводяться у вигляді мікропроцесорних комплектів інтегральних мікросхем, що мають єдиного конструктивно-технологічного виконання і призначених для сумісного застосування. Мікропроцесорний комплект крім самого мікропроцесора містить мікросхеми, що підтримують функціонування мікропроцесора і що розширюють його логічні можливості.

Розвиток мікропроцесорів ознаменувався змаганням біполярною і МОП-ТЕХНОЛОГІЙ мікроелектроніки.

МОП-структури електронних схем дозволяють розміщувати на одному кристалі велике число елементарних схем завдяки їх невеликим розмірам і невеликій потужності розсіяння.

Біполярні БІС, наприклад ТТЛ-схеми, володіли набагато більшою швидкістю, але значно меншою щільністю компонентів на кристалі. Тому досить важко було побудувати біполярний мікропроцесор на одному кристалі.

Для подолання обмежень, пов'язаних з порівняно невеликою щільністю компонентів у біполярних схем, був запропонований секційний метод конструювання мікропроцесорів.

По цьому методу мікропроцесор складається з декількох однакових 2-, 4- або 8-розрядних процесорних секцій, розміщених на окремих кристалах і об'єднаних загальним мікро програмним управлінням. Використання секційного методу у поєднанні з прогресивною технологією малопотужних ТТЛ-схем процесорні комплекти серій: К589, КР 1802, КР 1804, містили відповідно 2-, 4-, 8-розрядні процесорні кристали (секції) з швидкістю (часом циклу) 150 нс. До складу цих комплексів крім БІС процесорній секції входять п'ять - сім додаткових БІС (блоки мікропрограмного управління, синхронізації, зв'язки з периферійними пристроями, переривань). До даного типу мікропроцесорних засобів відноситься реалізована з використанням ЭСЛ-технології БІС К1800, що містить 4-розрядні секції АЛУ, мікропрограмного управління, синхронізатора, управління оперативною пам'яттю.

На основі секційних мікропроцесорних БІС можна будувати мікропроцесори із змінною розрядністю слова і мікропрограмним управлінням. Ці мікропроцесори не мають фіксованого набору команд, а мають тільки набір мікрокоманд, що дозволяє (хоча це пов'язано з певними труднощами для широкого користувача) реалізувати мікропрограмним шляхом оптимальний для даного завдання набір команд і процедур.

Секційні ТТЛШ і ЭСЛ-мікропроцесорні серії БІС використовуються при побудові пристроїв обчислювальної техніки і автоматики, яким пред'являються підвищені вимоги відносно швидкодії.

### **Архітектура мікропроцесора КР580ВМ80А**

Восьмирозрядний мікропроцесор з фіксованою системою команд КР580ВМ80А (у подальшому МП К580, є аналогом мікропроцесора Intel 8080, який з'явився в 1978 р.) виконаний у вигляді виготовленої по МОП-ТЕХНОЛОГІИ БІС, такої, що містить близько 6 тис. транзисторів на кристалі кремнію розміром близько 30 мм<sup>2</sup>. Виготовлений за 6-мкм технологією і поміщений в корпусі з 40 виводами. Працює з тактовою частотою 2 МГц (тривалість такту 500 нс), вимагає трьох рівнів напруги живлення: +5, -5, +12 В.

Довга машинного слова в Мп580 - 8 бітів, а розрядність шини адреси - 16 бітів. Мікропроцесор може працювати з оперативною і постійною пам'яттю, сумарна ємність яких не перевищує 64 Кбайт. Ширина вибірки з пам'яті - 1 байт. Можлива адресація будь-якого байта пам'яті. Швидкість виконання коротких (операцій типу реєстр - реєстр) 500 тис. операцій/с.

Створення високопродуктивного мікропроцесора з ефективними системою команд і процедурами обміну інформацією із зовнішнім по відношенню до процесора устаткуванням важко із-за обмежень, які викликалися

коротким словом МП і малим числом зовнішніх виводів його корпусу. Останнє, зокрема, зумовило малорозрядну шину даних до МП і вузького інтерфейсу обміну даними із зовнішнім устаткуванням. Для подолання цих обмежень було потрібно розробки ряду нових архітектурних рішень.

Характерними особливостями організації МП К580 є:

- трьохшинна структура з шинами даних, адреси і управління;
- магістральна структура зв'язків - наявність внутрішньої шини даних (8 бітів), яка зв'язує всі вузли усередині МП (ширина шини даних рівна розрядності слова, з якими оперує МП);
- регістрова пам'ять у вигляді блоку програмно-доступних загальних і деяких спеціалізованих регістрів, покажчик стека, регістр непрямої адреси, регістр ознак (прапорів);
- наявність 16-розрядної шини адреси, що дозволяє прямо адресувати зовнішню пам'ять ємністю 64 Кбайт;
- різноманітність вживаних методів адресації - пряма, регістрова пряма, регістрова непряма (зокрема що мається на увазі), безпосередня, індексна, стекова, - що в сукупності дозволяють при короткому 8-розрядному слові реалізовувати достатньо гнучку систему команд, що використовує три формати команд: 1, 2, 3 байти;
- наявність ефективних засобів роботи з підпрограмами і швидкодіючою системою переривання, що використовують стекову пам'ять («перевернутий стек»); спеціальні команди виклику підпрограм і повернення (зокрема умовного) з підпрограм; процедури переходу до тих, що переривають і повернення до перерваних програм;
- реалізація двобайтових (тандемних) передач і деяких інших двобайтових операцій, з тим щоб при 8-розрядних: шині даних, загальних регістрах і ширині вибірки з ОП спростити процедури обробки 16-розрядних слів, роботу з 16-розрядними адресами і трьома форматами команд (останнє досягається тим, що перший байт команди, що містить вказівку про її формат, завантажується завжди в регістр команди, а другий і третій, якщо вони є, - в певні регістри).

Крім згадуваних шин даних і адреси є шина управління, що містить лінії, призначені для передачі сигналів, що управляють, ознак стану процесора і периферійного устаткування. Шина містить наступні лінії:

- синхронізуючих сигналів для супроводу код при передачах їх в обох напрямках по мультиплексуємії шині даних;
- сигналів, що інформують МП про стан (готовності) периферійних пристроїв;
- сигналів запиту переривання від периферійних пристроїв і дозволу переривання і ін.

Структурна схема мікропроцесора К580 приведена на мал. 4.1.

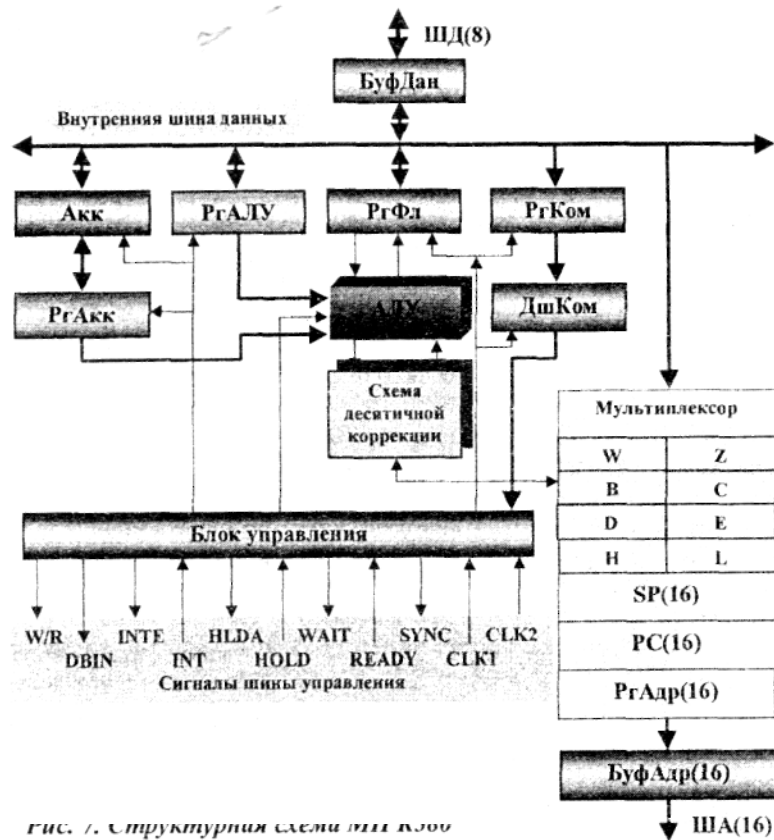


рис. 1. Структурная схема микр. К580

Мал. 4.1 Структурна схема МП К580.

У склад БІС МП580 входять:

- 8-розрядное арифметико-логічний пристрій (АЛУ);
- 8-розрядный регістр ознак (регістр прапорів Ргфл), що фіксує ознаки, формовані АЛУ в процесі виконання команд;
- 8-розрядный регістр акумулятор (Акк), який завжди містить результат операції над операндами;
- 8-розрядный регістр акумулятора (Ргакк), призначений для тимчасового зберігання операнда, що знаходиться в акумуляторі;
- 8-розрядный регістр тимчасового зберігання операндів (РгАЛП) при виконанні операцій в АЛУ;
- десятковий коректор, що виконує переклад код з двійкової в двійково-десяткову форму;
- 8-розрядный регістр команд (Ргком), призначений для зберігання першого байта (коди операції) команди;
- дешифратор команд, призначений для перетворення двійкової коди операції в сигнал управління запуском мікропрограми, що реалізовує команду;
- двонаправлений мультиплексор, призначений для перемикання внутрішньої шини даних між регістрами;
- блок регістрів загального призначення (РОН) для прийому, видачі і зберігання код в процесі виконання програм, що містить шість 8-розрядних регістрів (У, З, D, E, H, L);
- допоміжні регістри тимчасового зберігання W і Z;

- 16-розрядний реєстр адреси вершини стека SP, містить поточну адресу вершини стека;
- 16-розрядний лічильник команд (програмний лічильник) PC, формує адресу команди або операнда, автоматично збільшуючи на 1,2, або 3 поточна адреса залежно від формату виконуваної команди;
- 8-розрядний буфер шини даних (Буфдан);
- 16-розрядний реєстр адреси (Ргадр), використовується для зберігання адрес команд і даних;
- 16-розрядний буфер адреси, призначений для узгодження внутрішньої і зовнішньої шин адреси МП.

Мікропроцесор має 16-розрядну трьохстабільну шину адреси А, 8-розрядну трьохстабільну шину даних D(7-0), чотири вхідних і шість вихідних виводів управління і стану. МП К580 забезпечує адресацію пам'яті об'ємом 65536 байтів (С164 Кбайт), а також адресація 256 пристроїв введення і виводу.

## **Заняття 5**

### **Організація однокристальних 16-розрядних мікропроцесорів**

#### **Цілі заняття**

- Зрозуміти і засвоїти причини і шляхи вдосконалення мікропроцесорної техніки, що привели до появи МП Км1810.
- Вивчити архітектуру мікропроцесора Км1810, призначення і характеристики блоків і вузлів.
- Вивчити функціональне призначення вхідних і вихідних сигналів мікропроцесора.
- Отримати уявлення про способи адресації до пам'яті, реалізованих в МП Км1810.
- Засвоїти основні особливості системи команд МП Км1810.

#### **Загальні характеристики однокристальних 16-розрядних МП**

На основі п-МОП-технології з кремнієвими затворами вдалося реалізувати принцип пропорційного зменшення розмірів МОП-СХЕМ, досягти більшого ступеня інтеграції (близько 30 тис. транзисторів на кристалі розміром 5,5x5,5 мм) і високої швидкодії. Затримка на елемент зменшилася, вона стала того ж порядку, що і в ТТЛ-схемах з діодами Шотки, що мають значно великі розміри і споживану потужність.

На МОН БІС вказаного типу фірмою Intel (США) був створений в 1978 році однокристальний 16-розрядний МП 8086 (прототип вітчизняного МП Км1810вм86, надалі для скорочення іменованого Км1810). Прилади випускаються в 40-контактному корпусі.

Продуктивність МП КМ1810 складає при тактовій частоті 5 Мгц 2,5 млн. операцій типу реєстр-реєстр (і8086- має частоту синхронізації до 10 Мгц). Це досягнуто завдяки підвищенню швидкодії схем і архітектурним удосконаленням. Архітектура МП Км1810 має наступні особливості:

- виконання апаратурними засобами арифметичних операцій над 8- і 16-розрядними двійковими числами із знаком і без знаку, десятковими двійково-кодованими числами, логічних операцій під ланцюжками даних, розширені можливості роботи з окремими розрядами слів; наявність 16-розрядного АЛУ з апаратурною реалізацією множення і ділення;
- реєстрова структура з подвоєним в порівнянні з К580 числом загальних реєстрів, значне число рівнів векторного переривання;
- повна сумісність за системою команд з МП К580 (зокрема робота з 8-розрядними командами останнього) і одночасна наявність нових, ефективних 16-розрядних команд;
- сегментна адресація, що дозволяє прямо адресувати один мегабайт пам'яті (оперативну, дискову і тому подібне), проводити динамічне переміщення програм;
- використання одного рівня напруги живлення 5В.

Зв'язок між МП, ОП і периферійними пристроями здійснюється за допомогою інтерфейсу. Інтерфейс процесора має 20 ліній адреси, 16 з яких використовується і як лінії дані. Ця обставина приводить до того, що на системну шину не можна одночасно подавати адреси і дані. Мультиплексування адреси і даних в часі скорочує число контактів корпусу, але уповільнює швидкість передачі даних. МП має 16 ліній управління.

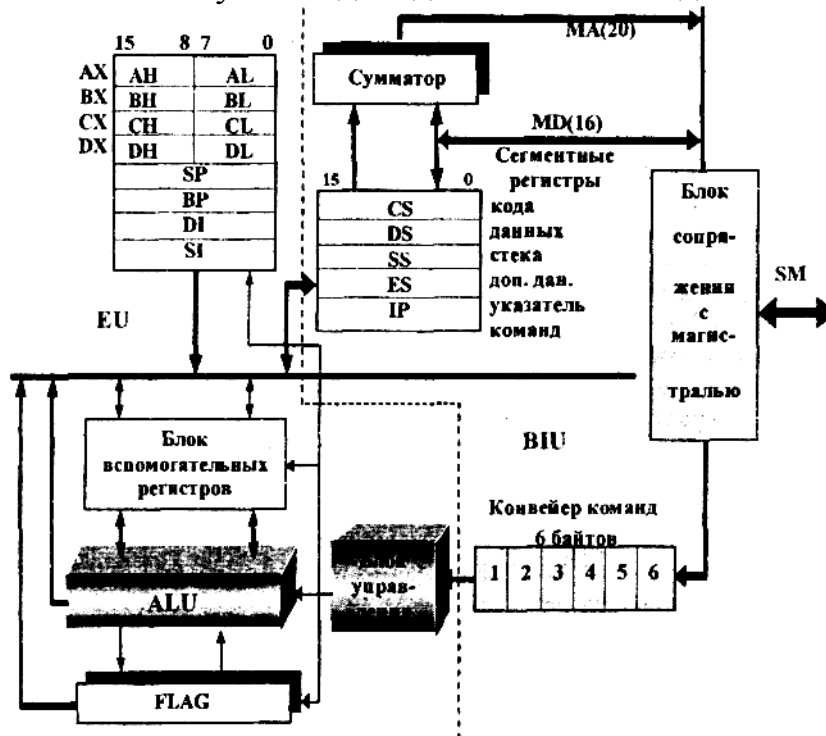
На мал. 5.1 представлена структурна схема МП Км1810, в якому є відносно автономні пристрої:

а) блок сполучення з магістраллю (Biu, Bus Interface Unit), що забезпечує випереджаючу вибірку команд і формування черги вибраних байт послідовності команд в спеціальній регістровій пам'яті (конвеєр команд (ємність 6 байт), а також формування фізичної адреси пам'яті, читання операндів з пам'яті або регістрів введення-виводу і запис результату операції в пам'ять або реєстри введення-виводу;

б) блок виконання команд (EU, Execution unit), що витягує команди з черги і реалізовує управління наказаними командами операції в 16-розрядному арифметико-логічному пристрої (ALU).

Пристрій сполучення з шиною крім регістрів черги команд має блок 16-розрядних регістрів переадресації, 16-розрядний суматор адреси. Сюди ж можна віднести покажчик (лічильник) команд.

Така структура при певному співвідношенні тактової частоти МП і тривалості циклу пам'яті дозволяє отримати ефективно поєднання процесів вибірки і виконання команд. Одному циклу основної пам'яті (800 нс) відповідають - чотири такти роботи МП Км1810. При поєднанні за час одного циклу основної пам'яті виконуються дві однобайтні команди.



Мал. 5.1. Програмна модель МП К1810.

Операційний пристрій включає блок 16-розрядних загальних регістрів, що містить чотири регістри даних: акумулятор АХ, базовий регістр ВХ, лічильник СХ і дані DX, регістри - покажчики стека SP і бази ВР, індексні регістри операнда SI і результату DI, а також АЛУ і 16-розрядний регістр ознак (прапорців) F.

Пристрій сполучення з магістраллю управляє процедурами формування і передачі адрес, операндів і команд, а також обміном даними з периферійними пристроями, включаючи обробку запитів переривання і реалізацію прямого доступу до пам'яті.

Мікропроцесор Км1810 має спеціальні апаратурні засоби (сегментна адресація і ін.), що підтримують мультипрограми режим роботи і що полегшують створення на основі цього МП багатопроцесорних систем, зокрема що містять арифметичні співпроцесори, що дозволяють у декілька разів збільшити швидкість виконання арифметичних операцій з плаваючою крапкою і обчислення деяких функцій.

Регістри даних АХ, ВХ, СХ, DX служать для зберігання операндів і результатів операцій. Можлива адресація, як цілих регістрів, так і їх молодшою L і старшою H частин. Деяким регістрам разом із загальним додається і спеціальне призначення. У останньому випадку можливо у відповідних командах адресувати ці регістри неявно самим кодом операції (адресація, що мається на увазі). Так, регістр АХ використовується як акумулятор, регістр ВХ - як базовий регістр, СХ - як лічильник в командах зрушень, управління обчислювальними циклами і в операціях з ланцюжками байт, а регістр DX неявно адресується в командах множення і ділення, а в деяких операціях введення-виводу зберігає адресу порту введення-виводу.

У МП Км1810 використовується сегментація пам'яті (уявлення у вигляді частин - сегментів), організовувана за допомогою 16-розрядних сегментних регістрів: коди CS, даних DS, стека SS і додаткового сегменту ES, що зберігають базові логічні адреси сегментів поточної програми. Ці базові адреси повинні бути кратні 16. Розмір одного сегменту - 64 Кбайт. Допускається перекриття сегментів.

Сегментна організація пам'яті дозволяє розділити по місцеположенню різні типи код: коди команд в сегменті коди; коди даних в сегментах даних і додаткових даних; стек розміщується в сегменті стека.

### **Способи адресації**

У командах МП До 1810 використовуються наступні способи адресації:

- безпосередня - дані завдовжки 8 або 16 біт є частиною команди;
- пряма - 16-бітова ефективна адреса даного вказується в адресній частині команди;
- регістрова пряма - дані містяться у визначуваному командою регістрі (16-бітовий операнд може знаходитися в регістрах АХ, ВХ, СХ, DX, Si, DI, SP, ВР, а 8-бітовий - в регістрах AL, BL, CL, DL, AH, BH, CH, DH);
- регістрова непряма - ефективна адреса операнда знаходиться в базовому регістрі ВХ або індексному регістрі DI або SI;

- регістрова відносна (базування) - ефективна адреса рівна сумі 8- або 16-бітового зсуву і вмісту базового (BX, BP) або індексного (SI, DI) регістрів;
- базова індексна - ефективна адреса рівна сумі вмісту базового (BX, BP) і індексного (SI, DI) регістрів, визначуваних командою;
- відносна базова індексна - ефективна адреса рівна сумі 8- або 16-бітового зсуву і вмісту базового (BX, BP) і індексного (SI, DI) регістрів.

### **Особливості структури мікропроцесора Intel 8088**

У 1979 році фірма Intel випустила мікропроцесор i8088. Його структура є декілька спрощений варіант структури МП i8086. Спрощення досягнуте за рахунок зменшення ширини шини даних - до 1 байта (8 ліній) замість 2 байт (16 ліній) і деяких інших, пов'язаних з вузькою специфікою змін (4-байтний буфер команд замість 6-байтного, ініціація вибірки команди, коли в буфері залишився один байт - замість двох в МП 8086 і ін.). Проте, були збережені повна програмна сумісність цих МП, 20-розрядна адреса, виконання операцій з 16-розрядними операндами, сегментні регістри, засоби підтримки мультипрограмного і багатопроцесорного режимів.

У 1981 році фірма IBM вийшла на ринок персональних комп'ютерів, використавши як процесора МП 8088. Завдяки такій могутній підтримці мікропроцесори фірми Intel стали найбільш популярними у виробництві персональних ЕОМ.

### **Процесори i80186/i80188**

Як і процесори i8086/i8088, МП i80186/i80188 є процесорами з 16-розрядною внутрішньою архітектурою і програмно сумісні з i8086. Відмітною особливістю є:

- вбудовані периферійні контролери переривань, прямого доступу в пам'ять, трьохканальний таймер і синхрогенератор;
- скорочена кількість тактів, потрібних для виконання деяких команд;
- мають засоби управління енергоспоживанням;
- є модифікації, у яких вбудовані послідовний порт і контроллер регенерації динамічного ОЗУ.

МП i80186/i80188 застосовуються як вбудовувані в устаткування контроллери.

## Заняття 6

### Архітектура процесора i80286

#### Цілі заняття

- Засвоїти основні загальні характеристики другого покоління 16-розрядних процесорів фірми Intel.
- Вивчити архітектуру процесора i80286, призначення і характеристики блоків і вузлів.
- Вивчити функціональне призначення вхідних і вихідних сигналів мікропроцесора.
- Отримати уявлення про стани процесора, реалізовані в i80286.
- Засвоїти особливості алгоритму функціонування i80286. Загальні відомості про параметри

#### Загальні відомості

Виробництво процесора i80286 почате в 1982 році. До складу i80286 входять 130 000 транзисторів, що дозволило змоделювати всі функції процесорів i808S/8086 і ввести нові функціональні можливості. Близько сорока відсотків нових транзисторів додано для прискорення операцій МП i8088/8086, а останні входять в схеми управління пам'яттю, управління завданнями і захисту. Підвищення продуктивності можна оцінити приблизно в 3,5 разу вище за МП i8088 і в 2,5 разу вище за МП i8086. Значне підвищення продуктивності МП i80286 пов'язане з конвеєризацією і підвищенням тактової частоти до 12,5 МГц. Завдяки конвеєризації декілька внутрішніх устроїв працюють одночасно, суміщаючи дешифрування команд, операції в АЛУ, обчислення ефективної адреси і цикли шини декількох команд. У складі процесора i80286 є 4 конвеєрних пристрої: командний пристрій (IU), операційний пристрій (EU), адресний пристрій (AU) і шинний пристрій (BU).

Процесор може працювати в двох режимах: реальному і захищеному.

**Real Address Mode** - режим реальної адресації повністю сумісний з роботою МП i8086. У цьому режимі по 20-розрядній шині адреси можлива адресація фізичної пам'яті до 1 Мб, представленою у вигляді декількох сегментів розміром по 64 Кбайта. Окрім цього, процесор може звертатися до 65536 портів введення виводу.

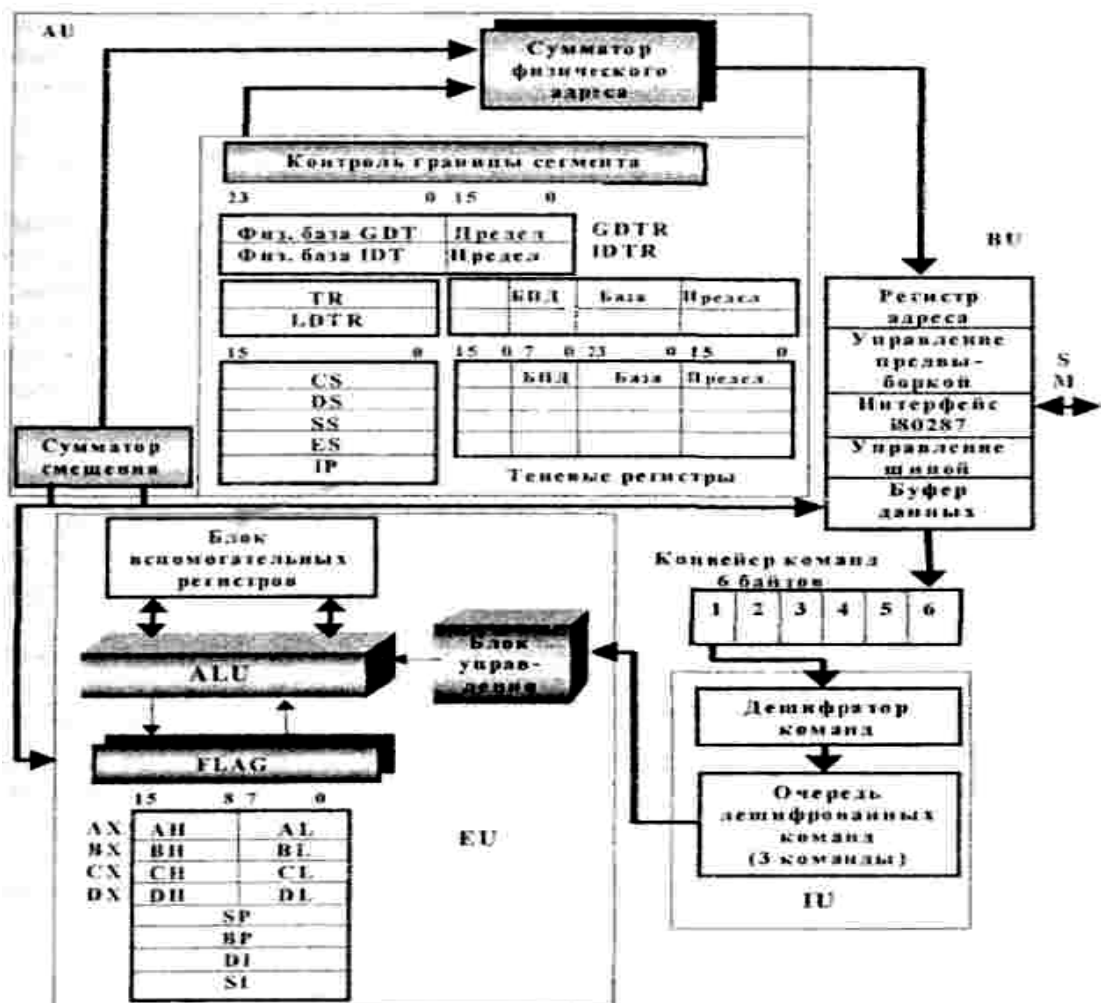
**Protected Virtual Address Mode** - захищений режим віртуальної адресації. Режим дозволяє мати доступ по 24-розрядній шині адреси до 16 Мбайтам фізичної пам'яті і 1Гбайту віртуальною для кожного завдання.

У архітектуру процесора i80286 вбудовано декілька новинок. Перш за все, це внутрішній устрій управління пам'яттю (MMU), яке дозволяє процесору виконувати декілька завдань.

Розроблені раніше зовнішні MMU виявилися більш обмеженими по можливостях. Внутрішнє MMU реалізує не тільки базові функції, але і забезпечує управління завданнями і захист завдань. Розглянемо коротко ті функції MMU, які були реалізовані до появи мікросхеми i80286.

Стандартні функції управління пам'яттю раніше виконувалися поза процесором. Наприклад, в «сімействі 86/380» фірм Intel була зовнішня плата управління пам'яттю для підтримки многопользовательской операційної системи XENIX. У інших сімействах також застосовувалися зовнішні MMU. Звичайно, можливості таких MMU були обмежені особливостями розробки. Оскільки MMU проводилися за замовленням, спроб стандартизації їх функцій не було, а це значно ускладнювало розробку "переносимих" програмних продуктів. Завдяки MMU процесор i80286 володіє двома достоїнствами для реалізації нових застосувань. По-перше, він не вимагає зовнішнього MMU, що підвищує продуктивність але доступу до пам'яті і гарантує транспортабельність програм. По-друге, можливості внутрішнього MMU ширші, ніж в інших розробках. Зокрема, MMU може управляти завданнями і володіє механізмом захисту. Це, у свою чергу, забезпечує апаратну підтримку мультизадачне і цілісність системи.

Завдяки вбудованій функції перемикавання завдань i80286 має нову мультизадачну архітектуру, що підтримує декілька програмних середовищ, званих завданнями, з можливістю простого перемикавання з одного завдання на інше. Наявність ієрархічних рівнів привілеїв приводить до захищеної архітектури, що підвищує надійність мікропроцесорних систем.



Мал. 6.1. Структура процессора i80286.

## **Структура процесора i80286**

Для підвищення продуктивності мікропроцесор i80286 розбитий на чотири блоки, що працюють незалежно. Ці блоки обробляють команди в конвеєрному режимі.

При проектуванні процесора i80286 була поставлена мета добитися виконання 1,5 мільйонів команд в секунду. За наслідками згаданого вище частотного аналізу середня команда має довжину 2,39 байт і проводить 0,63 читань і записів даних. Отже, на виконання середньої команди потрібний 1,83 шинного циклу. Шинний цикл (без тактів очікування) займає два такти. Таким чином, у i80286 пропускна спроможність шини достатня для підтримки 2,2 млн команд в секунду [ $8 \text{ МГц} / (1,83 * 2) = 2,2$ ].

При максимальній пропускній спроможності шини 2,2 млн команд в секунду блок підготовки команд повинен готувати команди щонайменше так само швидко, щоб використовувати швидкодію шини. Працюючи в режимі 1 байт/1 такт, блок підготовки команд може готувати середні команди (2,39 байта) з максимальною швидкістю 3,35 млн команд в секунду, що створює достатній запас швидкодії.

У мікропроцесора з високим рівнем конвеєризації існує можливість того, що часті передачі управління, після яких потрібний час для заповнення конвеєра, зведуть нанівець будь-який вигаш в продуктивності машини, визначуваний конвеєрною обробкою. Проте аналіз програм, написаних для мікропроцесора i8086, показує, що лише одна з семи команд проводить передачу управління, тобто близько 15%. Значить, конвеєризація, насправді, підвищує продуктивність машини, якщо заповнення конвеєра після передачі управління не дуже затягується. При проектуванні i80286 особлива увага була приділена скороченню цієї затримки. Коли черга код в шинному блоці порожня, а це трапляється після кожної передачі управління, нові байти команди проходять через шинний блок, не затримуючись. Так само команди, які готує блок підготовки команд, проходять через порожню черга команд до виконавчого блоку.

Порівняння мікропроцесорів i80286 з i8086 показує підвищення продуктивності першого в 1,8-6,3 разу при оцінці часу виконання команд по динамічній частотній вибірці, продуктивність i80286 складає 1,88 млн операцій в секунду. Хоча i80286 спроектований так, що забезпечує високу продуктивність без застосування дорогої пам'яті з високою швидкодією, все-таки можливі ситуації, в яких потрібні стани очікування. Але навіть якщо, їх потрібний три, все одно продуктивність i80286 перевищує 1 млн операцій в секунду.

## **Опис роботи мікропроцесора i80286**

Інтерфейс локальної шини підключає мікропроцесор i80286 до компонентів введення-виводу і пам'яті. Інтерфейс має 24-розрядну адресну шину, 16-розрядну шину даних і 8 сигналів стану і контролю.

Максимум 16 Мбайт фізичної пам'яті можуть бути адресовані в захищеному режимі. У реальному адресному режимі може бути адресоване 1

Мбайт. Звернення до пам'яті може виконуватися словом або байтом. Слово складається з двох послідовних байтів і адресується з байта, розташованого за молодшою адресою. Передача байтів може відбуватися по будь-якій половині 16-розрядній шині даних.

Процесор i80286 використовує подвійну частоту системного синхросигналу CLK для контролю синхронізації шини. Всі сигнали на локальній шині синхронізуються цим сигналом. Центральний процесор ділить сигнал CLK на 2 для отримання внутрішнього синхросигналу процесора, який і визначає стан шини. Кожен цикл процесорного синхросигналу складається з двох циклів сигналу CLK, званих фазою 1 і фазою 2.

Процесор i80286 використовує інтерфейс локальної шини з конвеєрною синхронізацією, що забезпечує максимально можливий час для доступу до даним. Конвеєрна синхронізація дозволяє ініціалізувати нові операції шини через кожних 2 процесорних циклу, тоді як виконання кожної окремої операції шини займає три процесорні цикли.

Синхронізація адресних виходів конвейеризована таким чином, що адреса наступної операції шини стає доступною під час поточної операції шини. Або, іншими словами, перший такт наступної операції шини перекривається з останнім тактом поточної операції.

## Заняття 7-8

### Призначення і характеристики команд передачі даних і двійкової арифметики

#### Цілі заняття

- Вивчити функції, які реалізуються командами передачі даних і двійкової арифметики.
- Зрозуміти і засвоїти відмінності між командами однієї і тієї ж групи.
- Навчитися визначати, які команди процесора найдоцільніше використовувати при реалізації конкретних алгоритмів мовами низького рівня.
- Засвоїти основні особливості команд передачі даних і двійкової арифметики.
- Навчитися використовувати команди передачі даних і двійкової арифметики в прикладному і системному програмуванні.

#### Команди передачі даних

Дані команди виконують передачу байт, слів, подвійних слів або збільшених учетверо слів між пам'яттю і регістрами процесора. Ці команди можна об'єднати в наступні підгрупи.

1. Команди передачі даних загального призначення.
2. Стекові команди.
3. Команди перетворення типів даних.

#### 1. Команди передачі даних загального призначення

**MOV** (Move) передає байт, слово або подвійне слово з джерела в приймач операнда. Команда MOV забезпечує передачу даних наступними трактами:

- у регістр з пам'яті;
- у пам'ять з регістра;
- між регістрами процесора;
- безпосередній операнд в регістр;
- безпосередній операнд в пам'ять.

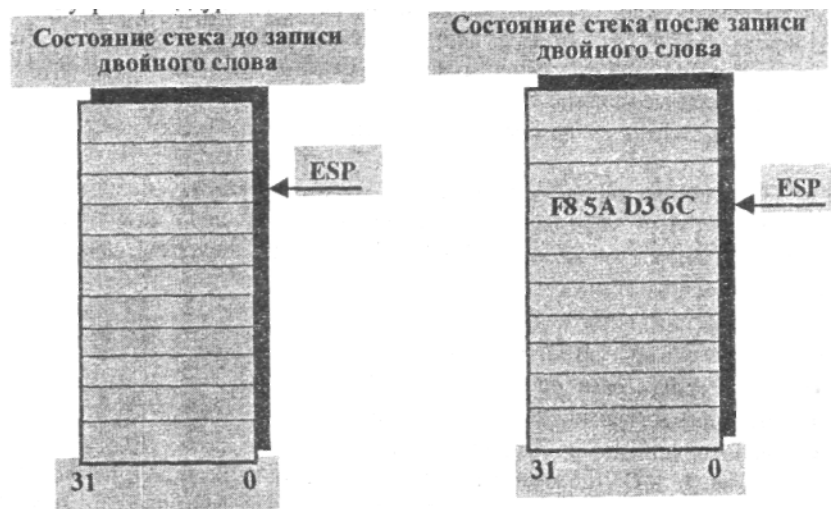
Команда MOV не може передавати дані з пам'яті в пам'ять або з сегментного регістра в сегментний регістр. Передачі пам'ять - пам'ять можуть бути виконані за допомогою команди передачі рядка MOVS. Спеціальна форма команди MOV забезпечує передачу даних між регістрами AL, AX, EAX і елементом пам'яті, що адресується 32-бітовим зсувом (offset), вказаним в команді. Ця форма команди не вирішує заміни сегменту, використання індексного регістра або масштабування. Ця команда на один байт коротша, ніж команда MOV загального призначення. Подібне ж кодування використовується для завантаження 8-, 16- або 32-бітового безпосереднього операнда в один з регістрів загального призначення.

**XCHG** (Exchange) обмінює два операнди. Ця команда замінює три команди MOV. Вона не вимагає тимчасового запам'ятовування одного з операндів при обміні. Команда XCHG особливо корисна для підтримки

семафорних механізмів або подібних структур даних для синхронізації процесів.

## 2. Стекові команди

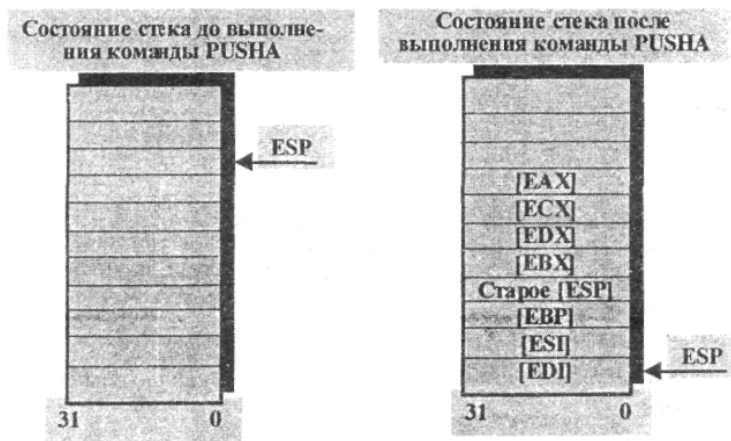
**PUSH (Push)** зменшує покажчик стека ESP і потім передає операнд-джерело до вершини стека (мал. 7.1). Команда PUSH часто використовується для передачі параметрів в стек до виклику процедури. Всередині процедури вона може бути використана для резервування простору в стеку для тимчасових змінних. Команда PUSH оперує з операндами, що зберігаються в регістрах, включаючи сегментні, пам'яті і з безпосередніми операндами. Спеціальна форма команди PUSH є в процесорі для збереження вмісту 32-бітового регістра загального призначення в стеку, вона коротше на один байт, чим команда PUSH загального вигляду.



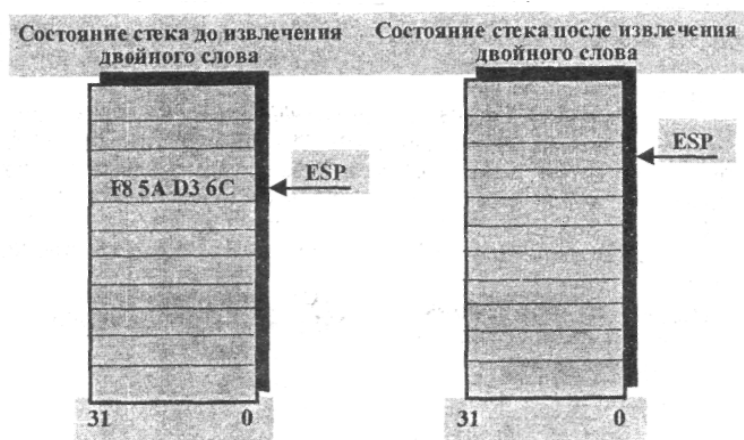
Мал. 7.1. Команда PUSH.

**PUSHA (Push All Registers)** зберігає вміст восьми регістрів загального призначення в стеку (мал. 7.2). Ця команда спрощує виклик процедури за рахунок зменшення числа команд, що звертаються до стека. Процесор запам'ятовує в стеку регістри в наступному порядку: EAX, ECX, EDX, EBX, ESP, EBP, ESI і EDI. При цьому завантажене в стек значення ESP відповідає ESP до завантаження регістра EAX, тобто старому POP (Pop) передає слово або подвійне слово з поточної вершини стека в регістр з подальшим інкрементом регістра ESP (мал. 7.3). Є спеціальна форма команди для передачі подвійного слова із стека в регістр загального призначення, яка на один байт коротше за команду POP.

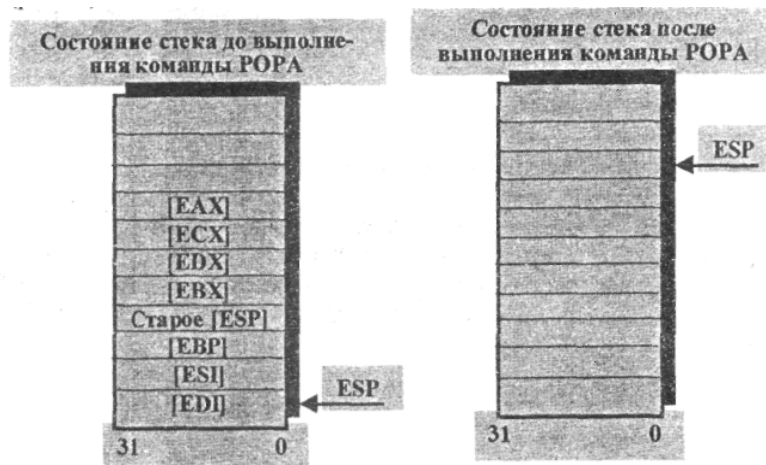
**POPA (Pop AH Registers)** передає дані, завантажені в стек командою PUSHA, із стека в регістри процесора, за винятком регістра ESP. Регістр ESP відновлюється за допомогою читання стека (мал. 7.4).



Мал. 7.2. Команда PUSHА.



Мал. 7.3. Команда POP.



Мал. 7.4. Команда POPА.

### 3. Команды перетворення типів даних

Дана група команд перетворить байти в слова, слова в подвійні слова і подвійні слова в збільшені учетверо слова (64 бита). Ці команди особливо корисні для перетворення знакових цілих, оскільки вони автоматично копіюють знаковий біт перетворюваного даного в старші біти формованого даного

більшій розрядності. Цей вид перетворення називається знаковим розширенням.

У процесорі є наступні два види команд перетворення:

- команди CWD, CDQ, CBW і CWDE, які оперують даними тільки в регістрі EAX;
- команди MOVSX і MOVZX, які допускають зберігання одного операнда в регістрах загального призначення, а другого — в регістрі або пам'яті.

**CWD (Convert Word to Doubleword) і CDQ (Convert Doubleword to Quad-Word)** подвоюють розрядність операнда-джерела. Команда перетворення слова в подвійне слово CWD копіює знаковий біт (15 битий) слова з регістра AX в кожен біт регістра DX. Команда перетворення подвійного слова в збільшене учетверо CDQ копіює знаковий біт (31 битий) подвійного слова в регістрі EAX в кожен біт регістра EDX. Команда CWD може бути використана для формування діленого перед виконанням операції ділення із слова в подвійне слово, а команда CDQ — для формування збільшеного учетверо діленого з подвійного слова перед початком ділення.

**CBW (Convert Byte to Word)** копіює знаковий біт (7 битий) байта з регістра AL в біти старшого байта регістра AX.

**CWDE (Convert Word to Doubleword Extended)** копіює знаковий біт (15 битий) слова, що зберігається в регістрі AX, в старших двох байтах регістра EAX.

**MOVSX (Move with Sign eXtention)** розширює 8-бітове дане в 16-бітове або 8-, 16-бітове дане в 32-бітове за допомогою заповнення знаком порожніх розрядів.

**MOVZX (Move with Zero eXtention)** розширює 8-бітове дане в 16-бітове або 8-, 16-бітове дане в 32-бітове з установкою в нуль вільних розрядів.

#### **Команди двійкової арифметики**

Команди даної групи виконують складання, віднімання, множення, ділення, а також інкремент, декремент, порівняння, зміна знаку над даними, представленими в двійковому коді, при цьому дані можуть бути як знаковими, так і без знаковими.

Команди обробки двійкових даних також використовуються при виконанні арифметичних операцій над десятковими даними, що використовують **BСD-кодирование (Binary Coded Decimal)**. Операнди-джерела можуть бути безпосередніми даними, зберігатися в регістрах або пам'яті. Результати операцій можуть формуватися в регістрах процесора або пам'яті. Модифікація пам'ять - пам'ять не підтримується системою команд. Основні арифметичні команди мають спеціальні формати при використанні безпосередніх операндів як джерела, а регістри AL, AX, EAX як приймачі результатів операцій. Дані команд коротше на один байт в порівнянні з арифметичними командами загального призначення.

Арифметичні команди модифікують прапори ZF, CF, SF і OF для формування ознак результатів операцій. Вид команди для аналізу прапорців залежить від інтерпретації даних — знакових або без знакових. Прапор CF

відображає інформацію для без знакових даних, а SF і OF — для знакових. Прапор ZF використовується для обох випадків представлення даних.

#### 4. Команди складання і віднімання

**ADD (Add Integers)** замінює операнд-приймач результатом складання цілих, прапори OF, SF, ZF, AF, PF і CF формуються відповідно до результату операції.

**ADC (Add Integers with Carry)** аналогічна попередньою, але враховує також і значення прапорця перенесення CF. Команда ADC використовується для передачі перенесення при використанні, наприклад, 32-бітових команд ADD для складання збільшених учетверо операндів.

**INC (Increment)** додає 1 до операнда, прапор CF не змінюється. Однобайтний формат даної команди використовується, коли операнд зберігається в регістрі. Команда впливає на прапори OF, SF, ZF, AF і PF.

**SUB (Subtract Integers)** віднімає операнд-джерело з операнда-приймача і замінює операнд-приймач результатом операції. Якщо позика виникає, то прапор CF встановлюється в 1. Операнди можуть бути знаковими або без знаковими байтами, словами або подвійними словами/флаги OF, CF, ZF, AF, PF, CF формуються командою.

**SBB (Subtract Integers with Borrow)** аналогічна попередній команді, але враховує і значення позики в прапорці CF. Якщо прапор CF встановлений в 1, то з операнда джерела віднімається 1; інакше команда SBB; якщо CF = 0 аналогічна команді SUB. Команда впливає на прапора OF, SF, ZF, AF, PF і CF.

**DEC (Decrement)** віднімає 1 з операнда-джерела і не змінює прапор CF. Прапора OF, SF, ZF, AF, PF формуються командою.

Використання команди SUB з безпосередньою величиною 1 забезпечує виконання декремента з формуванням прапора CF. Однобайтна форма цієї команди доступна худа, коли операнд зберігається в регістрі загального призначення.

#### 5. Команди порівняння і зміни знаку

**CMR (Compare)** віднімає операнд-джерело з операнда-приймача. При цьому операнди не змінюються, а формуються прапори OF, SF, ZF, AF, PF, CF, які перевіряються командами передачі управління для визначення результату порівняння операндів.

**NEG (Negate)** віднімає знакове ціле з нуля. Підсумком виконання команди NEG є зміна знаку операнда, представленого в додатковому двійковому, коді, без зміни його значення. Прапори OF, SF, ZF, AF, PF, CF формуються командою.

#### 6. Команди множення

Процесор має окремі команди множення для знакових і без знакових операндів.

**MUL (Unsigned Integer Multiply)** виконує множення без знакового операнда-джерела і вмісту регістра AL, AX або EAX. Якщо операнд-джерело є байтом, то команда множить його на вміст регістра AL і формує дві подвійної

довжини в регістрах AH і AL. Якщо операнд-джерело — слово, то він множиться на вміст регістра AX, а 32-бітовий результат формується в DX і AX. Операнд-джерело, відповідний подвоєному слову, множиться на вміст регістра EAX і 64-бітовий твір запам'ятовується в регістрах EDX і EAX. Команда множення MUL встановлює прапори CF і OF в 1, якщо старша половина результату не рівна нулю, інакше CF і OF прирівнюються нулю. Значення прапорів SF, ZF, AF і PF не визначене.

**IMUL (Signed Integer Multiply)** виконує операцію множення знакових цілих. Команда IMUL має наступні три форми.

1. Однооперандна форма команди, операндом може бути байт, слово або подвійне слово, що зберігаються в пам'яті або регістрі. Дана команда використовує операнди регістрів EAX і EDX або AX і DX за умовчанням, аналогічно команді MUL.

2. Двооперандна форма, коли один операнд зберігається в регістрі, а інший — в регістрі або пам'яті; результат операції формується в регістрі.

3. Трехоперандна форма, при якій два операнди використовуються як початкові, а третій - для результату.

## 7. Команди ділення

Процесор Pentium має роздільні команди для знакових і без знакових операндів. Для обох варіантів при діленні формується сигнал помилки (особливий випадок), якщо дільник рівний нулю або приватне дуже велике для представлення його в регістрах AL, AX, EAX.

**DIV (Unsigned Integer Divide)** виконує ділення цілого беззнакового операнда, що зберігається в регістрах AL, AX або EAX. Ділиме має розрядність в два рази більше розрядності дільника.

**IDIV (Signed Integer Divide)** виконує знакове ділення акумулятора на операнд-джерело і використовує ті ж регістри, що і команда DIV. При діленні знакового байта максимальне позитивне приватне рівне +127, а мінімальне негативне приватне -128. Залишок завжди має такий же знак, як дільник, і його величина менше дільника. Стани прапорів OF, SF, ZF, AF, PF і CF не визначені.

## Заняття 9-10

### Призначення і характеристики команд десятькової арифметики і логічних операцій

#### Цілі заняття

- Вивчити функції, які реалізуються командами корекції упакованих і не упакованих десятичових даних.
- Зрозуміти і засвоїти відмінності між командами булевих операцій.
- Навчитися визначати, які команди логічних операцій найдоцільніше використовувати при реалізації конкретних алгоритмів на мовах низького рівня.
- Засвоїти основні особливості команд логічних операцій.
- Навчитися представляти дії, тестування, що реалізуються командами, і модифікації битий, сканування битий.
- Навчитися використовувати команди зрушення, установки байта по умові і тестування в прикладному і системному програмуванні.

#### Команди десятичової арифметики

Десятькова обробка даних виконується за допомогою використання команд арифметичних операцій над двійковими даними і команд десятичової корекції для формування правильних результатів в упакованому або не упакованому форматах. Ці команди виконують операції тільки над вмістом регістрів AL і AH, більшість з них використовують прапор **AF**.

**1. Команди корекції упакованих десятичових даних DAA (Decimal Adjust after Addition)** коректує результат складання двох упакованих десятичових операндів в регістрі AL. Команда DAA повинна виконуватися за командою складання двох упакованих десятичових чисел (одна десятичова цифра займає одну тетраду). Прапор CF встановлюється в 1, якщо зустрічається перекіс, прапори SF, ZF, AF, PF і CF формуються, а значення прапора OF є невизначеним.

**DAS (Decimal Adjust after Subtraction)** коректує результат віднімання упакованих десятичових чисел в регістрі AL. Прапор CF встановлюється, якщо має місце заїм, значення прапорів SF, ZF, AF, PF і CF формуються відповідно до результату, а значення прапора OF не визначається.

#### 2. Команди корекції неупакованих десятичових даних

**AAA (ASCII Adjust after Addition)** змінюють вміст регістра AL в не упаковане десятичове число і зводить до нуля старші 4 бита. Команда AAA повинна виконуватися після команди складання не упакованих десятичових чисел, одне з яких зберігається в регістрі AL. Прапор CF встановлюється і вміст регістра AH збільшується, якщо зустрічається перенесення. Прапори AF і CF формуються, а значення прапорів OF, SF, ZF, і PF є невизначеним

**AAS (ASCII Adjust after Subtraction)** перетворить вміст регістра AL в дійсне не упаковане десятичове число і очищає старші 4 бита. Команда корекції AAS слідує за командою віднімання, прапор CF встановлюється і вміст регістра

АН зменшується на 1, якщо зустрічається позика. Прапори AF і CF формуються, а значення прапорів OF, SF, ZF і PF не визначаються.

**AAM (ASCII Adjust after Multiplication)** коректує результат множення двох не упакованих десяткових чисел і ця команда повинна виконуватися за командою множення. Старша цифра розміщується в регістрі АН, а молодша — в регістрі AL. Прапори SF, ZF, PF формуються, а стан прапорів AF, OF і CF є невизначеним.

**AAD (ASCII Adjust before Division)** модифікує дільник в регістрах АН і AL для підготовки операції ділення двох не упакованих десяткових операндів, так що приватне буде в не упакованій формі. Регістр АН повинен зберігати старшу, а регістр AL — молодшу цифру числа. Ця команда коректує значення результату і формує його в регістрі AL, а регістр АН зводиться до нуля. Прапори SF, ZF, PF формуються, а стани прапорів AF, OF і CF невизначені.

### Команди логічних операцій

Ці команди є двооперандними, операнд-джерело може бути безпосереднім даним, вмістом регістра або пам'яті. Приймачем результату може бути регістр або елемент пам'яті, якщо операнд-джерело не зберігається, в пам'яті. Логічні команди модифікують значення прапорів. Команди логічних операцій включають наступні:

- команди булевих операцій;
- команди тестування і модифікації битий;
- команди сканування битий;
- команди зрушення;
- команди установки байта по умові.

#### 1. Команди булевих операцій

Дані команди забезпечують виконання операцій кон'юнкції (**AND**), диз'юнкції (**OR**), що виключає АБО (**XOR**) і інвертування битий (**NOT**).

Команда **NOT** є однооперандний, операнд може зберігатися в регістрі або елементі пам'яті. Команда **NOT** не впливає на прапори і інвертує біти у формі доповнення до одиниці.

Команди **AND**, **OR** і **XOR** зводять до нуля прапори **OF** і **CF**, не змінюють прапор **AF** і модифікують прапори **SF**, **ZF**, **PF**.

#### 2. Команди тестування і модифікації битий

Дані команди оперують над одним бітом, що зберігається в пам'яті або регістрах. Команди **BT**, **BTS**, **BTR**, **BTC** вибирають з регістра або елемента пам'яті значення біта і завантажують його в прапор **CF**. Номер біта визначається безпосереднім операндом в команді або вмістом регістра.

#### 3. Команди сканування битий

Ці команди сканують слово або подвійне слово в регістрі або елементі пам'яті для пошуку першої 1, починаючи з молодшого або старшого розряду і запам'ятовують номер першого біта, рівного 1, в регістрі-приймачі. Прапор **ZF**

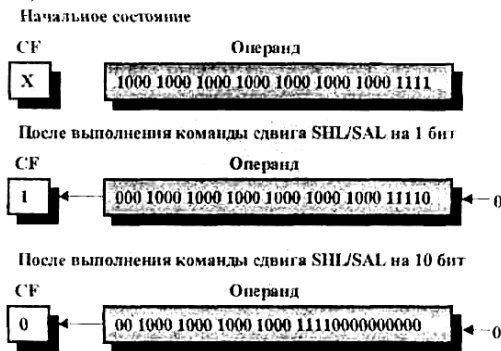
встановлюється в 1, якщо сканований операнд рівний Про і зводиться до нуля, якщо одиничний біт знайдений в даному операнді. У першому випадку значення регістра-приймача залишається невизначеним. Стани прапорів OF, SF, AF, PF і CF є невизначеними.

#### 4. Команди зрушення

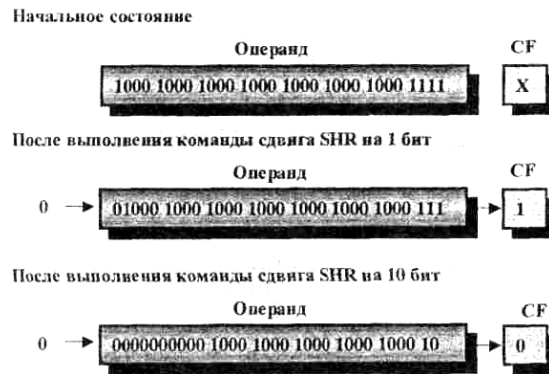
Дані команди виконують арифметичні або логічні зрушення байтів, слів або подвійних слів. Арифметичне зрушення управо копіює знаковий біт в попередній розряд, а логічне зрушення управо заповнює його нулем. Арифметичне зрушення є швидким шляхом виконання операції ділення операнда на два. Логічне зрушення управо ділить незначає ціле або позитивне ціле, але знакове негативне ціле при діленні приводить до втрати знакового біта.

**SAL (Shift Arithmetic Left)** зрушує байт, слово або подвійне слово вліво на один або більш битий, що визначається безпосереднім операндом або регістром CL (мал. 9.1). Порожні позиції біт зводиться до нуля..

**SHL (Shift Logical Left)** аналогічна команді **SAL**. **SHR (Shift Logical Right)** зрушує байт, слово або подвійне слово управо на один або більш битий (мал. 9.2).

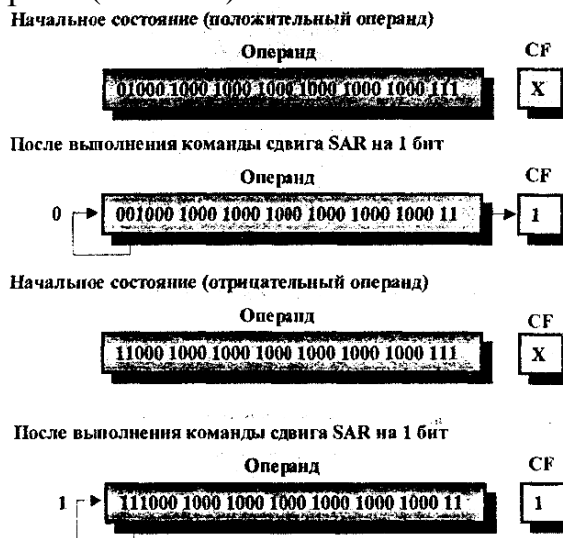


Мал. 9.1 Команда SHL/SAL.



Мал. 9.2 Команда SHR.

**SAR (Shift Arithmetic Right)** зрушує байт, слово або подвійне слово на один або більш битий управо (мал. 9.3).



Мал. 9.3. Команда SAR.

## Заняття 11-12

### Призначення і характеристики команд передачі управління і обробки рядків

#### Цілі заняття

- Вивчити функції, які реалізуються командами передачі управління і обробки рядків.
- Зрозуміти і засвоїти відмінності між командами повернення, умовної передачі управління, управління циклом.
- Навчитися визначати, які команди програмних переривань найдоцільніше використовувати при реалізації конкретних алгоритмів на мовах низького рівня.
- Засвоїти основні особливості команд обробки рядків.
- Навчитися представляти дії, повторення, що реалізуються командами.
- Навчитися використовувати команди передачі управління і обробки рядків в прикладному і системному програмуванні.

#### Команди передачі управління

Система команд процесорів сімейства i80x86 включає команди безумовної і умовної передачі управління. Умовні передачі використовують певні комбінації станів прапорів процесора.

#### 1. Команди безумовної передачі управління

Для команд даної групи передача управління може бути усередині сегменту (**near transfer** — **близька передача**) або в інший сегмент (**far transfer** — **дальня передача**). Різновиди команд передачі управління в інший сегмент обговорюються в окремому розділі цього розділу. Якщо модель пам'яті в системі не використовує сегментацію, то для прикладного програміста команди дальньої передачі управління будуть незатребуваними.

**JMP (Jump)** передає управління за адресою, вказаною в команді, реєстрі або елементі пам'яті. Розміщення адреси переходу визначає його інтерпретацію командою як відносну або абсолютну адресу.

При відносній адресі зсув зберігається в команді (**displacement**) і є знаковим байтом або подвійним словом. Адреса переходу формується шляхом складання зсуву і вмісту покажчика команд EIP.

При використанні абсолютної адреси 32-бітовий зсув може завантажуватися в EIP з реєстра процесора або з елемента пам'яті, визначуваної відповідно до режиму адресації.

**CALL (Call Procedure)** передає управління процедурі і зберігає адресу команди, наступної за командою CALL, для повернення. Команда CALL зберігає вміст реєстра EIP в стеку.

## 1.1. Команди повернення

**RET (Return from procedure)** завершує процедуру і передає виконання команді, наступній за командою CALL, що викликала процедуру. Команда RET відновлює вміст регістра EIP, який запам'ятав в стеку до виклику процедури.

Команди RET мають необов'язковий (optional) безпосередній операнд. Коли він присутній, ця константа додається до регістра EIP, що відповідає ефекту видалення всіх параметрів, завантажених в стек перед викликом процедури.

**IRET (Return from Interrupt)** повертає управління в перервану процедуру. Команда IRET відрізняється від команди RET тим, що вона відновлює регістр прапорів із стека. Вміст регістра прапорів запам'ятовується в стеку, коли зустрічається переривання.

## 2. Команди умовної передачі управління

### 2.1. Команди управління циклом

Команди управління циклом є командами умовної передачі, що використовують вміст регістра ECX як лічильник кількості повторень циклу. Команди управління циклом зменшують вміст регістра ECX при кожному виконанні циклу і завершують циклічну ділянку програми досягнувши нуля. Чотири з п'яти команд управління циклом використовують прапор ZF як умову для завершення циклу до досягнення лічильником нуля.

**LOOP (Loop While ECX Not Zero)** є командою умовної передачі управління, яка зменшує вміст ECX до тестування умови завершення циклу. Якщо вміст регістра ECX не рівний нулю, то програма переходить за адресою, вказаною в команді. Команда LOOP забезпечує виконання блоку команд програм, який повторюється, поки свідчення (вміст) лічильника не буде рівне нулю. Якщо вміст регістра ECX дорівнює 0, коли команда LOOP перший раз викликається, то лічильнику привласнюється код 0FFFFFFFH і цикл виконується 232 рази.

**LOOPE (Loop While Equal)** і **LOOPZ (Loop While Zero)** є синонімами. Ці команди є командами умовного переходу, що зменшують вміст регістра ECX до перевірки умови завершення циклу. Якщо вміст ECX рівний нулю і прапор ZF встановлений, то програма передає управління за адресою, вказаною в команді. Коли вміст ECX стає рівним нулю, або прапор ZF зводиться до нуля, тоді виконання передається команді, наступній за командою LOOPE/LOOPZ.

**LOOPNE (Loop While Not Equal)** і **LOOPNZ (Loop While Not Zero)** є синонімами. Ці команди є командами умовного переходу, які зменшують вміст ECX до перевірки умови завершення циклу. Якщо вміст ECX не рівний нулю і прапор ZF скинутий, то програма передає управління приймачу, вказаному в команді.

## 3. Програмні переривання

Команди INT, INTO і BOUND дозволяють програмістові специфікувати передачу управління обробникові особливих випадків або переривань.

**INTn (Software Interrupt)** викликає обробник переривання, визначуваний вектором переривання, закодованим в команді. Команда переривання INT може визначати будь-який тип переривання. Ця команда використовується для підтримки безлічі типів програмних переривань або тестування програм обробки переривань.

**INTO (Interrupt on Overflow)** викликає обробник особливого випадку по переповнюванню, якщо прапор OF встановлений. Прапор OF встановлюється командами арифметичних і логічних операцій і обробки рядків.

**BOUND**, команда контролю знаходження індексу масиву в заданих межах. Команда BOUND порівнює знакову величину в регістрі загального призначення з верхньою і нижньою межею.

### Команди обробки рядків

Дані команди маніпулюють великими структурами даних в пам'яті, такими, як рядки алфавітно-цифрових символів і використовуються спільно для організації програмних циклів з префіксами повторення. Команди обробки рядків включають наступні:

**MOVS** — передача рядка (Move String);

**CMPS** — порівняння рядка (Compare String);

**SCAS** — сканування рядка (Scan String);

**LODS** — завантаження рядка (Load String);

**STOS** — запам'ятовування рядка (Store String).

### Команди обробки рядка

**MOVS (Move String)** передає елемент рядка, що адресується регістром ESI, у елемент пам'яті, що адресується регістром EDI. Команда **MOVSB** передає байти, **MOVSW** — слова, а **MOVSD** — подвійні слова. Команда **MOVS** при використанні префікса **REP**, забезпечує передачу блоку даних типу пам'ять — пам'ять. Для реалізації даної операції необхідно ініціалізувати регістри ECX, ESI, EDI. Регістр ECX визначає кількість елементів в блоці.

**CMPS (Compare Strings)** забезпечує порівняння рядків за допомогою віднімання елемента рядка-приймача з елемента рядка-джерела і оновлює значення прапорів AF, SF, PF, CF, OF. Елементи рядків при цьому не змінюються.

**LODS (Load String)** забезпечує завантаження рядка шляхом приміщення елемента рядка-джерела, що адресується регістром ESI, в регістр EAX для рядків з подвійних слів, в регістр AX для рядків із слів і в регістр AL для рядків з байт.

**STOS (Store String)** забезпечує запам'ятовування рядка за допомогою пересилки елемента рядка з регістра EAX, AX або AL в рядок, що адресується регістром EDI. Ця команда зазвичай використовується в циклі, в якому вона записує в пам'ять результат обробки елемента рядка, що завантажується з пам'яті командою **LODS**.

## Заняття 13-14

### Призначення і характеристики команд підтримки мов з блоковою структурою, управління прапорами, виконання операцій з сегментними регістрами, спеціальних команд

#### Цілі заняття

- Вивчити функції, які реалізуються командами підтримки мов з блоковою структурою.
- Зрозуміти і засвоїти відмінності між командами ENTER і LEAVE, які спрощують вхід в процедуру і вихід з процедури в кодї, що формується компілятором.
- Засвоїти основні особливості доступу змінним для вкладених процедур.
- Навчитися визначати, які команди управління прапорами доцільно використовувати при реалізації конкретних алгоритмів на мовах низького рівня.
- Навчитися представляти дії, виконання операцій, що реалізуються командами, з сегментними регістрами.
- Навчитися використовувати команди обчислення адреси, перетворення перестановки байт, обміну і складання, порівняти і обміняти, ідентифікації процесора в прикладному і системному програмуванні.

#### Команди для мов з блоковою структурою

Дані команди забезпечують підтримку на рівні машинно-орієнтованої мови для блок-структурованих мов високого рівня, таких, як Сі і Паскаль. Вони включають команди BOUND, ENTER і LEAVE, які спрощують вхід в процедуру і вихід з процедури в кодї, що формується компілятором. Вони підтримують структуру покажчиків і локальних змінних в стеку, званому кадром стека.

Зазвичай команди цієї групи генеруються компіляторами, а програмісти на асемблері їх не застосовують. Такі команди спрощують розробку компіляторів і забезпечують апаратну підтримку деяких загальних операцій.

**BOUND (check array BOUNDaries).** Застосовується для контролю знаходження значення усередині або поза діапазоном. Якщо значення усередині діапазону, ніякої дії не робиться, інакше генерується переривання 5. Команда призначена для компіляторів з метою контролю індексу масиву.

Перший операнд (регістр) містить індекс масиву, а другою служить покажчиком пам'яті. У пам'яті знаходяться два слова: нижній і верхній індекси масиву, тобто мінімальне і максимальне значення індексу, а не адреси кінців масиву в пам'яті. Якщо індекс менше першого слова або більше другого, генерується переривання 5. Команда BOUND не впливає на прапорці і не показує, яка межа

Команда BOUND зручна для контролю меж масиву, але її реалізація заплутана. Простіше було встановити OF, коли порушена межа, а прапорцем SF вказати невірний індекс. Замість цього генерується переривання і необхідно поклопотатися про обробку переривання. Можна або повністю обробити

помилку, або встановити прапорець помилки, повернути управління програмі, а в ній передбачити перехід по прапорцю до потрібного обробника помилки.

**ENTER (Enter Procedure)** створює кадр стека, сумісний з правилами області дії блок-структурованих мов. У цих мовах процедура має доступ до її власних змінних і деякого числа інших змінних, визначених в програмі. Областю дії процедури є безліч змінних, до яких вона має доступ. Правила для області дії різні для окремих мов. Вони можуть ґрунтуватися на вкладенні процедур, діленні програми на окремо компільовані файли або використовувати інші форми модульності програм.

Команда **ENTER** має два операнди. Перший операнд визначає кількість байт, резервованих в стеку для динамічного зберігання в процедурі, що викликається. Динамічна пам'ять є областю для розміщення змінних, що створюються при виклику процедури, відомими також як автоматичні змінні. Другий операнд є рівнем лексичного вкладення (від 0 до 31) процедури. Рівень вкладення — це глибина процедури в ієрархії програми з блоковою структурою. Лексичний рівень не має певного взаємозв'язку з рівнем привілею захисту або з рівнем привілею введення - виводу.

Для того, щоб дозволити процедурі адресувати її індикатор, команда **ENTER** використовує регістр **EBP** як покажчик на перше подвійне слово в індикаторі. Оскільки стеки ростуть вниз, то це є фактично подвійним словом з вищою адресою в індикаторі. Команди обробки даних, що визначають регістр **EBP** як базовий, автоматично адресують осередки усередині стекового лічильника замість сегменту даних.

**LEAVE (Leave Procedure)** по дії протилежна розглянутій команді **ENTER**. Команда **LEAVE** не використовує операнди, вона копіює вміст регістра **EBP** в регістр **ESP** для того, щоб звільнити весь стековий простір, розподілений під процедуру. Потім команда **LEAVE** відновлює старе значення регістра **EBP** із стека. При цьому одночасно відновлюється первинний вміст регістра **ESP**.

Подальша команда повернення **RET** може видалити будь-які параметри і адресу повернення, поміщені в область стека зухвалою програмою для використання процедурою.

### Команди управління прапорами

Команди **LAHF** і **SAHF** маніпулюють п'ятьма прапорами, які формуються в основному арифметичними і логічними командами.

**LAHF (Load AH from Flags)** копіює прапори **SF**, **ZF**, **AF**, **PF**, **CF** в регістр **AH**, а вміст решти біт при цьому не змінюється.

**SAHF (Store AH into Flags)** копіює певні біти регістра **AH** в **SF**, **ZF**, **AF**, **PF**, **CF**.

Команди **PUSHF** і **POPF** не тільки корисні для запам'ятовування прапорів в пам'яті, а також відновлення, але і використовуються при виклику процедур і повернення з них.

**PUSHF (Push Flags)** зберігає молодше слово регістра прапорів в стеку. Команда **PUSHFD** зберігає весь регістр прапорів в стеку, при цьому прапори **RF** і **VM** читаються як обнулені.

**POPF (Pop Flags)** відновлює слово із стека в регістрі прапорів. Якщо рівень привілеїв поточного програмного сегменту дорівнює 0 (самий привілейований), то биті **IOPL** також змінюються. Якщо рівень привілеїв введення-виводу (**IOPL**) дорівнює 0, то прапор **IF** теж змінюється. Команда **POPFD** відновлює подвійне слово в регістрі прапорів, і вона може змінювати стан прапорів **AC** (битий 18) і **ID** (битий 21) так само, як і всіх біт, змінних командою **POPF**.

### 1. Команди передачі сегментних регістрів

Ряд форм команд **MOV**, **POP** і **PUSH** також використовується для завантаження і запам'ятовування сегментних регістрів. Ці форми відповідають командам роботи з регістрами загального призначення, але з тим лише відмінністю, що одним з операндів є вміст сегментного регістра. Проте команда **MOV** не може використовуватися для прямої передачі вмісту одного сегментного регістра в іншій.

### 2. Команди дальньої передачі управління

Дані команди передають управління іншому сегменту шляхом завантаження в регістр **CS** нового значення. Приймач визначається дальнім покажчиком, який є сукупністю 16-бітового селектора сегменту і 32-бітового зсуву в сегменті. Дальній покажчик може бути безпосереднім операндом або операндом пам'яті.

**Far Call**, ця команда міжсегментної передачі управління завантажує значення з регістрів **Eip** і **CS** в стек.

**Far RET**, міжсегментна команда повернення відновлює вміст регістрів **CS** і **EIP** із стека.

### 3. Команди завантаження покажчиків даних

Дані команди завантажують дальній покажчик в регістри процесора. Дальній покажчик складається з 16-бітового селектора сегменту, який завантажується в сегментний регістр, а 32-бітовий зсув в сегменті завантажується в регістр загального призначення.

**LDS (Load Pointer Using DS)** копіює дальній покажчик з операнда-джерела в регістр **DS** і регістр загального призначення. Операнд-джерело повинен бути операндом пам'яті, а операнд-приймач — регістром загального призначення. Наприклад, команда **LDS ESI, STRING\_X** завантажує регістр **DS** селектором сегменту для сегменту, що адресується змінною **STRING\_X** і завантажує зсув усередині сегменту **STRINGX** в регістр **ESI**. Визначення регістра **ESI**, як операнда-приймача, є зручним шляхом для обробки рядка, коли рядок-джерело не знаходиться в поточному сегменті даних.

Команди **LES, LFS, LGS, LSS** аналогічні команді **LDS**. Команда **LSS** особливо важлива, оскільки вона дозволяє двом регістрам, які ідентифікують стек (регістри **SS** і **ESP**), виконати дію протягом однієї безперервної операції.

### Спеціальні команди

Нижче дані команди не потрапляють в яку-небудь з розглянутих груп, але є не менш важливими.

Команди **BSWAP, XADD** і **CMPXCHG** відсутні в процесорі **Intel386**. Команди **CMPXCHGB8** і **CPUID** введені в процесор **Pentium** і підвищують функціональні можливості процесора в порівнянні з попередніми процесорами **Intel 486** і **Intel 386**.

#### 1. Команди обчислення адреси

**LEA (Load Effective Address)** завантажує 32-бітовий зсув операнда-джерела в пам'яті (а не його вміст) в операнд-приймач. Операнд-джерело повинен бути в пам'яті, а як приймач операнда повинен бути регістр загального призначення. Дана команда особливо корисна при ініціалізації регістрів **ESI** або **EDI** перед виконанням команд обробки рядків або ініціалізації регістра **EBX** перед виконанням команди **XLAT**. Команда **LEA** може виконати будь-яке індексування або масштабування, які потрібні.

#### 2. Команда немає операції

**NOP (No Operation)** збільшує вміст регістра **EIP** для формування адреси наступної команди і не викликає яких-небудь інших дій.

#### 3. Команда перетворення

**XLATB (Translate)** замінює вміст регістра **AL** на байт, завантажений з таблиці трансляції в пам'яті. Вміст регістра **AL** інтерпретується як беззнаковий індекс в цій таблиці, а вміст регістра **EBX** використовується як базова адреса. Команда **XLAT** виконує те ж саме, але вона отримує байт з пам'яті.

#### 4. Команда перестановки байт

**BSWAP (Byte Swap)** міняє порядок байт в 32-бітовому операнді регістра. Бити 7-0 обмінюються з бітами 31 - 24, а бити 15-8 обмінюються з бітами 23-16. Ця команда ефективна при перетворенні форматів даних «big — endian» і «little — endian». Виконання даної команди двічі підряд залишає в регістрі те ж саме значення. Команда **BSWAP** також може прискорити виконання команд десяткової арифметики за допомогою роботи над чотирма цифрами одночасно. Ця команда також прискорює виконання операцій десяткової арифметики шляхом одночасної обробки чотирьох десяткових цифр.

#### 5. Команда обміну і складання

**XADD (Exchange and ADD)** використовує два операнди: операнд-джерело в регістрі і операнд-приймач в регістрі або пам'яті. Операнд-джерело замінюється операндом-приймачем, а операнд-приймач замінюється сумою операндів джерела і приймача.

## 6. Команди порівняти і обміняти

**CMY1PXCNG (Compare and Exchange)** використовує три операнди: операнд-джерело в регістрі, операнд-приймач в регістрі або пам'яті і акумулятор (AL, AX або EAX, залежно від розміру операнда). Якщо значення в операнді-приймачі і акумуляторі рівні, операнд-приймач замінюється операндом-джерелом. Інакше первинне значення операнда-джерела завантажується в акумулятор. Прапори відображають результат, який був би отриманий шляхом віднімання операнда-приймача з акумулятора. Прапор ZF встановлюється в 1, якщо значення операнда-приймача і акумулятора рівні, інакше ZF обнуляється.

**CMPXCHG8B (Compare and Exchange 8 bytes)** використовує три операнди: операнд-приймач в пам'яті, 64-бітову величину в EDX:EAX і 64-бітовий величині в Ecx:evx. Команда CMPXCHG8B порівнює 64-бітовий операнд в EDX:EAX з приймачем. Якщо вони рівні, то 64-бітовий операнд в Ecx:evx запам'ятовується в приймачі. Якщо EDX.EAX і приймач не рівні, то операнд-приймач завантажується в EDX:EAX. Прапор ZF встановлюється в 1, якщо значення операндів в приймачі і EDX.EAX рівні; інакше прапор ZF обнуляється, прапори CF, PF, AF, SF, OF не встановлюються. Команда CMPXCHG8B може бути об'єднана з командою LOCK для виконання всіх шинних циклів в одній безперервній операції.

## 7. Команда ідентифікації процесора

**CPUID** забезпечує інформацію програмному забезпеченню про постачальника і модель процесора. При завантаженні нуля в регістр EAX і подальшому виконанні команди ідентифікації процесора CPUID регістри ECX, EDX, EBX міститимуть ідентифікацію постачальника.