

Розділ 10

Організація пам'яті

В комп'ютері використовується декілька різних типів пам'яті. Це пов'язано з тим, що неможливо одним типом пам'яті вирішити всі завдання, які стоять перед нею, в першу чергу забезпечити велику ємність та високу швидкодію. Разом з тим, постає завдання забезпечення ефективної взаємодії всіх типів пам'яті, щоб система пам'яті в цілому задовольняла всі вимоги з боку інших вузлів комп'ютера. Як буде показано в даному розділі, це можливо здійснити завдяки використанню при побудові системи пам'яті підходу, відомого як принцип ієрархічної організації пам'яті.

Відповідно до цього принципу пам'ять комп'ютера будується на основі пристройів пам'яті різних типів, які, залежно від характеристик, належать до певного рівня ієрархії. Пам'ять нижчого рівня має меншу ємність, швидша і має більшу вартість в перерахунку на біт, ніж пам'ять вищого рівня. Рівні ієрархії взаємозв'язані: всі дані на одному рівні можуть бути також знайдені на вищому рівні, і всі дані на цьому вищому рівні можуть бути знайдені на наступному вищому рівні і т. д. З рухом вверх по ієрархічній структурі зменшується співвідношення вартість/біт, зростає ємність та час доступу. Однак завдяки принципу локальності за зверненням з рухом вверх по ієрархічній структурі зменшується частота звернення до пам'яті з боку центрального процесора, що веде до зменшення загальної вартості при заданому рівні продуктивності.

Принципи забезпечення ефективної взаємодії між рівнями ієрархії пам'яті розглядаються в даному розділі.

10.1. Ієрархічна організація пам'яті комп'ютера

10.1.1. Різниця між продуктивністю процесора та пам'яті

Одним із вузьких місць комп'ютерів з архітектурою Джона фон Неймана є розрив в продуктивності процесора та пам'яті, причому цей розрив неухильно збільшується. Так, продуктивність процесора зростає вдвічі приблизно кожні 1,5 роки. В табл. 10.1 наведено ріст з роками тактової частоти роботи процесора на прикладі процесорів фірми Intel.

Таблиця 10.1

Роки	1980	1985	1990	1995	2000	2005	2005-1980
Тип процесора	8080	286	386	Pentium	P-III	P-IV	
Тактова частота (МГц)	1	6	20	150	750	3800	3800 разів
Час одного такту (нс)	1 000	166	50	6	1.6	0.26	3800 разів

Разом з тим, для пам'яті приріст швидкодії не є таким високим, як у процесорів. В табл. 10.2 наведено зміну з роками ціни зберігання одного МБ інформації, часу доступу до статичної (SRAM) і динамічної (DRAM) напівпровідникової пам'яті та дискової пам'яті, а також ємність основної та зовнішньої дискової пам'яті.

Таблиця 10.2

	Роки	1980	1985	1990	1995	2000	2005	Ріст протягом 2005-1980
RAM	Характеристики							
	Ціна (\$/МБ)	19200	2900	320	256	100	50	380 разів
DRAM	Час доступу (нс)	300	150	35	15	2	0.5	300 разів
	Ціна (\$/МБ)	8000	880	100	30	1	0.1	80 000 разів
DISC	Час доступу (нс)	375	200	100	70	50	30	12 разів
	Типовий об'єм ОП на базі DRAM (МБ)	0.064	0.256	4	16	64	258	4 000 разів
	Ціна (\$/МБ)	500	100	8	0.3	0.05	0.01	5 000 разів
	Час доступу (нс)	300	150	35	15	2	0.5	300 разів
	Типовий об'єм дискової пам'яті (МБ)	1	10	160	1 000	9 000	80000	80000 разів

Виходячи з наведених даних в табл. 10.1 та табл. 10.2, на рис. 10.2 показана зміна з роками часу доступу до статичної T_{SRAM} та динамічної T_{DRAM} напівпровідникової та дискової T_{DISK} пам'яті, а також тактової частоти роботи процесора T_{CPU} . Як видно з наведеного рисунку, існує значна різниця між тактовою частотою процесора і часом доступу до динамічної пам'яті. Причому з роками розрив у значеннях часових характеристик між процесором і динамічною напівпровідниковою пам'яттю та дисковою пам'яттю зростає.

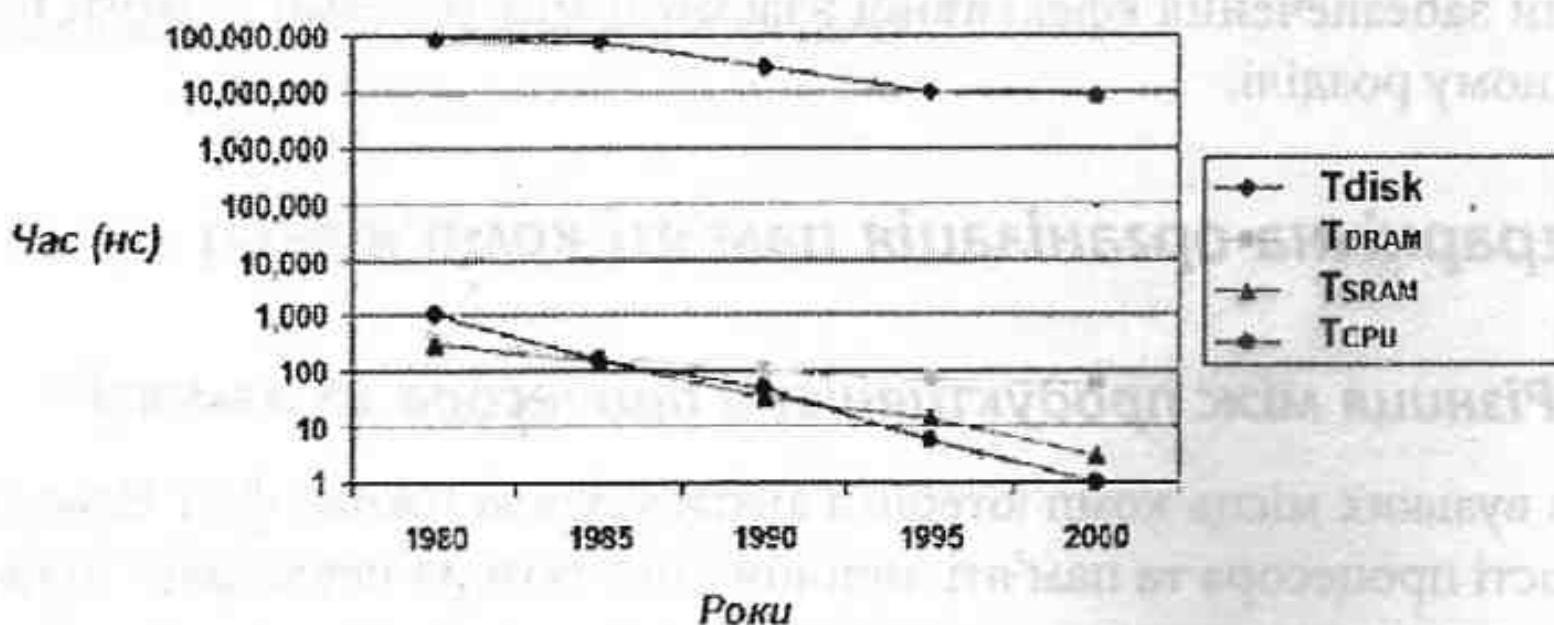


Рис. 10.2. Зміна з роками часових характеристик статичної і динамічної напівпровідникової та дискової пам'яті, а також процесора

Особливо важливим з точки зору архітектури комп'ютера є те, що зростає розрив між тактовою частотою процесора і часом доступу до динамічної напівпровідникової пам'яті DRAM, на основі якої будується основна пам'ять комп'ютера. Адже якраз між цими двома вузлами комп'ютера здійснюється основна частка обмінів інформацією. В той час, коли продуктивність процесора зростає щороку на 60 % (подвоєння за

1,5 року), ріст продуктивності динамічної напівпровідникової пам'яті не перевищує 9 % в рік (подвоєння за 10 років). Це виражається в збільшенні розриву в продуктивності між процесором і динамічною напівпровідниковою пам'яттю на 50 % в рік (рис. 10.3).



Рис. 10.3. Збільшення розриву в продуктивності між процесором і пам'яттю

Разом з тим, з рис. 10.2 видно, що розрив з роками між продуктивністю процесора і статичної напівпровідникової пам'яті є незначним. Виглядає доцільним побудова основної пам'яті комп'ютера на базі статичної напівпровідникової пам'яті. Однак це не так, оскільки, як видно з табл. 10.2, зберігання 1МБ інформації в такій пам'яті є приблизно в 100 разів дорожчим, аніж в динамічній напівпровідниковій пам'яті. Тому був знайдений наступний вихід з цієї ситуації – включення між основною пам'яттю комп'ютера і процесором додаткової швидкої пам'яті малої ємності, побудованої на основі статичної напівпровідникової пам'яті.

Іще більший розрив існує між продуктивністю зовнішньої дискової і динамічної напівпровідникової пам'яті. Однак, як видно з табл. 10.2, вартість зберігання 1МБ інформації в дисковій пам'яті приблизно в 20 разів менша, ніж в динамічній напівпровідниковій пам'яті. Крім того, вона забезпечує зберігання значно більших об'ємів інформації. Тому і тут виникає потреба пошуку механізму, який би забезпечив прискорення обміну між основною і зовнішньою пам'яттю.

10.1.2. Властивість локальності за зверненням до пам'яті

Якщо розглянути процес виконання більшості програм, то з дуже високою ймовірністю можна спрогнозувати, що адреса чергової команди програми або слідує безпосередньо за адресою, за якою була зчитана поточна команда, або розташована поблизу неї. Таке розташування адрес називається просторовою локальністю команд програм. Аналогічно можна стверджувати і про дані, які, як правило, є структурованими і, зазвичай, зберігаються в послідовних комірках пам'яті. Дано особливість називається просторовою локальністю даних.

Крім того, програми містять безліч невеликих циклів і підпрограм. Це означає, що невеликі набори команд можуть багато разів повторюватися протягом деякого інтервалу часу, тобто має місце часова локальність.

Таким чином, існує дві передбачувані властивості програм при зверненні до пам'яті:

- просторова локальність – якщо відбулося звернення до деякої комірки пам'яті, то з великою ймовірністю можна стверджувати, що в найближчий час відбудеться звернення до сусідньої комірки пам'яті;

- часова локальність – якщо відбулося звернення до деякої комірки пам'яті, то з великою ймовірністю можна стверджувати, що в найближчий час знову відбудеться звернення до тієї ж комірки пам'яті.

На рис. 10.4 наведено типовий розподіл звернень до пам'яті, який ілюструє наведену вище властивість часової та просторової локальності.

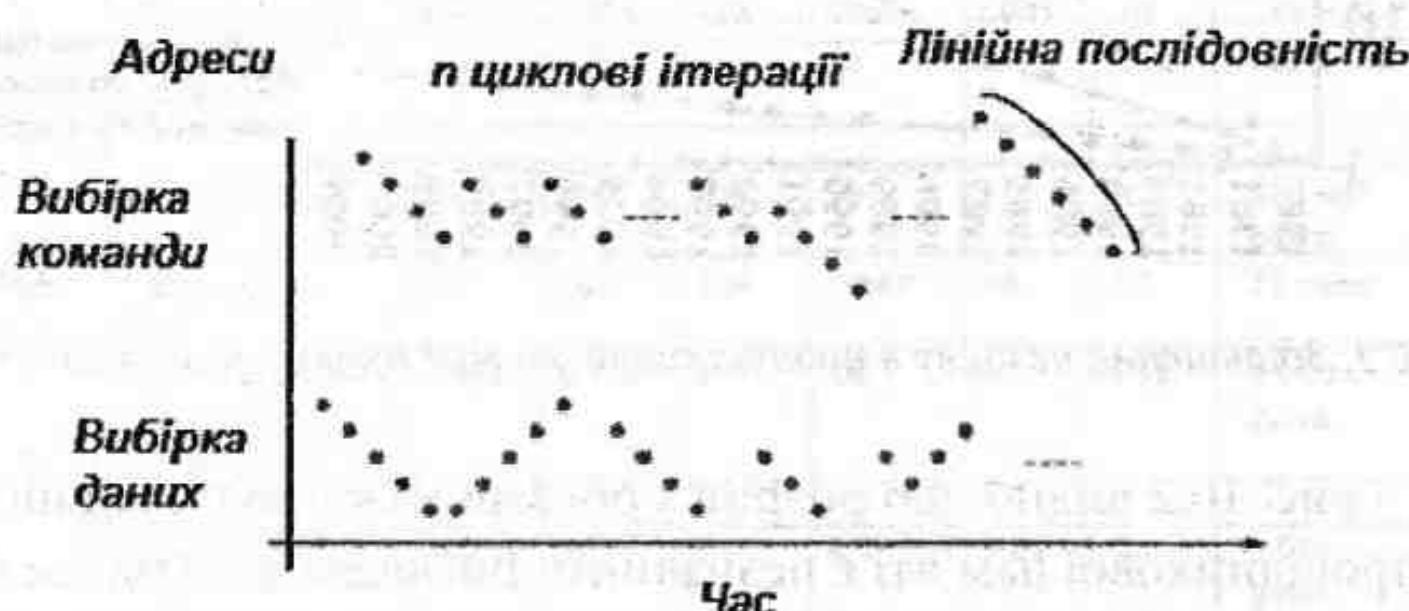


Рис. 10.4. Типовий розподіл звернень до пам'яті

Обидва види локальності об'єднує поняття локальності за зверненням. Властивість локальності можна виразити в чисельній формі і представити у вигляді так званого правила «90/10»: 90 % часу роботи програми пов'язано з доступом до 10 % адресного простору цієї програми.

З властивості локальності витікає, що програму розумно представити у вигляді послідовно оброблюваних фрагментів – компактних груп команд і даних. Поміщаючи такі фрагменти в швидшу пам'ять, можна істотно понизити загальні затримки на звернення до пам'яті, оскільки команди і дані, будучи один раз передані з повільного пристрою пам'яті в швидкий, потім можуть використовуватися багато разів, і середній час доступу до них в цьому випадку визначається вже швидшим пристроєм пам'яті. Це дозволяє зберігати великі програми і масиви даних на повільних, великої ємності, але дешевих пристроях пам'яті, а в процесі обробки активно використовувати швидку пам'ять порівняно невеликої ємності, збільшення якої пов'язане з великими витратами.

10.1.3. Принцип ієрархічної організації пам'яті

З проведеного вище аналізу можна зробити висновки про наступні фундаментальні та стабільні протягом тривалого часу властивості апаратних та програмних засобів сучасного комп'ютера:

- чим менший час доступу до пам'яті, тим менша її ємність та вища вартість зберігання в ній одного біта інформації;
- чим більша ємність пам'яті, тим більший час доступу до неї та нижча вартість зберігання в ній одного біта інформації;
- існує значна різниця між продуктивністю процесора і основної пам'яті, а також між продуктивністю основної та зовнішньої пам'яті;
- комп'ютерні програми наділені властивістю локальності за зверненням до пам'яті.

Ці фундаментальні властивості позитивно доповнюють одну одну з точки зору вирішення завдання забезпечення необхідної ємності та високої швидкодії пам'яті за прийнятну ціну. Вони є підґрунтям доцільності використання при побудові системи пам'яті підходу, відомого як принцип ієрархічної організації пам'яті.

Відповідно до цього принципу пам'ять комп'ютера складається із пристрій пам'яті різних типів, які, залежно від характеристик, належать до певного рівня ієрархії.

Пам'ять нижчого рівня має меншу ємність, швидша і має велику вартість в перерахунку на біт, ніж пам'ять вищого рівня. Рівні ієрархії взаємозв'язані: всі дані на деякому нижчому рівні можуть бути також знайдені на вищому рівні, і всі дані на цьому вищому рівні можуть бути знайдені на наступному вищому рівні і т. д.

З рухом вверх по ієрархічній структурі зменшується співвідношення вартість/біт, зростає ємність та час доступу. Однак завдяки властивості локальності за зверненням з рухом вверх по ієрархічній структурі зменшується частота звернення до пам'яті з боку нижчих рівнів, що веде до зменшення загальної вартості при заданому рівні продуктивності.

На кожному рівні ієрархії пам'ять розбивається на блоки, які є найменшою інформаційною одиницею, що пересилається між двома сусідніми рівнями ієрархії. Розмір блоків може бути фіксованим або змінним. При фіксованому розмірі блоку ємність пам'яті зазвичай кратна його розміру. Розмір блоків на кожному рівні ієрархії найчастіше різний і збільшується від нижчих рівнів до вищих.

Процесор взаємодіє з пам'яттю найнижчого рівня ієрархії, яка розміщена найближче до нього. При зверненні з боку процесора до пам'яті, наприклад, для зчитування команд або даних, проводиться їх пошук в пам'яті нижнього рівня. Факт виявлення потрібної інформації називають попаданням (hit), інакше говорять про промах (miss). При промаху проводиться пошук потрібної інформації в пам'яті наступного вищого рівня, де також можливі попадання або промахи. Після виявлення необхідної інформації виконується послідовне пересилання вмісту блоку з шукаютою інформацією з вищих рівнів на нижчі. Слід зазначити, що незалежно від числа рівнів ієрархії, пересилання інформації може здійснюватися лише між двома сусідніми рівнями, тобто пересилання інформації в обхід деяких рівнів не допускається.

При проведенні обміну інформацією блоками між пристроями пам'яті різних рівнів ієрархії необхідно вирішувати наступні питання:

- вибрати правило заміщення вмісту одних блоків вмістом інших блоків, оскільки через меншу ємність в пам'яті нижчого рівня не може бути така сама кількість блоків, як в пам'яті вищого рівня;
- визначити допустиме місце розташування нового вмісту блоку при записі з дотриманням правила заміщення;
- вибрати та дотримуватись стратегії запису з метою забезпечення узгодженості копій вмісту одних і тих же блоків, розташованих на різних рівнях, при записі нової інформації в копію, що знаходиться на нижчому рівні.

10.1.4. Характеристики ефективності ієрархічної організації пам'яті

При оцінці ефективності ієрархічної організації пам'яті використовують наступні характеристики:

- коефіцієнт попадань відповідного рівня ієрархії пам'яті – відношення числа звернень з пам'яті даного рівня ієрархії до пам'яті наступного вищого рівня, при яких

відбулося попадання, до загального числа звернень з пам'яті даного рівня ієрархії до пам'яті наступного вищого рівня ієрархії. Попадання – факт виявлення потрібної інформації при зверненні до пам'яті наступного вищого рівня;

- коефіцієнт промахів – відношення числа звернень до пам'яті наступного вищого рівня, при яких мав місце промах, до загального числа звернень до пам'яті даного рівня ієрархії. Якщо позначити коефіцієнт попадань через k_h , а коефіцієнт промахів через k_m , то залежність між ними можна виразити наступною формулою: $k_m = 1 - k_h$;
- час звернення при попаданні – час, необхідний для пошуку потрібної інформації в пам'яті нижчого рівня (включаючи з'ясування, чи є звернення попаданням), плюс час на фактичне зчитування даних;
- втрати на промахах – час, потрібний для заміни блоку в пам'яті нижчого рівня на блок з потрібними даними, розташований у пам'яті наступного (вищого) рівня.
- середній час доступу до пам'яті, який визначається з виразу: $t_{av} = t_h + k_m t_p$, де t_h – час звернення при попаданні, t_p – втрати на промахах. Імовірність попадання в сучасних комп'ютерах є досить високою (біля 95 % від загальної кількості звернень), і, відповідно, коефіцієнт промахів є низьким. Однак, через велику різницю між швидкодією пам'яті нижчого і вищого рівнів, навіть такий низький коефіцієнт промахів суттєво впливає на середній час доступу до пам'яті. Так, якщо прийняти, що час доступу до пам'яті нижчого рівня складає 1 такт, а час доступу до пам'яті вищого рівня в 10 разів більший, то середній час доступу буде рівним $t_{av} = 1 + 0.05 * 10 = 1.5$ такти.

10.1.5. Ієрархічна пам'ять сучасного комп'ютера

Структура ієрархічної пам'яті сучасного комп'ютера представлена на рис. 10.5.

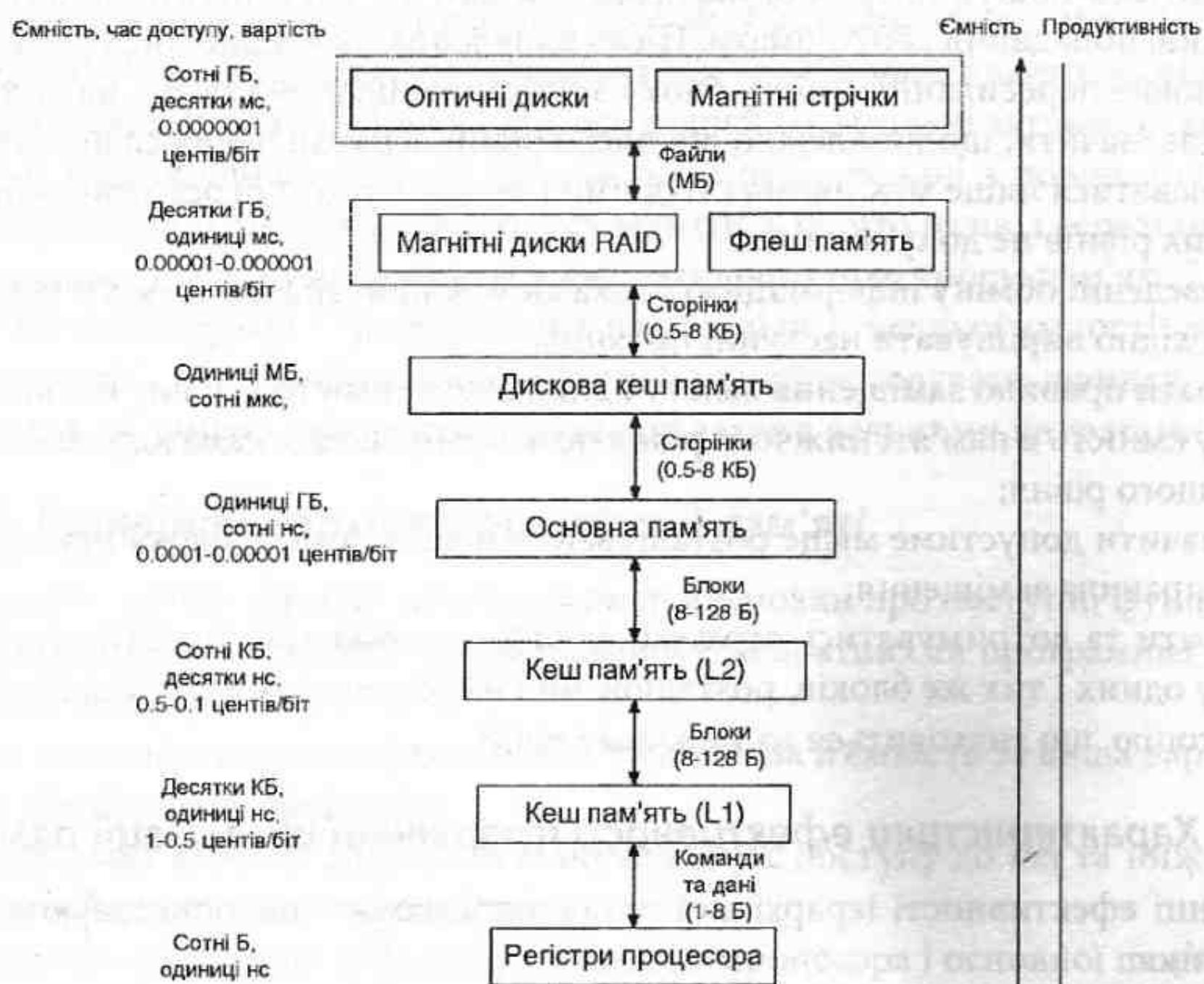


Рис. 10.5. Ієрархічна пам'ять

Найшвидший, але і мінімальний за ємністю, тип пам'яті – це внутрішні реєстри процесора, тобто реєстровий файл. Як правило, кількість реєстрів в реєстровому файлі невелика (типово 16-128). У архітектурі комп'ютера із спрощеною системою команд їх число може доходити до декількох сотень. Основна пам'ять має значно більшу ємність. Між реєстрами процесора і основною пам'яттю часто розміщують кеш пам'ять, яка за ємністю відчутно програє основній пам'яті, але істотно перевершує останню за швидкодією, поступаючись у той же час реєстровому файлу. У більшості сучасних комп'ютерів є декілька рівнів кеш пам'яті, які позначають буквою L з номером рівня кеш пам'яті. На рис. 10.5 показані два таких рівні. В останніх комп'ютерах все частіше з'являється також третій рівень кеш пам'яті (L3), причому розробники комп'ютерів говорять про доцільність введення і четвертого рівня – L4. Кожен наступний рівень кеш пам'яті має більшу ємність, але одночасно і меншу швидкодію в порівнянні з попереднім. За швидкодією будь-який рівень кеш пам'яті перевершує основну пам'ять. Чотири нижні рівні ієархії утворюють внутрішню пам'ять комп'ютера, а всі вищі за них рівні – це зовнішня або вторинна пам'ять.

Всі види внутрішньої пам'яті реалізуються на основі напівпровідниковых технологій і, в основному, є енергозалежними.

Довготривале зберігання великих об'ємів інформації (програм і даних) забезпечується зовнішньою пам'яттю, першим рівнем якої є пам'ять на базі магнітних дисків. Після дискової пам'яті йдуть пристрій архівної пам'яті, серед яких найпоширеніші пристрій на базі оптичних дисків, на магнітних стрічках та флеш пам'ять.

Нарешті, ще один рівень ієархії може бути доданий між основною пам'яттю і магнітними дисками. Цей рівень носить назву дискової кеш пам'яті і реалізується у вигляді пристрію, що входить до складу магнітного диску. Дискова кеш пам'ять істотно покращує швидкість обміну інформацією між магнітними дисками і основною пам'яттю.

Розглянемо далі принципи обміну інформацією між двома базовими рівнями: між процесором і основною пам'яттю та між основною і зовнішньою пам'яттю.

10.2. Організація обміну інформацією між процесором і основною пам'яттю через кеш пам'ять

10.2.1. Кеш пам'ять в складі комп'ютера

Звернення процесора до основної пам'яті завжди локалізовано в невеликому діапазоні змін її адрес. Застосування кеш пам'яті ґрунтуються на обох принципах локальності за зверненням: в ній використовується часова локальність, оскільки зберігається вміст комірок, до яких недавно відбулося звернення, та використовується просторова локальність, оскільки в блоках даних зберігається вміст множини сусідніх комірок.

Кеш пам'ять розташована між процесором і основною пам'яттю. Це швидка буферна пам'ять невеликої ємності. Кеш пам'ять працює на близькій тактовій частоті до процесора і не пригальмовує його роботу. Кеш пам'ять (cache в перекладі з англ. – тайник) залишається прозорою для програміста, тому що система команд процесора, як правило, не містить команд роботи з кеш пам'яттю.

Типовий фрагмент структури сучасного комп'ютера, в якому використовується два рівні кеш пам'яті, подано на рис. 10.6. Для зберігання даних і команд в кеш пам'яті пер-

шого рівня використано розділені кеш пам'яті даних і команд. В свою чергу, обидві кеш пам'яті першого рівня зв'язані з єдиною кеш пам'яттю другого рівня, яка взаємодіє з основною пам'яттю. Зрозуміло, що обмін в підсистемі “кеш пам'ять L2 – кеш пам'ять даних L1” є двостороннім, а в підсистемі “кеш пам'ять L2 – кеш пам'ять команд L1” – одностороннім.



Рис. 10.6. Фрагмент структури сучасного комп'ютера

Наведена структура ефективно поєднала риси Принстонської і Гарвардської архітектур. Такий підхід вигідніший, ніж побудова швидкої основної пам'яті без використання проміжної кеш пам'яті.

10.2.2. Порядок взаємодії процесора і основної пам'яті через кеш пам'ять

Основні правила організації кеш пам'яті та основної пам'яті, які забезпечують їх ефективну взаємодію, є наступними:

- Кеш і основна пам'яті діляться на блоки одинакового розміру, тобто вони можуть вміщувати одну і ту ж кількість слів.
- Базовою порцією інформації, яка переміщується між основною та кеш пам'яттю, є вміст одного блоку.
- Кожний блок має свій номер.
- Нумерація комірок в кожному блоці одна.
- В кожний момент часу в блоках кеш пам'яті знаходиться вміст декількох переписаних із основної пам'яті блоків.
- Кожне слово кеш пам'яті супроводжується адресним тегом, що вказує на те, вміст якого блоку основної пам'яті є переписаним до блоку кеш пам'яті, в якому знаходиться слово.
- Ідентичність вмісту блоків кеш пам'яті та основної пам'яті забезпечується використанням спеціальних методів оновлення вмісту блоків основної пам'яті.
- Заміна вмісту одних блоків в кеш пам'яті вмістом інших блоків з основної пам'яті здійснюється за правилами, які називають алгоритмом заміщення.
- Між блоками основної пам'яті та кеш пам'яті встановлюється відповідність, яка задається функцією відображення.

Якщо дотримано вищезазначених правил, то взаємодія між процесором, кеш пам'яттю та основною пам'яттю відбувається наступним чином.

Процесор формує адреси для взаємодії з основною пам'яттю і не враховує наявність кеш пам'яті. Він видає адресу комірки основної пам'яті і сигнали запису або зчитування. Контролер кеш пам'яті визначає, чи вміст блоку з даною коміркою знаходиться в кеш пам'яті.

ті. Якщо так (попадання), то при зчитуванні з процесором взаємодіє лише кеш пам'ять, а при записі для забезпечення ідентичності вмісту блоків кеш і основної пам'яті може бути два варіанти. При першому варіанті здійснюється одночасний запис операнда в комірку з тією ж адресою в кеш пам'яті та в основній пам'яті. При другому варіанті одночасний запис не здійснюється, а використовується біт модифікації. Саме ж заміщення вмісту даної комірки основної пам'яті відбувається пізніше, при заміщенні в кеш пам'яті вмісту даного блоку. Оскільки питання забезпечення ідентичності вмісту блоків кеш пам'яті та основної пам'яті є важливим, в наступному пункті воно буде розглянуто детальніше.

Якщо ж вміст блоку з заданою коміркою відсутній в кеш пам'яті (промах), то при зчитуванні він поступає із основної пам'яті в кеш пам'ять, а слово з заданої комірки поступає прямо в процесор, або після пересилки вмісту всього блоку. Перший підхід ефективніший але складніший. При операції запису в цьому випадку слово записується прямо в основну пам'ять.

Таким чином, кеш пам'ять перехоплює сигнали читання/запису, які процесор надсилає до основної пам'яті, а коли потрібно, то надає процесору копії даних, які тимчасово зберігає у власній робочій пам'яті. Якщо кеш пам'ять спроможна підмінити собою основну пам'ять (у понад 96–98 відсотків випадків), тоді вона за рахунок власних ресурсів задовольняє запит процесора. Процесор не пригальмовується і продовжує працювати на повній швидкості. Коли “підміна” основної пам'яті неможлива (менше від двох–чотирьох відсотків випадків), тоді кеш пам'ять залучає до роботи основну пам'ять, обмін з якою суттєво пригальмовує процесор.

Усі задачі, пов'язані з перехопленням запитів від процесора до основної пам'яті, вирішує контролер кеш пам'яті, який є її складовою частиною. Другою частиною кеш пам'яті є невелика робоча пам'ять, де зберігаються копії вмісту блоків основної пам'яті, що брали участь в обслуговуванні останніх запитів процесора.

Важливо, що вміст комірок основної пам'яті копіюється до кеш пам'яті разом із їхніми адресами. Саме ці адреси і дозволяють контролеру кеш пам'яті приймати рішення про спроможність задовільнити конкретний процесорний запит без залучення до обміну повільної основної пам'яті.

10.2.3. Забезпечення ідентичності вмісту блоків кеш пам'яті і основної пам'яті

Як ми вже бачили вище, в процесі обчислень процесор не лише зчитує з кеш пам'яті наявну інформацію, але і записує нову, оновлюючи тим самим вміст блоків кеш пам'яті, які є копіями вмісту відповідних блоків основної пам'яті. Виникає ситуація, коли вміст блоку кеш пам'яті та відповідного блоку основної пам'яті перестають співпадати, що створює проблему необхідності фіксування та усунення цього неспівпадіння шляхом оновлення вмісту комірок основної пам'яті. Для подолання цієї проблеми в комп'ютерах з кеш пам'яттю використовуються два методи оновлення вмісту блоків основної пам'яті: метод наскрізного запису і метод зворотного запису.

За методом наскрізного запису перш за все оновлюється вміст комірки основної пам'яті. Якщо в кеш пам'яті існує копія вмісту цієї комірки, то вона також оновлюється. Якщо ж в кеш пам'яті відсутня потрібна копія, то, або з основної пам'яті в кеш пам'ять пересилається вміст блоку, в якому знаходиться оновлене слово (наскрізний запис з відображенням), або цього не робиться (наскрізний запис без відображення).

Метод наскрізного запису, про який вище вже згадувалось, передбачає одночасний запис операнда в комірку з тією ж адресою як кеш, так і основної пам'яті. Основна перевага методу наскрізного запису полягає в тому, що коли вміст деякого блоку в кеш пам'яті має бути заміщений, то його можна не повернати в основну пам'ять, оскільки його копія там вже є. Метод достатньо простий в реалізації. На жаль, ефект від використання кеш пам'яті (скорочення часу доступу) стосовно операцій запису тут відсутній. Даний метод застосований в мікропроцесорах i486 фірми Intel.

Певний виграв дає модифікація методу наскрізного запису, відома як метод буферизованого наскрізного запису. Інформація спочатку записується в кеш пам'ять і в спеціальний буфер, що працює за схемою FIFO. Запис в основну пам'ять проводиться вже з буфера, а процесор, не чекаючи її закінчення, може відразу ж продовжувати свою роботу. Звичайно, відповідна логіка керування повинна піклуватися про те, щоб своєчасно очищувати заповнений буфер. При використанні буферизації процесор повністю звільнюється від роботи з основною пам'яттю.

Згідно з методом зворотного запису, слово заноситься лише в кеш пам'ять. Якщо вмісту потрібного блоку в кеш пам'яті немає, то він спочатку пересилається з основної пам'яті, після чого запис все одно виконується виключно в кеш пам'ять. При заміщенні вмісту блоку його попередній вміст необхідно заздалегідь переслати у відповідне місце основної пам'яті. Для методу зворотного запису, на відміну від методу наскрізного запису, характерним є те, що при кожному зчитуванні з основної пам'яті здійснюються два пересилання вмісту блоків між основною і кеш пам'яттю.

Ефективність зворотного запису підвищується при використанні біта модифікації. Відповідно до методу зворотного запису з бітом модифікації, коли в якомусь блоці кеш пам'яті проводиться заміщення вмісту, встановлюється пов'язаний з цим блоком біт модифікації (прапорець). При заміщенні вміст блоку з кеш пам'яті переписується в основну пам'ять лише тоді, коли його біт модифікації встановлений в 1. Такий метод використовується, наприклад, в мікропроцесорах класу i486 і Pentium фірми Cyrix.

В середньому зворотний запис на 10 % ефективніший за наскрізний запис, але для його реалізації потрібні додаткові апаратні витрати. З іншого боку, практика показує, що операції запису складають менше 30 % від загальної кількості звернень до пам'яті. Таким чином, відмінність за швидкодією між розглянутими методами невелика.

Крім розглянутих вище методів забезпечення ідентичності вмісту блоків кеш пам'яті та основної пам'яті існують й інші методи, пов'язані з тим, що пристрой введення/виведення можуть безпосередньо обмінюватися інформацією з основною пам'яттю без участі процесора, а значить і кеш пам'яті. Тобто в основну пам'ять з пристроя введення, минувши процесор, заноситься нова інформація і невірною стає копія, що зберігається в кеш пам'яті. Запобігти подібній неузгодженості дозволяють два підходи: забезпечити введення будь-якої інформації в основну пам'ять з проведенням відповідних змін в кеш пам'яті, або здійснювати доступ до основної пам'яті лише через кеш пам'ять.

10.2.4. Функція відображення

10.2.4.1. Типи функцій відображення

Відображенням називають відповідність між вмістом блоків кеш пам'яті та блоків основної пам'яті.

Існує три основних типи відображення: повністю асоціативне, пряме та частково-асоціативне. Рис. 10.7 дає пояснення кожному типу відображення. При асоціативному відображенії вміст будь-якого блоку основної пам'яті може знаходитись в будь-якому блоці кеш пам'яті. Тобто вміст 12-го блоку основної пам'яті, наведеної на рис. 10.7, може бути записаний до будь-якого з 8 блоків кеш пам'яті. При прямому відображенії вміст кожного блоку основної пам'яті можна копіювати не до будь-якого, а лише до наперед визначеного блоку кеш пам'яті. Наприклад, це може бути відображення вмісту блоку K основної пам'яті в блок (K)modr кеш пам'яті, де r – кількість блоків в кеш пам'яті. Тобто 12 блок основної пам'яті на рис. 10.7 може бути записаний лише до 4 блоку кеш пам'яті, так як $(12) \bmod 8 = 4$. При частково-асоціативному відображенії вміст кожного блоку основної пам'яті можна копіювати не до будь-якого, а до кількох наперед визначених блоків кеш пам'яті. Наприклад, це може бути відображення вмісту блоку K основної пам'яті в довільний блок сектора (K)mods кеш пам'яті, де s – кількість секторів в кеш пам'яті. Тобто вміст 12-го блоку основної пам'яті на рис. 10.7 може бути записаний лише до блоку 0 або блоку 1 сектора 0 кеш пам'яті, так як $(12) \bmod 4 = 0$.

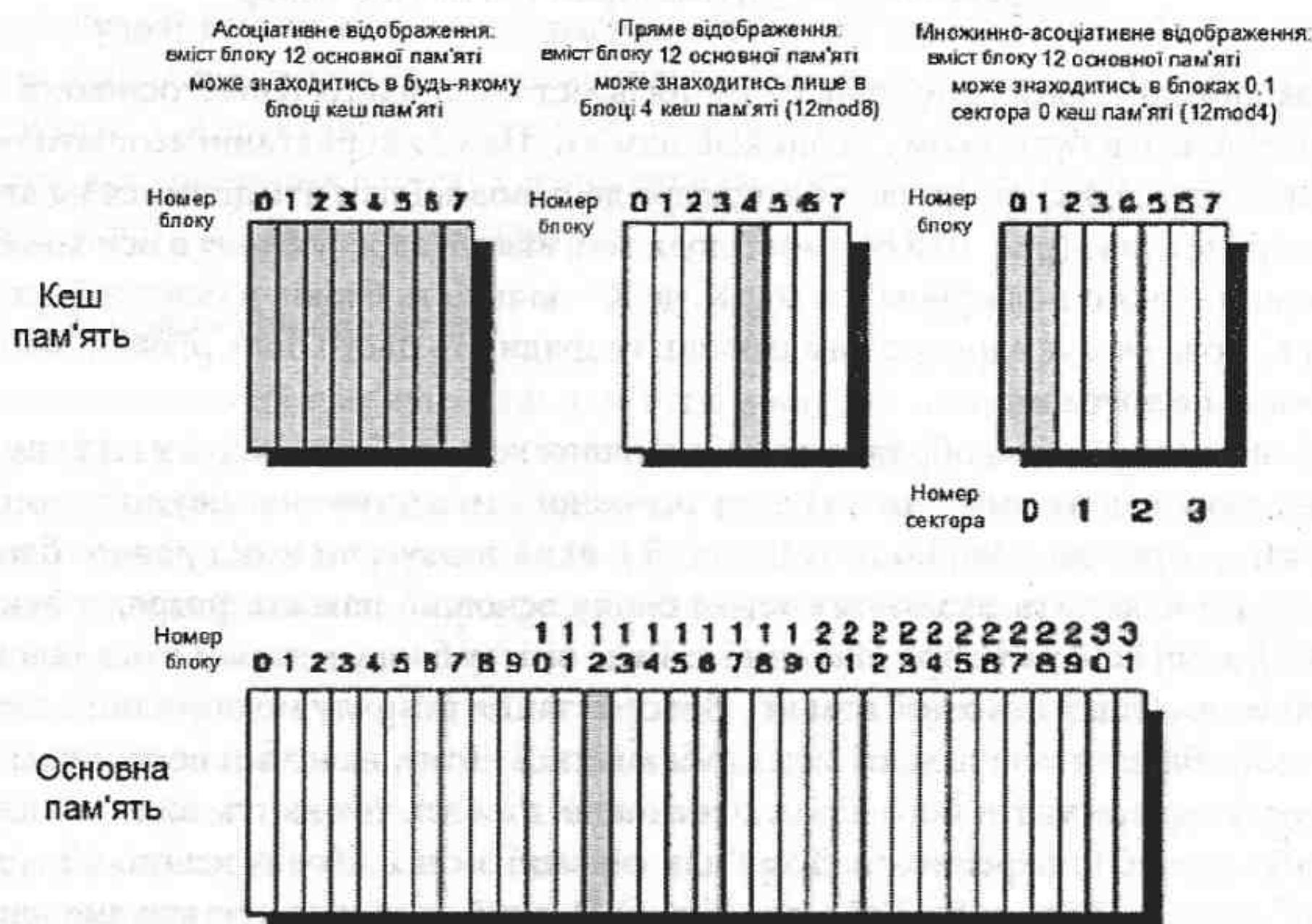


Рис. 10.7. Три основних типи відображення

Для детальнішого пояснення принципів обміну вмісту блоків основної пам'яті та кеш пам'яті при використанні кожного типу відображення приймемо, що основна пам'ять має ємність 1 ГБ, а кеш пам'ять має ємність 16 КБ, вони поділені на блоки ємністю по 32 Б, тобто в основній пам'яті є 2^{25} блоків, а в кеш пам'яті – 512 блоків.

10.2.4.2. Повністю асоціативне відображення

Взаємодія основної пам'яті з кеш пам'яттю з використанням повністю асоціативного відображення показана на рис. 10.8 а.

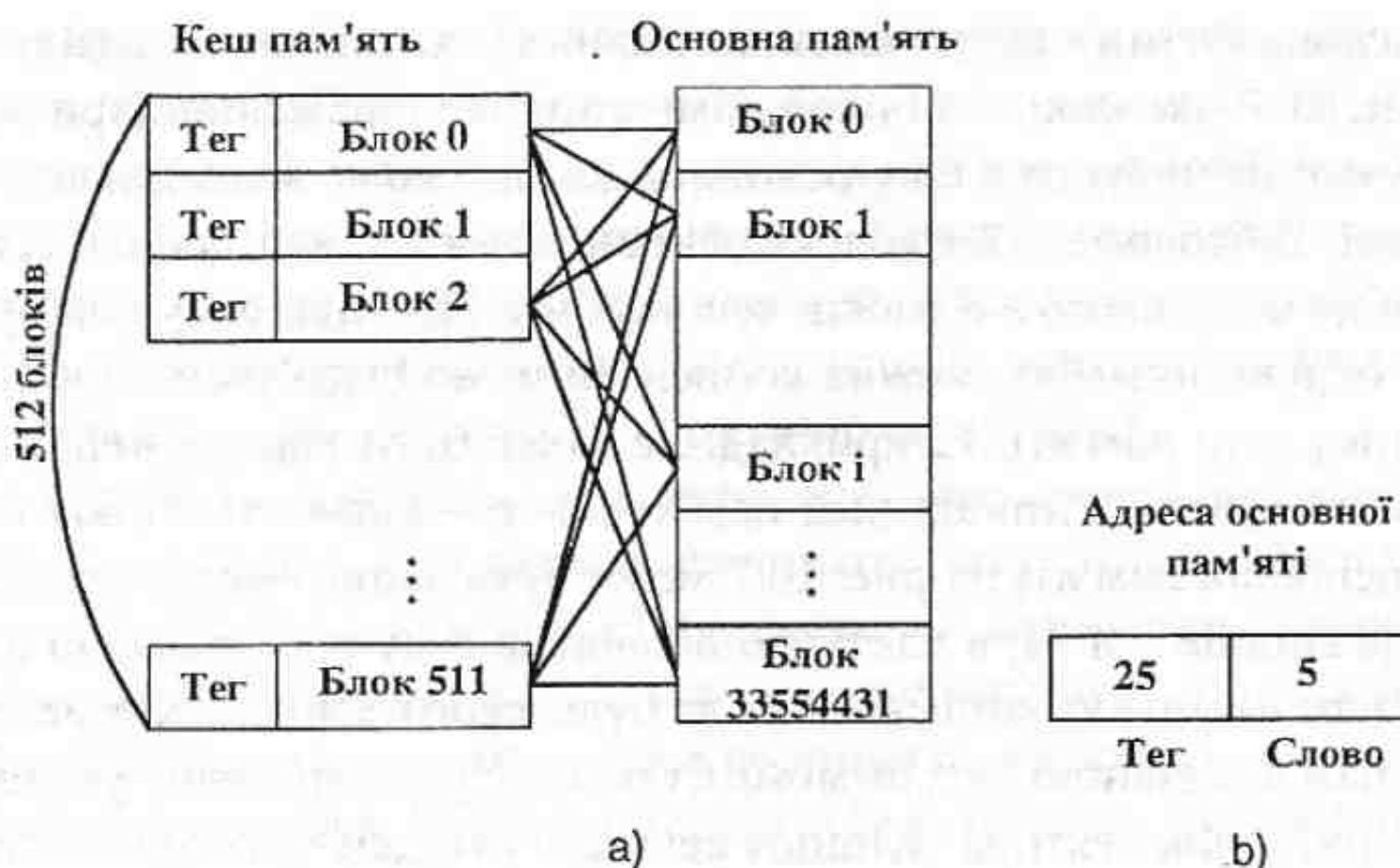


Рис. 10.8. Взаємодія основної пам'яті з кеш пам'яттю з використанням повністю асоціативного відображення a) та формат адреси основної пам'яті b)

Відповідно до цього типу відображення вміст будь-якого блоку основної пам'яті може знаходитись в будь-якому блоці кеш пам'яті. При використанні асоціативного відображення адреса, яка поступає з процесора до основної пам'яті, ділиться на два поля: поле тега і поле слова (рис. 10.8 b). Вміст поля тега вказує адресу блоку в основній пам'яті. Розрядність цього поля рівна $k = \log_2 K$, де K – кількість блоків в основній пам'яті. А вміст поля слова вказує адресу слова в блоці. Розрядність цього поля рівна $m = \log_2 M$, де M – кількість слів в блоці.

При даному способі відображення кожен рядок кеш пам'яті вміщує наступну інформацію: тег, який вказує вміст якого блоку основної пам'яті переписано до даного блоку кеш пам'яті, розряд достовірності (valid bit V), який вказує, чи вміст даного блоку кеш пам'яті дійсно належить вказаному тегом блоку основної пам'яті, розряд модифікації (dirty bit D), який інформує про внесення змін до вмісту блоку кеш пам'яті, а також вміст вказаного тегом блоку основної пам'яті. Використання розряду модифікації пов'язано з тим, що вміст блоку в кеш пам'яті може змінюватись. Тому, якщо він не змінився, немає необхідності переписувати його назад до основної пам'яті, так як там вже є копія. Якщо ж змінився, потрібно переписати. Тому для фіксації змін в блоках кеш пам'яті до рядка кеш пам'яті і вводиться розряд модифікації D, який дозволяє суттєво зменшити час обміну. Коли процесору потрібний операнд із блоку з певною адресою, контролер кеш пам'яті повинен шукати його шляхом порівняння відповідних розрядів адреси з тегами всіх блоків кеш пам'яті. Якщо таке порівняння здійснювати послідовно, то кеш пам'ять втрачає зміст через низьку швидкодію. Доцільніше таке порівняння робити одночасно зі всіма тегами, як це показано на рис. 10.9. В цьому випадку кожен блок кеш пам'яті повинен містити свій компаратор, тобто це буде пам'ять з асоціативним доступом. При наявності в кеш пам'яті відповідного тега, та при однічному значенні розряду достовірності, кеш пам'ять видає сигнал підтвердження попадання та надає доступ до відповідного блоку.

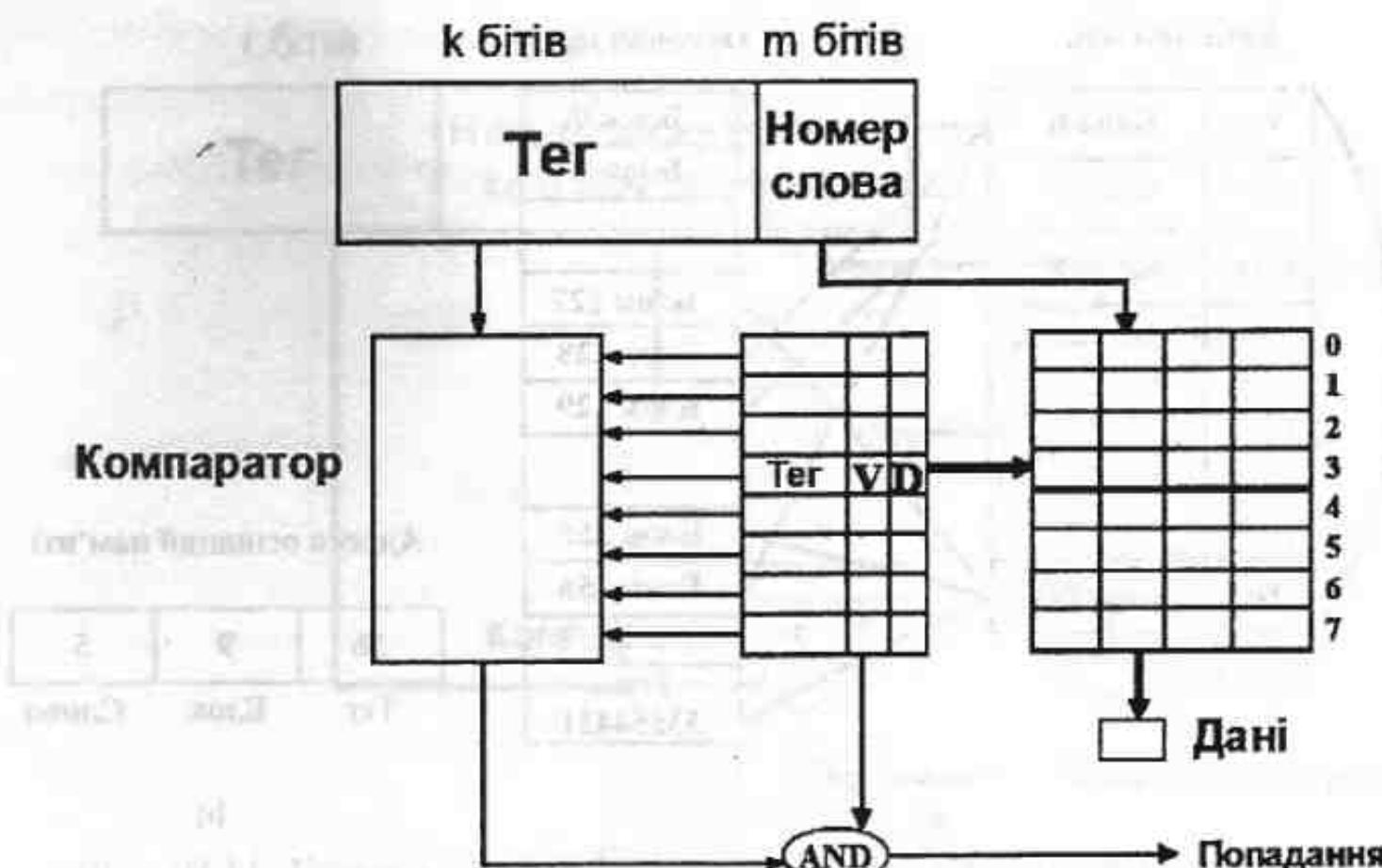


Рис. 10.9. Ідентифікація даних в кеш пам'яті з асоціативним доступом

При побудові кеш пам'яті малої ємності використання пам'яті з асоціативним доступом є доцільним, оскільки дозволяє швидко ідентифікувати потрібну інформацію та реалізувати найефективніший спосіб відображення. При побудові кеш пам'яті великої ємності використання пам'яті з асоціативним доступом стає проблематичним через складність реалізації та великі габарити.

10.2.4.3. Пряме відображення

Взаємодія основної пам'яті з кеш пам'яттю з використанням прямого відображення показана на рис. 10.10а. Тут вміст кожного блоку основної пам'яті можна копіювати лише до наперед визначеного блоку кеш пам'яті. Одна з можливих реалізацій використовує розрядне відображення. Тут додатково до того, як це було зроблено для асоціативного відображення, k -розрядне поле адреси блоку в основній пам'яті розбивається на дві частини: поле тега r та поле номера блоку s (рис. 10.10б). Тим самим основна пам'ять ділиться на секції, за кожною з яких закріплюється відповідна бирка (або ознака), яку називають тегом. Кількість R секцій (а значить і кількість тегів) рівна відношенню ємності основної пам'яті до ємності кеш пам'яті, або, що є тим самим, відношенню кількості K блоків в основній пам'яті, до кількості S блоків в кеш пам'яті. Розрядність поля тега визначається з виразу $r = \log_2 K/S = \log_2 (2^k/2^s) = k - s$. Кількість S блоків в кожній секції рівна кількості блоків в кеш пам'яті. Тобто ємність секції рівна ємності кеш пам'яті.

Наприклад, якщо в адресі блоку основної пам'яті присутні k розрядів, то молодші n з них вибирають, в який блок кеш пам'яті може копіюватись вміст блоку основної пам'яті. Отже, вміст всіх блоків з одинаковими молодшими адресами потрапляє в один і той самий блок кеш пам'яті. Інші $r=k-s$ розрядів адреси, що залишилися, використовуються як адресний тег, який використовується для порівняння з адресою, виданою процесором з метою пошуку номера секції в основній пам'яті.

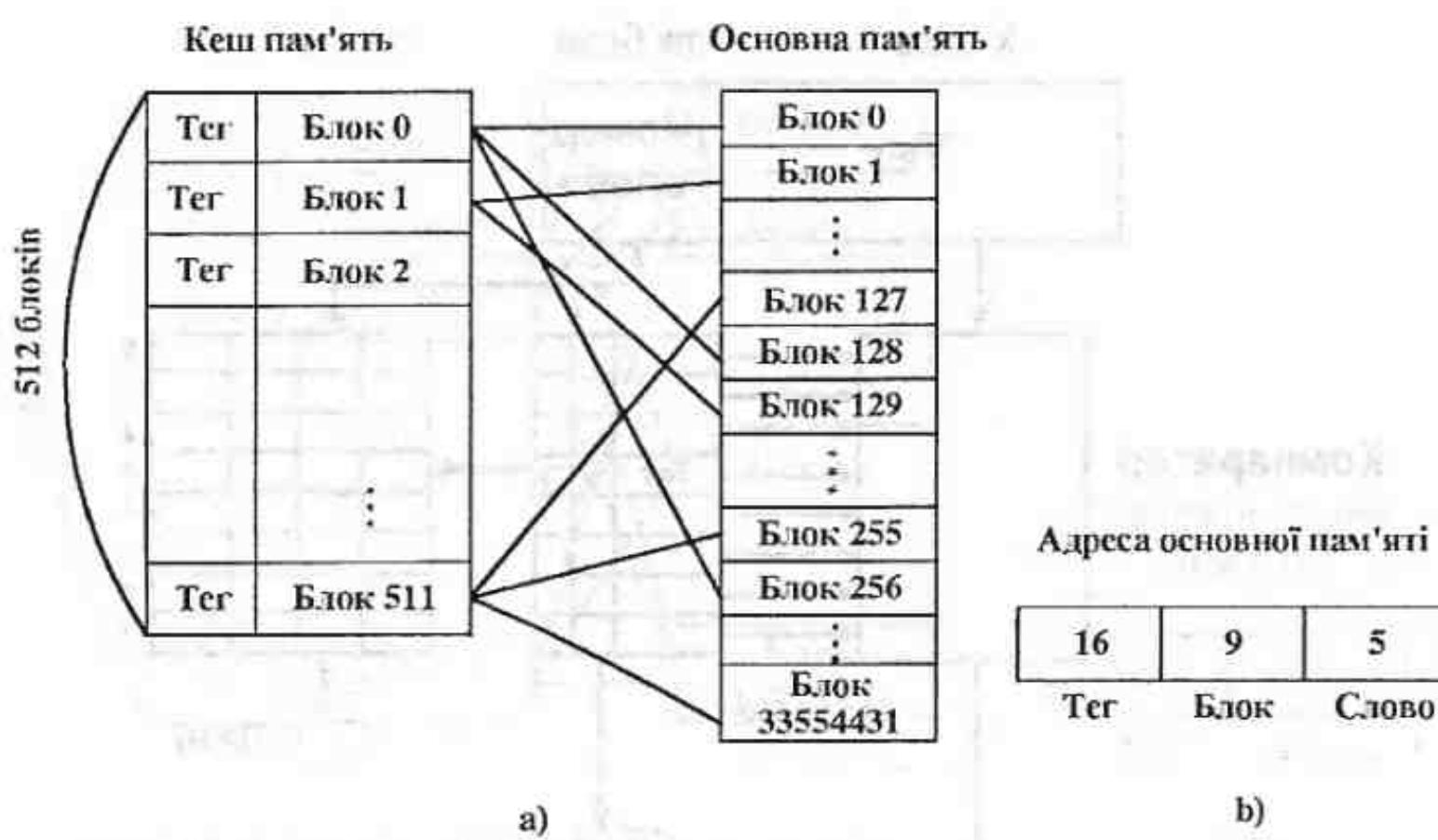


Рис. 10.10. Взаємодія основної пам'яті з кеш пам'яттю з використанням прямого відображення а) та формат адреси основної пам'яті б)

Для наведеної вище умови, коли що основна пам'ять має ємність 1 ГБ, а кеш пам'ять – 16 КБ, вони поділені на блоки ємністю по 32 Б, тобто кількість блоків в основній пам'яті рівна 2^{25} , а в кеш пам'яті – 2^9 , маємо: повна розрядність адреси $n = 30$, причому поле адреси блоків основної пам'яті займає $k = 25$ розрядів, поле номера тега (тобто номера секції в основній пам'яті) $r = 16$ розрядів, поле номера блоку в секції (в кеш пам'яті) $n = 9$ розрядів, поле адреси слова в блоці $m = 5$ розрядів (рис. 10.10b). Зауважимо, що процесор не сприймає структурної інтерпретації адреси, наведеної на рис. 10.10b. Таку інтерпретацію адресі надає лише контролер кеш пам'яті, коли перехоплює цю адресу, що призначена основній пам'яті (в гарвардській архітектурі це може бути як пам'ять даних, так і пам'ять команд). Контролер кеш пам'яті за допомогою дев'яти середніх розрядів слова адреси звертається до визначеного цими розрядами блоку власної пам'яті. Зрозуміло, що шуканий в такий спосіб блок завжди присутній в кеш пам'яті. Адже кеш пам'ять вміщує 512 блоків. Але вміст віднайденого блоку кеш пам'яті може бути копією не одного, а одного з декількох дозволених на копіювання блоків основної пам'яті. Наприклад, до нульового блоку кеш пам'яті дозволено копіювати вміст наступних блоків основної пам'яті: 0, 128, 256, 512 і т. д. Усього до кожного блоку кеш пам'яті можна скопіювати вміст одного з $2^{16} = 65536$ блоків основної пам'яті, оскільки ємність основної пам'яті в 65536 разів перевищує ємність кеш пам'яті. Постає завдання визначити, чи є поточне наповнення вказаного блоку кеш пам'яті відповідним до запиту процесора. Задання розв'язують за допомогою вмісту старших 16 бітів адреси основної пам'яті, які утворюють поле із назвою Тег.

Основною перевагою тут є те, що кеш пам'ять має структуру звичайної пам'яті з довільним доступом, тому потрібен лише один компаратор (рис. 10.11).

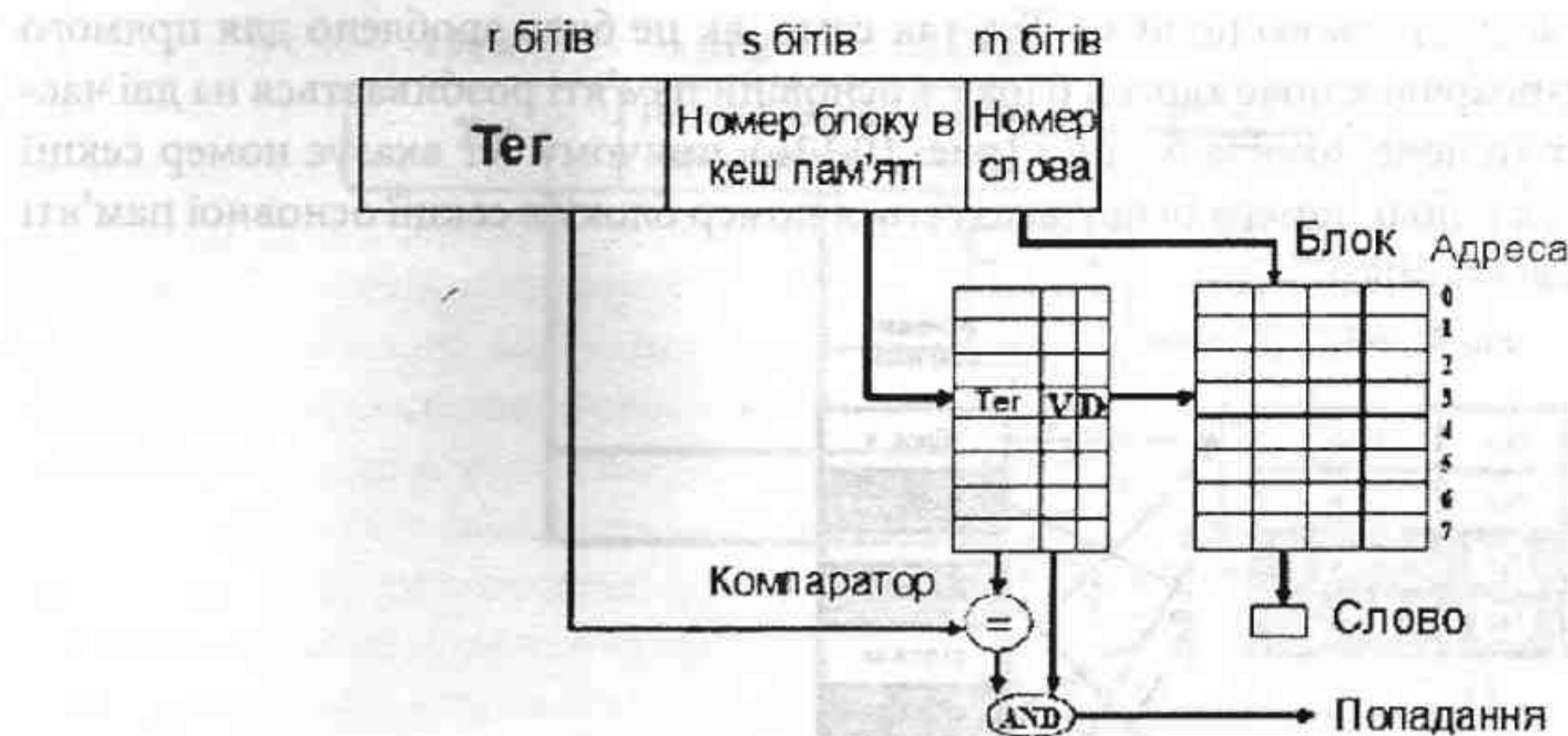


Рис. 10.11. Ідентифікація даних в кеш пам'яті з довільним доступом

При даному способі відображення кожен рядок кеш пам'яті вміщує наступну інформацію: тег, який вказує вміст блоку з якої секції основної пам'яті переписано до даного блоку кеш пам'яті, розряд достовірності (valid bit V), який вказує чи вміст даного блоку кеш пам'яті дійсно належить блоку основної пам'яті, вказаному s розрядами поля адреси, розряд модифікації (dirty bit D), який інформує про внесення змін до вмісту блоку кеш пам'яті, а також вміст блоку основної пам'яті, вказаного s розрядами поля адреси. Коли процесору потрібний операнд із блоку з певною адресою, контролер кеш пам'яті вибирає з пам'яті тегів відповідний номеру блоку в кеш пам'яті тег, та порівнює його з відповідними t розрядами адреси, в яких вказано тег. При наявності в кеш пам'яті відповідного тега та при однічному значенні розряду достовірності кеш пам'ять видає сигнал підтвердження попадання та надає доступ до відповідного блоку.

Отже пам'ять такого типу може мати велику ємність. Основний недолік закріплення за кожним блоком однієї секції основної пам'яті одного блоку в кеш пам'яті, тому в кеш пам'яті може заходитись обмежене число вмісту блоків основної пам'яті. Якщо процесор почергово звертається до даних з двох різних секцій основної пам'яті, які відображуються на той же блок кеш пам'яті, то ймовірність попадання буде низькою. Як наслідок, збільшується кількість промахів і, відповідно, час звернення до пам'яті.

10.2.4.4. Частково-асоціативне відображення

Взаємодія основної пам'яті з кеш пам'яттю з використанням частково-асоціативного відображення показана на рис. 10.12а. Тут послідовно розміщені блоки кеш пам'яті об'єднуються в сектори рівної ємності. Зазвичай один сектор містить 2, 4, 8 блоків. Основна пам'ять ділиться на секції, за кожною з яких закріплюється тег, що вказує номер секції. Кількість таких секцій рівна відношенню ємності основної пам'яті до ємності секції. При цьому ємність секції рівна відношенню ємності кеш пам'яті до кількості блоків в секторі кеш пам'яті. Тобто кількість блоків в секції основної пам'яті рівна кількості блоків в кеш пам'яті, поділеній на кількість блоків в секторі кеш пам'яті. Таким чином, в такій реалізації, як і при прямому відображення, використовується схема вибору сектора в кеш пам'яті за молодшими розрядами адреси блоку основної пам'яті, але в такому секторі кеш пам'яті знаходиться одразу вміст декількох блоків основної пам'яті, вибір між якими здійсню-

ється на основі асоціативного пошуку. Тут так само, як це було зроблено для прямого відображення, k-роздрядне поле адреси блоку в основній пам'яті розбивається на дві частини: поле тега r та поле номера блоку s (рис. 10.12b), причому тег вказує номер секції основної пам'яті, а в полі номера блоку вказується номер блоку в секції основної пам'яті та відповідно в кеш пам'яті.

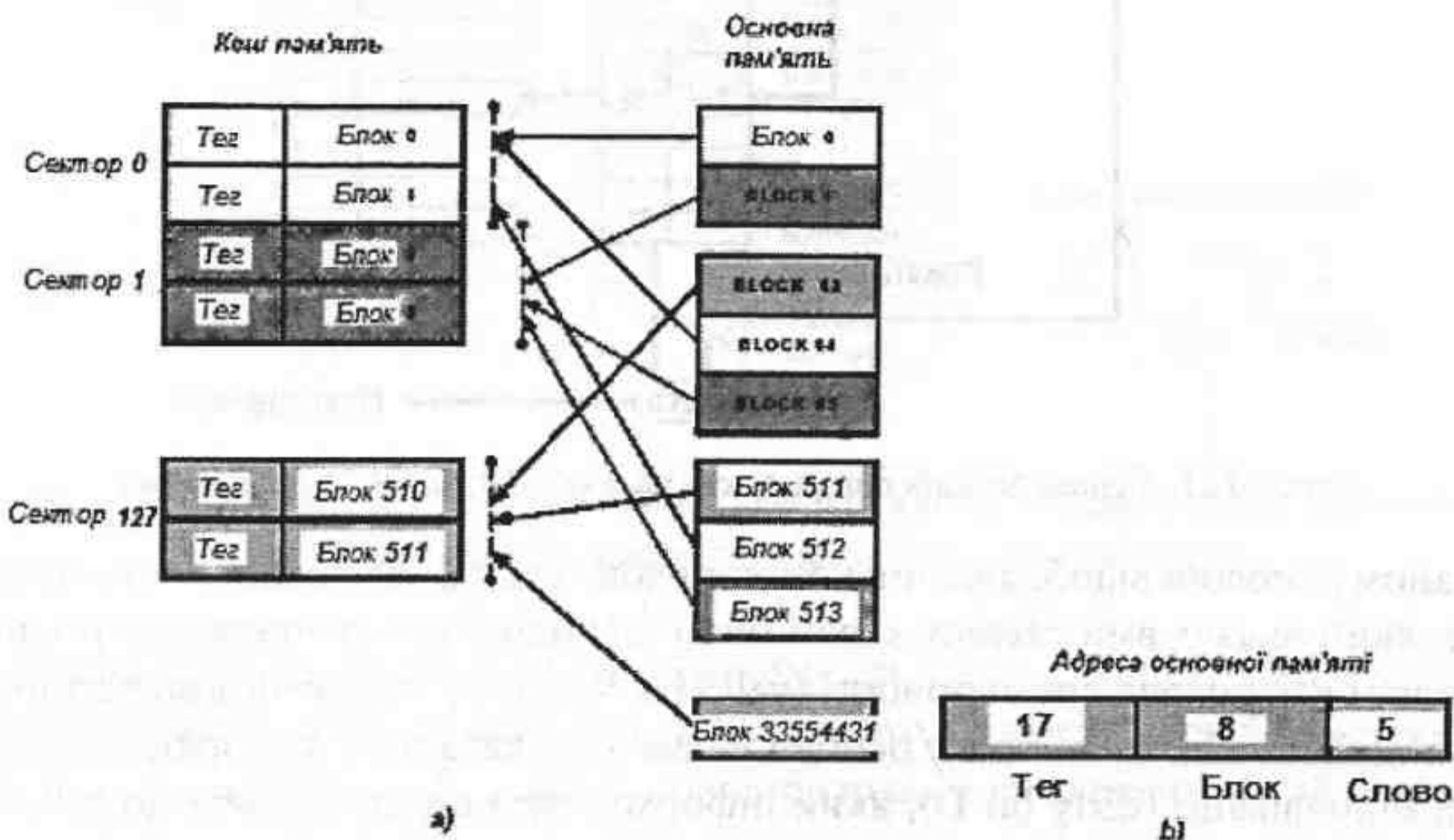


Рис. 10.12. Взаємодія основної пам'яті з кеш пам'яттю з використанням частково-асоціативного відображення a), коли кількість блоків в секторі кеш пам'яті рівна 2, та формат адреси основної пам'яті b)

Для наведеного вище прикладу, коли кількість блоків в основній пам'яті рівна 2^{25} , а в кеш пам'яті – 2^9 , причому кількість блоків в секторі кеш пам'яті рівна 2, маємо: повна розрядність адреси $n = 30$, причому поле адреси блоків основної пам'яті займає $k = 25$ розрядів, поле номера тега (тобто номера секції в основній пам'яті) $r = 17$ розрядів, поле номера блоку в секції (в кеш пам'яті) $n = 8$ розрядів, поле адреси слова в блоці $m = 5$ розрядів (рис. 10.12b).

Контролер кеш пам'яті за допомогою восьми середніх розрядів слова адреси звертається до визначеного цими розрядами блоку власної пам'яті. Зрозуміло, що шуканий в такий спосіб блок завжди присутній в кеш пам'яті. Адже кеш пам'ять вміщує 512 блоків. Але вміст віднайденого блоку кеш пам'яті може бути копією не одного, а одного з декількох дозволених на копіювання блоків основної пам'яті. Наприклад, до нульового блоку кеш пам'яті дозволено копіювати вміст наступних блоків основної пам'яті: 0, 64, 128, 192, 256 і т. д. Усього до кожного блоку кеш пам'яті можна скопіювати вміст одного з $2^{17} = 131072$ блоків основної пам'яті, оскільки ємність основної пам'яті в 131072 разів перевищує ємність кеш пам'яті. Для того, щоб визначити, чи є поточне наповнення вказаного блоку кеш пам'яті відповідним до запиту процесора, використовують вміст старших 17 бітів адреси основної пам'яті.

Потрібно відзначити, що якраз частково-асоціативне відображення найчастіше використовується в сучасних комп'ютерах, причому кількість блоків в одній секції кеш пам'яті зазвичай не перевищує чотирьох. Основною перевагою тут є те, що кеш пам'ять поєднує переваги пам'яті з довільним та з асоціативним доступом (рис. 10.13).

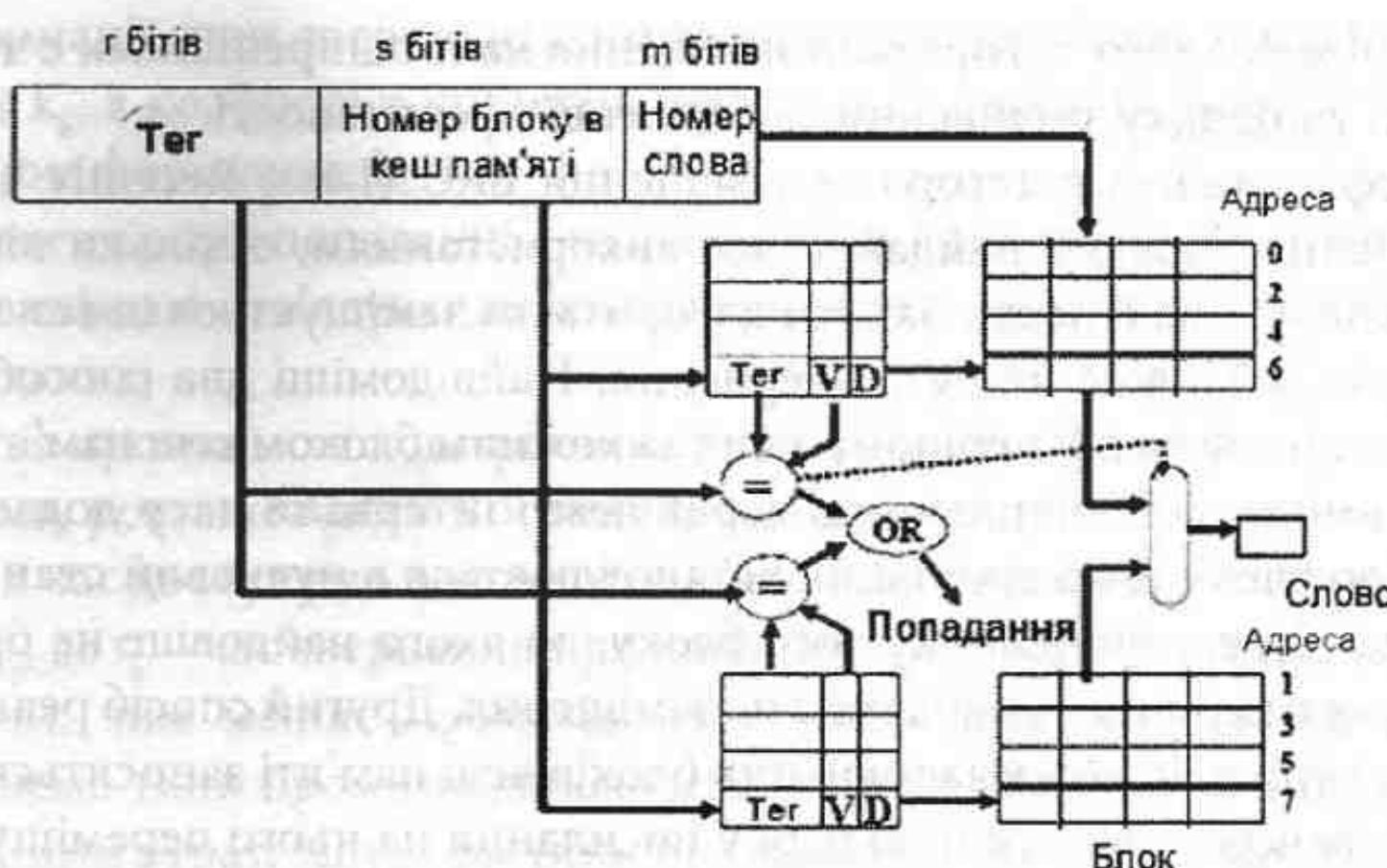


Рис. 10.13. Ідентифікація даних в кеш пам'яті з частково-асоціативним доступом

При даному способі відображення кожен рядок кеш пам'яті вміщує наступну інформацію: тег, який вказує вміст блоку якої секції основної пам'яті переписано до даного блоку кеш пам'яті, розряд достовірності (valid bit V), який вказує, чи вміст даного блоку кеш пам'яті дійсно належить блоку основної пам'яті, вказаному s розрядами поля адреси, розряд модифікації (dirty bit D), який інформує про внесення змін до вмісту блоку кеш пам'яті, а також вміст блоку основної пам'яті, вказаного s розрядами поля адреси. Коли процесору потрібний операнд із блоку з певною адресою, контролер кеш пам'яті вибирає з секторів пам'яті тегів відповідні номеру блоку в кеш пам'яті теги, та порівнює їх з відповідними g розрядами адреси, в яких вказано тег. При наявності в кеш пам'яті відповідного тега, та при однічному значенні розряду достовірності, кеш пам'ять видає сигнал підтвердження попадання та надає доступ до відповідного блоку.

10.2.5. Порядок заміщення блоків в кеш пам'яті з асоціативним відображенням

Для запису з основної пам'яті до кеш пам'яті вмісту нового блоку в ній потрібно знайти блок, вміст якого має бути заміщений. При використанні прямого відображення такий блок є наперед визначенім. При використанні повністю асоціативного та частково-асоціативного відображення, коли в деякий блок в кеш пам'яті може бути записаний вміст довільного блоку, або деякої множини блоків основної пам'яті, потрібно використати відповідний алгоритм заміщення.

Основна мета стратегії заміщення – утримувати в кеш пам'яті вміст блоків основної пам'яті, до яких найбільш імовірні звернення в найближчому майбутньому, і заміщувати вміст блоків, доступ до яких відбудеться пізніше, або взагалі не відбудеться. Очевидно, що оптимальним буде алгоритм, який заміщує вміст того блоку, звернення до якого в майбутньому відбудеться пізніше, ніж до будь-якого іншого блоку кеш пам'яті. На жаль, такий прогноз здійснити практично неможливо, тому використовують ряд стратегій, кожна з яких вигідніша в відповідному конкретному випадку. При цьому, для досягнення високої швидкості заміщення, алгоритми заміщення реалізуються апаратними засобами.

Серед безлічі можливих алгоритмів заміщення найпоширенішими є чотири, що розглядаються далі у порядку зменшення їх відносної ефективності.

Найбільш ефективним є алгоритм заміщення LRU (Least Recently Used), який передбачає заміщення блоків з найдавнішим використанням, оскільки він базується на властивості часової локальності. За цим алгоритмом заміщується вміст того блоку кеш пам'яті, до якого найдовше не було звернення. Найвідоміші два способи апаратурної реалізації цього алгоритму. У першому з них за кожним блоком кеш пам'яті закріплюють лічильник. До вмісту всіх лічильників через певні інтервали часу додається одиниця. При зверненні до блоку його лічильник встановлюється в нульовий стан. Таким чином, найбільше число буде в лічильнику того блоку, до якого найдовше не було звернень, і вміст цього блоку є першим кандидатом на заміщення. Другий спосіб реалізується за допомогою черги, куди у порядку заповнення блоків кеш пам'яті заноситься посилання на ці блоки. При кожному зверненні до блоку посилання на нього переміщується в кінець черги. В результаті першим в черзі кожного разу опиняється посилання на блок, до якого найдовше не було звернень. Вміст саме цього блоку перш за все і заміщується.

Інший можливий алгоритм заміщення – алгоритм, що працює за принципом “перший увійшов, перший вийшов” (FIFO – First In First Out). Тут заміщується вміст блоку, що найдовше знаходився в кеш пам'яті. Алгоритм легко реалізується за допомогою розглянутої раніше черги, з тією лише різницею, що після звернення до блоку положення відповідного посилання в черзі не змінюється.

Ще один алгоритм – заміна вмісту блоку, що найменше використовувався (LFU – Least Frequently Used). Заміщується вміст того блоку в кеш пам'яті, до якого було менше всього звернень. Принцип можна реалізувати, пов'язавши кожен блок з лічильником звернень, до вмісту якого після кожного звернення додавати одиницю. Головним претендентом на заміщення є вміст блоку, лічильник якого містить найменше число.

Простий алгоритм заміщення – довільний вибір блоку для заміни його вмісту. Блок, вміст якого замінюється, вибирається випадковим чином. Це може бути реалізовано, наприклад, за допомогою лічильника, вміст якого збільшується на одиницю з кожним тактовим імпульсом, незалежно від того, що мало місце – попадання чи промах. Значення в лічильнику визначає блок, вміст якого буде замінено в повністю асоціативній кеш пам'яті, або воно визначає блок в межах сектора, вміст якого буде замінено в частково-асоціативній кеш пам'яті. Даний алгоритм використовується вкрай рідко.

Частіше попереднього використовують алгоритм випадкового заміщення вмісту блоків кеш пам'яті за значенням лічильника випадкових чисел, оскільки за ефективністю він є близьким до алгоритму LRU та простішим за нього в реалізації.

10.2.6. Підвищення ефективності кеш пам'яті

Характеристики, які використовують при оцінці ефективності ієрархічної організації пам'яті, підходять і для оцінки ефективності кеш пам'яті. До цих характеристик належать наступні:

- коефіцієнт попадань – відношення числа попадань до загального числа звернень процесора до основної пам'яті, де під попаданням розуміється виявлення в кеш пам'яті потрібної інформації, при зверненні до основної пам'яті;
- коефіцієнт промахів – відношення числа промахів до загального числа звернень процесора до основної пам'яті, де під промахом розуміється відсутність в кеш пам'яті

потрібної інформації, при зверненні до основної пам'яті. Якщо позначити коефіцієнт попадань через k_h , а коефіцієнт промахів через k_m , то залежність між ними можна виразити наступною формулою: $k_m = 1 - k_h$;

- час звернення при попаданні – час, необхідний для пошуку потрібної інформації в кеш пам'яті (включаючи з'ясування, чи є звернення попаданням), плюс час на фактичне зчитування даних;
- втрати на промах – час, потрібний для заміни блоку в кеш пам'яті на блок з потрібними даними, розташований в основній пам'яті;
- середній час доступу до основної пам'яті, який визначається з виразу:

$$t_{av} = t_h + k_m t_p, \text{ де } t_h \text{ – час звернення при попаданні, } t_p \text{ – втрати на промахах.}$$

Таким чином, чим менше промахів, тим вища ефективність використання кеш пам'яті. Розглянемо типи промахів до кеш пам'яті та підходи до зменшення їх кількості й пов'язаних з ними втрат. Існує три типи промахів: обов'язкові, ємнісні та конфліктні.

Обов'язкові промахи – це промахи, які виникають при початковому зверненні до основної пам'яті. Зрозуміло, що перший доступ до кеш пам'яті обов'язково буде промахом. Для зменшення кількості цього типу промахів потрібно збільшувати розмір блоків в кеш пам'яті.

Ємнісні промахи пов'язані з обмеженим розміром кеш пам'яті. Для зменшення кількості цього типу промахів потрібно збільшувати розмір кеш пам'яті.

Конфліктні промахи пов'язані з ефективністю алгоритму заміщення блоків в кеш пам'яті. Для зменшення кількості цього типу промахів потрібно збільшувати асоціативність кеш пам'яті, аж до використання для її реалізації повністю асоціативної пам'яті.

На рис. 10.14 наведено залежність величини коефіцієнта промахів від ємності кеш пам'яті та від кількості блоків в одному секторі кеш пам'яті з використанням частково-асоціативного відображення при виконанні набору тестових програм SPEC 92.

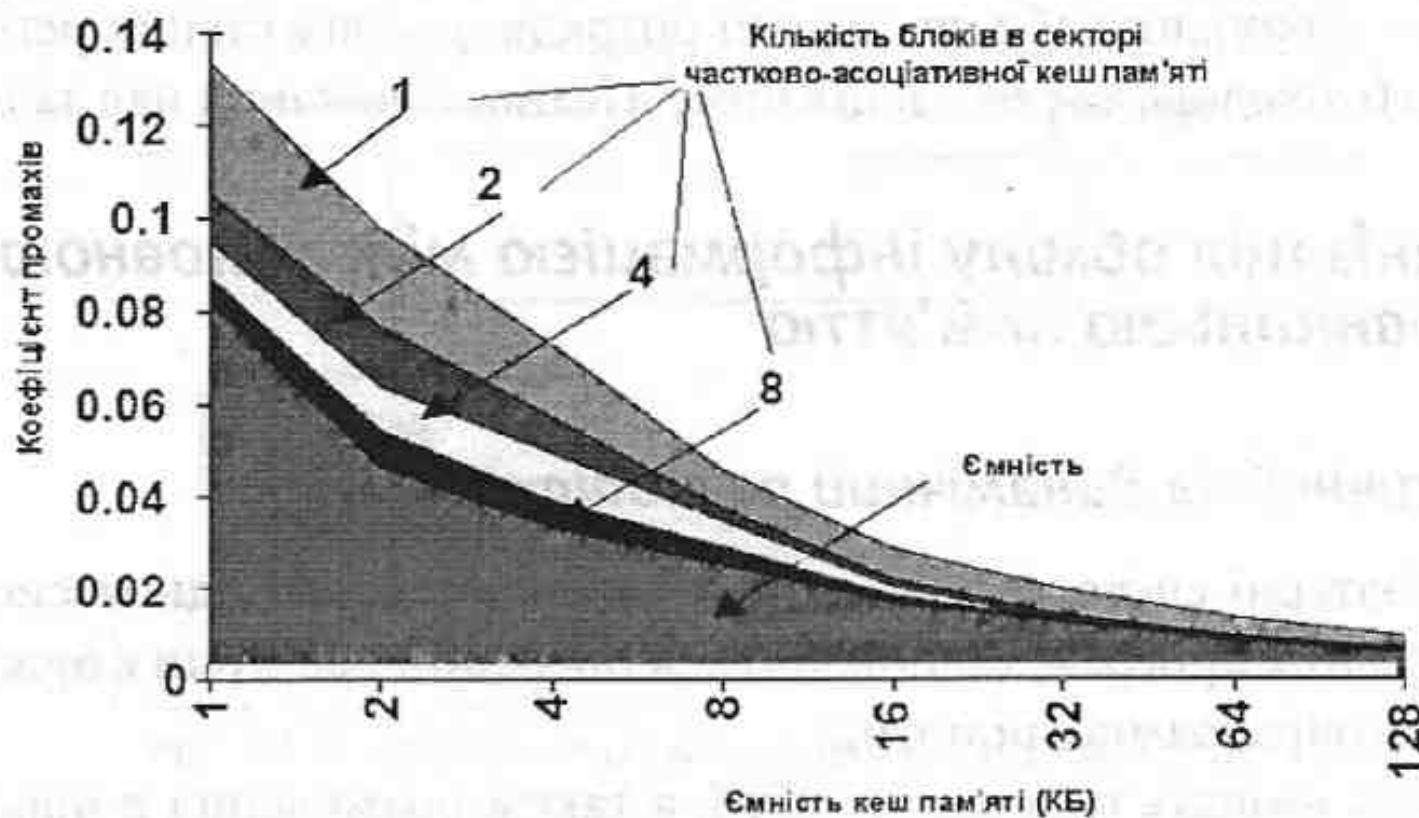


Рис. 10.14. Залежність значення коефіцієнта промахів від ємності та від кількості блоків в одному секторі частково-асоціативної кеш пам'яті

Таким чином, збільшення розміру блоку кеш пам'яті понижує кількість обов'язкових промахів, але збільшує втрати на промах, через зростання часу пересилання блоку інформації. Крім того, якщо ємність кеш пам'яті невелика, то збільшення розміру блоку збільшує кількість конфліктних промахів через зменшення кількості блоків та секторів.

Подібним чином збільшення ємності кеш пам'яті зменшує кількість ємнісних промахів, але збільшує час звернення при попаданні.

Для зменшення кількості конфліктних промахів потрібно збільшувати асоціативність кеш пам'яті, але, як показує практика, більша 4 кількість блоків в одному секторі кеш пам'яті з використанням частково-асоціативного відображення збільшує затрати обладнання без видимого виграшу, та збільшує час звернення при попаданні.

Часто для суттєвого зменшення кількості конфліктних промахів, в першу чергу для кеш пам'яті з прямим відображенням, використовують додаткову кеш пам'ять малого розміру (8-16 блоків), в якій зберігаються останні заміщені в кеш пам'яті блоки.

Для зменшення значення коефіцієнта промахів використовується попередня вибірка з основної пам'яті блоків, до яких може бути звернення. При цьому можуть засновуватись як апаратні, так і програмні засоби попередньої вибірки. При використанні апаратних засобів зазвичай при промаху додатково до вибірки відсутнього в кеш пам'яті блоку вибирається наступний, або кілька наступних блоків. Це дає виграну, оскільки процесор не чекає на попередньо вибраний блок. Такий підхід може бути використаний як до кеш пам'яті даних, так і команд, але є більш ефективним для кеш пам'яті команд. При використанні програмних засобів до програм, або до компілятора, додаються спеціальні команди попередньої вибірки.

Для зменшення значення коефіцієнта промахів використовуються також спеціальні підходи по оптимізації компіляторів, зокрема об'єднання даних та команд у блоки, об'єднання циклів, черговість циклів, злиття масивів і т. д.

В останніх комп'ютерах для підвищення ефективності кеш пам'яті вводяться більш складні механізми її функціонування за рахунок ускладнення контролера та організації внутрішньої пам'яті. Це, зокрема, створення так званої неблокуючої кеш пам'яті, яка при наявності промаху продовжує взаємодію з процесором, використовуючи механізм невпорядкованого виконання або спеціальні розряди в асоціативних реєстрах, будучи при цьому багато блоковою, та реалізація пріоритетності читання над записом.

10.3. Організація обміну інформацією між основною та зовнішньою пам'яттю

10.3.1. Статичний та динамічний розподіл пам'яті

Сучасні комп'ютерні системи є багатопрограмними. В них одночасно виконується велика кількість різних програм. Відповідно й їх операційні системи є орієнтованими на забезпечення багатопрограмної роботи.

Через обмежену ємність основної пам'яті, а також розміщення в ній, крім програм користувача, програм операційної системи, всі програми в багатопрограмних комп'ютерних системах не можуть розміщуватись в основній пам'яті.

Але в цьому немає великої необхідності, так як в певний момент часу виконується не вся програма, а певна її частина. Тому основна пам'ять використовується для зберігання активних частин виконуваних програм, а решта інформації зберігається в зовнішній пам'яті. При цьому може виникнути необхідність заміни тих частин виконуваних програм, які стали неактивними. Звідси випливає потреба розподілу пам'яті між програмами.

Якщо вся необхідна основна пам'ять для програм призначається до початку їх виконання – це статичний розподіл пам'яті. Цей тип розподілу основної пам'яті використовувався в перших комп'ютерах. Кожний програміст наперед замовляв потрібну йому ємність основної пам'яті. Зрозуміло, що статичний розподіл основної пам'яті не є досконалим. З одного боку, неефективно використовується пам'ять, так як важко наперед передбачити потрібну ємність основної пам'яті, а з іншого боку, на програміста покладається проблема заміни інформації в основній пам'яті.

В сучасних багатопрограмних комп'ютерних системах використовується динамічний розподіл пам'яті, відповідно до якого основна пам'ять розподіляється між програмами під час їх виконання. Для цього при підготовці цільових програм використовуються умовні адреси, а в процесі виконання програми комп'ютер перетворює умовні адреси в виконавчі, виділяючи програмі місце в пам'яті. Найчастіше це здійснюється з використанням апаратних засобів, щоб прискорити звернення до пам'яті. Існує декілька способів динамічного розподілу пам'яті. Далі розглянемо два методи динамічного розподілу основної пам'яті: за допомогою базових адрес та сторінкової організації.

10.3.2. Розподіл основної пам'яті за допомогою базових адрес

Відповідно до методу динамічного розподілу пам'яті за допомогою базових адрес кожній програмі виділяється деяка область основної пам'яті, і, крім того, програма пишеться з використанням умовних адрес. Для цього кожній програмі ставиться у відповідність базова адреса, що встановлюється операційною системою. При всіх зверненнях до пам'яті базова адреса додається на суматорі СМ (рис. 10.15) до умовної адреси, визначеній програмою, утворюючи виконавчу адресу пам'яті.



Рис. 10.15. Розподіл пам'яті за допомогою базових адрес

Як тільки деяка програма стає активною, операційна система знаходить для неї місце в пам'яті і встановлює в реєстрі базової адреси відповідне число (базову адресу).

Недоліки розподілу пам'яті за допомогою базових адрес:

- необхідність розміщувати програму в послідовних комірках основної пам'яті, що призводить до неефективного її використання;
- необхідність вводити програми в основну пам'ять повністю (або ті її частини, в яких використовується одна базова адреса);

- фрагментація пам'яті, тобто її поділ на використовувані і не використовувані частини (іншими словами – наявність в пам'яті дірок).

Як приклад на рис. 10.16а наведено розподіл основної пам'яті між трьома програмами А, В, С в деякий момент часу. Після завершення виконання програми В (рис. 10.16б) в пам'яті вивільнилося місце, яке разом з областю пам'яті D (вільні області В та D виділені штриховими лініями) є більшим за розміром, ніж потрібно програмі Е, однак ця програма не може бути записана до основної пам'яті, оскільки вона є більшою за кожну з областей В та Е, які розміщені в пам'яті не підряд.

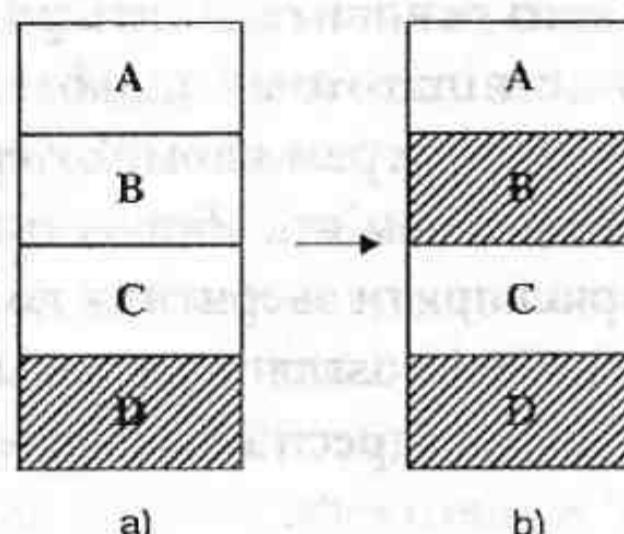


Рис. 10.16. Області основної пам'яті В та D є дірками, оскільки програма Е не вставляється на їх місце

Виходом з цієї ситуації є, наприклад, переміщення програми із області С ближче до області А, що дозволяє вивільнити місце програмі Е, як це показано, наприклад, на рис. 10.17а.

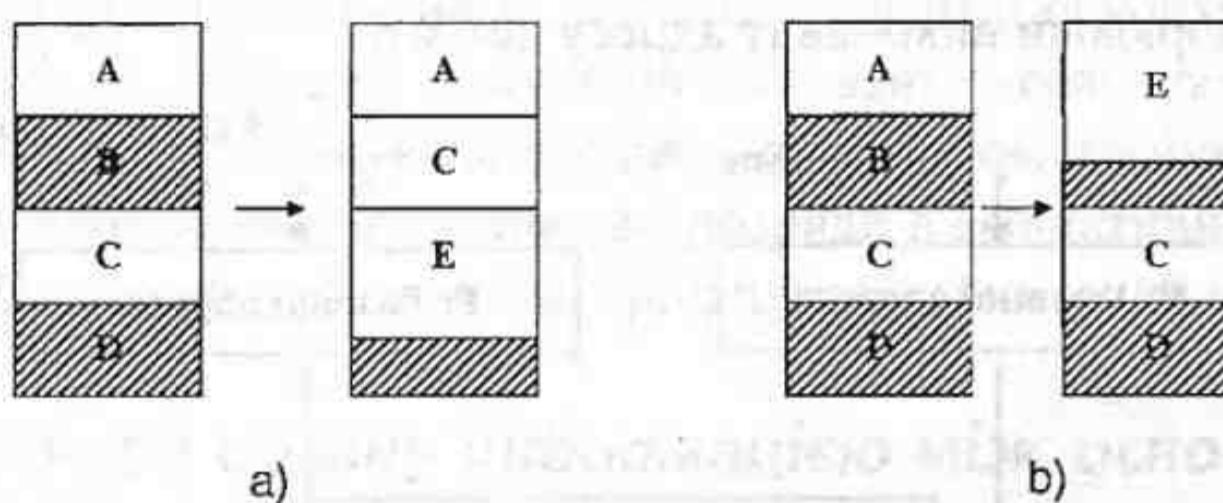


Рис. 10.17. Переміщення програми із області С а) та очікування місця для програми Е б)

Однак це пов'язано з втратами часу на переміщення програми, а часто і з необхідністю редагування програми і завантаження її заново, оскільки змінилися адреси.

Іншим варіантом є чекання, доки не звільниться область необхідної ємності, як це показано на рис. 10.17б, коли завершилось виконання програми А і з'явилось достатньо місця для програми Е. Виходить, що програма чекає своєї черги на завантаження до основної пам'яті, хоча є ділянки пам'яті великого розміру, але вони розміщені в різних областях основної пам'яті.

Таким чином, при використанні описаного способу розподілу в основній пам'яті завжди будуть наявні дірки, які в сумі можуть займати значну частину її ємності.

Відмічені недоліки методу динамічного розподілу пам'яті за допомогою базових адрес відсутні в другого методу динамічного розподілу пам'яті, а саме в віртуальній пам'яті з сторінковою організацією.

10.3.3. Віртуальна пам'ять

Принцип віртуальної пам'яті передбачає, що користувач при підготовці програми має справу не з багаторівневою фізичною основною пам'яттю, що є в комп'ютері і має фіксовану ємність, а з віртуальною (увяною) однорівневою пам'яттю, ємність якої дорівнює всьому адресному простору комп'ютера, визначеному розміром адрес в реєстрі адреси.

Користувач розпоряджається всім адресним простором комп'ютера, незалежно від ємності основної пам'яті і вимог до неї з боку інших користувачів. На всіх етапах підготовки програма представлена в віртуальних (умовних) адресах. При виконанні програм віртуальні адреси перетворюються у фізичні. Користувач не знає, де знаходиться його програма – в основній чи в зовнішній пам'яті, і в якій області. Це встановлюється автоматично в ході обчислювального процесу, тобто шляхом динамічного розподілу пам'яті. При цьому, оскільки для виконання програми необхідно, щоб та частина активної інформації, на яку вказують віртуальні адреси, знаходилась в основній пам'яті, то в процесі виконання програми необхідно перетворювати віртуальні адреси в фізичні та переписувати активну інформацію до основної пам'яті.

За своєю суттю віртуалізація пам'яті – це спосіб реалізації ієархічної організації пам'яті на рівні взаємодії основної та зовнішньої пам'яті.

Віртуальна організація пам'яті дозволяє здійснювати керування пам'яттю, коли виконується паралельно багато програм, з забезпеченням захисту даних та наданням в розпорядження кожної програми всього адресного простору комп'ютера.

При віртуальній організації пам'яті адреси, які формуються процесором, є віртуальними. Необхідним атрибутом віртуальної пам'яті є зовнішня пам'ять, яка здатна зберігати всю програму повністю.

Розглянемо приклад. Нехай процесор посилає до основної пам'яті віртуальну адресу 520 004 096, причому ємність основної пам'яті рівна 512МБ, а ємність зовнішньої пам'яті рівна 80 ГБ (рис. 10.18).



Рис. 10.18. Взаємодія процесора з основною пам'яттю при використанні віртуальної пам'яті

В комп'ютерах без застосування віртуальної пам'яті використовується пряма адресація. Тому звертання за вказаною адресою викликало б програмне переривання з таким повідомленням: "Адресована неіснуюча область пам'яті", оскільки вказана адреса виходить за межі основної пам'яті. В комп'ютерах з віртуальною пам'яттю виконується така послідовність кроків:

- віртуальна адреса перетворюється у фізичну;
- фізична адреса подається до фізичної пам'яті (основної або зовнішньої);
- в фізичній пам'яті знаходиться відповідна фізичній адресі комірка;
- до цієї комірки записується, або з неї читається, дане, причому, якщо ця комірка знаходиться в зовнішній пам'яті, то звернення до неї здійснюється через основну пам'ять шляхом перезапису інформації з зовнішньої пам'яті до основної.

10.3.4. Сторінкова організація пам'яті

10.3.4.1. Основні правила сторінкової організації пам'яті

Для спрощення перетворення віртуальних адресів в фізичні і позбавлення від фрагментації пам'яті, а крім того для прискорення обміну між основною та зовнішньою пам'яттю та вирішення питань захисту пам'яті, а також з метою реалізації принципу віртуальної пам'яті, використовується сторінкова організація пам'яті. Основні правила сторінкової організації пам'яті, які забезпечують ефективну взаємодію основної та зовнішньої пам'яті, є наступними:

- Програми представляють в віртуальних (умовних) адресах віртуальної однорівневої пам'яті, ємність якої дорівнює всьому адресному простору комп'ютера, визначеному розміром адрес в реєстрі адреси.
- Віртуальна пам'ять та фізична (основна та зовнішня) пам'ять розбиваються на сторінки рівного об'єму.
- Сторінка – це деяка множина послідовно розміщених комірок пам'яті.
- Сторінкам присвоюються номери.
- Кожна фізична сторінка здатна зберігати вміст однієї віртуальної сторінки.
- Нумерація комірок в обох сторінках (віртуальних і фізичних) однаакова.
- Базовою порцією інформації, яка переміщується між основною та зовнішньою пам'яттю, є вміст однієї сторінки.
- Ідентичність вмісту відповідних сторінок основної пам'яті і зовнішньої пам'яті забезпечується використанням спеціальних методів оновлення вмісту сторінок основної пам'яті.
- Заміщення вмісту сторінок в основній пам'яті вмістом сторінок з зовнішньої пам'яті здійснюється за правилами, які називають алгоритмом заміщення.
- Взаємодія між сторінками віртуальної і фізичної пам'яті задається сторінковою таблицею.
- Між різними процесами (програмами) розподіляється фізична (а не віртуальна) пам'ять.
- Основна пам'ять динамічно розподіляється між різними процесами (програмами) посторінково.

Переваги сторінкової організації пам'яті:

- не вимагаються додаткові переміщення програм в пам'яті, пов'язані з потребою звільнення місця через фрагментацію;
- скорочується кількість пересилань між основною та зовнішньою пам'яттю, так як вміст сторінки завантажується лише при необхідності;

зменшуються вимоги до ємності основної пам'яті, так як до неї завантажуються лише активні частини програми.

Про сторінкову організацію пам'яті говорять, що вона є прозорою, тобто програміст може працювати, не звертаючи уваги на факт її існування.

10.3.4.2. Реалізація сторінкової організації пам'яті

Відповідно до наведених вище правил сторінкової організації пам'яті спочатку потрібно розділити віртуальну та фізичну пам'ять на сторінки рівного об'єму, присвоїти цим сторінкам номери, та встановити відповідність між цими сторінками, як це для прикладу показано на рис. 10.19, де об'єм сторінки вибрано рівним 4КБ, а відповідні віртуальним сторінкам A, B, C, D фізичні сторінки розміщені як в основній, так і в зовнішній пам'яті. При цьому відразу необхідно зауважити, що довільній віртуальній сторінці може відповідати довільна фізична сторінка, тобто відповідність між сторінками є повністю асоціативною.

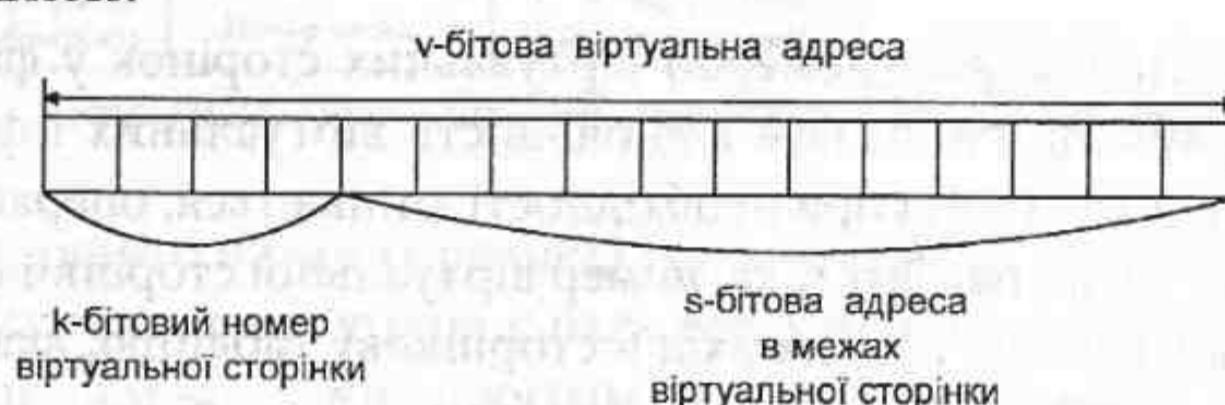


Рис. 10.19. Поділ віртуальної та фізичної пам'ятей

Нехай віртуальна пам'ять має ємність $V=2^v$ слів, та поділена на 2^k віртуальних сторінок, кожна з яких має об'єм 2^s слів, тобто розрядність віртуальної адреси $v = k + s$, де k – адреса (номер) віртуальної сторінки, s – номер слова в сторінці. Тоді формат віртуальної адреси буде мати вид, представлений на рис. 10.20.



Рис. 10.20. Формат віртуальної адреси

Аналогічно, нехай основна пам'ять має ємність $M=2^m$ слів, та поділена на 2^l віртуальних сторінок, кожна з яких має об'єм 2^s слів, тобто розрядність адреси основної пам'яті $m = l + s$, де l – адреса (номер) сторінки основної пам'яті, s – номер слова в сторінці. Тоді формат адреси основної пам'яті буде мати вид, представлений на рис. 10.21.

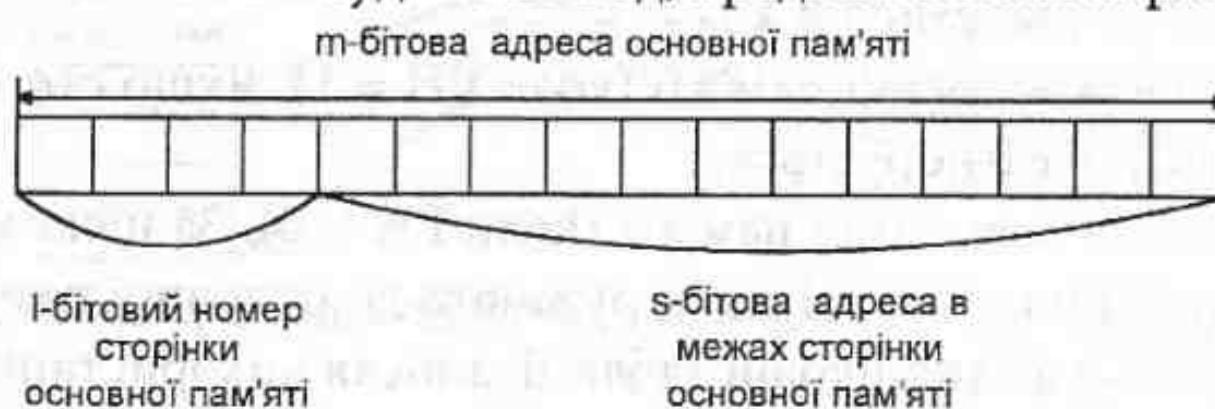


Рис. 10.21. Формат адреси основної пам'яті

Подібним чином, нехай зовнішня пам'ять має ємність $E=2^e$ слів, та поділена на 2^p віртуальних сторінок, кожна з яких має об'єм 2^s слів, тобто розрядність адреси основної пам'яті $e = p + s$, де p – адреса (номер) сторінки зовнішньої пам'яті, s – номер слова в сторінці. Тоді формат адреси зовнішньої пам'яті буде мати вид, представлений на рис. 10.22.

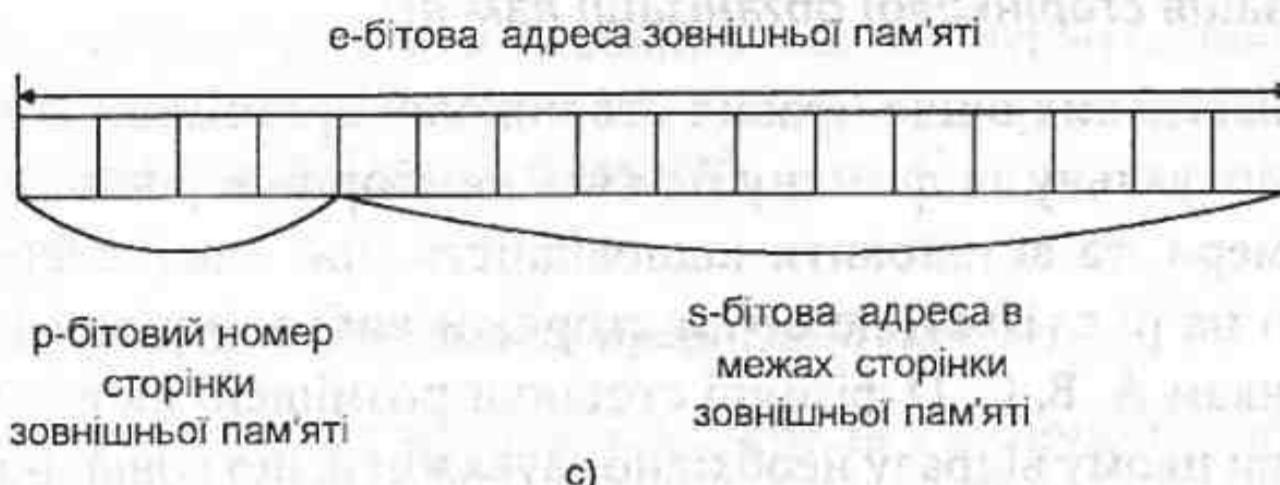


Рис. 10.22. Формат адреси зовнішньої пам'яті

Порядок перетворення адрес (номерів) віртуальних сторінок у фізичні задається у вигляді сторінкової таблиці, що вказує відповідність віртуальних і фізичних сторінок. Сторінкова таблиця формується, і при необхідності змінюється, операційною системою.

Процедура звернення до пам'яті така: номер віртуальної сторінки вибирається з віртуальної адреси і використовується як вхід в сторінкову таблицю, яка вказує номер фізичної сторінки (рис. 10.23).

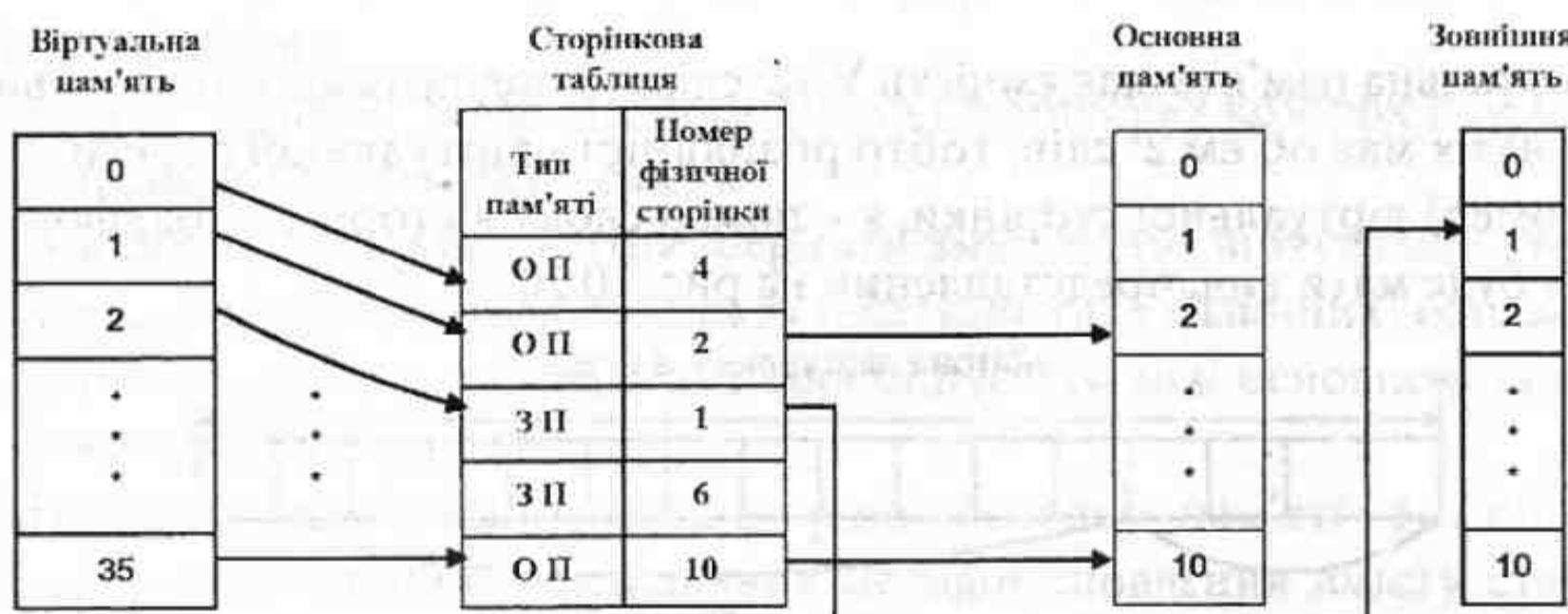


Рис. 10.23. Процедура звернення до пам'яті

Слово сторінкової таблиці складається з трьох основних полів (про допоміжні поля буде сказано пізніше):

- перше поле, яке займає один розряд, вказує тип пам'яті. Коли в цьому полі записано 0, це говорить про те, що сторінка відсутня в основній пам'яті. Коли записано 1 – це говорить про те, що сторінка наявна в основній пам'яті. Цей розряд зазвичай називають розрядом наявності (RH, або valid bit V).
- адреса сторінки в основній пам'яті (коли RH = 1). Якщо сторінка знаходиться в зовнішній пам'яті – це поле ігнорується.
- адреса сторінки в зовнішній пам'яті (коли RH = 0). За цією адресою вибирають вміст сторінки із зовнішньої пам'яті та загружають до основної пам'яті. Крім того, робиться відповідний запис в сторінковій таблиці, а після використання ця сторінка буде повернута назад в зовнішню пам'ять.

Перетворення віртуальної адреси у фізичну виконується так, як показано на рис. 10.24. Тобто номер віртуальної сторінки за допомогою сторінкової таблиці замінюється на номер фізичної сторінки, а номер слова в межах сторінки залишається без змін, причому залежно від значення розряду наявності (RH) формується або адреса основної пам'яті, або адреса зовнішньої пам'яті.

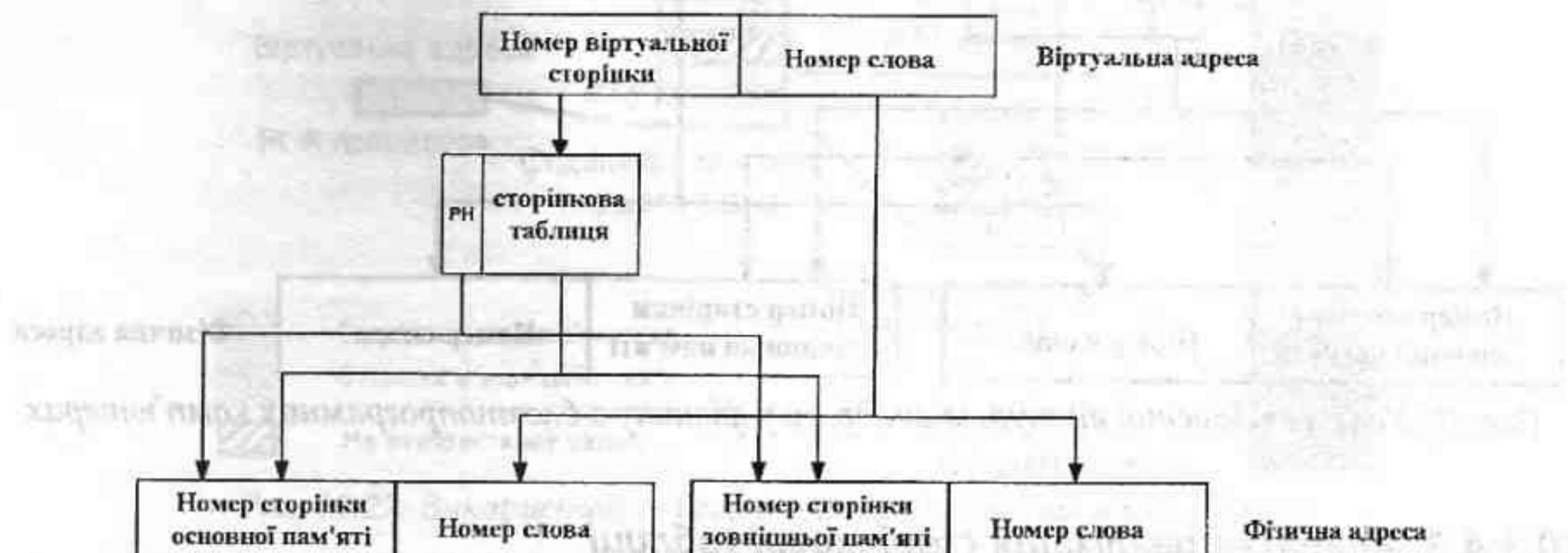


Рис. 10.24. Перетворення віртуальної адреси у фізичну

Вище було розглянуто питання розподілу пам'яті, коли виконується лише одна програма. Насправді сучасні комп'ютери є багатопрограмними. В цьому випадку при сторінковій організації пам'яті кожна програма має свою віртуальну пам'ять з адресацією від нуля і до максимальної адреси, яка допускається розрядною сіткою. Так як написання різних програм проводиться незалежно одна від одної, вони можуть виконуватись в одних і тих самих віртуальних адресах, які повинні належати до різних фізичних сторінок. Отже, сторінкова таблиця повинна враховувати також належність віртуальної сторінки різним програмам. Тобто номер фізичної сторінки є функцією двох змінних – номера віртуальної сторінки, та номера програми, як це показано на рис. 10.25. Іншими словами, для кожної програми має бути своя сторінкова таблиця.

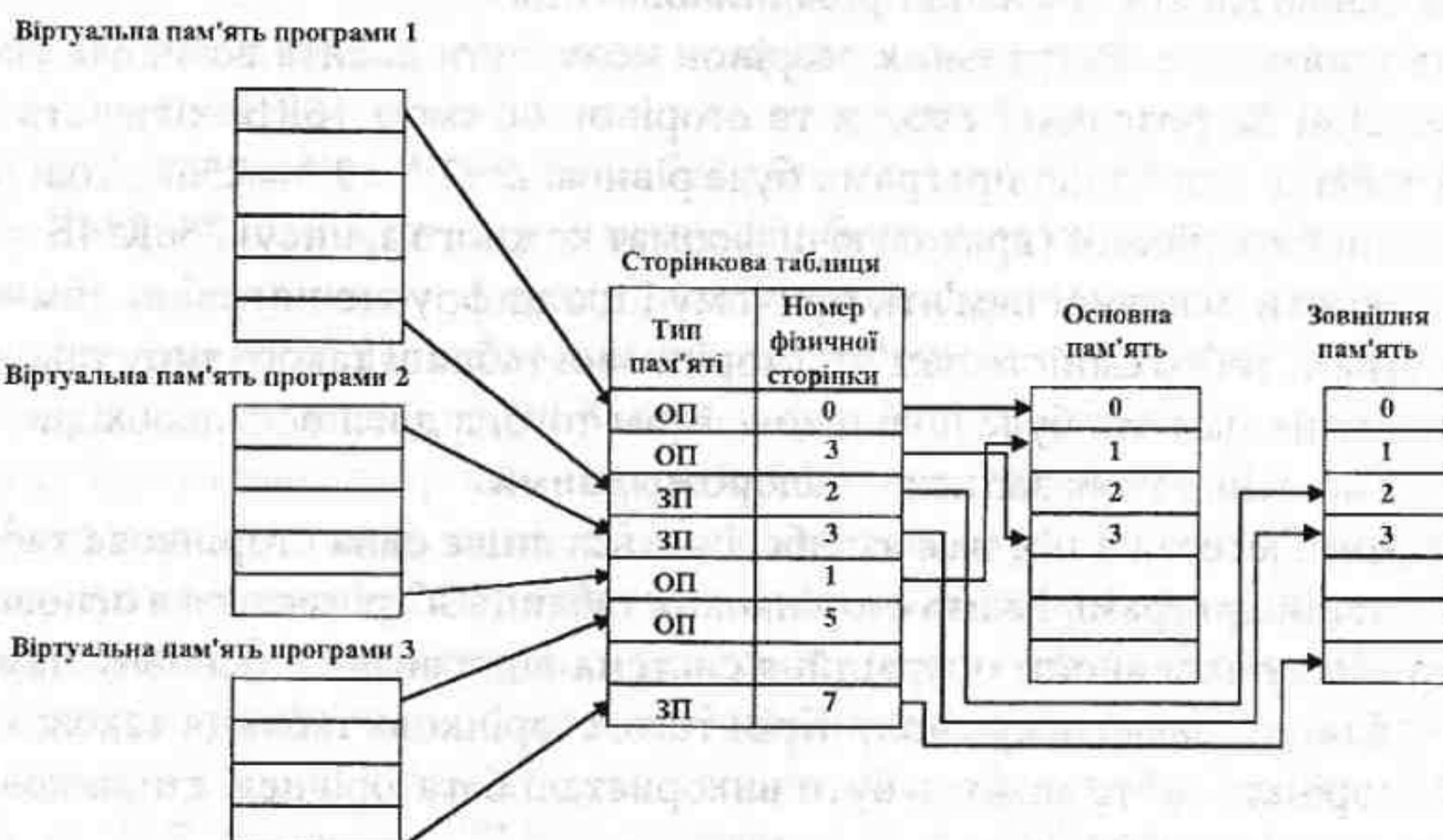


Рис. 10.25. Процедура звернення до пам'яті в багатопрограмних комп'ютерах

Перетворення віртуальної адреси у фізичну тут здійснюється відповідно до рис. 10.26.

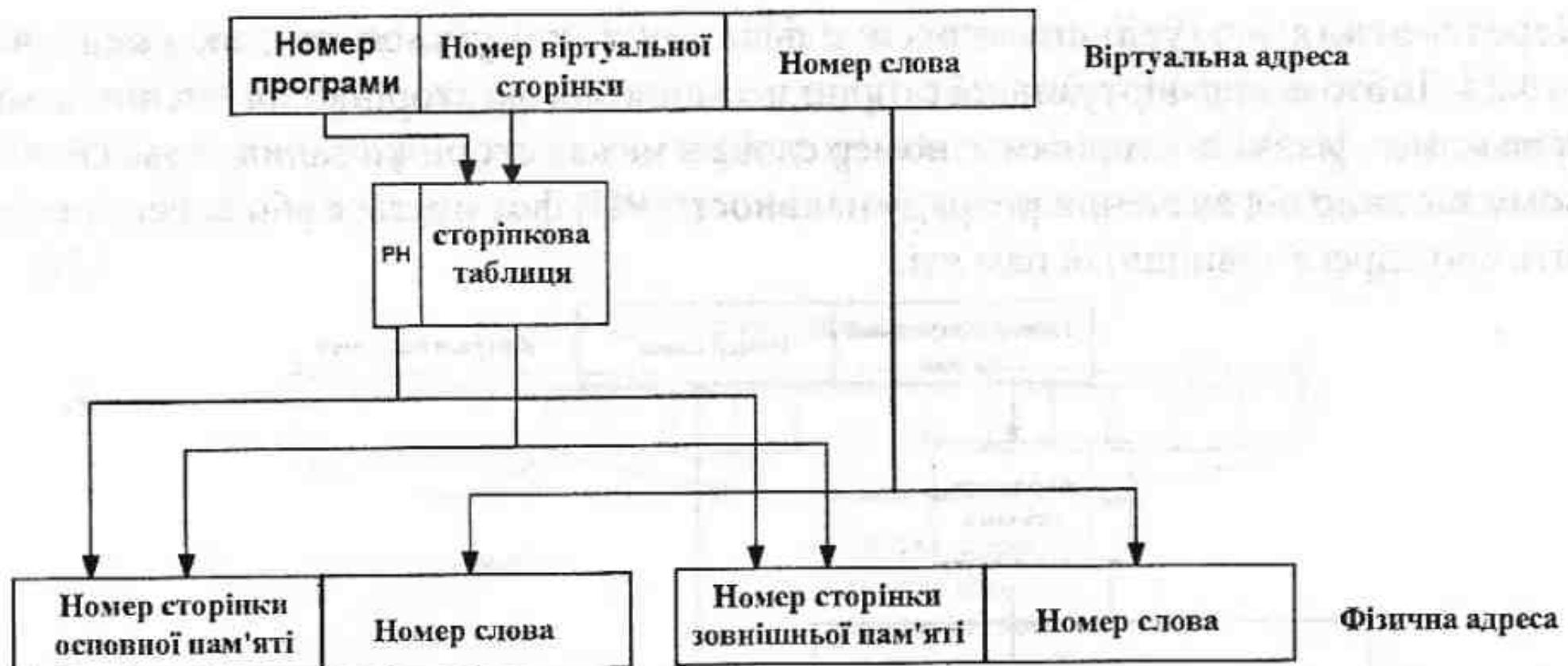


Рис. 10.26. Перетворення віртуальної адреси у фізичну в багатопрограмних комп'ютерах

10.3.4.3. Апаратна реалізація сторінкової таблиці

Вибір слова з пам'яті при сторінковій організації вимагає двох циклів звернення: спочатку до сторінкової таблиці, а потім до фізичної адреси пам'яті. Якщо сторінкова таблиця зберігається в основній пам'яті, це в два рази сповільнює вибірку слова. Тому часто сторінкову таблицю зберігають в швидкій пам'яті невеликої ємності. Склад і можливості цієї пам'яті залежать від вираного правила побудови сторінкової таблиці:

- кожній віртуальній сторінці відповідає одна стрічка сторінкової таблиці. В стрічці є номер фізичної сторінки, який зберігає дану віртуальну сторінку;
- кожній фізичній сторінці відповідає одна стрічка сторінкової таблиці, що зберігається в даній фізичній сторінці.

В першій структурі входом в сторінкову таблицю є номер віртуальної сторінки і номер програми, як це показано на рис. 10.26, і вона будеться як пам'ять з довільним доступом на основі мікросхем напівпровідникової пам'яті.

Кількість програм та віртуальних сторінок може бути досить великою. Наприклад, при використанні 32-розрядної адреси та сторінок об'ємом 16КБ, кількість записів в сторінковій таблиці для однієї програми буде рівною $2^{32}/2^{14} = 2^{18} = 256\text{K}$. Тоді об'єм сторінкової таблиці буде рівним (враховуючи формат кожного запису) $256\text{K} \cdot 4\text{Б} = 1\text{МБ}$, що складає 64 сторінки основної пам'яті, причому і цю цифру ще потрібно помножити на кількість програм. Тобто ємність пам'яті сторінкової таблиці такого типу також буде великою, причому ця пам'ять буде повільною. Крім того, в дійсності необхідно зберігати значно менше записів, так як записи є малорозрядними.

В деяких комп'ютерах в цій пам'яті зберігається лише одна сторінкова таблиця, яка належить активній програмі. решта сторінкових таблиць зберігаються в основній пам'яті. При зміні циклу активності операційна система відправляє в основну пам'ять дану сторінкову таблицю і завантажує нову. Крім того, сторінкова таблиця також може бути поділена на сторінки, тобто можуть бути використані багаторівневі сторінкові таблиці, як це показано на рис. 10.27.



Рис. 10.27. Використання багаторівневих сторінкових таблиць

Тут спочатку іде звернення до сторінкової таблиці першого рівня, а з неї до сторінкової таблиці другого рівня. Як видно з рисунка, взамін однієї сторінкової таблиці об'ємом 2^{20} записів (кількість записів рівна кількості сторінок в пам'яті), тут потрібна одна сторінкова таблиця першого рівня з 1024 записами, та 1024 сторінкові таблиці другого рівня з такою ж кількістю записів. Однак при цьому додається ще одне звернення до пам'яті.

В другій структурі довжина сторінкової таблиці визначається лише кількістю фізичних сторінок в основній пам'яті і не залежить від числа цільових програм. Ємність пам'яті зменшується, проте ускладнюється структура – застосовується асоціативна пам'ять, що дозволяє вибирати дані за їх змістом, а не за місцезнаходженням. Але, як було показано вище, кількість фізичних сторінок в сучасних комп'ютерах є досить великою, що робить проблематичним застосування асоціативної сторінкової таблиці. В зв'язку з цим, в сучасних комп'ютерах поступають наступним чином: виділяють сторінкові таблиці області в основній пам'яті, хоча це приводить до подвоєння часу звернення, та використовують додаткову кеш пам'ять, яку називають буфером перетворення з передісторією (TLB – Translation Look-aside Buffer). Використання TLB дозволяє зменшити час перетворення віртуальних адрес у фізичні. Він будується як кеш пам'ять та може бути організованим за принципом прямого, асоціативного та частково-асоціативного відображення. При кожному перетворенні номера віртуальної сторінки в номер фізичної сторінки результат заноситься в TLB: номер фізичної сторінки до асоціативного реєстра, а номер віртуальної сторінки – до реєстрів тегів. Таким чином, до TLB попадають результати декількох останніх операцій перетворення адрес. При кожному зверненні до основної пам'яті спочатку потрібна віртуальна сторінка шукається в TLB, і при попаданні номер відповідної фізичної сторінки береться з цього буфера. Якщо зафіксовано промах, то процедура перетворення адрес здійснюється за допомогою сторінкової таблиці, яка зберігається в основній пам'яті, після чого номер віртуальної та фізичної сторінок заноситься до TLB. Структура TLB, побудованої як повністю асоціативна кеш пам'ять, наведена на рис. 10.28.

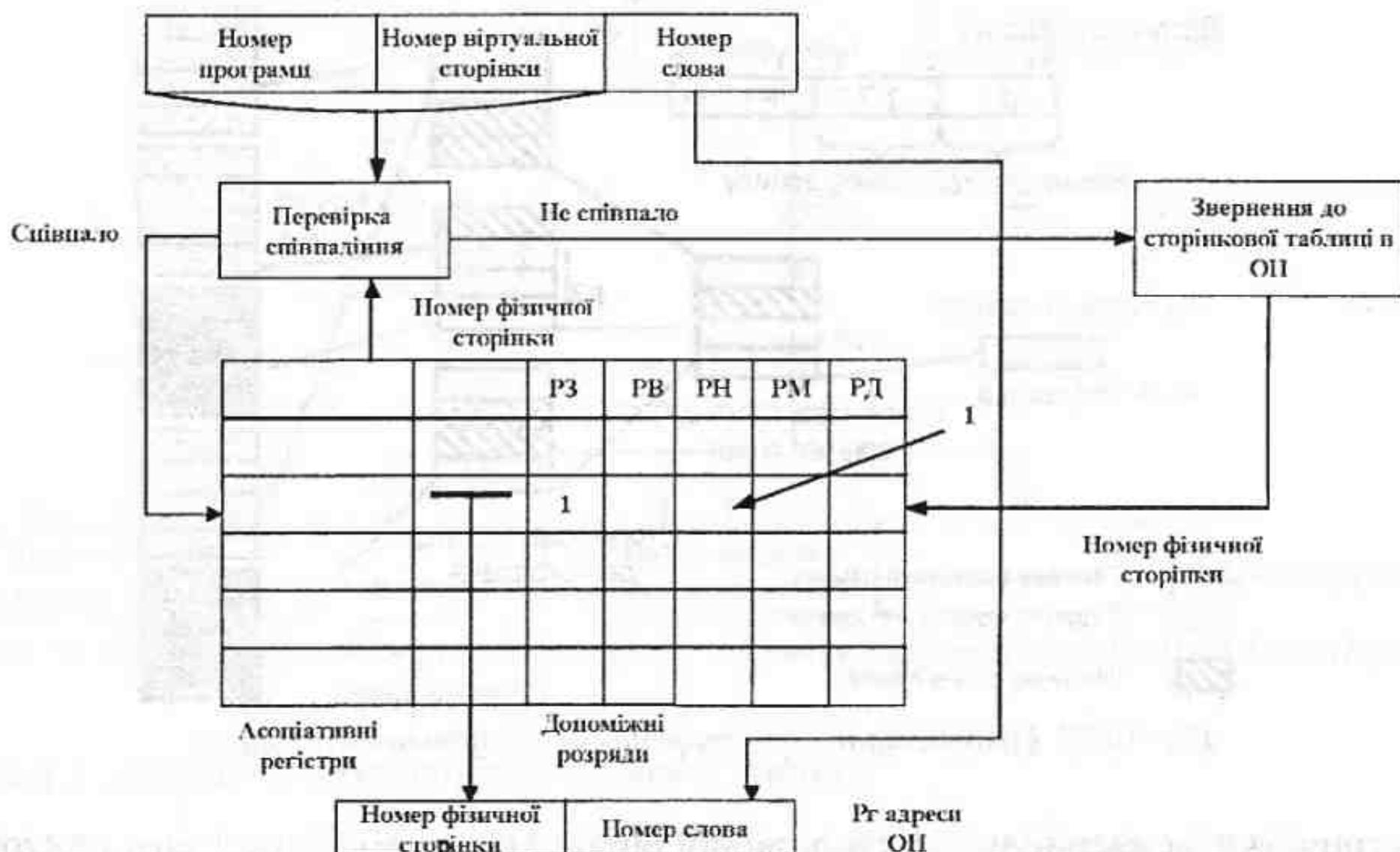


Рис. 10.28. Структура асоціативної кеш пам'яті буфера перетворення з передісторією

Зазвичай число записів (входів) в TLB є невеликим (64-256), наприклад в комп'ютері Pentium III є 64 входи, при розмірі сторінки 4 КБ, що дозволяє отримати швидкий доступ до пам'яті ємністю 256 КБ.

Структура TLB, побудованої як частково-асоціативна кеш пам'ять, наведена на рис. 10.29.

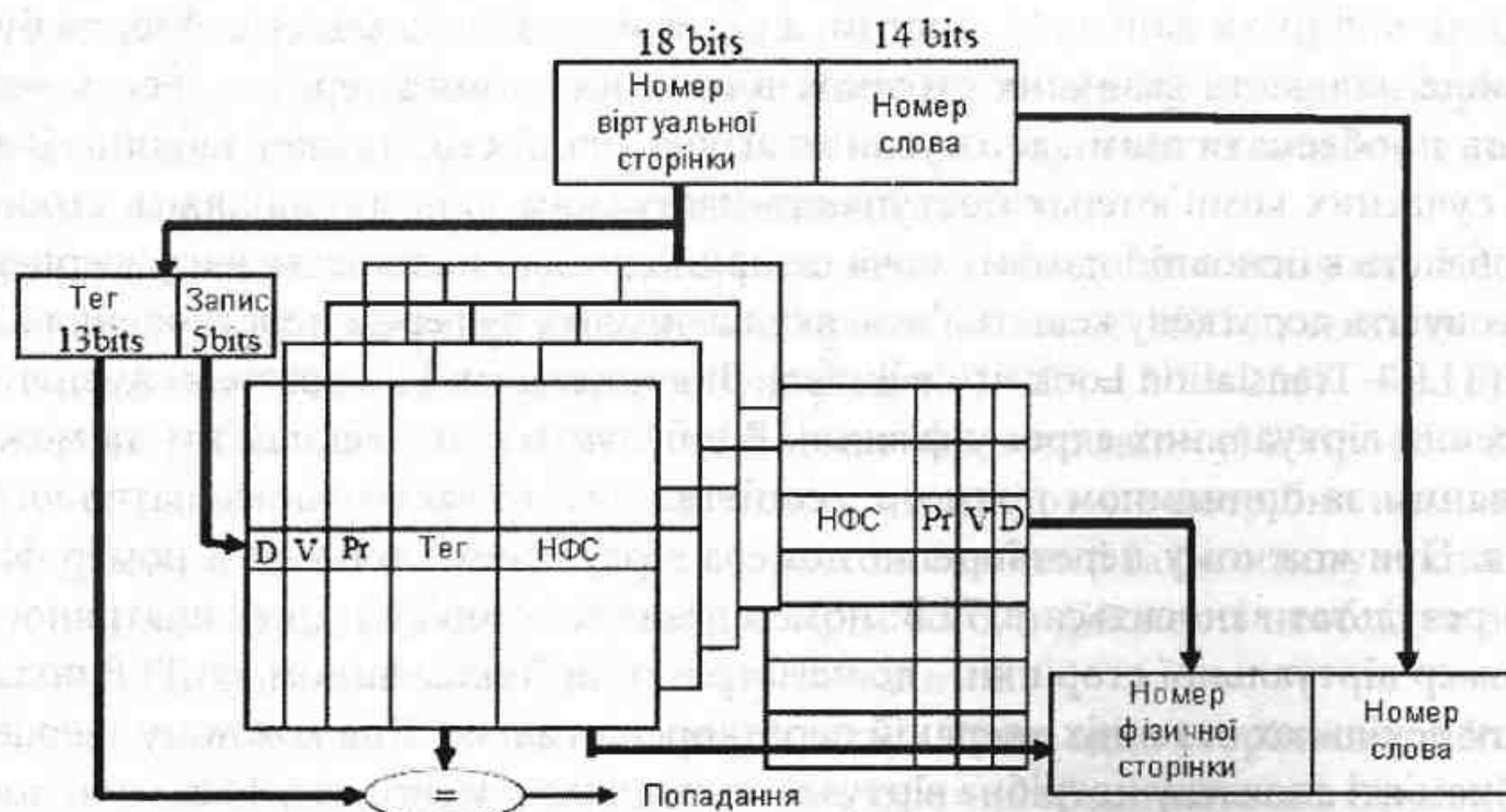


Рис. 10.29. Структура частково-асоціативної кеш пам'яті буфера перетворення з передісторією

Крім номерів програми, віртуальної сторінки та фізичної сторінки слово сторінкової таблиці включає п'ять ознак: РЗ, РВ, РН, РМ, та РД.

Для керування занесення інформації в асоціативні реєстри до їх складу вводять додаткові розряди: розряд зайнятості (РЗ) і розряд використання (РВ).

Ознака РЗ (роздряд зайнятості) потрібна для реалізації стратегії заміщення вмісту асоціативних реєстрів. РЗ встановлюється в 1 при записі інформації в асоціативні реєстри. РЗ говорить про те, вільний чи зайнятий даний асоціативний реєстр.

Так як зазвичай всі асоціативні реєстри зайняті інформацією, то для запису нової необхідно звільнити якийсь асоціативний реєстр. Звільняється той реєстр, інформація в якому не використовувалась. Для фіксації звернень до даної віртуальної сторінки використовується ознака РВ (роздряд використання), який встановлюється в 1 при звертанні до віртуальної сторінки. Якщо всі РЗ = 1, то нова інформація засилается в реєстр, в якому РВ = 0. Якщо всі РВ = 1, то вони автоматично скидаються.

Ознака РН (роздряд наявності) встановлюється в одиницю, якщо віртуальна сторінка в даний момент знаходиться в основній пам'яті. В цьому випадку в асоціативному реєстрі знаходиться номер фізичної сторінки. Якщо РН = 0, то при спробі звернення до даної віртуальної сторінки операційна система завантажує сторінку з зовнішньої пам'яті до основної. При цьому в асоціативному реєстрі наявна інформація про місце розміщення сторінки в зовнішній пам'яті. Завантаження сторінки з зовнішньої пам'яті супроводжується записом до відповідного асоціативного реєстра номера фізичної сторінки, до якої буде записана віртуальна сторінка.

Сторінка в ОП може змінюватись. Тому, якщо вона не змінилася, немає необхідності її переписувати в ЗП, так як там вже є копія. Якщо ж змінилася, потрібно переписати. Для фіксації змін в сторінках ОП вводиться в сторінкову таблицю роздяд модифікації РМ, який дозволяє суттєво зекономити час обміну.

Ознака прав доступу ОД вказує вид доступу до сторінки: запис, читання чи дозволені або заборонені обидві операції, і призначений для вирішення питань захисту інформації.

Потрібно відзначити, що при сторінковій організації пам'яті виникають ті ж проблеми з заміщенням сторінок в основній пам'яті, як це було для кеш пам'яті. Зазвичай в ОП немає вільних сторінок і при завантаженні нової інформації якусь сторінку доводиться видаляти з ОП. Необхідний алгоритм визначення сторінки, що підлягає знищенню. Тут використовуються ті ж самі методи заміщення сторінок, що і в кеш пам'яті: LRU – знищенння сторінки з максимальним простоєм, LFU – знищенння сторінки з мінімальною частотою звернення, FIFO – знищенння сторінки згідно з чергою поступлення, RAND – знищенння сторінки випадковим чином. При цьому потрібно відзначити, що плата за промах (коли вміст потрібної сторінки відсутній в основній пам'яті) є досить великою (десятки мілісекунд), оскільки потрібно робити звернення до повільної зовнішньої пам'яті.

Важливим питанням сторінкової організації пам'яті є вибір об'єму сторінки. На користь малого об'єму сторінки говорять такі аргументи:

- зменшуються втрати через фрагментацію за рахунок неповного використання об'єму виділеної сторінки програмою;
- зменшується об'єм пересилань.

На користь великого об'єму говорять наступні аргументи:

- зменшуються затрати на сторінкову таблицю;
- зменшується час підготовчих операцій в ЗП.

Зазвичай вибирають об'єм сторінки в діапазоні 8КБ – 32КБ.

Таким чином використання сторінкової організації пам'яті дає наступні переваги:

- Віртуальна пам'ять представляється для програміста як пам'ять одного рівня, що спрощує програмування.

- Об єм віртуальної пам'яті визначається довжиною адресного поля і може набагато перевищувати ємність основної пам'яті.
- Виключаються протиріччя, пов'язані з розподілом основної пам'яті між системними і прикладними програмами. Немає обмеження на кількість програм.
- Виключається фрагментація основної пам'яті.
- При мультипрограмуванні не потрібне послідовне розташування сторінок однієї програми в основній пам'яті.
- Не потрібно переміщення інформації, як це є при розподілі з допомогою базових реєстрів.
- В основну пам'ять загружаються активні сторінки за потребою, неактивні сторінки залишаються в зовнішній пам'яті.

Разом з тим, використання сторінкової організації пам'яті вимагає додаткових апаратних та програмних засобів, та ускладнює роботу комп'ютера.

10.3.5. Сегментна організація віртуальної пам'яті

Зазвичай програма складається з декількох частин, розміри яких наперед невідомі та можуть змінюватись в процесі виконання програми. Дляожної з цих частин повинна бути відведена область в просторі віртуальних адрес, оскільки користуватися віртуальною пам'яттю з неперервною нумерацією байтів всіх частин не завжди зручно. Більш зручно, коли кожна частина має свою нумерацію байтів починаючи з нуля. Бажано також, щоб складена таким чином програма могла працювати при динамічному розподілі пам'яті, не вимагаючи від програміста зусиль по об'єднанню різних її частин в єдиний масив. Це завдання розв'язується в багатьох комп'ютерах шляхом використання особливого методу перетворення віртуальних адрес в фізичні та називається сегментною організацією пам'яті.

Принципи сегментної організації пам'яті є наступними:

- віртуальна пам'ятьожної програми ділиться на частини, що називаються сегментами;
- всередині сегменту адресація байтів є незалежною, починаючи від нуля до якогось максимального значення;
- різні сегменти можуть мати різну довжину;
- довжина сегмента може змінюватись в процесі роботи;
- так як кожний сегмент займає незалежний адресний простір, сегменти можуть рости і скорочуватися незалежно один від одного.

Як приклад на рис. 10.30 наведено сегменти пам'яті деякої програми.

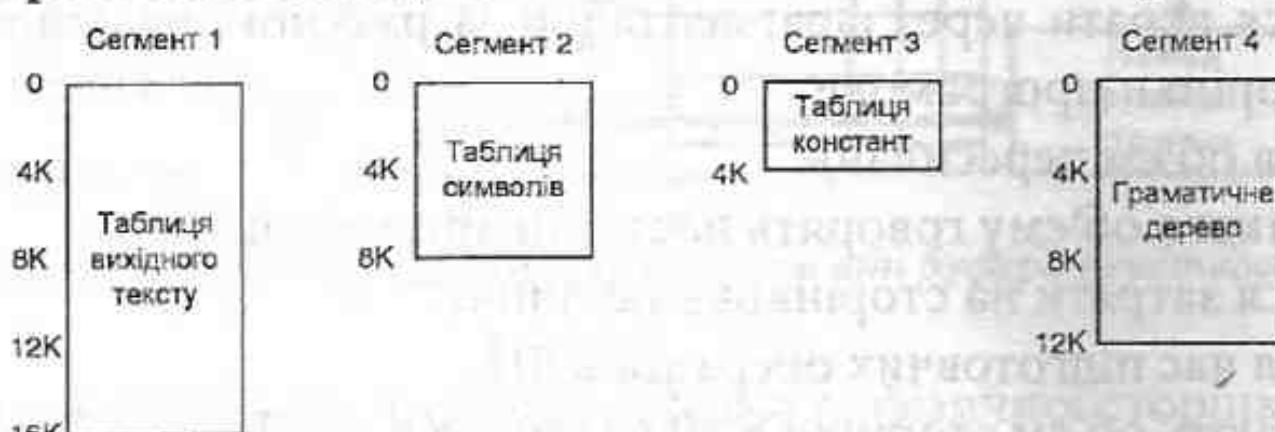


Рис. 10.30. Сегменти пам'яті із чотирьох таблиць

Щоб виконати звернення до такої сегментованої, або двовимірної пам'яті, програма повинна видати адресу, яка складається з двох частин: номера сегмента і внутрішньої

адреси сегмента. Тобто до віртуальної адреси необхідно додати додаткові розряди лівіше номера сторінки, які визначають номер сегмента.

Крім спрощення обробки змінних за об'ємом структур даних, сегментована пам'ять значно спрощує зв'язок процедур, компіляція яких виконана окремо. Якщо процедура в деякому сегменті зазнала змін і повторної компіляції, то решта процедур змінювати не потрібно. Сегментація спрощує і спільне використання даних локальними процесами.

Так як з точки зору програміста сегменти є самостійними логічними об'єктами, припустимо застосування для них різних видів захисту: дозволяється лише читання, запис і т. п.

Таким чином виникає певна ієархія в організації програм, яка складається з чотирьох ярусів: програма, сегмент, сторінка, слово. Цій ієархії програм відповідає й ієархія таблиць перетворення віртуальних адрес у фізичні, як це показано на рис. 10.31.

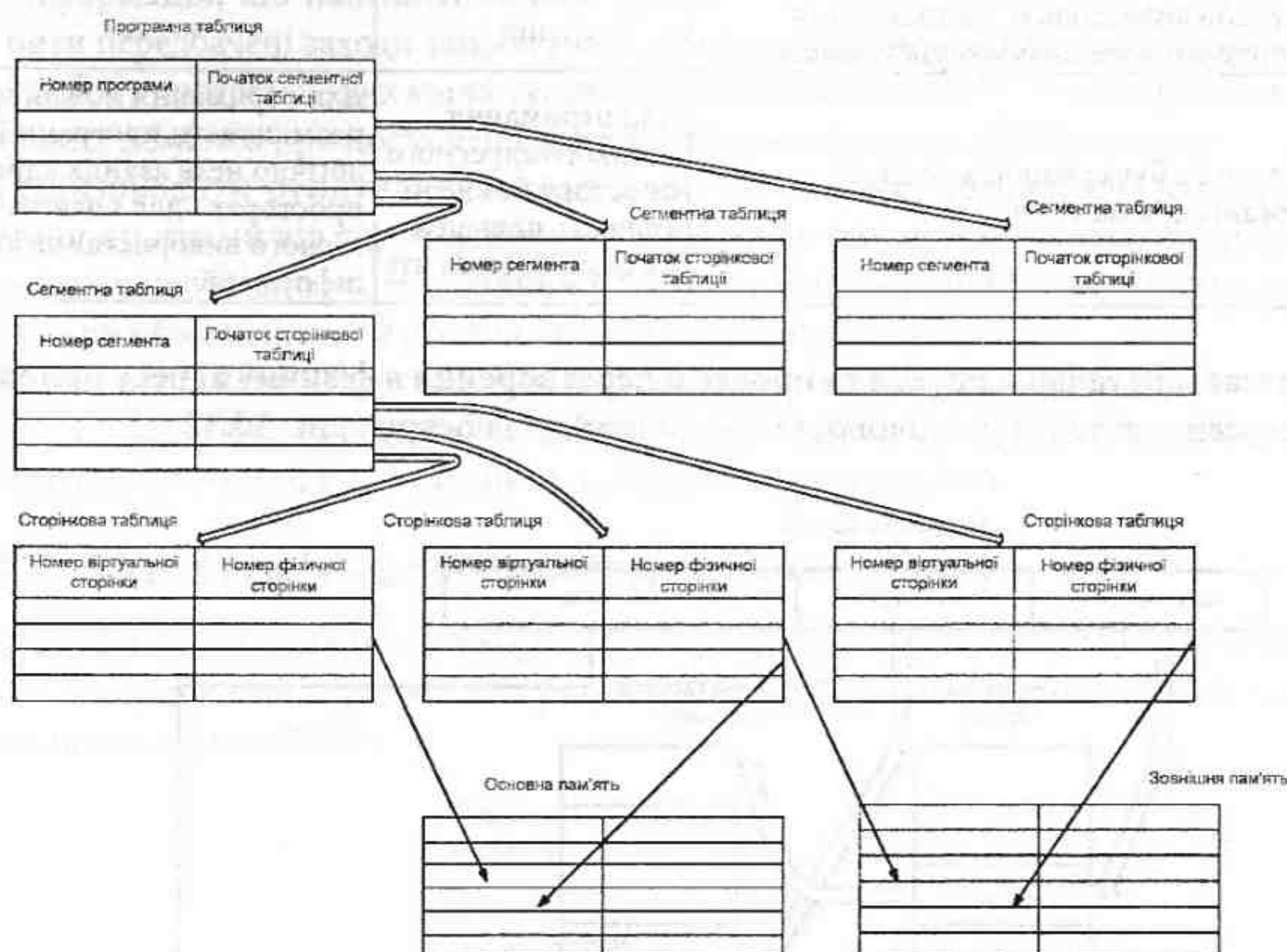


Рис. 10.31. Ієархія таблиць перетворення віртуальних адрес у фізичні

Програмна таблиця включає всі завантажені в комп'ютер програми. Кожній програмі відповідає своя сегментна таблиця. Сегментна таблиця включає сегменти даної програми. Кожному сегменту відповідає своя сторінкова таблиця. Сторінкова таблиця визначає фізичне розташуванняожної сторінки в пам'яті (основній і зовнішній).

Хоча в логічному відношенні сегментна і сторінкова організація пам'яті тісно пов'язані між собою та подібні в реалізації, цілі їх застосування різні. Порівняння сегментної і сторінкової організації пам'яті наведено в табл. 10.1.

Таблиця 10.1

№ п/п	Характеристика	Сторінкова організація пам'яті	Сегментна організація пам'яті
1.	Чи повинен програміст знати, який вид організації пам'яті використовується?	ні	так
2.	Скільки є лінійних адресних просторів?	1	багато
3.	Чи може адресний простір перевищувати ємність пам'яті?	так	так
4.	Чи можуть бути розпізнані та окремо захищені процедури і дані?	ні	так
5.	Чи можна розміщувати таблиці змінного об'єму?	ні	так
6.	Чи можливе спільне застосування процедур декількома користувачами?	ні	так
7.	Для чого була розроблена дана організація пам'яті?	Для отримання великого адресного простору без необхідності збільшення фізичної пам'яті	Для отримання можливості розміщувати програми і дані в логічно незв'язаних адресних просторах і для полегшення сумісного використання і захисту інформації

Формат віртуальної адреси та процес її перетворення в фізичну адресу при використанні сегментної та сторінкової організації пам'яті ілюструє рис. 10.32.



Рис. 10.32. Перетворення віртуальної адреси у фізичну адресу при використанні сегментної та сторінкової організації пам'яті

Як видно з рисунка, для виконання перетворення необхідно два додаткових звернення до пам'яті. Для скорочення кількості звернень для побудови сегментної таблиці може бути застосована, як і при сторінковій організації пам'яті, асоціативна пам'ять.

10.4. Захист пам'яті від несанкціонованих звернень

10.4.1. Задачі захисту пам'яті

Сучасні комп'ютери, як правило, одночасно виконують багато завдань для багатьох користувачів, коли в основній пам'яті одночасно знаходяться програми, що належать як до різних користувачів, так і до різних завдань одного користувача. Крім того, в основній пам'яті завжди присутні фрагменти операційної системи. Кожному завданню в основній пам'яті виділяється свій адресний простір. Такі простори, якщо це спеціально не передбачено, зазвичай є незалежними. В той же час в програмах можуть міститися помилки, що може привести до вторгнення в адресний простір інших завдань. В результаті може бути створена інформація, що належить іншим програмам. Отже, у комп'ютері обов'язково повинні бити передбачені заходи запобігання несанкціонованій дії програмами одного користувача на роботу програм інших користувачів і на операційну систему. Особливо небезпечні наслідки таких помилок при порушенні адресного простору операційної системи.

Щоб перешкодити руйнуванню одних програм іншими, досить захистити область пам'яті даної програми від спроб запису в неї з боку інших програм (захист від запису). У ряді випадків необхідно мати можливість захисту і від читання з боку інших програм, наприклад при обмеженнях на доступ до системної інформації.

Захист від вторгнення програм в чужі адресні простори реалізується цілим рядом способів, які поєднують програмні та апаратні засоби, що керуються операційною системою. Розглянемо нижче найуживаніші способи захисту пам'яті.

10.4.2. Захист пам'яті за допомогою реєстра захисту

Цей спосіб захисту зазвичай використовують при відлагодженні нових програм паралельно з функціонуванням інших програм. Він передбачає захист окремих комірок або блоків пам'яті (рис. 10.33).

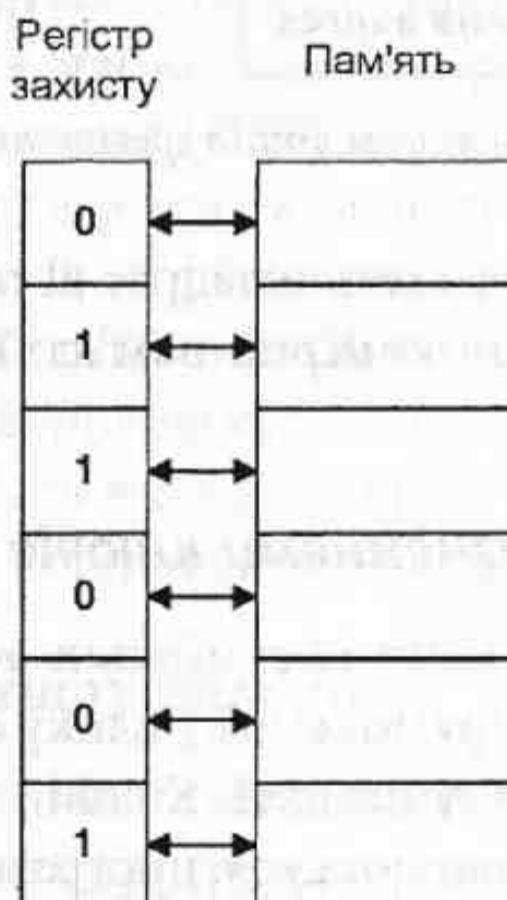


Рис. 10.33. Захист пам'яті з допомогою реєстра захисту

Коли розряд захисту рівний 1, то запис до пам'яті заборонено. Може бути декілька розрядів, які вказують режими роботи.

Реалізувати цей спосіб можна також шляхом виділення в кожній комірці пам'яті спеціального розряду захисту і під'єднання його до пристрою керування записом в пам'ять. Встановлення цього розряду в 1 блокує запис в дану комірку. Подібний спосіб використовувався в обчислювальних машинах попередніх поколінь. Зокрема він був застосований в системі *Atlas*.

10.4.3. Захист пам'яті за граничними адресами

Даний метод захисту є найпоширенішим. Метод припускає наявність в процесорі двох граничних регістрів, вміст яких визначає нижню і верхню межі області пам'яті, куди програма має право доступу (рис. 10.34). Заповнення граничних регістрів проводиться операційною системою при завантаженні програм. При кожному зверненні до пам'яті перевіряється, чи потрапляє використовувана адреса у встановлені межі. Таку перевірку, наприклад, можна організувати на етапі перетворення віртуальної адреси у фізичну. При порушенні межі доступ до пам'яті блокується і формується запит переривання, що викликає відповідну процедуру операційної системи. Нижню межу дозволеної області пам'яті визначає сегментний регистр. Верхня межа підраховується операційною системою відповідно до розміру розміщуваного в пам'яті сегменту.

У розглянутій схемі необхідно, щоб у комп'ютері підтримувалися два режими роботи: привілейований і призначений для користувача. Запис інформації в граничні регістри можливий лише в привілейованому режимі.



Рис. 10.34. Захист пам'яті за граничними адресами

Як вже було вказано, значення граничних адрес встановлюється операційною системою. Цей спосіб вимагає займання комірок пам'яті підряд. Застосований в системі *Stretch* і *IBM7090*.

10.4.4. Захист пам'яті за значеннями ключів

Метод дозволяє організувати захист несуміжних областей пам'яті. Пам'ять умовно ділиться на блоки однакового розміру. Кожному блоку ставиться у відповідність деякий код, який називають ключем захисту пам'яті. Кожній програмі, у свою чергу, присвоюють код захисту програми. Умовою доступу програми до конкретного блоку пам'яті служить збіг ключів захисту пам'яті і програми, або рівність одного з цих ключів нулю. Нульове значення ключа захисту програми надає доступ до всього адресного простору і використовується лише програмами операційної системи. Розподілом ключів захисту програми відає операційна система. Ключ захисту програми зазвичай представлений у

вигляді окремого поля слова стану програми, що зберігається в спеціальному реєстрі. Ключі захисту пам'яті зберігаються в спеціальній пам'яті. При кожному зверненні до основної пам'яті схема порівняння проводить порівняння ключів захисту пам'яті і програми. При збігу доступ до пам'яті дозволено. Дії у разі неспівпадіння ключів залежать від того, який вид доступу заборонений: при записі, при читанні або в обох випадках. Якщо з'ясувалося, що даний вид доступу заборонений, то так само як і в методіграничних адрес формується запит переривання і називається відповідна процедура операційної системи.

Описаний спосіб є більш гнучким порівняно з попереднім, так як дозволяє звертатися до областей пам'яті, розміщених не підряд. Він був застосований в системі IBM 360. Схема захисту пам'яті за описаним способом в системі IBM 360 наведена на рис. 10.35.

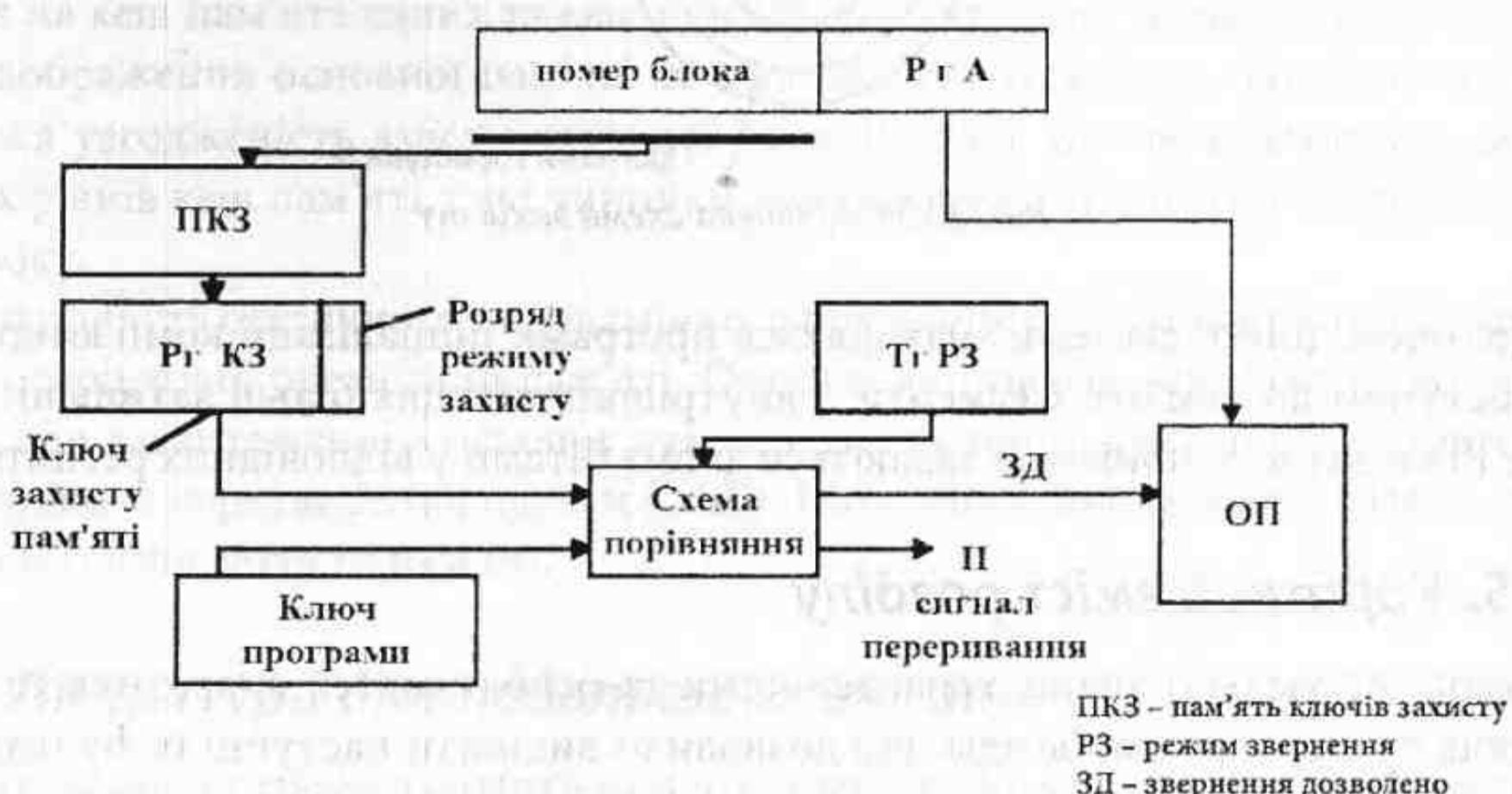


Рис. 10.35. Захист пам'яті за значеннями ключів

В цій системі пам'ять складається з блоків. Кожний блок має код – ключ захисту пам'яті (КЗП). Кожна програма теж має код – ключ програми (КП). Ключі захисту пам'яті зберігаються в пам'яті ключів захисту (ПКЗ). Крім того, є іще тригер режиму захисту (ТгРЗ), в якому зберігається розряд режиму захисту (РРЗ). Доступ дозволений, якщо КЗП = КП. Коли КЗП = КП, а РРЗ = 0, то дозволено лише зчитування. Коли ж КЗП = КП, а РРЗ = 1, то доступ до пам'яті заборонений. Об'єм блоку рівний 2048 байт. Одночасно обробляється 16 програм і, відповідно, використовується 16 варіантів КЗП і КП, тобто коди є 4-розрядними. КП вказується в спеціальному полі слова стану програми (ССП) або каналу (ССК). Коди КП і КЗП встановлює операційна система.

10.4.5. Кільцева схема захисту пам'яті

Захист адресного простору операційної системи від несанкціонованого вторгнення з боку призначених для користувача програм зазвичай організовують шляхом надання системного і призначеного для користувача рівнів привілеїв. Таку структуру прийнято називати кільцевою системою захисту. На рис. 10.36 ця система зображена у вигляді концентричних кіл, де призначений для користувача режим представлений зовнішнім кільцем, а системний – внутрішнім колом. У системному режимі доступні всі

ресурси комп'ютера, а можливості призначеного для користувача режиму істотно обмежені. Перемикання з призначеного для користувача режиму в системний здійснюється спеціальною командою. Вперше описаний підхід був застосований в системі MULTICS на комп'ютері GE 645, де крім ключів захисту використовувалась кільцева схема захисту з 32 рівнями привілеїв. У більшості сучасних комп'ютерів число рівнів привілеїв (кілець захисту) невелике, зокрема в процесорах фірми Intel передбачено чотири рівні привілеїв.



Рис. 10.36. Кільцева схема захисту

В ядрі операційної системи знаходяться програми ініціалізації комп'ютера та керування доступом до пам'яті. Сегменти в внутрішніх кільцях більш захищені, ніж в зовнішніх. Рівні захисту привілеїв задаються двома бітами у відповідних реєстрах.

10.5. Короткий зміст розділу

В розділі проведено аналіз характеристик та особливостей апаратних та програмних засобів сучасного комп'ютера, що дозволило виділити наступні їх фундаментальні властивості, які позитивно доповнюють одна одну з точки зору вирішення задачі забезпечення необхідної ємності і високої швидкодії пам'яті за прийнятну ціну:

- чим менший час доступу до пам'яті, тим менша її ємність та вища вартість зберігання в ній одного біта інформації;
- чим більша ємність пам'яті, тим більший час доступу до неї та нижча вартість зберігання в ній одного біта інформації;
- існує значна різниця між продуктивністю процесора і основної пам'яті, а також між продуктивністю основної та зовнішньої пам'яті;
- комп'ютерні програми володіють властивістю локальності за зверненням до пам'яті.

Ці властивості є підґрунтям доцільності використання при побудові системи пам'яті підходу, відомого як принцип ієрархічної організації пам'яті.

Відповідно до цього принципу пам'ять комп'ютера складається із пристрій пам'яті різних типів, які, залежно від характеристик, належать до певного рівня ієрархії. Пам'ять нижчого рівня має меншу ємність, швидша і має велику вартість в перерахунку на біт, ніж пам'ять вищого рівня. Рівні ієрархії взаємозв'язані: всі дані на деякому нижчому рівні можуть бути також знайдені на вищому рівні, і всі дані на цьому вищому рівні можуть бути знайдені на наступному вищому рівні і т. д. З рухом вверх по ієрархічній структурі зменшується співвідношення вартість/біт, зростає ємність та час доступу. Однак завдяки властивості локальності за зверненням з рухом вверх по ієрархічній структурі зменшу-

ється частота звернення до пам'яті з боку нижчих рівнів, що веде до зменшення загальної вартості при заданому рівні продуктивності. На кожному рівні ієархії пам'ять розбивається на блоки, які є найменшою інформаційною одиницею, що пересилається між двома сусідніми рівнями ієархії. Розмір блоків може бути фіксованим або змінним. При фіксованому розмірі блоку ємність пам'яті зазвичай кратна його розміру. Розмір блоків на кожному рівні ієархії найчастіше різний і збільшується від нижчих рівнів до вищих.

Виходячи з принципу ієархічної організації пам'яті в розділі описано організацію взаємодії між рівнями ієархічної пам'яті. Зокрема, наведено принципи обміну інформацією між рівнями ієархічної пам'яті, характеристики, які використовуються для оцінки ефективності ієархічної пам'яті, пояснено місце кеш пам'яті в складі комп'ютера, призначення і логіку роботи кеш пам'яті, описана вигода від поділу кеш пам'яті первого рівня на кеш пам'ять даних та кеш пам'ять команд. Обґрунтована різноманітність методів відображення основної пам'яті на кеш пам'ять, показано, якими методами забезпечується узгодженість вмісту основної і кеш пам'яті, та чим обумовлене введення додаткових рівнів кеш пам'яті, і які чинники впливають на вибір ємності кеш пам'яті та розміру блоку.

Введено поняття статичного та динамічного розподілу пам'яті та поняття віртуальної пам'яті та сторінкової організації пам'яті. Описані алгоритми заміщення, які використовуються при завантаженні в основну пам'ять вмісту зовнішньої пам'яті, призначення буфера швидкого перетворення адреси (TLB). Наведена сегментна організація пам'яті. Розглянуті питання захисту пам'яті.

10.6. Література для подальшого читання

Обґрунтування потреби в ієархії пам'яті і сам термін, а також принципи керування двома рівнями пам'яті вперше запропоновано в роботі [17]. Віртуальна пам'ять була реалізована в комп'ютерах системи IBM 370, в якій вперше був використаний буфер швидкого перетворення адреси, інформацію про що можна знайти в роботі [6].

Питання побудови кеш пам'яті, організацію взаємодії між рівнями ієархічної пам'яті, принципи обміну інформацією між рівнями ієархічної пам'яті, характеристики, які використовуються для оцінки ефективності ієархічної пам'яті, різноманітність методів відображення основної пам'яті на кеш пам'ять, методи забезпечення узгодженості вмісту основної і кеш пам'яті, та які чинники впливають на вибір ємності кеш пам'яті і розмір блоку, розглянуті в роботах [1-5,7,11-16,18-34].

Питання сторінкової організації пам'яті, зокрема як це було зроблено в системі Атлас, та захисту в ній інформації на прикладі систем IBM 360, Intel 80286 та Multics описані в роботах [10, 35-37].

10.7. Література до розділу 10

1. AGARWAL, A. [1987]. *Analysis of Cache Performance for Operating Systems and Multiprogramming*, Ph.D. Thesis, Stanford Univ., Tech. Rep. No. CSL-TR-87-332 (May).
2. AGARWAL, A. AND S. D. PUDAR [1993]. "Column-associative caches: A technique for reducing the miss rate of direct-mapped caches," 20th Annual Int'l Symposium on Computer Architecture ISCA '93, San Diego, Calif., May 16-19. *Computer Architecture News* 21:2 (May), 179-90.

3. BAER, J.-L. AND W.-H. WANG [1988]. "On the inclusion property for multi-level cache hierarchies", *Proc. 15th Annual Symposium on Computer Architecture* (May-June), Honolulu, 73-80.
4. BHANDARKAR, D. P. [1995], *Alpha Architecture Implementations*, Digital Press, Newton, Mass.
5. BORG, A., R. E. KESSLER, AND D. W. WALL [1990]. "Generation and analysis of very long address traces", *Proc. 17th Annual Int'l Symposium on Computer Architecture* (Cat. No. 90CH2887-8), Seattle, May 28-31, IEEE Computer Society Press, Los Alamitos, Calif., 270-9.
6. CASE, R. P. AND A. PADEGS [1978]. "The architecture of the IBM System/370", *Communications of the ACM* 21:1, 73-96. Also appears in D. P. Siewiorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples* (1982), McGraw-Hill, New York, 830-855.
7. CLARK, D. W. [1983]. "Cache performance of the VAX-W780", *ACM Trans. on Computer Systems* 1:1, 24-37.
8. CONTI, C., D. H. GIBSON, AND S. H. PITKOWSKY [1968]. "Structural aspects of the System/360 Model 85, Part I: General organization", *IBM Systems J.* 7:1, 2-14.
9. CRAWFORD, J. H. AND P. P. GELSINGER [1987]. *Programming the 80386*, Sybex, Alameda, Calif.
10. FABRY, R. S. [1974]. "Capability based addressing", *Comm. ACM* 17:7 (July), 403-412.
11. FARKAS, K. I. AND N. P. JOUPPI [1994]. "Complexity/performance tradeoffs with non-blocking loads", *Proc. 21st Annual Int'l Symposium on Computer Architecture*, Chicago (April).
12. GAO, Q. S. [1993]. "The Chinese remainder theorem and the prime memory system", 20th Annual Int'l Symposium on Computer Architecture ISCA '93, San Diego, May 16-19, 1993. *Computer Architecture News* 21:2 (May), 337-40.
13. GEE, J. D., M. D. HILL, D. N PNEVMATIKATOS, AND A. J. SMITH [1993]. "Cache performance of the SPEC92 benchmark suite", *IEEE Micro* 13:4 (August), 17-27.
14. HILL, M. D. [1987]. *Aspects of Cache Memory and Instruction Buffer Performance*, Ph.D. Thesis, University of Calif, at Berkeley, Computer Science Division, Tech. Rep. UCB/CSD 87/381 (November).
15. HILL, M. D. [1988]. "A case for direct mapped caches", *Computer* 21:12 (December), 25-40.
16. JOUPPI, N. P. [1990]. "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers", *Proc. 17th Annual Int'l Symposium on Computer Architecture* (Cat. No. 90CH2887-8), Seattle, May 28-31, 1990. IEEE Computer Society Press, Los Alamitos, Calif., 364-73.
17. KILBURN, T., D. B. G. EDWARDS, M. J. LANIGAN, AND F. H. SUMNER [1962], "One-level storage system", *IRE Trans. on Electronic Computers* EC-11 (April) 223-235. Also appears in D. P. Siewiorek, C. G. Bell, and A. Newell, *Computer Structures: Principles and Examples* (1982), McGraw-Hill, New York, 135-148.
18. KROFT, D. [1981]. "Lockup-free instruction fetch/prefetch cache organization", *Proc. Eighth Annual Symposium on Computer Architecture* (May 12-14), Minneapolis, 81-87.
19. LAM, M. S., E. E. ROTHBERG, AND M. E. WOLF [1991]. "The cache performance and optimizations of blocked algorithms", Fourth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems, Santa Clara, Calif., April 8-11. *SIGPLAN Notices* 26:4 (April), 63-74.
20. LEBECK, A. R. AND D. A. WOOD [1994]. "Cache profiling and the SPEC benchmarks: A case study", *Computer* 27:10 (October), 15-26.
21. LIPTAY, J. S. [1968]. "Structural aspects of the System/360 Model 85, Part II: The cache", *IBM Systems J.* 7:1, 15-21.
22. MCFARLING, S. [1989]. "Program optimization for instruction caches", *Proc. Third Int'l Conf. on Architectural Support for Programming Languages and Operating Systems* (April 3-6), Boston, 183-191.
23. MOWRY, T. C., S. LAM, AND A. GUPTA [1992]. "Design and evaluation of a compiler algorithm for prefetching", Fifth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), Boston, October 12-15, *SIGPLAN Notices* 27:9 (September), 62-73.

24. PALACHARLA, S. AND R. E. KESSLER [1994]. "Evaluating stream buffers as a secondary cache replacement", *Proc. 21st Annual Int'l Symposium on Computer Architecture*, Chicago, April 18-21, IEEE Computer Society Press, Los Alamitos, Calif., 24-33.
25. PRZYBYLSKI, S. A. [1990]. *Cache Design: A Performance-Directed Approach*, Morgan Kaufmann Publishers, San Mateo, Calif.
26. PRZYBYLSKI, S. A., M. HOROWITZ, AND J. L. HENNESSY [1988], "Performance tradeoffs in cache design", *Proc. 15th Annual Symposium on Computer Architecture* (May-June), Honolulu, 290-298.
27. SAAVEDRA-BARRERA, R. H. [1992]. *CPU Performance Evaluation and Execution Time Prediction Using Narrow Spectrum Benchmarking*, Ph.D. Dissertation, University of Calif., Berkeley (May).
28. SAMPLES, A. D. AND P. N. HILFINGER [1988]. "Code reorganization for instruction caches", Tech. Rep. UCB/CSD 88/447 (October), University of Calif., Berkeley.
29. SITES, R. L. (ED.) [1992]. *Alpha Architecture Reference Manual*, Digital Press, Burlington, Mass.
30. SMITH, A. J. [1982]. "Cache memories", *Computing Surveys* 14:3 (September), 473-530.
31. SMITH, J. E. AND J. R. GOODMAN [1983]. "A study of instruction cache organizations and replacement policies", *Proc. 10th Annual Symposium on Computer Architecture* (June 5-7), Stockholm, 132-137.
32. STRECKER, W. D. [1976]. "Cache memories for the PDP-11?", *Proc. Third Annual Symposium on Computer Architecture* (January), Pittsburgh, 155-158.
33. TORRELLAS, J., A. GUPTA, AND J. HENNESSY [1992], "Characterizing the caching and synchronization performance of a multiprocessor operating system", Fifth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), Boston, October 12-15, SIGPLAN Notices 27:9 (September), 162-74.
34. WANG, W.-H., J.-L. BAER, AND H. M. LEVY [1989]. "Organization and performance of a two-level virtual-real cache hierarchy", *Proc. 16th Annual Symposium on Computer Architecture* (May 28-June 1), Jerusalem, 140-148.
35. WILKES, M. [1965]. "Slave memories and dynamic storage allocation", *IEEE Trans. Electronic Computers* EC-14:2 (April), 270-271.
36. WILKES, M. V. [1982]. "Hardware support for memory protection: Capability implementations", *Proc. Symposium on Architectural Support for Programming Languages and Operating Systems* (March 1-3), Palo Alto, Calif., 107-116.
37. WULF, W. A., R. LEVIN, AND S. P. HARBISON [1981], *Hydra/C.mmp: An Experimental Computer System*, McGraw-Hill, New York.

10.8. Питання до розділу 10

1. Чим пояснюється різниця між продуктивністю процесора та пам'яті?
2. Що показує аналіз зміни з роками швидкодії статичної і динамічної напівпровідникової та дискової пам'яті і процесора?
3. Як зростає з роками розрив між швидкодією процесора і динамічної пам'яті? На що це впливає?
4. Як зростає з роками розрив між швидкодією динамічної і дискової пам'яті? На що це впливає?
5. Чим викликана необхідність побудови системи пам'яті за ієрархічним принципом?
6. Що включає поняття «локальність за зверненням»?
7. Що таке просторова локальність?
8. Що таке часова локальність?
9. Які висновки витікають з властивості локальності?
10. Наведіть принцип ієрархічної організації пам'яті
11. Завдяки чому середній час доступу в ієрархічній системі пам'яті визначається більш швидкодіючими видами пам'яті?

12. Що в ієрархічній системі пам'яті визначають терміни «промах» і «попадання»?
13. На які питання необхідно відповісти, щоб охарактеризувати певний рівень ієрархічної пам'яті?
14. Наведіть принципи обміну інформацією між рівнями ієрархічної пам'яті.
15. Які характеристики використовуються для оцінки ефективності ієрархічної пам'яті?
16. Поясніть місце кеш пам'яті в складі комп'ютера.
17. Поясніть призначення і логіку роботи кеш пам'яті.
18. В чому вигода від поділу кеш пам'яті першого рівня на кеш пам'ять даних та кеш пам'ять команд?
19. Які проблеми породжує включення кеш пам'яті в ієрархію пам'яті?
20. Чим обумовлена різноманітність методів відображення основної пам'яті на кеш пам'ять?
21. Якій вимозі повинен відповісти «ідеальний» алгоритм заміщення вмісту кеш пам'яті?
22. Якими методами забезпечується узгодженість вмісту основної і кеш пам'яті?
23. Чим обумовлене введення додаткових рівнів кеш пам'яті?
24. Які чинники впливають на вибір ємності кеш пам'яті та розміру блоку?
25. Як співвідносяться характеристики звичайної і дискової кеш пам'яті?
26. Якими засобами забезпечується віртуалізація пам'яті?
27. Чи існує обмеження на розмір віртуального простору?
28. Що визначає об'єм сторінкової таблиці?
29. Якими прийомами досягають скорочення об'єму сторінкових таблиць?
30. Які алгоритми заміщення використовуються при завантаженні в основну пам'ять нової віртуальної сторінки?
31. Поясніть призначення буфера швидкого перетворення адреси (TLB).
32. Чим мотивується розбиття віртуальних секторів на сторінки?
33. Яка частина віртуальної адреси залишається незмінною при його перетворенні у фізичну адресу?
34. Наведіть способи апаратної реалізації сторінкової таблиці.
35. Наведіть алгоритми заміщення сторінок в основній пам'яті.
36. Для чого призначена сегментна організація пам'яті?
37. Дайте порівняння сторінкової та сегментної організації пам'яті.
38. Наведіть ієрархію таблиць перетворення віртуальних адрес у фізичні.
39. Чим обумовлена необхідність захисту пам'яті?
40. Назвіть способи захисту пам'яті.
41. Поясніть спосіб захисту пам'яті за допомогою ключів захисту.