4. ОРГАНІЗАЦІЯ ДОСТУПУ ДО ДАНИХ В ОПЕРАТИВНІЙ ПАМ'ЯТІ

4.1. ОРГАНІЗАЦІЯ ПАМ'ЯТІ

Фізична пам'ять, до якої мікропроцесор (МП) має доступ по адресній шині називається **оперативною пам'яттю** (ОП) (або оперативним запам'ятовувальним пристроєм – ОЗП). На найнижчому рівні пам'ять комп'ютера можна розглядати як масив бітів. Один біт може зберігати значення 0 або 1. Для фізичної реалізації бітів і роботи з ними ідеально підходять логічні схеми. Однак мікропроцесору незручно працювати з пам'яттю на рівні бітів, тому реально ОЗП організоване як послідовність комірок – байтів.

Кожному байту відповідає своя унікальна адреса (його номер), яку називають ***фізичною***. Діапазон значень фізичних адрес залежить від розрядності шини адреси мікропроцесора. Для i486 і *Pentium* він становить від 0 до 2^{32}–1 (4 *Гбайт*). Для мікропроцесора сім'ї *Pentium Pro/ II/ III* цей діапазон ширше – від 0 до 2^{36}–1 (64 *Гбайт*).

Механізм управління пам'яттю повністю апаратний. Це означає, що програма сама не може сформувати фізичну адресу пам'яті на адресній шині. Вона змушена працювати за правилами мікропроцесора. Цей механізм дозволяє забезпечити:

– компактність збереження адреси в машинній команді;

– гнучкість механізму адресації;

– захист адресних просторів у багатозадачній системі;

– підтримку віртуальної пам'яті.

Мікропроцесор апаратно підтримує дві моделі використання ОП:

– ***сегментовану*** – програмі виділяються неперервні області пам'яті (сегменти), а сама програма може звертатись тільки до даних, які розміщуються в цих сегментах;

– ***сторінкову*** (можна розглядати як надбудову над сегментованою моделлю) – ОП розглядається як сукупність блоків фіксованого розміру (4 *Кбайт*). Основне застосування цієї моделі пов'язано з організацією віртуальної пам'яті, що дає змогу операційній системі використовувати для роботи програм простір пам'яті, більший за об'єм фізичної пам'яті. Для мікропроцесорів i486 та *Pentium* розмір віртуальної пам'яті може досягати 4 *Тбайт*.

Особливості використання й реалізації цих моделей залежать від режиму роботи мікропроцесора.

1. Режим реальних адрес, або **реальний режим**. Це режим, у якому працював i8086. Наявність цього режиму в i486 та *Pentium* обумовлено тим, що фірма *Intel* намагається забезпечити в нових моделях мікропроцесорів функціонування програм, які розроблені для ранніх моделей мікропроцесорів.

Донедавна це був єдиний режим, у якому функціонувала операційна система MS-DOS. Для неї було розроблено великий об'єм програмного забезпечення. Розуміючи це й не бажаючи втрачати ринок, фірма *Intel* у всіх модернізаціях свого мікропроцесора підтримує цей режим.

Характеристика режиму:

– простір ОП поділяється на сегменти по 64 *Кбайт*. Сегменти в пам'яті можуть перекриватись;

– сторінкове перетворення адреси заборонено, тобто її фізична адреса дорівнює лінійній і формується як сума двох складових:

а) 16-розрядної ефективної адреси, яка, своєю чергою, є сумою трьох складових: бази, зміщення та індексу;

б) 20-розрядного зсуву вмісту конкретного сегментного регістра на 4 розряди вліво;

– максимальне значення фізичної адреси дорівнює 0FF FFFh, тобто 1 *Мбайт*, однак фактично в реальному режимі МП адресується на 64 *Кбайт* більше, що витікає з таких розрахунків:

FFFF0
+0FFFF
=10FFEF

де FFFF0 – максимальне значення сегментної частини адреси, зсунуте на 4 розряди вліво;

0FFFF – максимальне значення зміщення;

10FFEF = 1 114 096 *байт* – максимальна фізична адреса в реальному режимі;

– у реальному режимі схема розподілу ОП фіксована й містить системні області, деякі з них перелічені нижче:

а) у діапазоні адрес 00000H–003FFH (перший мегабайт оперативної пам'яті) розміщується таблиця векторів переривань (ТВП). Вона містить 256 векторів переривань розміром по 4 байти (покажчиків на програми обробки переривань);

б) у діапазоні адрес 00400H–006FFH відразу за таблицею векторів переривань розміщується область пам'яті, у якій містяться жорсткоструктуровані дані, які забезпечують роботу BIOS та MS DOS;

в) з адреси 0B800H розміщується область відеопам'яті, у якій формується зображення.

2. Захищений режим. Цей режим дозволяє максимально реалізувати всі архітектурні ідеї, закладені в моделі мікропроцесора *Intel*, починаючи з i80286. Програми, що розроблені для i8086 (реального режиму), не можуть функціонувати в захищеному режимі.

Усі сучасні багатозадачні операційні системи працюють у захищеному режимі.

Реальний режим підтримував виконання лише однієї програми. У зв'язку із цим не потрібно було здійснювати захист програм від взаємного впливу. Усе, що потрібно було знати програмі, – це адреси сегментів коду, даних та стеку. *Якщо виникла необхідність розмістити в одне програмно-апаратне середовище декілька незалежних програм, то автоматично виникло питання їх захисту від взаємного впливу*.

Через те, що кожна задача в системі займає один або декілька сегментів у пам'яті, то логічно мати більше інформації про них, як про об'єкти, що реально існують у системі в цей момент. Сегментам надають визначені ***атрибути***. До них належать:

– розміщення сегмента в пам'яті;

– розмір сегмента;

– рівень привілеїв – визначає права даного сегмента відносно інших сегментів;

– тип доступу – визначає призначення сегмента;

– деякі інші.

Отже, в захищеному режимі мікропроцесор підтримує два типи захисту – за привілеями та доступом до пам'яті. На відміну від реального режиму в захищеному режимі програма вже не може просто звернутися за будь-якою фізичною адресою пам'яті: для цього їй потрібні певні повноваження та вона повинна задовольняти деякі вимоги.

Ключовим об'єктом захищеного режиму є спеціальна структура – ***дескриптор*** *сегмента, який являє собою 8-байтовий дескриптор (короткий опис) безперервної області пам'яті, яка містить перелічені атрибути*. Будь-яка область пам'яті, яка логічно може бути сегментом даних, стеку або коду, повинна бути описана таким дескриптором. Усі дескриптори об'єднуються разом в одну з трьох дескрипторних таблиць відповідно до призначення. Адреса, за якою розміщуються

ці дескрипторні таблиці, може бути будь-якою. Вона зберігається в спеціальному регістрі, який саме для цього призначений.

3. Режим віртуального 8086. Перехід у цей режим можливий, якщо мікропроцесор вже знаходиться в захищеному режимі. Відмінною рисою цього режиму є можливість одночасної роботи декількох програм, розроблених для i8086. Незважаючи на те, що мікропроцесор знаходився в захищеному режимі, в режимі віртуального i8086 можлива робота програм реального режиму. Це пояснюється тим, що процес формування фізичної адреси для цих програм здійснюється за правилами реального режиму.

4. Режим системного управління – це новий режим роботи мікропроцесора, який вперше з'явився в мікропроцесорі *Pentium*. Він забезпечує операційну систему механізмом для виконання машинозалежних функцій, таких як переключення комп'ютера в режим зниженого енергоспоживання або виконання дій щодо захисту системи. Для переходу в цей режим мікропроцесор повинен отримати спеціальний сигнал – SMI від удосконаленого програмованого контролера переривань APIC (*Advanced Programmable Interrupt Controller*), при цьому зберігається стан обчислювального середовища мікропроцесора. Функціонування мікропроцесора в цьому режимі подібне до його роботи в режимі реальних адрес. Вихід із цього режиму здійснюється спеціальною командою мікропроцесора.

4.2. СЕГМЕНТНА МОДЕЛЬ ПАМ'ЯТІ

Сегментація – механізм адресації, який забезпечує існування декількох незалежних адресних просторів як у межах однієї задачі, так і в сегменті в цілому, для захисту задач від взаємного впливу. В основі сегментації лежить поняття ***сегмента***, який являє собою незалежний блок пам'яті, який підтримується на апаратному рівні.

Кожна програма може складатися з будь-якої кількості сегментів, але безпосередній доступ вона має лише до трьох основних сегментів: *коду, даних та стеку* – та від одного до трьох додаткових сегментів даних.

Програма сама не визначає, за якими фізичними адресами буде розміщено її сегменти. Це здійснює операційна система. Саме операційна система розміщує сегменти програми в ОП за визначеними фізичними адресами, після чого розміщує значення цих адрес у визначені місця, які залежать від режиму роботи мікропроцесора. Так, у реальному режимі

ці адреси розміщуються безпосередньо у відповідні сегментні регістри, а в захищеному – у дескрипторні таблиці.

Усередині сегмента програма звертається до адрес відносно початку сегмента лінійно, тобто починаючи з 0 і закінчуючи адресою, яка дорівнює розміру сегмента. Ця відносна адреса або зміщення, яку мікропроцесор використовує для доступу до даних усередині сегмента, називається *ефективною*.

Розрізняють три основні моделі сегментованої організації пам'яті:

– сегментована модель пам'яті реального режиму;

– сегментована модель пам'яті захищеного режиму;

– неперервна модель пам'яті захищеного режиму.

Яким є порядок формування фізичної адреси в реальному режимі? Під **фізичною адресою** розуміють адресу пам'яті, яка видається на адресну шину МП. Інша назва цієї адреси – **лінійна адреса**. Ця подвійність у назвах обумовлена наявністю сторінкової моделі організації оперативної пам'яті. Ці назви є синонімами лише в разі відключення сторінкового перетворення адрес. У реальному режимі сторінкова адресація відключена. У сторінковій моделі лінійна та фізична адреси мають різні значення.

4.3. ФОРМУВАННЯ ФІЗИЧНОЇ АДРЕСИ В РЕАЛЬНОМУ РЕЖИМІ

У реальному режимі механізм адресації фізичної пам'яті має такі характеристики:

а) діапазон зміни фізичної адреси від 0 до 1 *Мбайт*. Це визначається адресною шиною i8086, яка має 20 ліній;

б) максимальний розмір сегмента – 64 *Кбайт*. Це пояснюється 16-розрядною архітектурою i8086. Нескладно підрахувати, що максимальне значення, яке можуть містити 16-розрядні регістри складає 2^{16}–1, що визначає величину 64 *Кбайт*;

в) для звернення до конкретної фізичної адреси оперативної пам'яті необхідно визначити адресу початку сегмента (сегментну складову) і зміщення всередині сегмента.

Однак сегментна складова адреси (або база сегмента) являє собою лише 16-бітне значення, яке розміщується в одному із сегментних регістрів. Максимальне значення, яке при цьому отримується відповідає 2^{16}–1. Внаслідок цього адреса початку сегмента може бути тільки в діапазоні 0–64 *Кбайт* від початку оперативної пам'яті. Виникає

питання, як адресувати решту частини ОП, включно до 1 *Мбайт*, з урахуванням того, що розмір самого сегмента не перевищує 64 *Кбайт*.

У сегментному регістрі містяться тільки старші 16 бітів фізичної адреси початку сегмента. Молодші 4 біти 20-бітної адреси отримують зсувом значення в сегментному регістрі вліво на 4 розряди. Ця операція зсуву виконується апаратно й для програмного забезпечення абсолютно зрозуміла.

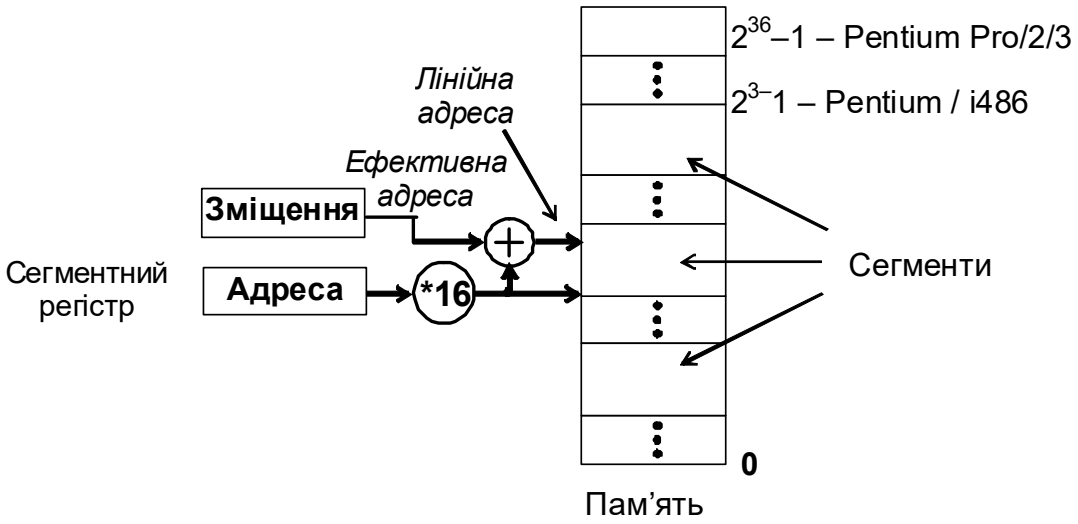

Рис.4.1. Сегментована модель пам'яті реального режиму

Отримане в такий спосіб 20-бітне значення і є справжньою фізичною адресою, яка відповідає початку сегмента.

Зміщення являє собою 16-бітне значення. Це значення може розміщуватись явно в команді або непрямо в одному з регістрів загального призначення. У мікропроцесорі ці дві складові додаються на апаратному рівні, внаслідок чого формується фізична адреса розмірністю 20 *біт*. Даний механізм створення фізичної адреси дозволяє зробити програмне забезпечення переміщуваним, тобто таким, що не залежить від конкретних адрес завантаження його в оперативній пам'яті. Цей механізм більш детально зображений на рис. 4.2.

Пристрій сторінкового перетворення адрес призначений для суміщення двох принципово різних моделей організації ОП та видачі на адресну шину істинного значення фізичної адреси пам'яті.

Значення зміщення можна отримати мінімум з одного й максимум із трьох джерел. Кількість джерел визначається кодуванням конкретної машинної команди і, якщо таких джерел декілька, то значення в них додаються.

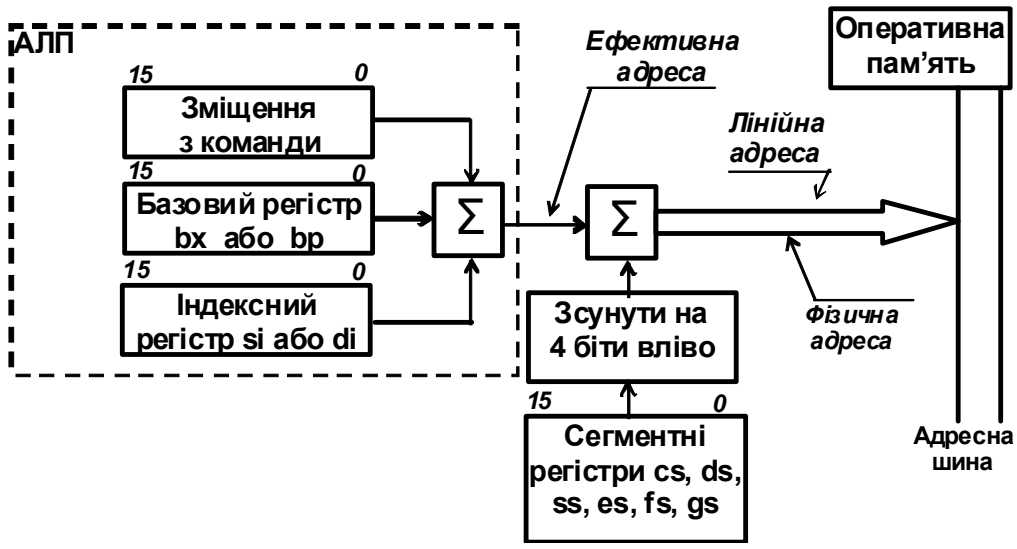

Рис. 4.2. Механізм формування фізичної адреси в реальному режимі

Недоліки такої організації пам'яті:

– сегменти безконтрольно розміщуються з будь-якої адреси, що кратна 16 (через те, що вміст сегментного регістра апаратно зміщується на 4 розряди). Внаслідок цього, програма може звертатись за будь-якими адресами, зокрема й такими, що не існують;

– сегменти мають максимальний розмір 64 *Кбайт*;

– сегменти можуть перекриватися з іншими сегментами.

Із метою усунення вказаних недоліків з'явився захищений режим, у якому працюють всі сучасні операційні системи, зокрема *Windows* та *Linux*.

4.4. ЗАХИЩЕНИЙ РЕЖИМ РОБОТИ ПРОЦЕСОРА

4.4.1. Системні регістри захищеного режиму

Уперше захищений режим з'явився в мікропроцесорі i80286 фірми *Intel*. Усі сучасні багатозадачні операційні системи працюють тільки в цьому режимі. Такі операційні системи на сьогодні стали стандартом.

Будь-який сучасний процесор у захищеному режимі майже не відрізняється від i8086. Це лише його більш швидкодіючий аналог зі збільшеною до 32 розрядів розміром усіх регістрів крім сегментних. Щоб отримати доступ до всіх архітектурних та функціональних нововведень мікропроцесора необхідно перейти в захищений режим.

У захищеному режимі змінено принципи роботи мікропроцесора з пам'яттю. Пам'ять залишається сегментованою, однак змінюються функції та номенклатура програмно-апаратних компонентів, які беруть участь у сегментації. Як вже зазначалось, у реальному режимі роботи мікропроцесора розмір сегмента не перевищував 64 *Кбайт*,

а адреса області пам’яті сегмента розміщувалась в одному із сегментних регістрів. Функціональне призначення сегмента визначалось тим, у якому із шести сегментних регістрів розміщувалась його адреса. Апаратні засоби контролю доступу до сегмента були відсутні. Реальний режим підтримував виконання тільки одної програми. Для цього достатньо було простих механізмів розподілу оперативної пам’яті й не було необхідності захисту програм від взаємного впливу. Тому, все що було потрібно знати програмі – це адреси, за якими розміщуються її сегменти коду, даних та стеку. Якщо виникла б необхідність розмістити в одне програмно-апаратне середовище декілька незалежних програм, то автоматично стало б питання про їх захист від взаємного впливу. Для розв’язування цієї проблеми мікропроцесору вже недостатньо, використовуючи сегментні регістри, знати, де розміщені сегменти програм. Для забезпечення сумісної роботи декількох програм необхідно захистити їх від взаємного впливу, а якщо виникає необхідність взаємодії між ними, то воно повинно обов’язково регулюватися.

Щоб вести таке регулювання слід мати більше інформації про самі програми. Можна пропонувати декілька варіантів структурної організації та розміщення такої інформації. Фірма *Intel* не стала порушувати принцип сегментації. Через те, що кожна програма в системі займає один або декілька сегментів у пам’яті, то логічно мати більше інформації про них як про об’єкти, які реально існують у системі в даний момент. Якщо кожному з об’єктів привласнити визначені атрибути, то частину контролю за доступом до них можна перекласти на сам процесор. Що й було зроблено. Будь-який сегмент пам’яті в захищеному режимі має такі атрибути:

- розміщення сегмента в пам’яті;
- розмір сегмента;
- рівень привілеїв, який визначає права даного сегмента відносно інших сегментів;
- тип доступу, який визначає призначення сегмента;
- деякі інші.

Склад перерахованих атрибутів показує, що в захищеному режимі мікропроцесор підтримує два типи захисту: за привілеями та доступом до пам’яті. На відміну від реального режиму програма вже не може вільно звернутись за будь-якою фізичною адресою пам’яті. Для цього вона повинна мати певні повноваження та задовольняти відповідні вимоги.

Основним об’єктом захищеного режиму є спеціальна структура – **дескриптор сегмента**, який являє собою 8-байтовий (короткий опис) безперервної ділянки пам’яті, який містить перераховані атрибути. Будь-яка область пам’яті, яка логічно може бути сегментом даних, коду або

стеку повинна бути описана таким дескриптором. Усі дескриптори об'єднуються разом в одну з трьох дескрипторних таблиць. У яку саме таблицю повинен бути розміщений дескриптор визначається його призначенням. Адреса, за якою розміщуються ці дескрипторні таблиці, може бути будь-якою. Вона зберігається в системному регістрі, який спеціально призначений для цієї адреси.

При вивченні реального режиму мікропроцесора зазначалося, що програмна модель мікропроцесора складається з 32 доступних користувачеві регістрів. Ці регістри поділяються на дві великі групи: користувацькі та системні. Роботу з користувацькими регістрами розглянуто. Далі викладено матеріал стосовно системних регістрів.

Системні регістри мікропроцесора. Системні регістри виконують специфічні функції в системі. Використання цих регістрів суворо регламентовано. Саме вони забезпечують роботу захищеного режиму. Ці регістри є доступними для програмістів і можуть бути використані.

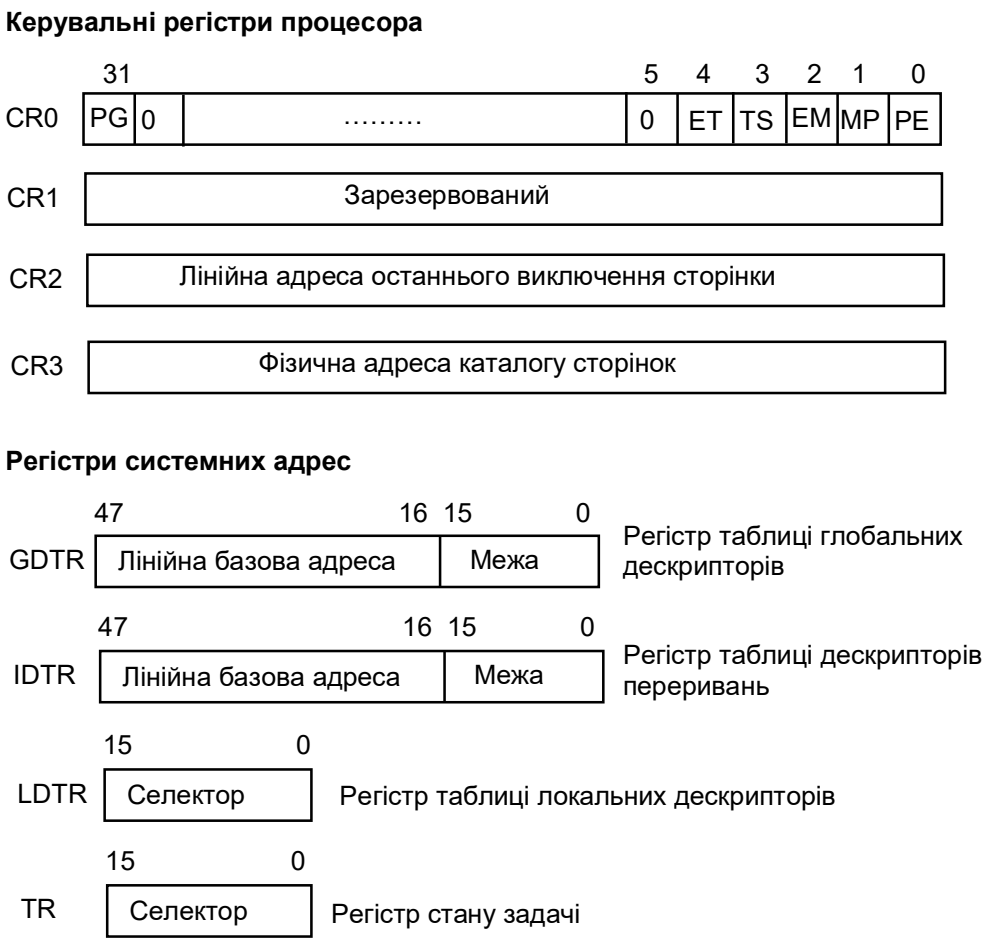

Рис. 4.3. Системні регістри процесора

Системні регістри поділяються на три групи:

- чотири регістри управління;
- чотири регістри системних адрес;

- вісім регістрів налагоджування.

До складу системних регістрів мікропроцесорів ряду *Pentium* введено такі зміни:

- задіяно раніше зарезервований регістр cr4;

- введено групу MSR-регістрів (MSR – *Model Specific Register*, модельно-залежні регістри процесора), призначення та можливості яких стосуються конкретної моделі мікропроцесора. Для доступу до цих регістрів введено спеціальні команди.

Регістри управління. До складу регістрів управління входять п’ять регістрів: cr0, cr1, cr2, cr3, cr4. Ці регістри призначені для загального управління системою. Регістри управління доступні тільки програмам із рівнем привілеїв 0. Мікропроцесор має п’ять регістрів управління, хоча доступними є лише чотири. Незадіяним є cr1, функції якого ще не визначені (зарезервований для майбутнього використання).

Регістр cr0 містить системні прапорці, які керують режимами роботи мікропроцесора й відбивають його стан глобально, незалежно від конкретних задач, що виконуються. Призначення системних прапорців:

pe (Protect Enable), біт 0 – дозвіл захищеного режиму роботи. Стан цього прапорця показує, у якому з двох режимів – реальному (pe = 0) або захищеному (pe = 1) – працює мікропроцесор у даний момент часу;

mp (Math Present), біт 1 – наявність сопроцесора. Завжди 1;

ts (Task Switched), біт 3 – перемикання задач. Процесор автоматично встановлює цей біт при переключенні на виконання іншої задачі;

am (Alignment Mask), біт 18 – маска переривання. Цей біт дозволяє (am = 1) або забороняє (am = 0) контроль переривання;

cd (Cach Disable), біт 30 – заборона кеш-пам’яті. За допомогою цього біта можна заборонити (cd = 1) або дозволити (cd = 0) використання внутрішньої кеш-пам’яті першого рівня;

pg (PaGing), біт 31 – дозвіл (pg = 1) або заборона (pg = 0) сторінкового перетворення. Регістр cr0 використовується при сторінковій моделі організації пам’яті.

Регістр cr2 використовується при сторінковій організації оперативної пам’яті для реєстрації ситуації, коли поточна команда звернулась за адресою, що міститься в сторінці пам’яті, яка в даний момент у пам’яті відсутня. У такій ситуації в мікропроцесорі виникає виключна ситуація з номером 14, і лінійна 32-бітна адреса команди, яка викликає це виключення, записується до регістра cr2. Маючи цю інформацію, оброблювач виключення 14 визначає потрібну сторінку, здійснює її підкачування в пам’ять та поновлює нормальну роботу програми.

Регістр cr3 також використовується при сторінковій організації пам'яті. Це так званий регістр каталогу сторінок першого рівня. Він містить 20-бітову фізичну адресу каталогу сторінок поточної задачі. Цей каталог містить 1024 32-бітних дескрипторів, кожен з яких містить адресу таблицю сторінок другого рівня. Своєю чергою кожна з таблиць сторінок другого рівня містить 1024 32-бітних дескрипторів, які адресують сторінкові кадри в пам'яті. Розмір сторінкового кадру 4 *Кбайт*.

Регістр cr4 містить переважно дозволяючі ознаки, які характеризують ті або інші архітектурні елементи, які вперше з'являються в різних моделях мікропроцесорів. Як приклад, підтримка 36-розрядної адресації, підтримка сторінок розміром 4 *Мбайт* та ін. Встановлюючи в регістрі cr4 визначені біти, можна включати або відключати підтримку цих властивостей.

Регістри системних адрес. Ці регістри ще називають *регістрами управління пам'яттю*. Вони призначені для захисту програм та даних у мультипрограмному режимі роботи мікропроцесора. При роботі в захищеному режимі мікропроцесора адресний простір поділяється:

- на *глобальний* – загальний для всіх;
- на *локальний* – окремий для кожної задачі.

Цим розподілом пояснюється те, що в архітектурі мікропроцесора існують такі системні регістри:

регістр таблиці глобальних дескрипторів gdtr (Global Descriptor Table Register) має розмір 48 *біт* та містить 32-бітну (біти 16–47) базову адресу глобальної дескрипторної таблиці GDT та 16-бітне значення межі, яка являє собою розмір таблиці GDT в байтах;

регістр таблиці локальних дескрипторів ldtr (Local Descriptor Table Register) має розмір 16 біт та містить так званий селектор дескриптора локальної дескрипторної таблиці LDT. Цей селектор є вказівником в таблиці GDT, який і описує сегмент, що містить локальну дескрипторну таблицю LDT;

регістр таблиці дескрипторів переривань idtr (Interrupt Descriptor Table Register) має розмір 48 *біт* та містить 32-бітну (біти 16–47) базову адресу дескрипторної таблиці переривань IDT та 16-бітне (біти 0–15) значення межі, яка являє собою розмір таблиці IDT в байтах;

16-бітний регістр задачі tr (*Task Register*) так само, як і регістр ldtr містить селектор, тобто покажчик на дескриптор в таблиці GDT. Цей дескриптор описує поточний сегмент стану задачі (TSS – *Task Segment Status*). Цей сегмент створюється для кожної задачі в системі, має жорстко регламентовану структуру та містить контекст (поточний

стан) задачі. Основне призначення сегментів TSS – зберігати поточний стан задачі в момент переключення на іншу задачу.

Регістри відлагоджування. Це досить цікава група регістрів, які призначені для апаратного налаштовування. Засоби апаратного налаштовування вперше з'явилися в мікропроцесорі i486. Апаратно мікропроцесор містить вісім регістрів налаштовування, однак реально використовуються тільки шість.

Регістри d0, d1, d2, d3 мають розрядність 32 біти та призначені для задавання лінійних адрес чотирьох точок переривання. При цьому використовується такий механізм: будь-яка формована поточна адреса порівнюється з адресами в регістрах d0…d3 і в разі відповідності генерується виключення з номером 1.

Регістр d6 називають регістром стану налагоджування. Біти цього регістра встановлюються відповідно до причин, які викликали виникнення останнього виключення за номером 1. Біти регістра d6:

b0 – якщо цей біт встановлено в 1, то останнє виключення (переривання) виникло внаслідок досягнення контрольної точки, яка визначена в регістрі dr0;

b1 – аналогічно до b0, однак для контрольної точки dr1;

b2 – аналогічно до b0, однак для контрольної точки dr2;

b3 – аналогічно до b0, однак для контрольної точки dr3;

bd – (біт 13) призначений для захисту регістрів відлагоджування;

bs – (біт 14) встановлюється в 1, якщо виключення 1 було викликане станом прапорця tf = 1 в регістрі eflags;

bt – (біт 15) встановлюється в 1, якщо виключення 1 було викликане переключенням на задачу зі встановленим бітом пастки в TSS $t = 1$.

Решта бітів у цьому регістрі заповнюються нулями. Оброблювач виключення 1 за вмістом dr6 повинен визначити причину, через яку виникло виключення, та виконати необхідні дії.

Регістр dr7 називається регістром управління відлагоджуванням. У ньому для кожного з чотирьох регістрів контрольних точок відлагоджування є поля, за допомогою яких можна уточнити такі умови, за яких необхідно згенерувати переривання:

- місце реєстрації контрольної точки – тільки в поточній задачі або в будь-якій задачі. Відповідні біти займають молодші вісім бітів регістра dr7 (по два біти на кожну контрольну точку (фактично точку переривання), яка задається регістрами dr0, dr1, dr2, dr3). Перший біт з кожної пари – це так званий локальний дозвіл, його встановлення свідчить про те, що точка переривання діє, якщо вона міститься в ме-

жах адресного простору поточної задачі. Другий біт кожної пари визначає глобальний дозвіл, який говорить про те, що дана контрольна точка діє в межах адресних просторів усіх задач, які є в системі;

тип доступу, за яким ініціюється переривання: тільки при виборі команди, записуванні або записуванні/читанні даних. Біти, які визначають природу виникнення переривання, локалізуються в старшій частині даного регістра.

Більшість системних регістрів програмно доступні. Коротко розглянуто лише деякі з них із метою зацікавлення подальшого вивчення архітектури мікропроцесора.

4.4.2. Структури даних захищеного режиму

У захищеному режимі будь-яке звернення до пам'яті як з боку прикладних програм, так і з боку операційної системи, повинне бути санкціонованим. Мікропроцесор апаратно контролює доступ програм до будь-якої адреси в оперативній пам'яті. Для отримання доступу цільова адреса, до якої хоче отримати доступ програма, повинна бути описана для програми. Це означає, що ділянка фізичної пам'яті, яка містить потрібну адресу, повинна описуватись за допомогою деякого дескриптора сегмента, який розміщується в одній із трьох дескрипторних таблиць. Локалізація цих таблиць здійснюється з використанням одного з розглянутих системних регістрів: gdtr, ldtr, idtr. Програмі, яка бажає використати дану ділянку пам'яті, повинен бути наданий покажчик на відповідний дескриптор в одній із двох таблиць – GDT або LDT. Робота з таблицею IDT здійснюється за іншими принципами, тому тут не розглядається. Покажчик на дескриптор сегмента в одній з таблиць GDT або LDT залежно від функціонального призначення ділянки пам'яті (сегмента), яка описується дескриптором, розміщується в один із шести сегментних регістрів. Отже, в захищеному режимі змінюється роль сегментного регістра – тепер він містить не адресу, а селектор, або індекс у таблиці дескрипторів сегментів. Однак саме призначення сегментних регістрів не змінюється – вони так само вказують на сегменти команд, даних та стеку, однак здійснюють це використовуючи інші механізми.

Структурно дескриптор сегмента являє собою 8-байтову структуру, яка зображена на рис. 4.4. Поля в дескрипторі описані в табл. 4.1.

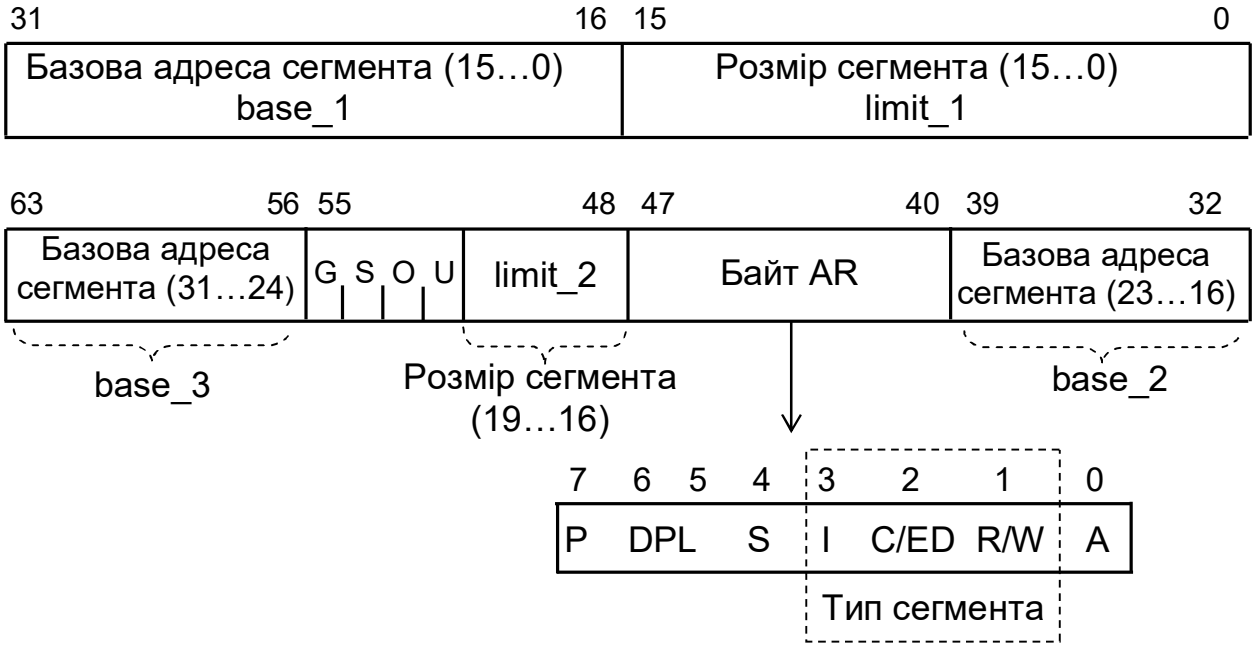

Рис. 4.4. Структура дескриптора сегмента захищеного режиму процесора

Таблиця 4.1.

Призначення полів дескриптора сегмента

Номер байта	Кількість бітів у полі	Символічне позначення	Призначення та вміст полів дескриптора
0…1	16	limit_1	Молодші біти 0…15 20-розрядного поля межі сегмента, який визначає розмір сегмента в одиницях вимірювання, які визначаються бітом гранулярності G
2…4	23	base_1	Біти 0…23 32-розрядної бази сегмента, яка визначає значення лінійної адреси початку сегмента в пам’яті
5	8	AR	Байт, поля якого визначають права доступу до сегмента (табл. 4.2)
6	0…3	limit_2	Старші біти 16…19 20-розрядної межі сегмента
6	1	U	Біт користувача (user). Спеціально не використовується, може використовуватись користувачем
6	1	–	0 – біт не використовується
6	1	D	Біт розрядності операндів та адрес: 0 – у програмі використовуються 16-розрядні операнди та режими 16-розрядної адресації; 1 – у програмі використовуються 32-розрядні операнди та режими 32-розрядної адресації

Закінчення табл. 4.1

Номер байта	Кількість бітів у полі	Символічне позначення	Призначення та вміст полів дескриптора
6	1	G	Біт гранулярності: 0 – розмір сегмента дорівнює значенню в полі *limit* у байтах; 1 – розмір сегмента дорівнює значенню в полі *limit* у сторінках
7	8	base_2	Біти 24…31 32-розрядної бази сегмента

З табл. 4.1 видно, що в захищеному режимі розмір сегмента нефіксований, його розмір можна задавати в межах 4 *Гбайт*. З рис. 4.4 видно, що поля, які визначають розмір сегмента та його базову (початкову) адресу, розірвані. Це є наслідком еволюції мікропроцесора. Захищений режим вперше з'явився в мікропроцесорі i80286. Цей мікропроцесор мав 24-розрядну адресну шину й тому міг адресувати до 16 *Мбайт* оперативної пам'яті. Для цього йому достатньо було мати в дескрипторі поле базової адреси 24 біти та поле розміру сегмента 16 *біт*. Після появи мікропроцесора i80386 з 32-розрядною шиною команд та даних із метою сумісності програм розробники не змінили формат дескриптора, а почали використовувати вільні поля. Усередині мікропроцесора ці поля об'єднані. Зовні вони залишилися розділеними й при програмуванні це потрібно враховувати.

Відомо, що в захищеному режимі розмір сегмента може досягати 4 *Гбайт*, тобто займати весь простір фізичної пам'яті. Однак розмір поля величини сегмента дорівнює 20 *біт*, що відповідає 1 *Мбайт*. На розмір сегмента впливає біт гранулярності G. Якщо G = 0, то значення в полі розмір сегмента означає розмір сегмента в байтах, а якщо G = 1, то в сторінках. Розмір сторінки – 4 *Кбайт*. Можна підрахувати, що коли максимальне значення поля розміру сегмента складає fffffh, то це відповідає 1М сторінок, що відповідає величині 1М*4 *Кбайт* = 4 *Гбайт*.

Тепер зрозуміло, що виведення інформації про базову адресу сегмента та його розміру на рівень мікропроцесора дозволяє апаратно контролювати роботу програм з пам'яттю та забороняти звернення до неіснуючих адрес або до адрес, які містяться за межею, яка визначена полем розміру сегмента limit.

Інший аспект захисту полягає в тому, що сегменти рівноправні в правах доступу до них. Інформація про це міститься в спеціальному байті AR, який входить до складу дескриптора. Формат цього байта показано на рис. 4.4, а в табл. 4.2 викладено його характеристику.

Таблиця 4.2

Байт AR дескриптора сегмента

Номер біта	Символічне позначення	Призначення та вміст
0	A	Біт доступу (*Accessed*) до сегмента. Встановлюється апаратно при зверненні до сегмента.
1	R W	Для сегмента коду це біт доступу за читанням (*Readable*); визначає чи можливе читання із сегмента коду при здійсненні заміни префікса сегмента: 0 – читання із сегмента заборонене; 1 – читання дозволене. Для сегментів даних це біт запису: 0 – модифікація даних у сегменті заборонена; 1 – модифікація дозволена.
2	C ED	Для сегментів кодів це біт підлеглості (*conforming*): 1 – підлеглий сегмент коду; 0 – звичайний сегмент коду. Для багатозадачного режиму визначає особливості зміни поточного рівня привілеїв. Для сегментів даних це біт розширення вниз (*Expand Down*); служить для розрізнення сегментів стеку та даних, а також визначає трактування поля *limit*: 0 – сегмент даних; 1 – сегмент стеку.
3	I	Біт призначення (*Intending*): 0 – сегмент даних або стеку; 1 – сегмент коду.
4	S	Якщо S = 1, то це біт сегмента (*Segment*). Для будь-яких сегментів у пам'яті дорівнює 1. Призначення та порядок використання сегмента уточнюється бітами *C* та *R*. Якщо S = 0, то це біт системний (*System*). Такий стан свідчить про те, що даний дескриптор описує спеціальний системний об'єкт, який може і не бути сегментом в пам'яті.
5…6	dpl	Поле рівня привілеїв сегмента (*Descriptor Previlege Level*). Містить числове значення від 0 до 3, яке визначає привілейованість сегмента, який описується даним дескриптором.
7	P	Біт присутності (*Present*): 0 – у даний момент сегмента немає в оперативній пам'яті; 1 – у даний момент сегмент знаходиться в оперативній пам'яті.

Найважливішими полями байта AR є dpl та біти R/W, C/ED, I, які разом визначають тип сегмента. Поле dpl – частина механізму захисту за привілеями. Суть цього механізму полягає в тому, що конкретному сегменту може бути властивий один із трьох рівнів привілейованості з номерами 0, 1, 2 та 3. Найбільш привілейованим є рівень із номером 0. Існує ряд обмежень на взаємодію сегментів із різними рівнями привілейованості. У межах цього навчального посібника вважається, що всім сегментам властивий нульовий рівень привілеїв.

Розглянувши детальніше поле типу сегмента (біти I, C/ED, R/W) видно, що це поле визначає цільове призначення сегмента. У табл. 4.2

ці біти відокремлені, однак для полегшення їх сумісної інтерпретації слід розглянути комбінації цих бітів разом. У табл. 4.3 наведено призначення деяких комбінацій цих бітів.

Таблиця 4.3

Комбінації в полі тип сегмента

Значення бітів			Призначення сегмента
I	C/ED	R/W	
000			Сегмент даних, тільки для читання
001			Сегмент даних із дозволом читання та запису
010			Не визначено
011			Сегмент стеку з дозволом читання та запису
100			Сегмент коду з дозволом лише виконання
101			Сегмент коду з дозволом виконання та читання з нього
110			Підпорядкований сегмент коду з дозволом виконання
111			Підпорядкований сегмент коду з дозволом виконання та читання з нього

Із таблиці видно, що можливі два принципово різні види сегментів: даних та коду. Сегмент стеку є різновидом сегмента даних, однак, з особливим розумінням поля розміру сегмента. Це пояснюється специфікою використання стеку: він зростає в напрямку молодших адрес. Видно, що поле типу сегмента обмежує використання оголошених сегментів. Зокрема програмні сегменти не можуть бути модифікованими без застосування спеціальних підходів. Доступ до сегмента даних також може обмежуватись лише для читання.

Отже, у захищеному режимі використанню будь-якої ділянки пам’яті повинні передувати певні дії з ініціалізації відповідного дескриптора. Цю роботу виконує операційна система або програма, сегменти якої також описуються подібними дескрипторами.

При ввімкненні живлення або натисканні кнопки reset мікропроцесор починає свою роботу в реальному режимі. У цьому режимі він виконує дії з тестування апаратури комп’ютера. Після вдалого завершення тестування мікропроцесор виконує початкове завантаження системи, використовуючи програму початкового завантаження, яка зберігається на нульовій доріжці диска. Програма початкового завантаження зчитує з диска програму ініціалізації операційної системи та передає їй управління. Дії цієї програми залежать від того, у якому режимі роботи мікропроцесора буде здійснюватись подальше функціонування системи. Якщо в реальному режимі, то операційна система формує середовище та структури даних для роботи в цьому режимі.

Якщо операційна система, яка завантажується, буде працювати в захищеному режимі, то вона повинна перейти в нього відповідним чином. Однак перед цим операційна система формує спеціальні структури даних (наприклад дескрипторні таблиці) для роботи в захищеному режимі. Тільки після цього можна переходити в захищений режим та виконувати подальші дії.

4.4.3. Формування фізичної адреси в захищеному режимі

Найважливішою відмінністю захищеного режиму від реального є інший принцип формування фізичної адреси. Як вже зазначалося, в реальному режимі адреса будь-якої комірки пам'яті, що задається в програмі, складається з двох компонентів – сегментної адреси та зсуву. Цю адресу часто називають віртуальною, а всю сукупність адрес, які можуть зустрічатися в програмі, – віртуальним адресним простором. Обидва компоненти віртуальної адреси мають довжину 16 розрядів, і процесор, звертаючись до пам'яті, користується правилом обчислення фізичної адреси:

*Фізична адреса = сегментна адреса * 16 + зсув.*

Як сегментна адреса, так і зсув не можуть бути більше величини `0×FFFF`, звідки випливають два найважливіші обмеження реального режиму: об'єм адресного простору складає лише 1 *Мбайт*, а сегменти не можуть мати розміру, що перевищує 64 *Кбайт*.

У захищеному режимі програма як і раніше складається із сегментів, що адресуються за допомогою 16-розрядних сегментних регістрів, і місце розташування байта, що адресується, визначається, як і раніше, завданням зсуву в сегменті, проте, перетворення віртуальної адреси у фізичну здійснюється не за наведеною вище формулою, а шляхом більш складних перетворень (рис. 4.5).

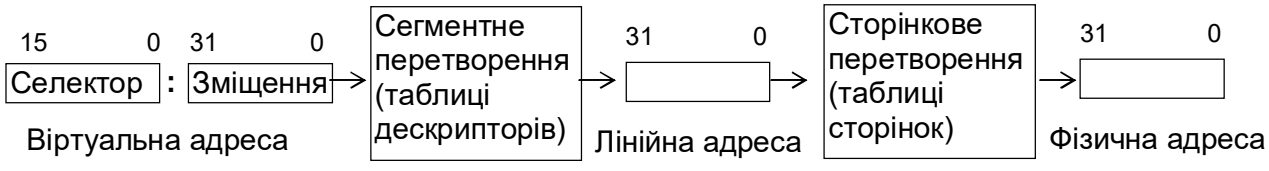

Рис. 4.5. Перетворення віртуальної адреси у фізичну

Спочатку процесор перетворює віртуальну адресу в лінійну. У процесі цього перетворення процесор звертається до таблиць дескрипторів, які заздалегідь будуються операційною системою захищеного режиму. На другому етапі за лінійною адресою визначається

фізична адреса пам’яті. У цьому перетворенні бере участь інший набір системних таблиць – таблиці сторінкової трансляції, які також складаються операційною системою. Обидва набори системних таблиць можуть динамічно змінюватися операційною системою в процесі роботи програм, забезпечуючи максимально ефективне використання ОП.

Слід підкреслити, що, хоча таблиці перетворення будуються операційною системою, саме перетворення віртуальної адреси у фізичну здійснює процесор на апаратному рівні. Самі таблиці знаходяться у фізичній пам’яті, і в процесі перетворення процесор вимушений додатково звертатися до пам’яті за додатковою інформацією. Проте з урахуванням кешування ділянок фізичної пам’яті, що адресуються, цей процес здійснюється достатньо швидко.

Детальніше розглянуто процес перетворення віртуальної адреси у фізичну, почавши з першого етапу цього механізму – визначення лінійної адреси.

У сегментні регістри в захищеному режимі записуються не сегментні адреси, а так звані селектори, до складу яких входять номери (індекси) елементів спеціальної (дескрипторної) таблиці, що містить дескриптори сегментів програми. Кожен дескриптор таблиці дескрипторів має розмір 8 *байт*, і в ньому зберігаються всі характеристики, необхідні процесору для обслуговування цього сегмента: базова лінійна адреса сегмента, його межа, яка є номером останнього байта сегмента, а також атрибутами сегмента, які визначають його властивості (рис. 4.6). Отже, селектори характеризують сегменти пам’яті.

Процесор за допомогою селектора визначає індекс дескриптора сегмента, що адресується, витягає з нього базову лінійну 32-розрядну адресу сегмента і, додавши до нього 32-розрядний зсув, утворює лінійну адресу комірки пам’яті, яка адресується.

Ця адреса називається лінійною, оскільки на відміну від віртуальної адреси, що складається з двох частин, вона є просто 32-розрядним числом, яке може набувати будь-якого значення від 0 до 0×FFFFFFFF, тобто до 4 *Гбайт* – 1.

Тут доречно зупинитися на відмінності двох видів прикладних програм, які можуть виконуватися в захищеному режимі, а саме 16- і 32-розрядних застосувань.

На відміну від 32-розрядних 16-розрядні додатки використовують лише 16-розрядні зсуви. Адресація в сегменті команд здійснюється за допомогою регістра IP (а не EIP); адресація в сегменті стеку – за допомогою регістра SP. Для вказівки зсувів у сегментах даних

використовуються молодші половини розширених регістрів процесора або 2-байтові поля в складі команди. Внаслідок цього розміри сегментів не можуть перевищувати 64 *Кбайт* і для побудови великої програми доводиться використовувати декілька сегментів команд або декілька сегментів даних.

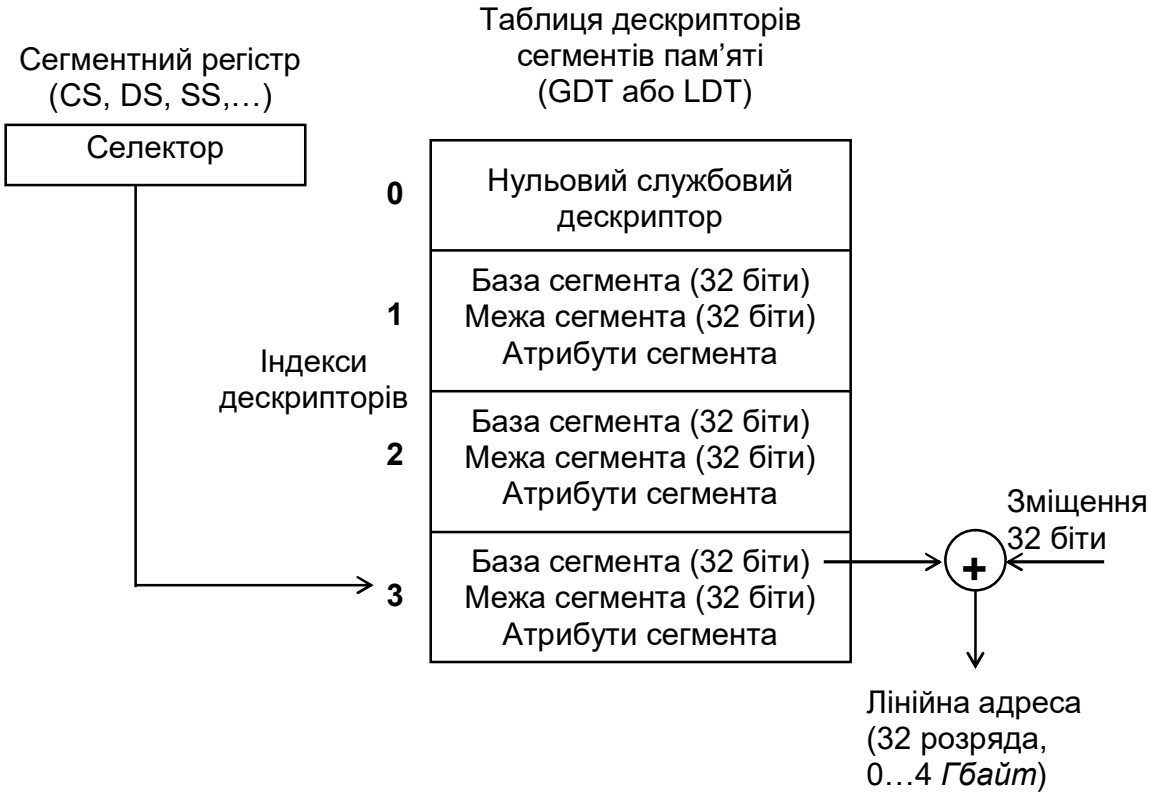

Рис. 4.6. Перетворення віртуальної адреса в лінійну

Інакше функціонують 32-розрядні застосування захищеного режиму. У них для завдання зсувів використовуються розширені регістри процесора, зокрема EIP для адресації в сегменті команд і ESP для роботи зі стеком. Обмеження 64 *Кбайт* вже не діє, сегменти можуть бути будь-якого розміру, і необхідності у побудові багатосегментних програм немає. На сьогодні майже всі нові застосування роблять 32-розрядними.

Згідно з рис. 4.6. архітектурою процесора передбачено таблиці дескрипторів двох типів – таблиця глобальних дескрипторів GDT (від *Global Descriptor Table*) і таблиця локальних дескрипторів LDT (від *Local Descriptor Table*). Відмінність між ними полягає в тому, що сегменти, описані в GDT, можуть бути доступні всім виконуваним задачам (як прикладним, так і системним); сегменти, описані в LDT, доступні лише тій задачі, до якої належить дана локальна таблиця. У глобальній таблиці описуються сегменти операційної системи,

а також сегменти, у яких розміщено самі LDT; у локальних таблицях (їх може бути стільки, скільки додатків виконується наразі) описуються сегменти конкретних задач. Реально *Windows* будує одну таблицю глобальних дескрипторів GDT, заповнену системними дескрипторами, і одну таблицю локальних дескрипторів LDT, у якій, крім системних дескрипторів, передбачаються місця для дескрипторів застосувань, що запускаються.

У міру запуску 16-розрядних застосувань *Windows* для їх сегментів (команд, даних і стеку) в LDT виділяє дескриптори (і відповідно, селектори) із числа резервних. При цьому в дескриптори заносяться характеристики сегментів конкретних застосувань, зокрема їх реальні розміри. Отже, усі запущені 16-розрядні застосування мають унікальні селектори, а закріплені за ними дескриптори описують ділянки єдиного лінійного адресного простору, що не перекриваються.

За допомогою таблиці сторінкової трансляції (процедуру трансляції буде описано нижче) ці ділянки лінійного простору відображуються на фізичну пам’ять.

На рис. 4.7 показано варіант такого відображення для конкретного прикладу двох 16-розрядних застосувань, що виконуються в системі *Windows* 98.

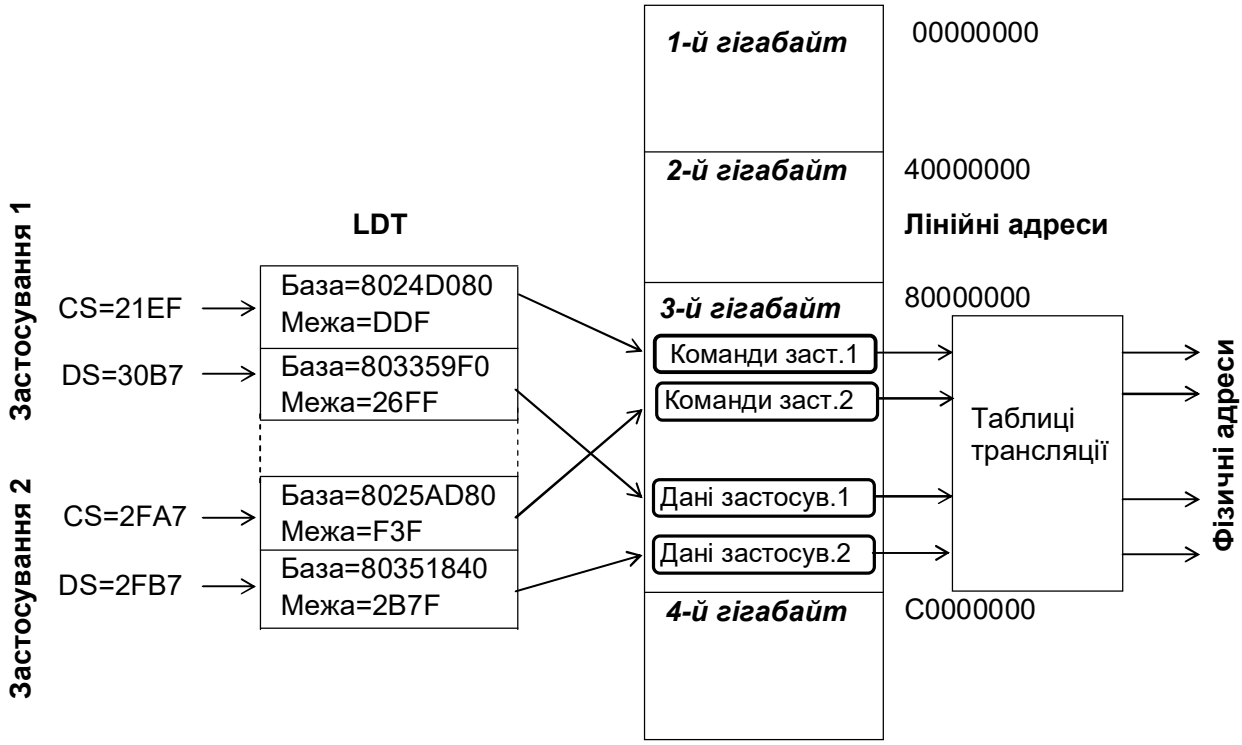

Рис. 4.7. Приклад перетворення адрес для 16-розрядних застосувань у *Windows*

При запуску застосувань система призначила їх сегментам довільні дескриптори із числа вільних в LDT (із селекторами 0×21EF, 0×30B7

і т.д.). У дескриптори були поміщені характеристики сегментів: межі, що відповідають реальним розмірам сегментів застосувань, і бази, які відповідають вільним ділянкам лінійного простору. Слід зауважити, що всі сегменти розміщені в 3-му гігабайті (в області лінійних адрес від 0×80000000 до 0×C0000000).

Лінійні адреси не відповідають будь-яким фізичним об'єктам. Можна сказати, що це фіктивні адреси, що виникають на півдорозі від віртуальних адрес, з якими працює застосування, до фізичних адрес реальної оперативної пам'яті комп'ютера. Проте вони мають велике значення й використовуються певним чином. Так, 16-розрядні застосування *Windows* 98 завжди працюють в 3-му гігабайті лінійного адресного простору.

Отримавши лінійну адресу байта, що адресується, процесор за допомогою таблиць трансляції перетворить його в 32-розрядну фізичну адресу того байта пам'яті, де знаходиться конкретна команда або конкретні дані. Ця адреса залежить від об'єму пам'яті, встановленої на комп'ютері, і може розташовуватися в будь-якому місці фізичної пам'яті (крім 1-го мегабайта, де розміщуються програми MS-DOS, відеопам'ять і ПЗП BIOS).

Чому селектори на рис. 4.7 мають такі значення? Як селектор пов'язаний із номером дескриптора дескрипторної таблиці? Слід розглянути формат селектора, зображеного на рис. 4.8.

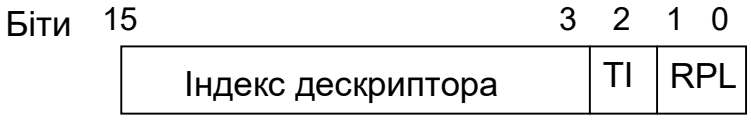

Рис. 4.8. Селектор дескриптора

Порядковий номер (індекс) дескриптора в таблиці дескрипторів записується в селектор починаючи з біта 3, що еквівалентно множенню його на 8. Якщо пригадати, що дескриптори мають довжину 8 *байт*, то стає зрозумілим, що в селекторі (якщо не брати до уваги 3 молодші біти) зберігається зсув дескриптора від початку таблиці дескрипторів.

Біт TI (*Table Indicator* – індикатор таблиці) дорівнює нулю, якщо селектор належить до глобальної таблиці, і одиниці, якщо відповідний селектору дескриптор входить в локальну таблицю.

Поле RPL (*Request Privilege Level* – запитаний рівень привілеїв), що займає біти 0...1 селекторів, може набувати значень від 0 до 3 відповідно до числа рівнів привілеїв процесора. Програми *Windows*, що належать до ядра операційної системи, виконуються на найбільш пріоритетному рівні 0; менш відповідальні системні програми, а також

всі застосування виконуються на найнижчому пріоритетному рівні привілеїв, який дорівнює трьом.

Щоб розшифрувати значення селектора 0×21EF, призначеного системою сегмента команд застосування 1 (див. рис. 4.7), слід записати його у вигляді двійкового числа (рис. 4.9).

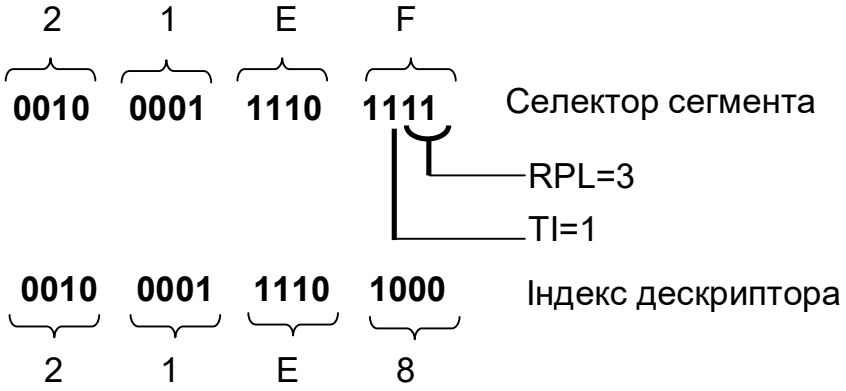

Рис. 4.9. Розшифрування значення селектора

Із рис. 4.9 видно, що рівень привілеїв RPL = 3, TI = 1 (дескриптор належить таблиці LDT), а зсув дескриптора в LDT складає 0×21E8, що відповідає номеру дескриптора 0×21E8/8 = 0×43D = 1085.

Цікаво зазначити, що в системах *Windows* NT/2000 перетворення адрес 16-розрядних застосувань здійснюється так само, проте, в лінійному адресному просторі для них виділений не 3-й, а 1-й гігабайт.

По-іншому відбувається з 32-розрядними застосуваннями *Windows*. Система надає всім таким застосуванням, скільки б їх не було, той самий селектор для сегмента команд і той самий селектор для сегмента даних (рис. 4.10). *Windows* 98 призначає всім сегментам команд селектор 0×167, а сегментам даних – селектор 0×l6F. У *Windows* NT/2000 виділяються відповідно селектори 0×1B і 0×23. Далі, бази обох сегментів дорівнюють нулю, а межі – 4 *Гбайт* – 1. Іншими словами, кожному запущеному застосуванню ніби надається весь лінійний адресний простір. Заведено говорити, що такі застосування працюють у моделі плоскої пам'яті (інколи для позначення плоскої пам'яті використовують англомовний термін "FLAT", що є, до речі, ключовим словом мови *Assembler*).

Зрозуміло, що жодне застосування не може використовувати всю плоску пам'ять. По-перше, лінійний адресний простір – це, як вже наголошувалося, фікція, а застосування, щоб воно могло виконуватися, повинне знаходитися в ОП, об'єм якої зазвичай значно менше 4 *Гбайт*. По-друге, помітну частину лінійного простору займають

програми операційної системи. По-третє, системи *Windows* надають для прикладних 32-розрядних програм лише половину адресного простору, а саме 1-й і 2-й гігабайти, резервуючи 3-й і 4-й гігабайти для системних програм.

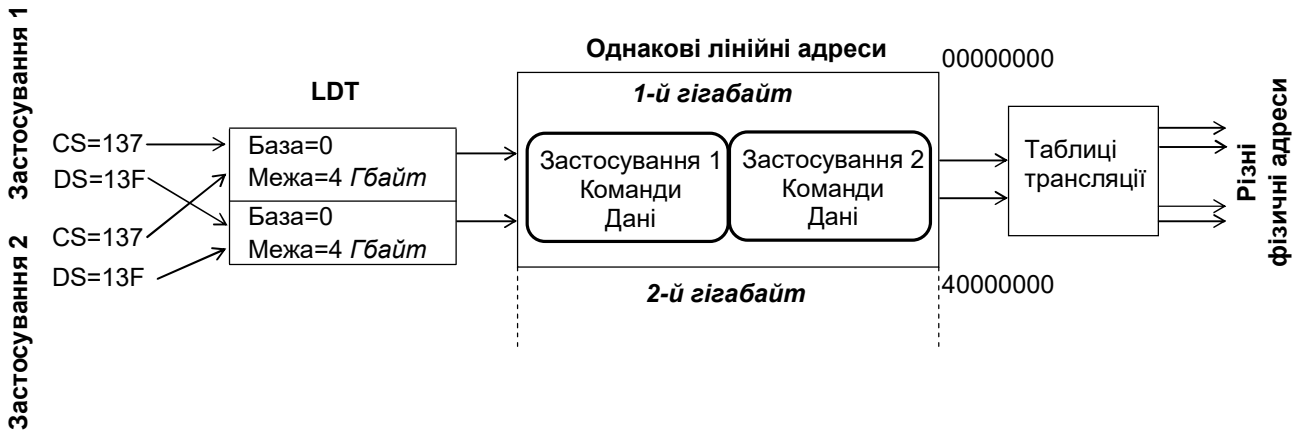

Рис. 4.10. Формування фізичних адрес застосування

Оскільки базові лінійні адреси всіх (але фактично лише двох) сегментів програми дорівнюють нулю, віртуальні зсуви, з якими працює програма, збігаються з лінійними адресами. Іншими словами, плоский *віртуальний* адресний простір програми збігається з плоским *лінійним* адресним простором. При цьому всі застосування використовують ті самі діапазони лінійних адрес. Для того, щоб при однакових лінійних адресах різні застосування займали окремі ділянки фізичної пам'яті, не затираючи один одного, *Windows* при зміні поточного застосування змінює таблиці сторінкової трансляції, за допомогою яких і відбувається перетворення лінійних адрес у фізичні адреси ОП.

Сторінкова трансляція є достатньо складним механізмом, у якому беруть участь апаратні засоби процесора й таблиці сторінкової трансляції, що знаходяться в пам'яті. Призначення та взаємодію елементів системи сторінкової трансляції схематично зображено на рис. 4.11.

Система сторінкових таблиць складається з двох рівнів. На першому рівні знаходиться каталог таблиць сторінок (або просто каталог сторінок) – це резидентна в пам'яті таблиця, що містить 1024 4-байтових полів з адресами таблиць сторінок. На другому рівні – таблиці сторінок, кожна з яких містить також 1024 4-байтових поля з адресами фізичних сторінок пам'яті. Оскільки розмір фізичної сторінки складає 4 *Кбайт*, 1024 таблиці по 1024 сторінки перекривають весь адресний простір (4 *Гбайт*).

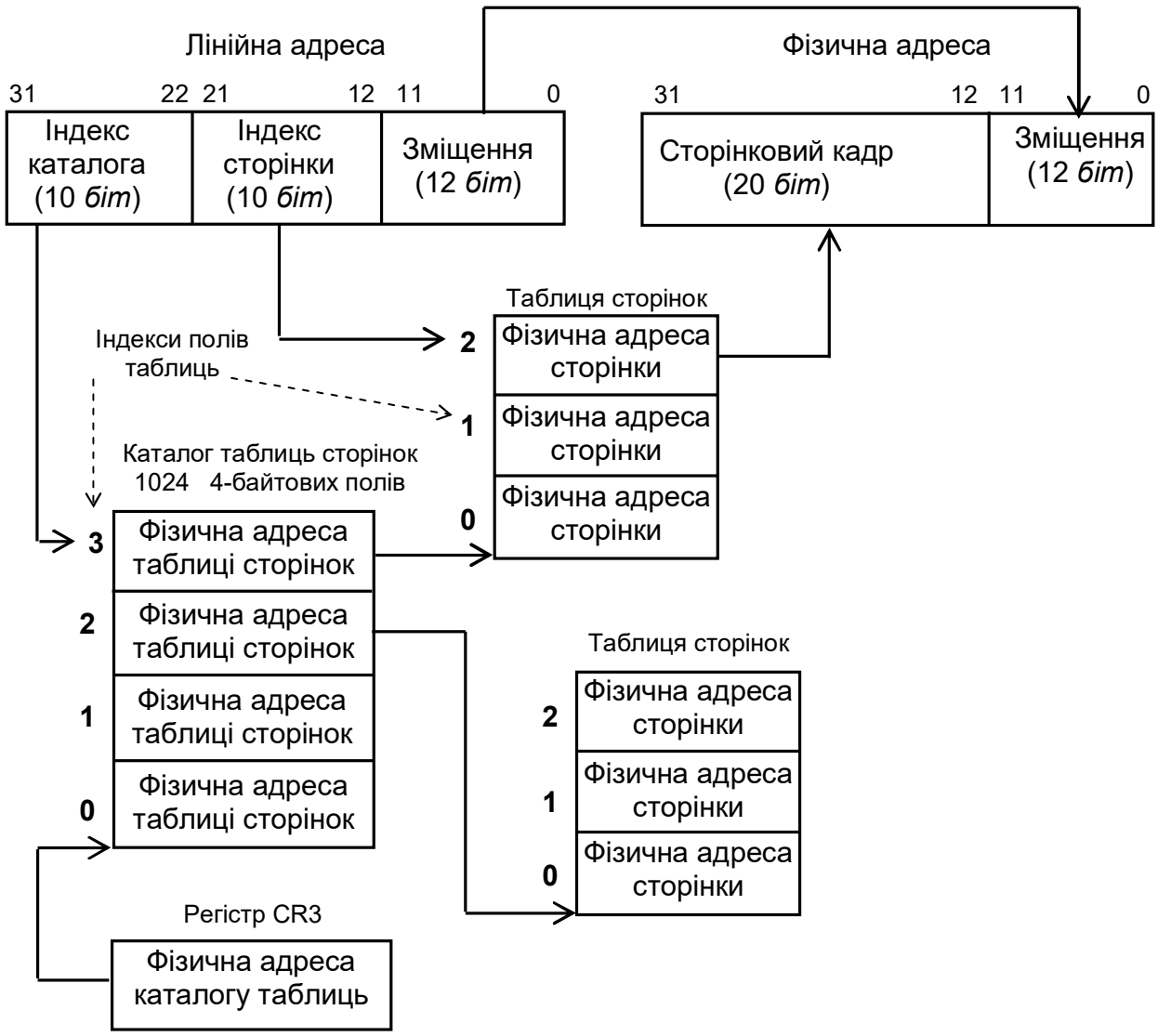

Рис 4.11. Сторінкова трансляція адрес

Не всі 1024 таблиці сторінок повинні обов'язково бути в наявності (вони зайняли б в пам'яті достатньо багато місця – 4 *Мбайт*). Якщо програма реально використовує лише частину можливого лінійного адресного простору (так завжди і відбувається) то невикористовувані поля в каталозі сторінок позначаються як відсутні. Для таких полів система, економлячи пам'ять, не виділяє сторінкові таблиці. З іншого боку, каталог сторінок (або сторінка каталогу, як його інколи називають, оскільки весь каталог має розмір 4 *Кбайт*) повинен завжди знаходитися у фізичній пам'яті.

При включеній сторінковій трансляції лінійна адреса розглядається як сукупність трьох полів: 10-бітового індексу в каталозі сторінок, 10-бітового індексу у вибраній таблиці сторінок і 12-бітового зсуву у вибраній сторінці.

Старші 10 бітів лінійної адреси утворюють номер елемента в каталозі сторінок. Базова фізична адреса каталогу зберігається в одному з регістрів управління процесора – в регістрі CR3. Через те, що каталог сам є сторінкою й вирівняний у пам'яті по межі 4 *Кбайт*, в регістрі CR3 для адресації до каталогу використовуються лише старші 20 бітів, а молодші 12 бітів зарезервовані для майбутнього використання.

Елементи каталогу мають розмір 4 *байт*, тому індекс, що вилучається з лінійної адреси, зсувається вліво на 2 біти (тобто множиться на 4) і отримана величина додається до базової адреси каталогу, утворюючи адресу конкретного елемента каталогу. Кожен елемент каталогу містить фізичну базову адресу однієї з таблиць сторінок, причому, оскільки таблиці сторінок самі є сторінками й вирівняні в пам'яті по межі 4 *Кбайт*, в цій адресі значущими є лише старші 20 бітів.

Далі з лінійної адреси вилучається середня частина (біти 12...21), зсовується вліво на 2 біти і додається до базової адреси, що зберігається у вибраному полі каталогу.

Як наслідок утворюється фізична адреса сторінки в пам'яті, у якій знову ж таки використовуються лише старші 20 бітів. Ця адреса, що розглядається як старші 20 бітів фізичної адреси комірки, що адресується, має назву *сторінкового кадру*. Сторінковий кадр доповнюється з правого боку молодшими 12 бітами лінійної адреси, які проходять через сторінковий механізм без зміни і відіграють роль зсуву всередині вибраної фізичної сторінки. Внаслідок цього кожні 4 *Кбайт* лінійних адрес відображуються на 4 *Кбайт* фізичних, причому операційна система, змінюючи вміст таблиць трансляції адрес, може легко змінювати відображення загального для всіх програм лінійного адресного простору на необхідні ділянки фізичної ОП, що містять елементи тієї або іншої активної програми.

Переваги сегментації пам'яті:

1. Можливість організувати декілька незалежних сегментів пам'яті для процесу і використати їх для зберігання даних різного призначення. При цьому права доступу до кожного такого сегмента можуть бути задані по-різному.

2. Окремі сегменти можуть спільно використовуватися різними процесами, для цього їх таблиці дескрипторів сегментів повинні містити однакові елементи, що описують такий сегмент.

3. Фізична пам'ять, що відповідає адресному простору процесу, повинна необов'язково бути неперервною. Сегментація дає змогу окремим частинам адресного простору процесу відображатися не

в основну пам’ять, а на диск, і довантажуватися з нього за потреби, забезпечуючи виконання процесів будь-якого розміру. Але має деякі недоліки:

1. Необхідність введення додаткового рівня перетворення пам’яті спричиняє зниження продуктивності (цей недолік властивий будь-якій повноцінній реалізації віртуальної пам’яті). Для ефективної реалізації сегментації потрібна відповідна апаратна підтримка.

2. Управління блоками пам’яті змінної довжини з урахуванням необхідності їх збереження на диску може бути досить складним.

3. Вимога, щоб кожному сегменту відповідав неперервний блок фізичної пам’яті відповідного розміру, спричиняє зовнішню фрагментацію пам’яті. Внутрішньої фрагментації у цьому разі не виникає, оскільки сегменти мають довжину й завжди можна виділити сегмент довжини, необхідної для виконання програми. На сьогодні сегментацію застосовують доволі обмежено, передусім, через фрагментацію й складність реалізації ефективного звільнення пам’яті та обміну з диском. Більш широкого використання набув розподіл пам’яті на блоки фіксованої довжини – *сторінкова організація пам’яті.*

Порівняльний аналіз сторінкової організації пам’яті та сегментації. Сторінкова організація пам’яті та сегментація мають більше спільних рис, ніж відмінностей. Основна відмінність між цими двома підходами полягає в тому, що всі сторінки мають фіксовану довжину, а сегменти – змінну. Інші базові моменти (відсутність вимоги неперервності фізичної пам’яті, можливість вивантаження блоків пам’яті на диск, необхідність підтримувати таблиці перетворення тощо) принципово не відрізняються.

Основні переваги сторінкової організації пам’яті порівняно із сегментацією, визначаються, насамперед, тим, що всі сторінки мають ту саму довжину.

1. Реалізація розподілу й звільнення пам’яті спрощується. Усі сторінки з погляду процесу рівноправні, тому можна підтримувати список вільних сторінок, у разі необхідності виділяти першу сторінку із цього списку, а після звільнення повертати сторінку до списку. Із сегментами так чинити не можна, оскільки кожен сегмент можна використати лише за його призначенням (спроба використати сегмент з іншою метою призведе, швидше за все, до того, що виникне потреба у сегменті іншої довжини).

2. Реалізація обміну даними з диском також спрощується. Для організації такого обміну ділянка на диску, яку використовують для

зберігання інформації про сторінки, вивантажені з пам'яті (простір підтримки, *backing store*) може бути теж розбита на блоки фіксованого розміру, що дорівнює розміру фрейму.

Сторінкова організація пам'яті має такі недоліки:

1. Насамперед, цей підхід спричиняє внутрішню фрагментацію, пов'язану з тим, що розмір сторінки завжди фіксований, і в разі необхідності виділення блока пам'яті конкретної довжини його розмір буде кратним розміру сторінки. У середньому розмір невикористовуваної пам'яті становить приблизно половину сторінки для кожного виділеного блока пам'яті (аналогічного до сегмента). Така фрагментація може бути знижена зменшенням кількості та збільшенням розміру блоків, що виділяються.

2. Таблиці сторінок повинні бути більші за розміром, ніж таблиці сегментів. Так, для виділення неперервного діапазону пам'яті розміром 100 *Кбайт* знадобиться один елемент таблиці сегментів, що описує сегмент, виділений для цього діапазону. З іншого боку, у разі використання сторінок розміром 4 *Кбайт* для опису такого діапазону знадобиться 25 елементів таблиці сторінок – по одному елементу для кожної сторінки.

Багаторівневі таблиці сторінок. Щоб адресувати логічний адресний простір значного обсягу за допомогою однієї таблиці сторінок, її доводиться робити надто великою. Наприклад, в архітектурі IA-32 для стандартного розміру сторінки 4 *Кбайт* (для адресації всередині такої сторінки потрібно 12 бітів) на індекс у таблиці залишається 20 бітів, що відповідає таблиці сторінок на 1 мільйон елементів.

Щоб уникнути таких великих таблиць, запропоновано технологію *багаторівневих таблиць сторінок.* Таблиці сторінок самі розбиваються на сторінки, інформацію про які зберігають у таблиці сторінок верхнього рівня. Кількість рівнів рідко перевищує 2, але може доходити й до 4.

Коли є два рівні таблиць, логічну адресу розбивають на індекс у таблиці верхнього рівня, індекс у таблиці нижнього рівня і зсув.

Ця технологія має дві основні переваги. По-перше, таблиці сторінок менші за розміром, тому пошук у них можна робити швидше. По-друге, не всі таблиці сторінок повинні перебувати в пам'яті у конкретний момент часу. Наприклад, якщо процес не використовує який-небудь блок пам'яті, то вміст усіх сторінок нижнього рівня невикористовуваного блока може бути тимчасово збережений на диску.

Реалізація таблиць сторінок в архітектурі IA-32. Архітектура IA-32 використовує дворівневу сторінкову організацію, починаючи з моделі *Intel* 80386. Таблицю верхнього рівня називають *каталогом сторінок* (*page directory*), для кожної задачі повинен бути заданий окремий каталог сторінок, фізичну адресу якого зберігають у спеціальному керувальному регістрі cr3 і куди він автоматично завантажується апаратним забезпеченням при переключенні контексту. Таблицю нижнього рівня називають просто *таблицею сторінок* (*page table*).

Механізм сторінкового управління пам'яттю включається встановленням біта PG в регістрі CR0. В регістрі CR2 зберігається лінійна адреса відмови та адреса пам'яті, за якою було виявлено останню відмову сторінки. Регістр CR3 містить фізичну адресу каталогу сторінок. Молодші 12 бітів цього регістра завжди дорівнюють нулю (вирівнювання по границі сторінок). Каталог сторінок складається з 32 бітних елементів та має об'єм 4 *Кбайт*. Структура елемента каталогу наведено на рис. 4.12.

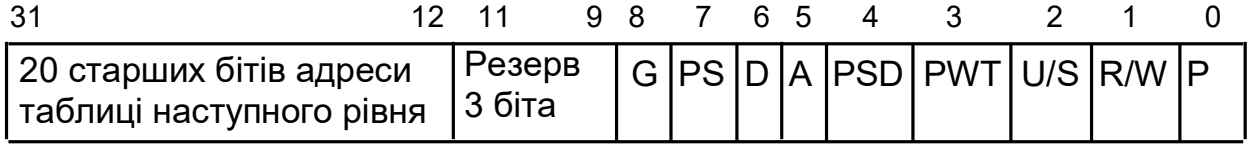

Рис. 4.12. Елемент каталогу (таблиці) сторінок

Кожна таблиця сторінок також має розмір 4 *Кбайт* і елементи аналогічного формату. Однак ці елементи містять базову адресу самих сторінок та атрибути сторінок. Фізична адреса складається з базової адреси та молодших 12 бітів лінійної адреси. Значення атрибутів сторінок:

G – глобальна сторінка. Сторінка не видаляється з буфера;

PS – розмір сторінки. Якщо 1, то розмір сторінки – 2 *Мбайт* або 4 *Мбайт*, інакше 0;

D – прапорець зміни (*Dirty*), такий, що дорівнює одиниці під час кожної операції записування в сторінку;

A – біт доступу (*Accessed*), такий, що дорівнює одиниці під час кожного звертання пристрою сторінкової підтримки до відповідної сторінки;

PCD – біт заборони кешування;

PWT – біт дозволу наскрізного записування;

U/S – біт привілейованого режиму (*User/Superversion*), який визначає режим процесора, необхідний для доступу до сторінки. Якщо цей прапорець дорівнює нулю, сторінка може бути адресована тільки

із привілейованого режиму, якщо одиниці – доступна також і з режиму користувача (для програміста з рівнем доступу 3);

R/W – біт читання-записування (*Read/Write*), що задає права доступу до цієї сторінки або таблиці сторінок (для читання й записування або тільки для читання);

P – біт присутності (*present*), дорівнює одиниці, якщо сторінка перебуває у фізичній пам'яті (їй відповідає фрейм); рівність цього прапорця нулю означає, що сторінки у фізичній пам'яті немає, при цьому операційна система може використати інші поля елемента для своїх цілей.

Прапорці присутності, доступу і зміни можна використовувати для організації віртуальної пам'яті.

Контрольні питання

1. Назвати режими роботи процесора.
2. Дати характеристику реального режиму роботи процесора.
3. Дати характеристику захищеному режиму роботи процесора.
4. Як адресуються дані в реальному режимі роботи процесора?
5. Як адресуються дані в захищеному режимі роботи процесора?
6. Дати характеристику сегментованої моделі пам'яті.
7. Дати характеристику сторінкової моделі пам'яті.
8. Яке призначення дескриптора сегмента? Яка його розрядність?
9. Назвати основні поля дескриптора сегмента.
10. Назвати переваги та недоліки сегментованої моделі пам'яті.
11. Назвати переваги та недоліки сторінкової моделі пам'яті.