Тема 5. ***ПЕОМ на базі 32-розрядних МП***

Серію 32-розрядних процесорів фірми Intel започаткував мікропроцесор Intel 80386. Ці процесори увібрали в себе всі властивості попередників, тобто 16-розрядних процесорів і забезпечили програмну сумісність з значним доробком програмного забезпечення, написаного для них. Процесори забезпечують чотирирівневу систему захисту пам'яті і уведення/виведення, перемикання задач. Вони мають розширену систему команд, яка також включає всі команди 8086 та 80286. Можливо, найціннішим є те, що у цих процесорах значно відсунуто верхню межу адресування оперативної пам'яті. Процесори можуть працювати у двох режимах:
реального та захищеного віртуального адресування.

У реальному режимі процесор працює як дуже швидкий 8086, проте з 32-розрядним розширенням. Реальний режим також використовують для підготовки мікропроцесора до роботи в захищеному режимі.

Віртуальний режим забезпечує доступ до дуже складного, сучасного способу керування пам’яттю, а також для підкачування (заміщення чи свопінгу) сторінок та інших можливостей мікропроцесора.

У віртуальному режимі програмне забезпечення процесора може розв’язувати задачі за степенем складності такі ж, як у 8086 і 80286. Цей режим дає змогу виконувати програмне забезпечення та прикладні програми 8086 одночасно з операційною системою та прикладним забезпеченням 32-розрядного процесора.

Віртуальні задачі 32-розрядних МП можуть бути захищені одна від іншої та операційної системи шляхом використання заміщення сторінок, емуляції команд уведення-виведення.

Для суміщення комп’ютера з високоефективними системами інтерфейс шини 32-розрядних МП має конвеєрне опрацювання даних, динамічне резервування шини даних і прямі сигнали BYTE ENABLE.

Головною особливістю є апаратна реалізація багатосистемного програмного середовища, яке забезпечує сумісну роботу програм користувачів, зорієнтованих на операційні системи UNIX, DOS і APX86.

Ознайомимося з архітектурою цих процесорів детальніше.

5.1. Загальна характеристика мікропроцесора Intel 80386. Опис регістрів

Мікропроцесор Intel80386 складається з центрального процесора, блока керування пам’яттю та шини інтерфейсу (див. рис. 5.1).

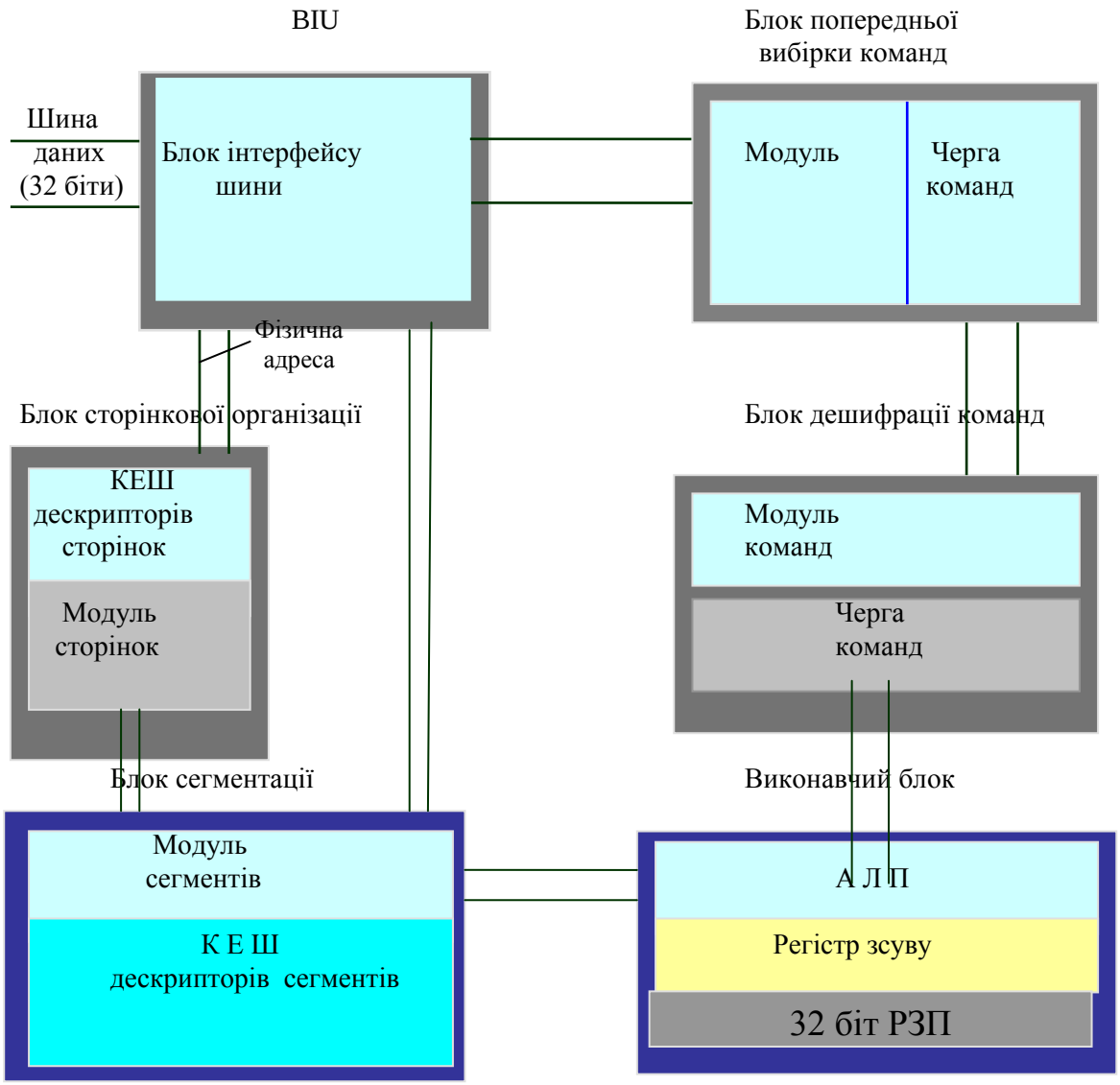
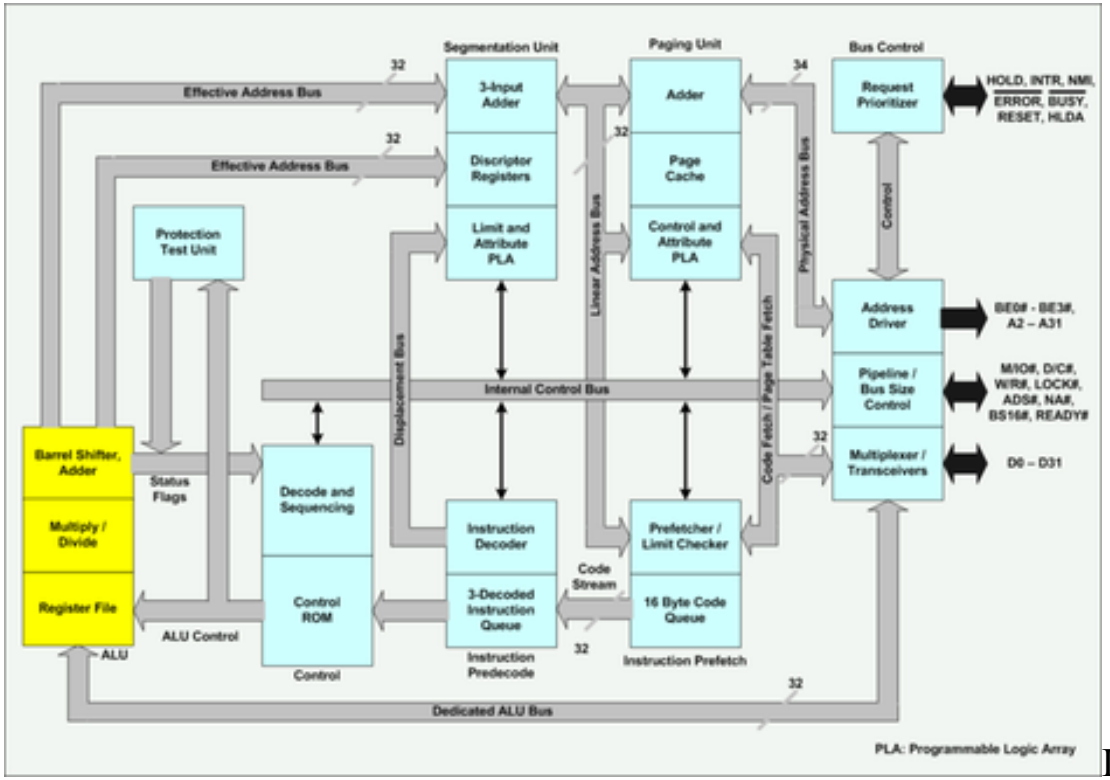


Рис. 5.1. Структурна схема мікропроцесора Intel80386



Центральний процесор має виконавчий блок, що містить вісім 32-розрядних регістрів загального призначення, які використовують як для обчислення адреси, так і для операцій з даними, та арифметико-логічний пристрій.

Пристрій опрацювання команд дешифрує коди операцій команд і поміщає їх у чергу для невідкладного виконання виконавчим блоком.

Пристрій керування пам'яттю складається із блока сегментації та пристрою заміщення сторінок (підкачування).

Сегментація дає змогу керувати простором логічної адреси.

Механізм підкачування працює "внизу" і паралельний до процесу сегментації, що дає змогу керувати простором фізичної адреси. Одна сторінка його становить 4 Кбайти.

Для використання системи віртуальної пам’яті 80386 підтримує повну відновлюваність для всіх помилок на сторінці і сегменті. Пам’ять може бути організована в один або більше сегментів різної довжини до 4 Гбайт.

Будь-яке завдання на 80386 може мати 16 381 сегмент, довжиною до 4 Гбайт кожний, тобто забезпечує 64 Тбайти віртуальної пам’яті для кожної задачі.

Блок сегментації забезпечує чотири рівні захисту для ізоляції та захисту пристроїв і операційної системи одне від одного. Конструкція мікропроцесора дає змогу всі системи сполучати в одне ціле.

5.1.1. Огляд регістрів

МП 80386 має 32 програмно-доступні регістри у таких категоріях:

1) регістри загального призначення;
2) регістри сегментування;
3) регістр ознак;
4) регістри керування;
5) регістри системної адреси;
6) регістр тестів;
7) регістр відлагоджування.

Ці регістри є також набором 8086, 80286, тому всі 16-розрядні регістри всіх процесорів попередніх випусків "покриваються" 32- розрядними регістрами 80386.

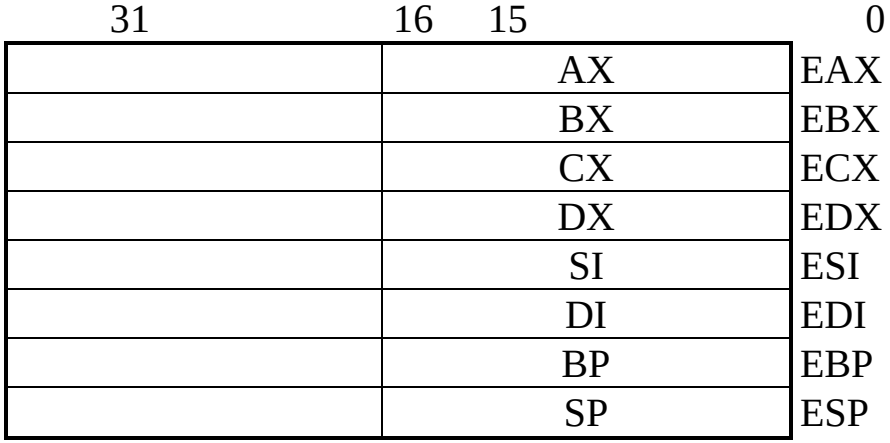


Рис. 5. 2. Регістри даних і адресування

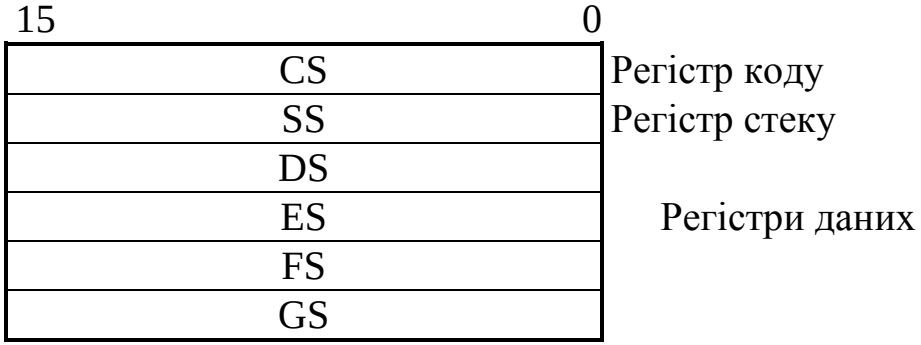


Рис. 5. 3. Сегментні регістри

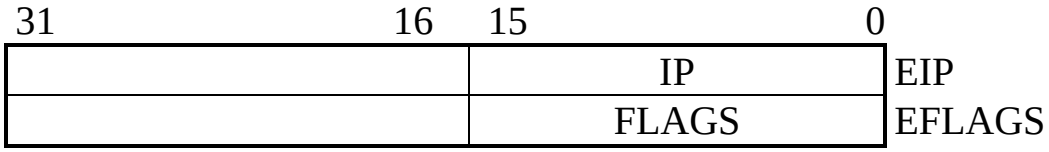


Рис. 5. 4. Регістри вказівника команд і ознак

На рис. 5.2 - 5.4 показані всі регістри базової архітектури мікропроцесора, які містять регістри загальної адреси і даних, вказівник команд, регістр ознак. Ці регістри у разі включення нової задачі завантажуються новим вмістом.

Базова архітектура містить шість прямо доступних сегментів, кожний розміром до 4 Гбайт. Сегменти відображені значеннями селектора, поміщеними в регістр сегмента.

Селектори автоматично завантажуються під час операції включення задач.

Інший тип регістрів – це регістри керування, системної адреси, відлагоджування і тестів. Їх використовують для налагодження операційної системи.

5.1.2. Опис регістрів

Регістри загального призначення (РЗП). Мікропроцесор має вісім 32-розрядних РЗП. Вони підтримують 16-, 32-розрядні адресні операнди, 1, 8, 16, 32, 64 операнди даних і бітові поля від 1 до 32 бітів. Їх називають: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP.

Молодші 16 розрядів кожного регістра можна використовувати окремо, це AX, BX, CX, DX, SI, DI, BP, SP. Також можна індивідуально застосовувати молодший (0-7) і старший (8-15) байти РЗП AX, BX, CX, DX: відповідно, їх називають AL, BL, CL, DL та AH, BH, CH, DH (рис. 5. 5).

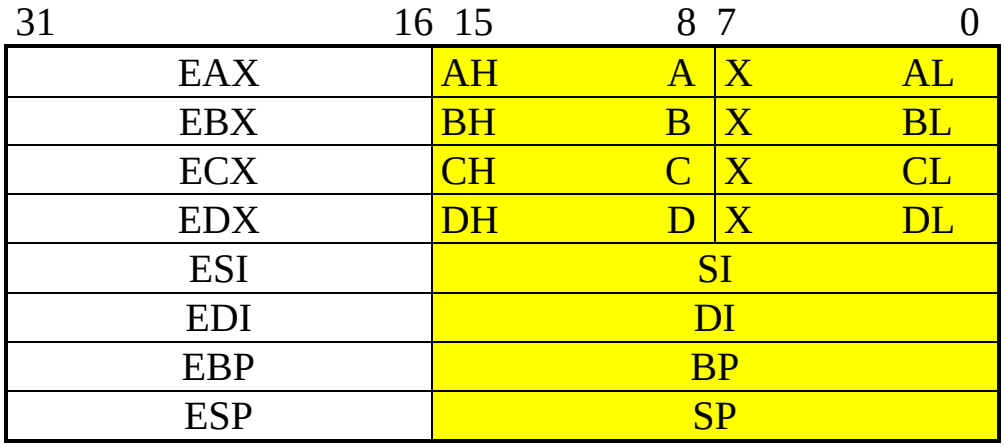


Рис. 5. 5. Регістри загального призначення

Індивідуальна доступність байтів РЗП забезпечує додаткову гнучкість для операцій з даними, однак це не використовують під час адресування.

Вказівник команд (IP). Регістр вказівника команд (рис. 5. 4) є 32-розрядним, його називають EIP. Він містить зміщення наступної команди, яку потрібно виконати. Зміщення завжди визначене щодо сегмента коду. Його молодші 16 біт називають IP і використовують для 16-розрядного адресування.

Регістр ознак є 32-розрядним, його називають EFLAGS. Частина бітів EFLAGS, як показано на рис. 5. 6, керують певними операціями і відображають статус 32-розрядного процесора. Молодші 16 розрядів EFLAGS становлять регістр FLAGS, який можна використовувати під час виконання операцій з процесором 8086.

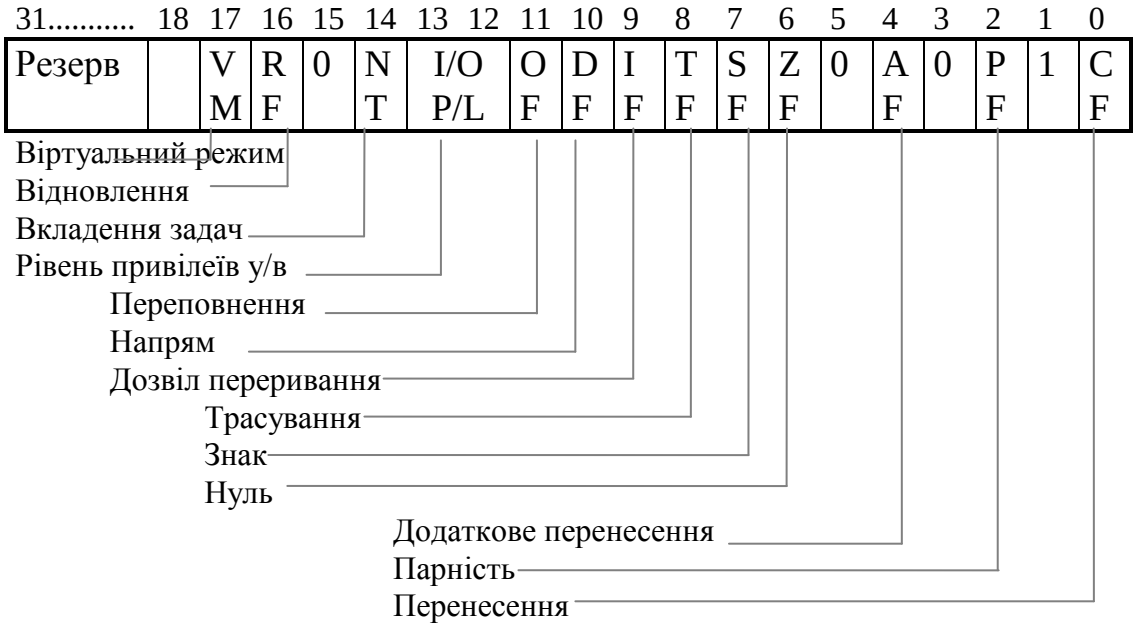


Рис. 5. 6. Регістр ознак

Найліпший спосіб вивчення регістра ознак — опис його побітно. Структура регістра ознак показана на рис. 5. 6, звідки легко зауважити, що не кожний біт є визначений. Невизначені біти є зарезервованими, тобто в цей момент не мають значення, однак можуть бути використані для спеціальних цілей у наступних версіях мікропроцесора.

Усі ознаки молодшого байта регістра задані арифметичними або логічними операціями процесора. За винятком ознаки переповнення, всі ознаки старшого байта молодшого слова відображають стан мікропроцесора і впливають на характер виконання програм.

Ознаку перенесення CF використовують для обчислень підвищеної точності, логічних операцій та інших команд. Вона відображає, було перенесення зі старшого розряду операнда чи ні, і відповідно, становить 1 або 0.

Ознака парності PF позначає парна чи непарна кількість одиниць молодших 8 біт результату. Вона дорівнює 1, якщо результат операції має парну кількість одиниць, інакше – 0.

Ознака додаткового перенесення AUX (AF), дає змогу мікропроцесору виконувати команди десяткової арифметики. Вона відображає чи правильно виконалася команда десяткової арифметики, чи ні, і, відповідно, дорівнює 0 або 1.

Ознака нуля ZF відображає, що результат останньої операції дорівнював нулю.

Ознака знака SF показує: додатним чи від'ємним був результат останньої арифметичної або логічної операції.

Ознака спеціального переривання TF допомагає відлагоджувати програми. Ця ознака не постає внаслідок роботи мікропроцесора, а її задає програма за допомогою спеціальної команди. Інша назва – ознака трасування, або покрокової роботи. Якщо вона задана, то після виконання кожної команди виникає переривання по трасуванню. Під час процедури переривання мікропроцесор скидає ознаку трасування. Це дає змогу опрацьовувати переривання по трасуванню без переривання після кожної команди. Після стандартного опрацювання ознака набуває початкового стану, тобто після виконання наступної команди програми знову виникає переривання по трасуванню. Цей процес буде тривати доти, доки програма користувача не скине ознаки TF або не завершиться виконання задачі.

Ознака переривань IF керує зовнішніми перериваннями. Доки ознака переривань дорівнює 0, жодні зовнішні переривання мікропроцесор не опрацьовуватиме (за винятком немаскованих). Коли вона дорівнює 1, то опрацьовуватимуться будь-які переривання, що виникають.

Ознаку напряму DF мікропроцесор використовує під час роботи з рядками, які працюють з великими блоками даних, для визначення напряму просування по блоку, тобто в бік збільшення або зменшення адреси (інкремент/декремент). Якщо ознака дорівнює 1, то команди опрацьовують рядки у бік зменшення адреси, якщо ж ознака дорівнює 0 – то в бік збільшення. Розглянуті біти EFLAGS повністю співпадають з бітами регістра FLAGS мікропроцесора 8086/8088.

Ознака переповнення OF інформує про те, відбулося переповнення під час виконання операцій чи ні, тобто про правильність результату. Вона дорівнює 0, якщо переповнення не було, в іншому випадку – 1.

Біт визначення рівня привілеїв уведення/виведення (I/O P/L) (розряди 12,13). Це двобітове поле належить до захищеного режиму. Рівень привілеїв уведення/виведення (I/O P/L) відображає максимальне значення поточного рівня привілеїв (ПРП). Для максимально допустимого значення ПРП, у разі виконання команд уведення/виведення без генерування спеціального переривання, він також вказує максимальне значення ПРП, яке дає змогу змінити біт IF, якщо нові значення завантажуються зі стеку в регістри FLAGS або EFLAGS. Команди POPF та IRET, якщо вони виконуються при ПРП=0, можуть змінювати поле I/O P/L. Операції вмикання задач завжди змінюють поле I/O P/L, коли нове значення ознаки завантажується з сегмента стану задачі.

Ознака вкладення задач **(**NT). Цю ознаку використовують у захищеному режимі. NT задають для того, щоб відобразити, що виконання конкретної задачі вкладено в іншу задачу. Якщо її задано, то сегмент стану поточної вкладеної задачі має достовірний обернений зв’язок з сегментом стану попередньої задачі. Цей біт задають або відмінюють командами, що передають керування іншим завданням. Значення NT в EFLAGS перевіряють командою IRET, яка у випадку NT=1 виконує перемикання задачі, а у випадку NT=0 – звичайне повернення з переривання.

Ознаку відновлення (RF) використовують у покроковому режимі або сумісно з точками переривань регістрів налагодження. Її перевіряють на межі команди, перед опрацюванням точки зупинки. Якщо RF=1, то будь-яка помилка налагодження буде проігнорована на наступній команді. RF автоматично скидається в разі успішного закінчення кожної команди (помилки не зафіксовані), крім команд IRET, POPF, JMP, CALL, INTER, які спричинюють включення задачі. Ці команди задають RF відповідно до збереженої копії EFLAGS у пам'яті.

Наприклад: наприкінці обслуговування програми переривань команда IRET може відновити копію EFLAGS, що має RF=1. Тоді МП відновить виконання програми в адресі точки переривання без генерування іншого переривання в тому ж місці.

Ознака віртуального режиму VM забезпечує віртуальний режим МП8086 у межах захищеного. Якщо VM=1, то 32-розрядний процесор перетворюється у високопродуктивний МП 8086. Біт VM може бути заданий тільки в захищеному режимі: командою IRET, якщо поточний привілейований рівень дорівнює 0, або шляхом перемикання задач на будь-якому рівні привілеїв. Біт VM не підпорядкований дії команди POPF, а команда PUSHF завжди встановлює цей розряд в 0, навіть, якщо працює у віртуальному 8086 режимі. Копія EFLAGS, збережена у стеку під час опрацювання переривання або під час вмикання задачі, міститиме одиницю в цьому біті, якщо переривання опрацьоване як віртуальне 8086 завдання.

Сегментні регістри. Шість 16-розрядних сегментних регістрів містять базові адреси сегментів, що визначають сегменти пам'яті поточного адресування (16-бітові вказівники у реальному режимі та дескриптори – у захищеному). Сегментні регістри показані на рис. 5. 7. У захищеному режимі кожний сегмент може мати розміри від одного байта до всього лінійного і фізичного простору машини (до 4Гбайт). У режимі реального адресування максимальний розмір сегмента обмежений 64 Кбайтами.

Рис. 5.7. Регістри дескриптора сегмента

Шість сегментів, які адресують пам'ять у будь-який конкретний момент, визначені вмістом регістрів CS, SS, DS, ES, FS і GS. Значення в CS визначає поточний сегмент коду; вміст SS визначає поточний сегмент стеку, а значення в DS, ES, FS, GS – сегменти даних.

У захищеному режимі регістри сегментів містять дескриптори сегментів, які завантажуються автоматично (про дескриптори див. 5.7.1). Регістри дескриптора сегмента невидимі для програміста, однак їхній вміст потрібно знати. Регістр дескриптора співвіднесений з кожним видимим регістром селектора, як показано на рис. 5.7. Кожний з них містить 32-бітову базову адресу сегмента, його межу та інші необхідні ознаки. Коли адреса сегмента завантажується в сегментний регістр, то асоціативний (співвіднесений) регістр дескриптора автоматично модифікується відповідно до нової інформації. В режимі реального адресування модифікується тільки базова адреса шляхом зсування значення сегментного регістра на чотири розряди ліворуч, оскільки максимальна межа й ознаки сегменту фіксовані. В захищеному режимі базова адреса, межа, всі ознаки модифіковані вмістом регістра дескриптора сегмента, індексованого селектором. Кожного разу, коли відбувається посилання на комірку пам'яті, регістр дескриптора сегмента автоматично "втягується" з посиланням на комірку пам’яті; 32-бітова базова адреса сегмента стає компонентою обчислення лінійної

адреси, 32-бітове значення межі буде використане для операцій контролю межі, а ознаки – перевірені щодо відповідності типу посилання на комірку пам’яті. До ознак (атрибутів) зокрема, належать: рівень привілеїв, наявність доступу до пам'яті, грануляції, читання чи запис, розмір стека та ін.

Регістри керування. МП має три регістри керування довжиною по 32 біти (CR0, CR2 і CR3), які зберігають ознаки процесора і є загальними для всіх задач. Ці регістри, поряд з регістрами системної адреси підтримують стан машини і діють на всі задачі в системі.

31						0
PG	Резерв (біти невизначені)	ET	TS	EM	MP	PE

Рис. 5.8. Регістр контролю CR0

CR0: Регістр керування машиною (рис. 5.8). Він містить шість розрядів для керування і визначення стану процесора; 16 молодших розрядів CR0 також відомі як *Слово Стану Машини* (MSW), яке є присутнє у МП 80286. Для завантаження і зберігання в пам’яті CR0 використовують команди LMSW і SMSW. З метою сумісності з операційними системами МП 80286, команди працюють аналогічно до команд 80286, тобто нові розряди CR0 ігноруються. Нові операційні системи мікропроцесора повинні використати команди для завантаження регістра CR0 повністю: MOV CR0.

Тепер опишемо всі розряди регістра CR0.

PG (розбиття на сторінки, біт 31). PG біт задають для того, щоб розблокувати пристрій заміщення сторінок. PG=0, якщо його треба блокувати.

ET (тип розширення процесора, біт 4). ET відображає тип розширення процесора (80287 або 80387). За бажанням ET біт може бути скинутий шляхом завантаження CR0 при виконанні програми. Коли ET=1, то застосовується 32-бітовий протокол, інакше – 16-бітовий.

TS (перемикання задачі, біт 3). МП задає TS кожного разу, коли виконується операція перемикання задачі та перевіряє його при виконанні команд співпроцесора. Якщо TS=1, то операційний код співпроцесора приводить до операційного переривання “Співпроцесор не готовий” за умови, що біт MP також заданий. Опрацьовувач переривання звичайно зберігає контекст 80287/80387, що належить поточній задачі, і очищає біт TS перш ніж повернутися до помилкового коду операції співпроцесора.

EM (емуляція). Біт EM задають для того, щоб змусити всі коди арифметичних операцій співпроцесора виробляти переривання "Співпроцесор не готовий" (виняток 7). Комбінацію EM=0, MP=1 застосовують у випадку наявності співпроцесора, а EM=1, MP=0 – для його програмної емуляції.

MP (монітор співпроцесора, біт 1). MP біт використовується разом з TS бітом для визначення того, чи виробляє код операції WAIT помилку "Співпроцесор не готовий" (виняток 7), коли TS=1. Якщо MP=1 і TS=1, то код операції WAIT виробляє виняток 7. В інших випадках цього не відбувається. Зверніть увагу, що TS задано автоматично кожного разу, коли виконується операція перемикання задачі.

PE (дозвіл захисту, біт 0). PE біт задають для створення можливості роботи в захищеному режимі (на рівні сегментів). Коли PE=0, то процесор працює в реальному режимі. PE може бути встановлений шляхом завантаження CR0 або MSW, а скинутий тільки в разі завантаження CR0.

CR1: резервний. CR 1 зарезервований для використання в майбутніх моделях процесорів INTEL.

CR2: *Лінійна адреса переривання внаслідок відсутності сторінки* (рис. 5. 9). CR2 містить 32-бітову адресу, яка визначає наявність помилки на останній сторінці. Код помилки опрацьованої сторінки, поміщений у стек за запитом, може надати додаткову інформацію про статус помилки на сторінці.

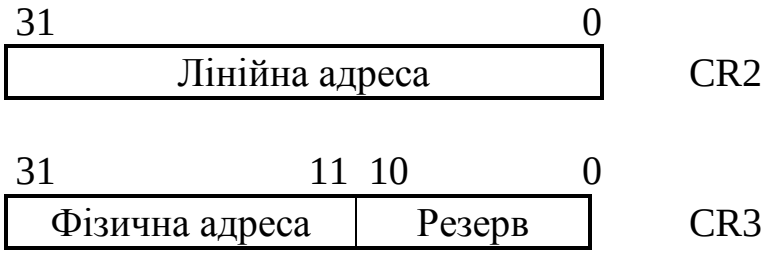


Рис. 5.9. Регістри керування

CR3: *Базова адреса таблиці сторінок*. CR3 містить фізичну базову адресу таблиці сторінок. Таблиця сторінок завжди посторінково вирівняна (тобто розташована через 4 Кбайти). Тому молодші 12 біт CR3 ігноровані і завжди поміщені в пам’ять як невизначені. У процесорі intel486 два розряди у цьому полі (PWT та PCD) задіяні і відповідають за режим запису і кешування сторінок.

Регістри системної адреси. У захищеному режимі є чотири спеціальні регістри для звертання до таблиць або сегмента (рис. 5.10):

GDTR – до таблиці глобального дескриптора GDT;
IDTR – до таблиці дескриптора переривань IDT;
LDTR – до таблиці локального дескриптора LDT;
TR – до сегмента стану задачі TSS.

Рис. 5. 10. Регістри системної адреси і сегмента

Регістри GDTR і IDTR містять 32-бітову лінійну базову адресу, 16-бітову межу таблиці глобального дескриптора і таблиці дескриптора переривання, відповідно. Сегменти GDT та IDT, оскільки вони глобальні для всіх задач у системі, визначені 32- бітовими лінійними адресами і 16-и бітовими значеннями меж.

Регістри LGTR і TR містять 16-бітові адреси сегментів для LDT і TSS. Сегменти LDT і TSS, оскільки вони є сегментами специфічних задач, визначені адресою, що поміщена в регістри системного сегмента.

Зверніть увагу, що регістр дескриптора сегмента (не видимий для програміста) співвідноситься з кожним регістром системного сегмента.

Регістри налагодження: Шість доступних для програміста регістрів налагодження забезпечують підтримку процесу налагодження програм. Регістри налагодження DR0-DR3 класифікують чотири лінійні зупинки, тобто містять лінійні адреси заданих контрольних

точок. Регістр стану налагодження DR6 використовують для задання точок зупинки, а регістр статусу налагодження DR7 виконує функції керування, а також відображає поточний стан точок зупинки. Використання регістрів налагодження детально описано у [2].

Тестові регістри. МП 80386 використовує тільки два регістри для керування і тестування сторінкової переадресації. TR6 –регістр тестування команд, а TR7 – регістр даних, який зберігає результат тестування буфера асоціативної трансляції. Їх використання розглянуто в розділі 5.6.

5.1.3. Доступність та сумісність регістрів

У доступах до регістрів у реальному і захищеному режимах є певні відмінності (табл. 5. 1).

Таблиця 5.1. Доступність регістрів 32-розрядних процесорів

Регістри	Реальний режим		Захищений режим		Режим віртуал.8086	
	Заванта жения	Збере-жения	Заванта жения	Збере-жения	Заванта жения	Збере-жения
Загального призначення	+	+	+	+	+	+
Сегментів	+	+	+	+	+	+
Ознак	+	+	+	+	IO PL	IO PL
Керівні	+	+	PL=0	PL=0	-	+
GDTR, IDTR	+	+	PL=0	+	-	+
LDTR, TR	-	-	+	-	-	-
Налагодження	+	+	PL=0	PL=0	-	-
Тестування	+	+	PL=0	PL=0	-	-

Варто пам'ятати про те, що деякі біти регістрів 32-розрядних процесорів невизначені. Це необхідно для сумісності програмного забезпечення, написаного для молодших моделей процесорів, з програмним забезпеченням наступних процесорів. У разі викликання невизначених бітів треба дотримуватися таких рекомендацій.

1. У разі тестування значень бітів регістрів уникати залежності від стану невизначених бітів. Ліпше їх замаскувати.

2. Уникати залежності від стану будь-якого із невизначених бітів у випадку розміщення їх у пам’яті або в іншому регістрі.

3. Уникати залежності від можливості розміщувати інформацію, записану у будь-якому з невизначених розрядів.

4. Під час завантаження регістрів завжди завантажувати їх як нулі.

5. Регістри, поміщені раніше, можуть бути перезавантажені без маскування.

5.2. Організація пам'яті 32-розрядних процесорів

Пам’ять 32-розрядних процесорів розділена на величини: 8-бітові (байти), 16-бітові (слова) і 32-бітові (подвійні слова). Слова зберігаються в двох послідовних байтах пам'яті так, що байт молодшого порядку розміщений у молодших адресах, а старший – у старших. Подвійні слова зберігаються в чотирьох послідовних байтах пам’яті з байтом молодшого порядку в наймолодшій адресі, а байт старшого порядку – у найстаршій. Адресою слова або подвійного слова є адреса байта молодшого порядку.

Крім цих головних типів даних, 32-розрядний процесор підтримує два великі блоки пам'яті: сегменти і сторінки. Пам'ять може бути розбита на один або декілька сегментів різної довжини, які можуть бути перенесені на диск або розподілені між програмами, або ж організована в одну чи більше 4 Кбайтових сторінок. Нарешті, сегментація і розбиття пам’яті

на сторінки можуть бути об’єднані, що дає змогу використати переваги обидвох систем. Сегментація корисна в разі організації пам’яті в логічні модулі, і є інструментом для програміста прикладних програм, тоді як сторінки використовують на системному рівні для керування фізичною пам'яттю.

5.2.1. Адресування простору

Тридцятидворозрядний процесор має три різні адресні простори: логічний, лінійний і фізичний. Логічна адреса (її також називають віртуальною адресою) складається із селектора сегмента і зміщення. *Селектор* – це вміст сегментного регістра. Зміщення формується шляхом сумування всіх адресних компонентів (Бази, Індекса, Зміщення) у виконавчу адресу (тобто виконавча адреса визначена у будь-якому режимі адресування).

Оскільки кожне завдання 32-розрядного процесора має максимально 16К (2^{14} - 1) селекторів, а зміщення можуть бути 4 Гбайти (2^{32} біт), то це дає повністю 2^{46} біт або 64 Тбайти логічного адресного простору на задачу. Цей віртуальний адресний простір доступний для програміста. Блок сегментації перетворює логічний адресний простір у 32-розрядний лінійний. Якщо блоку розбиття на сторінки нема, то 32-бітова лінійна адреса відповідає фізичній. Блок розбиття на сторінки перетворює лінійний адресний простір у фізичний. *Фізична адреса* – це те, що є на адресній шині. Головною відмінністю між реальним і захищеним режимом є те, як блок сегментації виконує перетворення логічної адреси в лінійну. У реальному режимі блок сегментації зсуває селектор ліворуч на 4 біти і додає сформовану в будь-якому режимі адресування *ефективну адресу* до зсуву для утворення лінійної адреси. В захищеному режимі кожний селектор має співвіднесену з ним лінійну базову адресу і зберігається в одній або двох таблицях операційної системи (тобто локальній таблиці дескриптора або глобальній таблиці

дескриптора). Лінійну базову адресу селектора додають до зміщення для утворення кінцевої лінійної адреси.

На рис. 5. 11 показаний взаємозв'язок між різними адресними просторами.

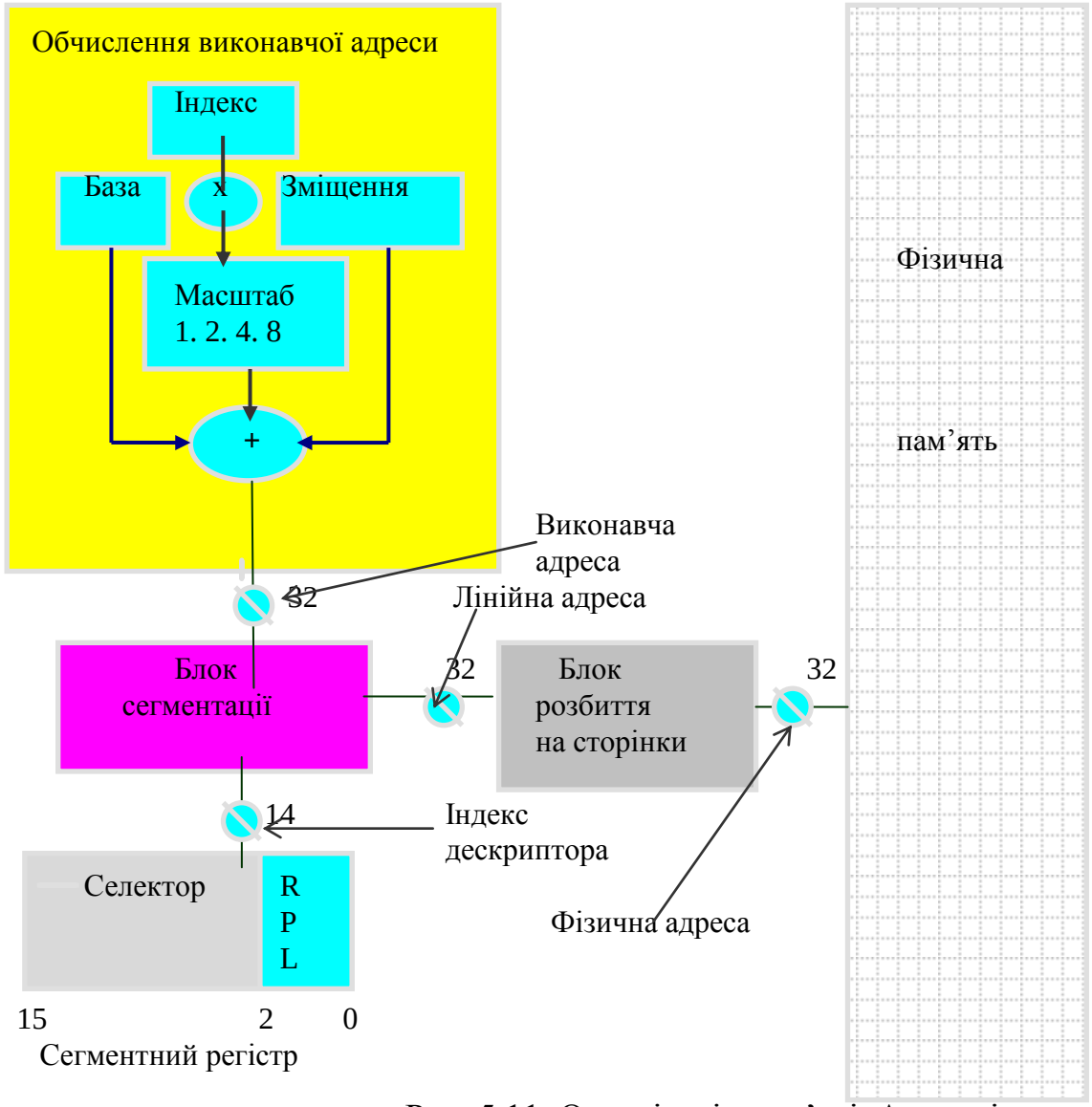


Рис. 5.11. Організація пам’яті. Адресні простори

5.2.2. Використання сегментного регістра

Як зазначено, головною структурою даних, які використовують для організації пам'яті, є сегмент. *Сегменти* – це різні за розміром блоки лінійних адрес, які мають певні співвіднесені з ними ознаки. Три головні типи сегментів: коду, даних і стеку, можуть мати розміри від 1 байта до 4 Гбайт.

Для забезпечення компактного кодування команд і поліпшення робочих характеристик процесора, немає потреби завжди обмежувати, який сегментний регістр буде використано для команд. Сегментні регістри вибираються автоматично за замовчуванням відповідно до правил (табл. 5. 2).

Таблиця 5.2 Правила вибору сегментного регістра

Тип посилання на комірку пам'яті	Використовуваний сегментний регістр (за замовчуванням)	Можливі префікси заміни сегмента (альтернативний)
Вибір коду	CS	нема
Приймач команд PUSH, PUSHA	SS	нема
Джерело команд POP, POPA	SS	нема
Будь-які посилання на дані з адресуванням, що використовує базові регістри:		
[EAX], [EBX], [ECX], [EDX], [ESI], [EDI]	DS	CS, SS, ES, FS, GS
[EBP], [ESP]	SS	CS, DS, ES, FS, GS

Звичайно, посилання на дані використовують селектор, який міститься в регістрі DS. Посилання на дані в стеку застосовують регістр SS і регістр SP як зміщення, а вибирання команд – регістр CS. Вміст вказівника команд IP забезпечує формування зміщення. Спеціальні префікси відміни сегментного регістра дають змогу явно використати заданий сегментний регістр і відмінити неявні правила, перечислені в табл. 5. 2. Префікси відміни дають змогу також застосувати сегментні регістри FS и GS.

Нема жодних обмежень, щодо перекриття будь-яких сегментів, тобто сегменти в пам’яті перекриваються довільно. Отже, всі шість сегментів можуть мати базову адресу, що дорівнює нулю, і створити систему з адресним простором у 4 Гбайти. Це формує систему, у якій віртуальний адресний простір є таким самим, як і лінійний. Детальніше сегментація описана в [2].

5.2.3. Простір уведення/виведення

Тридцятидворозрядні процесори мають два різні фізичні адресні простори: пам'яті і уведення/виведення. Звичайно, зовнішні пристрої поміщаются у простір уведення/виведення, хоча 80386 чи 80486 також підтримують зовнішні пристрої розподілу пам'яті. Простір уведення/виведення складається із 64 Кбайт і може бути поділений на 64 К 8-бітових портів, 32 К 16-бітових портів, 16 К 32-бітових портів або іншу комбінацію портів. 64 К-адресний простір уведення/ виведення стосується фізичної пам’яті, оскільки команди уведення/виведення не проходять через пристрої, які виконують сегментацію або розбиття на сторінки. Пам’ять уведення/виведення працює як додаткова адресна лінія, даючи змогу в такий спосіб легко визначити, до якого адресного простору в конкретний час має доступ процесор. Доступ до портів уведення/виведення можна отримати через команди IN (операнд) або OUT (операнд), де операнд містить адресу порту у вигляді безпосереднього операнда (якщо не

перевищує 256) або розташований у регістрах DL, DX чи EDX. Усі 8 і 16 біт адреси порту мають нульове розширення (тобто не розширюються) на верхніх адресних лініях. Команди уведення/виведення викликають цикл очікування, тому що працюють значно повільніше від мікропроцесора.

Адреси портів уведення/виведення від 00F8h до 00FFh зарезервовані для використання цифровими співпроцесорами.

5.3. Режими адресування

Система команд 32-розрядних процесорів забезпечує 11 режимів адресування переходів і даних. Режими адресування оптимізовані, що дає змогу ефективно працювати з такими мовами високого рівня, як Сі, ФОРТРАН та іншими, крім того, вони охоплюють велику кількість посилань даних, потрібних для мов високого рівня.

5.3.1. Регістровий і безпосередній режими

Команди, які використовують регістрові і безпосередні операнди, забезпечують два режими адресування:

Регістровий режим адресування: операнд розташований в одному з 8-, 16- або 32-розрядному регістрі загального призначення;

Режим безпосереднього операнда: операнд уведений у команду як частина операційного коду.

5.3.2. Інші режими адресування пам'яті

Решта дев'ять режимів забезпечують механізм для визначення виконавчої адреси операнда. Лінійна адреса складається з двох компонентів: сегментної базової адреси і виконавчої адреси. Виконавчу адресу обчислюють шляхом додавання будь-якої комбінації таких чотирьох адресних елементів:

Зміщення: 8 або 32 біти безпосереднього значення, яке визначене командою (16 бітів зміщення можуть бути використані перед командою з адресним префіксом);

База: вміст будь-якого з регістрів загального призначення. Базові регістри звичайно використовує компілятор для задання початку локальної змінної області;

Індекс: вміст будь-якого із регістрів загального призначення, крім ESP. Індексні регістри використовують для доступу до елементів масивів або ланцюжка символів;

Масштаб: значення індексного регістра може бути помножено на коефіцієнт масштабу 1, 2, 4, або 8. Режим масштабного індексу особливо корисний для доступу до масивів структур.

Комбінації цих чотирьох компонентів становлять дев'ять додаткових режимів адресування. У разі використання будь-якої із цих адресних комбінацій робочі характеристики не погіршуються, оскільки обчислення виконавчої адреси передається по конвеєру з виконанням інших команд. Єдиним винятком є одночасне використання компонентів Бази, Індексу і Зміщення, що потребує додаткового машинного такту (циклу очікування).

Виконавчу адресу (ВА) операнда обчислюють за такою формулою:

ВА = Базовий регістр
+ Індексний регістр помноженний на Масштабний коефіцієнт
+ Зміщення.

Прямий режим: Адреса операнда міститься як частина команди у вигляді 8, 16 або 32 бітів зміщення.

Приклад: INC WORD PTR[500], інкремент (вихідний операнд збільшується на 1).

Регістровий непрямий режим: Базовий або Індексний регістр містить адресу операнда.

Приклад: MOV [ECX], EDX, пересилання (вміст другого операнда пересилається у перший).

Базовий відносний: Вміст Базового регістра додається до зміщення для формування ВА.

Приклад: MOV ECX[EAX+4], EDX.

Індексний режим: вміст Індексного регістра додається до зміщення

Приклад: MOV EAX,TABLE[ESI].

Масштабний індексний режим: вміст Індексного регістра множиться на Масштабний коефіцієнт, який додається до зміщення.

Приклад: IMUL EBX,TABLE[ESI*4],7 цілочислове множення.

Базовий індексний режим: вміст Базового регістра сумується з вмістом Індексного регістра.

Приклад: MOV EAX, [ESI][EBX].

Базовий індексний масштабний режим: вміст Індексного регістра множиться на коефіцієнт масштабу і результат додається до вмісту Базового регістра.

Приклад: MOV ECX, [ESI*8][EAX].

Базовий індексний режим зі *Зміщенням*: вміст Базового регістра складається з вмістом Індексного регістра та зі зміщенням операнда для формування ВА.

Приклад: ADD EDX, [ESI][EBP+00FFFFF0h] , тобто
додавання (1) + (2) -> (1).

Базовий індексний масштабний режим зі *Зміщенням*: вміст Індексного регістра множиться на коефіцієнт масштабу і результат додається до вмісту Базового регістра, який підсумований зі зміщенням.

Приклад: MOV EAX, LOCAL TABLE[EDI*4][EBP+87].

Відмінності між 16- і 32-розрядними адресами. З метою забезпечення сумісності з програмами 8086 і 80286, 32-розрядний процесор може виконувати 16-бітові команди в реальному і захищеному режимах. Мікропроцесор визначає розмір виконуваної команди, аналізуючи біт D у дескрипторі сегмента. Якщо D=0, то довжина всіх операндів і виконавчих адрес становить 16 біт. Якщо ж D=1, то довжина за замовчуванням для операндів і адрес буде 32 біти. В реальному режимі розмір операндів і адрес за замовчуванням – 16 біт.

Незалежно від визначеної за замовчуванням довжини операндів 32-розрядний процесор здатний виконувати як 16- так і 32-бітові команди. Цього досягають завдяки префіксу відміни. Два префікси: *префікс розміру операнда* і *префікс довжини адреси,* відмінюють значення біта D на підставі заданого в самій команді значення. Ці префікси автоматично додаються асемблерами INTEL.

Приклад: Процесор працює в реальному режимі, і програмісту необхідно отримати доступ до регістра EAX. Команда асемблера для цього може бути такою: MOV EAX, 32bitMEMORYOP (32- розрядний операнд у пам'яті). ASM 386 автоматично визначає необхідність *префікса розміру операнда* і виробляє його.

Приклад: Біт D=0, і програміст хоче використати масштабний індексний режим адресування, щоб отримати доступ до масиву. Префікс довжини адреси дає змогу використати MOV DX, TABLE[ESI*2]. Асемблер використовує *префікс довжини адреси* оскільки при D=0, за замовчуванням використана 16-бітова адреса.

Приклад: Біт D=1, і программіст хоче розмістити в пам'ять 16- розрядну величину. Префікс розміру операнда буде використано для визначення тільки 16-бітового значення: MOV 16 біт пам'яті, DX.

5.3.3. Адресування пам'яті в реальному режимі

Після вмикання живлення або скидання 32-розрядний процесор переходить у *реальний режим* роботи (іноді кажуть, що процесор стає в "нуль"). Цей режим має таку саму базу, як 8086, однак надає доступ до 32-розрядних регістрів. У реальному режимі максимально можна заадресувати 1 мегабайт пам’яті. Оскільки розбиття пам'яті на сторінки в реальному режимі неможливо виконати, то лінійні адреси є такими ж, як і фізичні. Фізичні адреси формуються в реальному режимі шляхом додавання вмісту відповідного сегментного регістра, який зсувається ліворуч на 4 біти, до виконавчої адреси. Таке додавання приводить до створення адресного простору, який дорівнює 20-бітовій фізичній адресі або 1 мегабайтному адресному простору. Оскільки сегментні регістри зсуваються ліворуч на 4 біти, то це означає що сегменти реального режиму завжди починаються на межах параграфа.

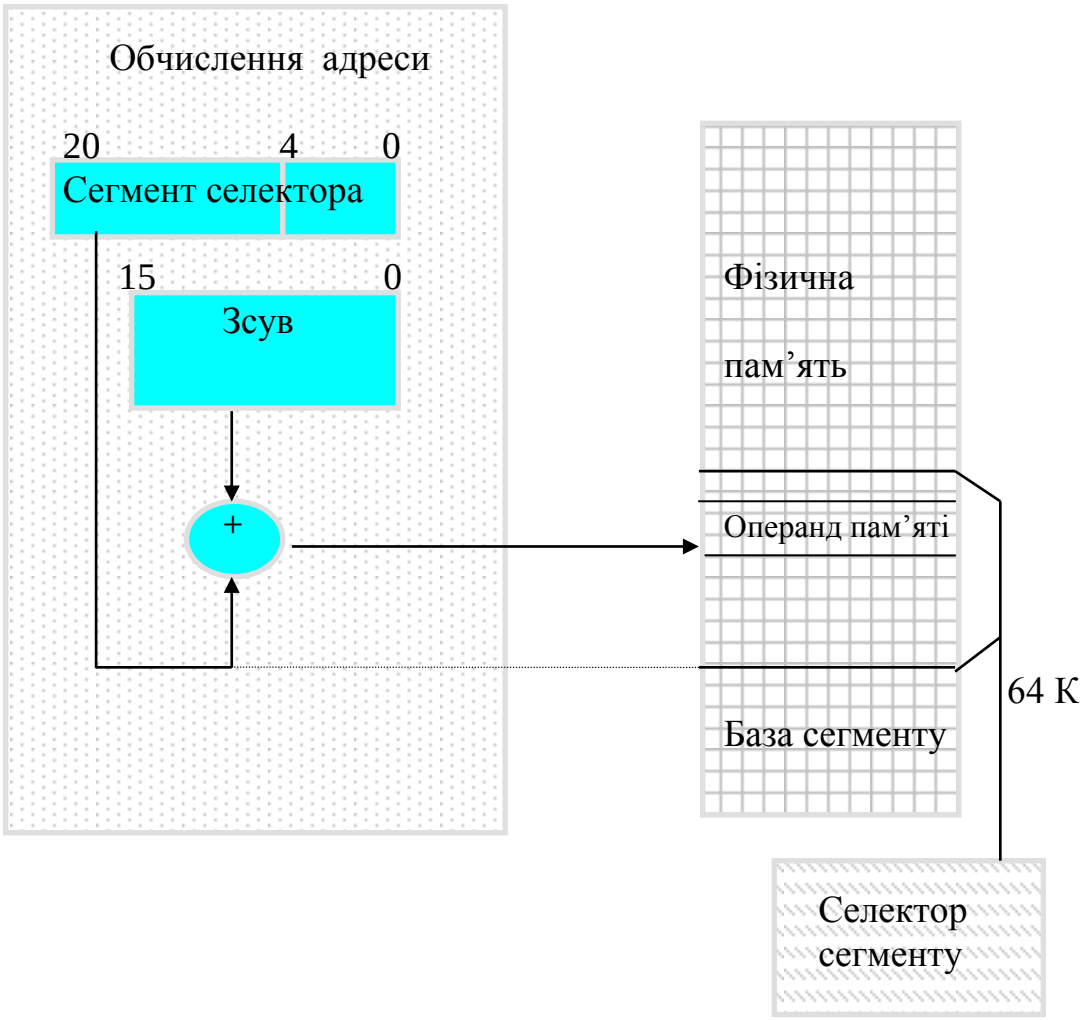


Рис. 5. 12. Адресування в реальному режимі

Усі сегменти в реальному режимі мають довжину 64 Кбайти і можуть бути прочитані, написані або виконані. МП 80836 (80486) може генерувати переривання 13, якщо адреса операнда даних або виклик команди відбуваються після кінця сегмента. (Тобто адреса операнда має зміщення більше ніж FFFFh, або, наприклад, слово з молодшим байтом з адресою FFFFh та старшим байтом з адресою 0000h).

Сегменти в реальному режимі можуть перекриватися, тобто якщо певний сегмент не використовує всі 64 Кбайти, то інший сегмент може бути накладений на невикористану частину попереднього сегмента. Це дає змогу програмісту зменшити кількість фізичної пам’яті, необхідної для програми.

5.3.4. Запасні комірки пам'яті

Є дві фіксовані області в пам’яті, які резервовані в режимі реального адресування:

1) область ініціалізації системи;

2) область таблиці переривання.

Комірки від 00000h до 003FFh резервовані для векторів переривань. Кожне з 256 можливих переривань має зарезервовану 4-байтову адресу переходу.

Комірки від FFFFFFF0h до FFFFFFFFh резервовані для ініціалізації системи.

5.4. Типи даних

Тридцятидворозрядні мікропроцесори підтримують усі типи даних, які використовують у мовах високого рівня.

Біт: величина в один біт.

Бітове поле: група, що охоплює до 32 суміжних бітів, що займають максимально до 4 байтів.

Рядок бітів: множина суміжних бітів, у 80386 рядок бітів може займати простір до 4 Гбайт.

Байт: 8-бітова величина зі знаком.
Байт без знака: 8-бітова величина без знака.
Ціле (*слово*): 16-бітова величина зі знаком.
Ціле (слово) без знака: 16-бітова величина без знака.
Довге ціле (*подвійне слово*): 32-бітова величина зі знаком.
Довге ціле (*подвійне слово*) *без знака*: 32-бітова величина без знака.
Почетверенне слово зі знаком: 64-бітова величина зі знаком.
Почетверенне слово без знака: 64-бітова величина без знака. (Зауважимо, що в усіх числах зі знаком старший біт є знаковим. Як і у випадку 16-розрядних процесорів для позначення додатнього числа біт дорівнює 0, а від'ємного – 1).
Короткий вказівник: 16 або 32 біти зміщення щодо базової адреси сегмента, що міститься в селекторі сегмента (адресація тільки в межах одного сегмента).
Довгий вказівник: повний вказівник, який складається із 16 бітів селектора сегмента і 16 або 32 бітів зміщення.
Рядок: послідовність суміжних байтів, слів або подвійних слів. Рядок може містити від 1 байта до 4 Гбайт.
BCD (двійково-десяткове число) розпаковане: зображення байтом дясяткової цифри 0-9.
Упаковане BCD: зображення одним байтом двох десяткових цифр від 0 до 9 з розташуванням кожного числа у напівбайті.

При наявності арифметичного співпроцесора підтримуються типи чисел з плаваючою комою:

Плаваюча кома: 32-, 64- або 80-розрядне дійсне число зі знаком.

5.5. Переривання і винятки

Переривання і винятки (виняткові ситуації) змінюють нормальне виконання задачі, щоб відреагувати на зовнішні умови, повідомити про помилки або виняткові обставини. Відмінності між перериваннями і винятками полягають в тому, що переривання опрацьовують асинхронні зовнішні умови, тоді як винятки – помилки команд. Хоч переривання *n* програма може виконати через команду INT *n*, однак процесор розглядає програмне переривання як виняток.

5.5.1. Типи переривань і винятків

Як зазначено, переривання, спричинені машинним обладнанням, виникають унаслідок зовнішніх подій і бувають двох типів: масковані і немасковані. Вони опрацьовуються після виконання поточної команди. Після того, як програма опрацювання переривань закінчує обслуговувати переривання, виконання програми продовжується з команди, яка є наступною. Відмінності між маскованими і немаскованими перериваннями описано у темі 3.

Винятки класифікують як помилки (або відмови), пастки або (передчасні) припинення залежно від способу їхнього повідомлення і від того, чи підтримується повторний запуск команди, яка викликає виняток.

Помилки (fault) – це винятки, які виявляються і виконуються до виконання команди з помилкою. Наприклад, *помилка* відбудеться в системі віртуальної пам'яті, коли процесор зішлеться на сторінку або сегмент, яких нема. Операційна система вибере сторінку або сегмент з диска, після чого процесор перезапустить команду.

Пастки (trap) – винятки, про які повідомлення надходять негайно після виконання тої програми, яка викликає виняток. До класу пасток належать і програмні переривання.
Припинення, або *аварійні завершення (abort)* – винятки, які не дозволяють визначати точно місце знаходження команди, яка викликає виняток. Їх використовують для повідомлення про грубі помилки, такі як апаратна помилка або неправильні значення в системних таблицях.

Отже, коли програма обслуговування переривання закінчує роботу, то виконання продовжується з команди, яка є наступною після команди, що спричинила переривання. З іншого боку, адреса повернення з програми помилки винятку завжди вкаже на команду, яка викликає виняток, і буде містити будь-який з префіксів провідної програми.

У табл. 5. 3 наведені всі можливі переривання для 80386, і відображено, куди відсилає адреса повернення. Процесор здатний опрацювати до 256 різних переривань/винятків. Для обслуговування переривань необхідно означити (скласти) таблицю з 256 векторами переривань. **Вектори переривань – це вказівники на відповідні програми обслуговування**; у реальному режимі вектори – це 4- байтові величини: сегмент коду плюс 16 бітів зміщення; у захищеному режимі вектори переривань є 8-байтовими величинами, які поміщені в таблицю дескрипторів переривань. Із 256 можливих переривань 32 зарезерворовані для використання INTEL, решта 224 вільні для застосування користувачем.

Багато винятків, що відображені в таблиці переривань і розглянуті у першій частині, не застосовують у реальному режимі. Зокрема переривання 10, 11, 12, 14 не можуть відбутися в реальному режимі. Інші переривання у реальному режимі мають деякі відмінності.

Таблиця 5.3. Вектори переривань

Функція	Номер пере-ривання	Команда, яка може викликати виняток	Адреса означає повернення на помилкову команду	Тип
Помилка ділення	0	DIV, IDIV	Так	Помилка
Виняток налагодження	1	Будь-яка	Так	Пастка
NMI переривання	2	INT2 або NMI	Ні	NMI
Однобайтове переривання	3	INT3	Ні	Пастка
Переповнення	4	INT0	Ні	Пастка
Контроль межі масиву	5	Межа	Так	Пастка
Неправильний код операції	6	Будь-яка заборонена команда	Так	Пастка
Відсутність пристрою	7	ESC, WAIT	Так	Пастка
Подвійна помилка	8	Будь-яка команда, що здатна виробити виняток	Так	Припи-нення
Перевантаж. сегмента співпроц.	9			Пастка
Неправильний TSS	10	JMP, CALL, IRET, INT	Так	Помилка
Відсутній сегмент	11	Команди з сегментними рег.	Так	Помилка
Помилка стека	12	Посилання на стек	Так	Помилка
Загальна помилка захисту	13	Будь-яке звертання до пам’яті	Так	Помилка
Перерив. внаслідок відсутності сторінки	14	Будь-який доступ до пам’яті для вибору коду	Так	Помилка
Помилка співпроцесора	16	ESC, WAIT	Так	Помилка
Контроль вирівнювання (486+)	17			Помилка
Машинний контроль (P5+)	18			Авар.прип.
Резерв	19-32			
Програмні переривання	0-255	INTn	Так	Пастка

5.5.2. Опрацювання переривань

Алгоритм опрацювання переривань процесора описаний у 4.2. Згадаємо, що адреса поточної команди і регістр ознак EFLAGS зберігаються в стеку, що дає змогу відновити перервану програму. Процедура опрацювання переривання визначена 8-бітовим вектором таблиці переривань, який містить початкову адресу програми опрацювання переривань. Після цього виконується викликана програма опрацювання. Старий стан процесора відновлюється командою IRET і за адресою повернення (тобто за адресою, збереженою у стеку) відновлюється виконання програми. Восьмибітовий вектор переривання подається на процесор кількома способами:

- винятки генерують і передають вектор переривання внутрішньо;
- команди INT містять або включають в себе вектор;
- переривання, масковані обладнанням, "поставляють" 8-бітовий вектор через шину підтвердження переривань від зовнішнього контролера.

5.5.3. Масковані переривання

Масковані переривання – найбільш загальний засіб, що його використовує процесор для відповіді на асинхронні зовнішні події в обладнанні. Апаратне переривання відбувається тоді, коли біт IF ознаки переривань розблокований (дорівнює 1), і рівень INTR на виході процесора є високим. Процесор реагує на масковані переривання тільки між виконанням команд. Рядкові команди мають "вікна переривань" між переміщеннями пам'яті, що робить можливим переривання під час опрацювання довгих рядків. Коли виникає переривання, процесор зчитує 8-бітовий вектор, що надходить від обладнання, який визначає джерело переривання. (Одне із 224 визначених користувачем переривань).

Коли відбувається обслуговування переривання, то біт IF у регістрі EFLAGS скинутий. Це дає змогу ефективно запобігти додатковим перериванням під час опрацювання обслуговування переривань. Однак IF може бути заданий опрацьовувачем переривань, щоб дозволити вкладення переривань. Коли команда IRET виконана, відновлюється початковий стан ознак, а отже, і біта IF.

5.5.4. Немасковані переривання

Немасковані переривання забезпечують обслуговування переривань дуже високого рівня. Як і для попередніх процесорів, одним із загальних прикладів немаскованих (NMI) переривань може слугувати переривання за збоєм живлення. Коли активізований контакт NMI, то відбувається переривання згідно до вектора 2. На відміну від звичайних переривань, для NMI не виконується послідовність підтверджень приймання переривань.

Під час процедури обслуговування NMI процесор не обслуговуватиме ні подальший запит NMI, ні запити INT доти, доки не буде виконана команда повернення із переривання (IRET) або поки процесор не буде скинутий. Якщо NMI відбудеться під час обслуговування NMI, то його (переривання) присутність буде збережена для опрацювання після опрацювання поточного NMI, тобто після першої ж команди IRET. Біт IF очищається на початку NMI для блокування подальших INTR команд.

5.5.5. Програмні переривання

Третім типом переривань/винятків є програмні переривання. Команда INT *n* змушує процесор виконувати програму обслугову-вання, яку визначає вектор *n* у таблиці переривань.

Особливим випадком двобайтового програмного переривання INT з номером *n* є однобайтове переривання INT3 або переривання зупинки. Шляхом уведення цієї однобайтової команди в програму користувач має змогу задати точки зупинки в своїй програмі для її налагодження.

Ще одним типом програмного переривання є переривання покрокового режиму. Воно корисне для відлагоджування програм.

5.5.6. Пріоритетність переривань

Оскільки переривання розпізнають тільки на межах команд (тобто тоді, коли одна команда закінчується, а інша починається), то можливо, що активними одночасно можуть бути кілька переривань. У випадку одночасних переривань вони опрацьовуватимуться згідно з таким пріоритетом:

Пріоритет опрацювання	Переривання/виняток
1 (вищий)	Помилка винятку
2	Команда пастки
3	Пастка налагодження для заданої команди
4	Помилка налагодження для наступної команди
5	MNI переривання
6	INTR переривання

Приклад: задана команда викликає системне переривання налагоджень і виняток "сегмент відсутній". Процесор насамперед відреагує на виняток "сегмент відсутній" (11), спробувавши активізувати опрацьовувач винятку 11. Програма опрацювання винятку 11 буде перервана,

унаслідок чого адреса опрацьовувача винятку 11 збережеться в стеку. Після цього буде викликаний опрацьовувач налагодження і керування знову повернеться опрацьовувачу винятку 11. Це дає змогу системотехніку відлагоджувати свої опрацьовувачі винятків.

5.5.7. Повторний запуск

Процесор повністю підтримує відновлення всіх команд після помилок. Операційна система не бере участі в процесі перезапуску, оскільки процесор повідомить про помилку сегмента або сторінки за такого стану машини, який забезпечує перезапуск помилкової команди після того, як опрацьовувач помилки виправив помилкову умову.

Повторний запуск команди гарантований за винятком двох умов:

1) якщо команди викликають уведення задачі в сегмент стану поточної задачі, що розташований у сторінці, якої нема.

2) якщо один із операндів розташований нижче від будь-якого з поточних вказівників стеку, тобто за адресою пам’яті меншою, ніж вершина стеку, або операнд у формі з плаваючою комою здійснює циклічний перехід у пам’яті.

5.5.8. Подвійні помилки

Подвійна помилка трапляється тоді, коли процесор пробує опрацювати виняток і отримує інший виняток під час програми опрацювання. Подвійна помилка викликає виняток 8. Більшість винятків у 32-розрядних процесорах не викликають умови подвійної помилки (типи 1, 2, 3, 4, 5, 6, 7, 9, 14 и 16). Це роблять тільки помилки внаслідок ділення (переривання 0) і сегментні винятки (10, 11, 12, 13). Отже, отримання винятку "сегмент відсутній" під час опрацювання винятку налагодження не призведе до подвійної помилки, тоді як помилка сегмента, що виникає під час опрацювання помилки внаслідок ділення на нуль, – призведе.

Переривання внаслідок відсутності сторінки (в оперативній пам’яті) не вважається подвійною помилкою. Наприклад, якщо команда викликає виняток "сегмент відсутній" (11) і переривання внаслідок відсутності сторінки (переривання 14), то обидва переривання будуть правильно опрацьовані. Опрацьовувач винятку "сегмент відсутній" буде активізований, унаслідок чого з диска завантажиться правильний сегмент. Після того команда буде повторно запущена, і викличе переривання внаслідок відсутності сторінки. Тоді опрацьовувач переривання подасть правильну сторінку і виконання програми продовжиться. Ще однією причиною подвійних помилок є рекурсивні помилки (наприклад, нема опрацьовувача переривання внаслідок відсутності сторінки). Це приводить до винятку 8.

5.5.9. Скидання та ініціалізація (RESET)

У разі ініціалізації або скидання процесора (високий рівень на виході RESET) регістри набувають значень, які відображені в примітках. Процесор після того розпочне виконання команд поблизу вершини фізичної пам’яті, у комірці FFFFFFF0h. Коли перша команда міжсегментного переходу або CALL виконана, то адресні лінії A20-31 у циклах вибирання команд мають одиничні значення, тобто "опускаються" вниз фізичної пам’яті, і процесор виконуватиме команди тільки в нижньому мегабайті пам’яті. Це дає змогу виконати команди BIOS у вершині фізичної пам’яті для ініціалізації системи та встановлення процесора у вихідний стан.

Наприклад, для 80386 поява високого рівня на вивід Reset як мінімум 78 тактів скидатиме процесор. У разі скидання процесор припиняє всі виконання і дії з локальними шинами. Між 350 і 450 тактовими періодами, коли скидання перестає діяти, 80386 починає виконувати команди у фізичній вершині пам’яті.

Примітки.

1. Регістр EFLAGS = 0002h. Старші 14 бітів EFLAGS невизначені. VM (біт 17) і RF (біт 16) дорівнюють нулю, як і всі інші визначені біти.

2. CR0 (Слово стану машини). Усі з визначених полів у CR0 дорівнюють 0 (PG біт 31, TS біт 3, EM біт 2, MP біт 1 и PE біт 0), крім ET (біт 4, тип розширення процесора). ET біт буде заданий час скидання згідно з типом співпроцесора в системі. Якщо співпроцесор типу 80387, то біт буде 1; якщо співпроцесор 80287 або його взагалі нема, то ET – 0. Усі інші біти невизначені.

3. Регістр сегмента коду (CS) буде мати свою базову адресу FFFF0000h і межу 0FFFFh. Усі невизначені біти зарезервовані для INTEL і не повинні використовуватися.

4. DS = ES = SS = FS = GS = 0000h.

5. EIP = 0000FFF0h.

6. Регістр DH після скидання містить ідентифікатор типу процесора Component Id (03 – 386, 04 – 486, 05 – Pentium, 06 – Pentium Pro або Pentium II). Регістр DL містить номер моделі (Revision Id).

5.5.10. Вимикання і зупинка

Команда HLT (зупинка) припиняє виконання програми і не дозволяє процесору використовувати локальну шину, доки він не буде запущений знову. Команди NMI, INTR (уведення) з перериваннями (IF=1) або скидання виведуть процесор зі стану зупинки. Якщо він зупинений, то пара регістрів CS:IP визначить іншу команду після HLT.

Вимикання МП відбудеться тоді, коли виявиться груба помилка, яка заважає подальшому опрацюванню. В реальному режимі вимикання може трапитися в двох випадках:

1) коли відбувається переривання, і вектор переривання більший, ніж таблиця опису переривань (тобто для заданого переривання нема опрацьовувача переривання).

2) коли виклик команди, уведення або проштовхування намагаються повернути сегмент стеку, якщо SP непарний (наприклад: завантаження значення зі стеку, коли SP=0001); унаслідок цього сегмент стеку більший ніж FFFFh.

5.6. Тестування. Самоконтроль

У тридцятидворозрядних процесорах розвинені засоби тестування і налагодження. Змогу виконувати самоконтроль мають процесори, починаючи з 80386. Самоконтроль перевіряє функціонування всіх пристроїв постійної пам'яті і більшість нерегулярної логіки частини машини. Під час самоконтролю може бути перевірена приблизно половина мікропроцесора. У 80386 самоконтроль ініційований тоді, коли відбувається перехід від високого сигналу до низького на виводі Reset, а на виводі BUSY низький сигнал. Самоконтроль займає приблизно 30 мілісекунд для процесора з тактовою частотою 16 МГц. Після закінчення самоконтролю процесор виконує скидання і починає звичайну роботу. Елемент уважається таким, що успішно пройшов контроль, якщо вміст регістрів EAX і EDX дорівнює 0. Якщо ж вміст EAX і EDX відрізняється від нуля, то це означає, що в процесі самоконтролю було виявлено дефект в елементі.

Регістр DH після скидання містить інформацію про тип процесора, а регістр DL – номер моделі (див. 5. 5. 9).

З деяких моделей 80486 у процесори почали вводити підтримку тестового інтерфейсу JTAG, який відповідає стандарту IEEE 1149.1 Boudary Scan Architecture. Тестуванню підлягає внутрішня логіка процесора за методом сканування меж. Цей інтерфейс є і в процесорах типу Pentium, де він удосконалений і надає доступ до регістрів процесора.

5.6.1. Тестування буфера асоціативної трансляції

Мікропроцесор 80386 має механізм для контролю буфера асоціативної трансляції (TLB) (див. 5. 8. 3). Ця можливість потрібна передусім тим, хто пише тестові програми для мікропроцесора. Тестування TLB є унікальною властивістю 80386 і в майбутніх процесорах можуть не використовувати. Тестування TLB потребує застосування тестера і програми мовою асемблера для надання дії моделі тесту. Сторінкова організація пам'яті у цьому разі повинна бути заблокована.

Два тестові регістри (див. рис. 5. 13) забезпечені засобами написання зразка в TLB і зчитуванням результату. TR6 є регістром тестових команд, а TR7 – контрольних даних.

31 12 11 0

Лінійна адреса	V	D	D	U	U	W	W	0	0	0	0	0
Фізична адреса	0	0	0	0	0	0	0	PL	REP		0	0

Рис. 5. 13. Тестові регістри TR6 та TR7.

Тестові регістри дають змогу провадити два види операцій з TLB: *напиши нове TLB уведення*; *виконай шукання TLB*.

Запис у регістр тестової команди через команду MOV TR0, (регістр) приводить до виконаня операції TLB. Якщо біт 0 дорівнює 1, то виконуватиметься шукання TLB.

5.6.2. Забезпечення налагодження

Тридцятидворозрядні процесори мають розвинені засоби налагодження. Більшість цих засобів призначені передусім для програмного налагодження. (INTEL забезпечує повний набір

інструментів налагодження, таких як ICE-386 (внутрішньосхемна емуляція) і PTM-386 (переглядовий монітор) для закінчення побудови налагоджувальних ознак).

Три головні типи засобів налагодження мікропроцесора такі:

- ✓ точки зупинки програмного забезпечення;
- ✓ покроковий режим роботи;
- ✓ регістри налагодження.

Однобайтове переривання INT3 відладники програмного забезпечення використовують для реалізації точок зупинки. Після того, як команду INT3 введено й опізнано, виконання продовжиться на опрацьовувачі переривання 3. Одноступеневого переривання досягають шляхом задання біта (TF) в регістрі EFLAGS. TF біт задають шляхом зміни копії регістра ознак у стеку і виконанням команди POPF або IRET. У разі задання біта TF одноступеневе переривання відбудеться після виконання наступної команди. Команда переривання помістить у стек поточний регістр ознак (з заданим бітом TF), а після того очистить цей біт (забезпечуючи можливість нормального виконання програми опрацювання одноступеневого переривання). Це дає змогу створити опрацьовувач переривання, який виконуватиме ступеневі рухи всередині команди. Одноступеневе переривання використовує вектор переривання 1, який надходить до процесора внутрішньо. Після завершення програми опрацювання одноступеневого переривання команда IRET добуде регістр ознак зі стеку і передасть керування наступній команді, яка також буде розбита на ступеневі кроки.

На відміну від традиційних точок зупинки, які підтримують тільки команди виклику переривань, регістри налагодження дозволяють задавати точки зупинки для доступу до даних. Отже, якщо змінна величина випадково перезаписана (або виконано накладення), то точка зупинки може бути задана, щоб припинити виконання незалежно від того, чи буде змінено значення цієї змінної. На рис. 5. 14 показані регістри налагодження. DR0-3 містять лінійні адреси точок переривання.

Лінійна адреса може не відповідати адресі фізичній, якщо розблокована посторінкова організація пам'яті. DR6 містить стан регістрів точок зупинки. Біти в регістрах мають такі значення:

біт BT: задано, якщо відбувається введення задачі в задачу, де TSS має заданий біт пастки налагодження;

біт BS: дає змогу опрацьовувачам налагодження розрізняти одноступеневі пастки від інших умов налагодження;

біт BD: заданий обладнанням, якщо наступна команда отримує доступ до регістра налагодження;

біти B0-B3: ці біти задані, якщо відбулося переривання обмеженого використання. B0 задано, якщо відбулося переривання 0 (точка зупинки) і т. д.:

Лінійна адреса точки зупинки 0																									DR0
Лінійна адреса точки зупинки 1																									DR1
Лінійна адреса точки зупинки 2																									DR2
Лінійна адреса точки зупинки 3																									DR3
Резерв. Значення бітів невизначені																									DR4
Резерв. Значення бітів невизначені																									DR5
0	0	0	0..0	0	0	0	0	0	0	BT	BS	BD	0	0	0	0	0	0	0	0	0	B3	B2	B1	B0	DR6
LEN3	R3	W3		LEN1	R1	W1	LEN0	R0	W0	0	0	GD	0	0	0	GE	LE	G3	L3	G2	L2	G1	L1	G0	L0	DR7
31	30	29	....	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Рис. 5.14.Регістри засобів налагодження

DR7 – регістр керування налагодженням. Його використовують для дозволу або уточнення різних точок зупинки. Біти мають такі призначення:

LEN – це двобітове поле, яке визначає довжину переривання. Всі переривання повинні бути вирівняні: двобайтові вектори – по межах слова; чотирибайтові – по межах подвійних слів. Значення поля: 00 – довжина в один байт, 01 – довжина в один байт, 10 –невизначена, 11 – довжина в чотири байти.

RWE: це двобітове поле визначає тип вибірки з пам'яті, яка повинна відбутися для того, щоб активізувати опрацьовування переривання:

Поле RWE	Тип вибірки
00	Тільки виконання команди
01	Тільки записування даних
10	Невизначений
11	Тільки зчитування та записування даних (команди не вибираються)

GE/LE – глобальна і локальна верхні точки зупинки: ці біти повинні завжди дорівнювати 1 у разі використання переривань.

Gi/Li – розблокування глобальної і локальної точок зупинки: якщо або Gi=1, або Li=1, то переривання розблокуються. Якщо ці біти задані, то будь-яке переривання обмеженого використання (тобто точка зупинки, яка відповідає умовам, що визначені бітами LWE) змусить процесор виконувати програму опрацювання налагодження. Біти Li дають змогу задавати локальні точки зупинки для індивідуальної задачі, не впливаючи на іншу задачу. Gi біти допомагають задавати переривання, які діють на всі задачі.

Для того, щоб задати точку зупинки, мікропроцесор повинен працювати на нульовому рівні привілеїв у реальному режимі. Після того точку зупинки потрібно задати шляхом завантаження регістра точки переривання (засобом MOV DRi, (операнд в пам'яті або регістрі)

команди) з адресою точки зупинки. Згодом потрібно задати відповідний LEN та RWE, і нарешті, біти розблокування точки переривання Gi і/або Li.

Молодші чотири біти Bi в DR6 стосуються чотирьох апаратних контрольних точок; одиничні стани відобразять досягнення точки зупинки, лінійна адреса якої міститься у регістрі DRi. Однак доти, доки не будуть задані Gi або Li, процесор не виконуватиме програми налагодження з цими перериваннями (детальніше див. [1, стор. 254]).

Генерування винятків за контрольними точками можна заборонити заданням ознаки RF у регістрі ознак.

5.7. Захищений режим. Механізми адресування

Усі можливості процесора розкриваються у захищеному режимі віртуальної адреси (захищений режим). Цей режим значно збільшує адресний простір (до 4 Гбайт) і дає змогу виконання програм практично необмеженого розміру (64 Тбайт). Крім того він дає змогу працювати з усіма 8086 і 80826 програмами, забезпечуючи керування величезною пам'яттю та апаратним механізмом захисту. Захищений режим забезпечує використання додаткових команд, спеціально оптимізованих для захищених багатозадачних операційних систем. Базова архітектура 32-розрядного процесора незмінна; регістри, команди і режими адресування, що описані в попередніх розділах, збережені.

Головна відмінність між захищеним і реальним режимом полягає з погляду програміста в тому, що збільшується адресний простір і змінюється механізм адресування.

Як і в реальному, у захищеному режимі використовують дві компоненти для формування логічної адреси: 16-бітовий селектор для визначення лінійної базової адреси сегмента, і 32-бітова виконавча адреса, яку додають до базової адреси для формування 32-бітової лінійної

адреси. Після того лінійну адресу використовують або як 32-бітову адресу або, якщо є можливість сторінкової організації пам'яті, то механізм підкачування перетворює 32-бітову лінійну адресу в 32-бітову фізичну.

Відмінність між двома режимами полягає у способі обчислення базової адреси. У захищеному режимі селектор використовують для специфікації індексу в таблиці операційної системи. Таблиця містить 32-бітову базову адресу цього сегмента. Фізична адреса формується шляхом додавання базової адреси, отриманої із таблиці, до зміщення.

Підкачування забезпечує додатковий механізм керування пам’яттю, який працює тільки в захищеному режимі, а також керування дуже великим сегментом процесора. Механізм підкачування переводить захищену лінійну адресу, яку видає блок сегментування, у фізичну.

5.7.1. Сегментування

Сегментування – один із способів керування пам’яттю, що забезпечує основу для захисту пам'яті. Сегменти використовують для об'єднання ділянок пам'яті, які мають загальні ознаки. Наприклад, у сегменті можуть міститися всі коди заданої програми або постійно зберігатися таблиця операційної системи. Вся інформація про сегменти зберігається у 8-байтовій структурі даних, яку називають *дескриптором*. Усі дескриптори в системі містяться в таблицях, які розпізнає обладнання. Для керування процесом використання привілейованих інструкцій доступу до дескрипторів, застосовують чотирирівневу ієрархічну систему привілеїв. Рівні привілеїв нумерують від 0 до 3. Правила використання рівнів привілеїв контролюють вентилі (Gate) або шлюзи.

Для детальнішого ознайомлення із взаємодією дескрипторів, рівнів привілеїв і захисту варто знати такі терміни:

PL – рівень прівелеїв, один із чотирьох ієрархічних рівнів.

Рівень 0 є найпривілейованішим, а рівень 3 – найменш привілейованим. Більш привілейовані рівні за числовим значенням більші, ніж менш привілейовані;

RPL – рівень привілеїв ініціатора запиту (за запитом) – рівень привілеїв оригінального постачальника селектора. RPL визначений найменшими за значенням двома бітами селектора;

DPL – рівень привілеїв дескриптора – найменш привілейований рівень, на якому завдання може вибрати заданий дескриптор і сегмент, співвіднесений з цим дескриптором. DPL визначений бітами 5 і 6 у байті доступу дескриптора;

CPL – поточний рівень привілеїв – рівень, на якому в поточний момент виконується програма; він є рівнем привілеїв сегмента коду, який виконується. CPL може бути також визначений шляхом вивчення двох наймолодших бітів CS регістра, за винятком узгодження сегментів коду;

EPL – ефективний (виконавчий) рівень привілеїв – менш привілейований, ніж RPL і DPL. Оскільки менші значення привілейованих рівнів підкреслюють більший привілей, EPL за числовим значенням є максимальним щодо RPL і DPL.

5.7.2. Дескрипторні таблиці

Таблиці дескриптора визначають всі сегменти, які використовують процесори, починаючи з 80286. У 32-розрядних процесорах є три типи таблиць, які містять дескриптори: таблиця глобального дескриптора (GDT), таблиця локального дескриптора (LDT) і таблиця дескриптора переривання (IDT). Усі таблиці є матрицями пам’яті різної довжини, вони можуть займати від 8 байт до 64 Кбайт. Кожна таблиця може містити 8192 восьмибайтових дескрипторів. Верхні 13 біт селектора використовують як індекс у таблиці дескриптора. Таблиці мають регістри, співвіднесені з ними, які містять 32-бітову лінійну базову адресу (у 80286 24-бітову фізичну адресу) і 16-бітову межу кожної таблиці. Кожній таблиці відповідає регістр GDTR, LDTR і IDTR.

Для локалізації таблиці LDT використовують 16-бітовий регістр LDTR, який містить тільки селектор. Таблиці LDT не є обов'язковими.

Команди LGDT та LIDT завантажують базу і межу GDT і IDT у відповідний регістр. Команди SGDT і SIDT зберігають значення бази і межі в пам'яті. Команда LLDT завантажує сегмент селектора у регістр LDTR, а команда SLDT зберігає селектор у регістрі. Всі таблиці опрацьовує операційна система, отже, команди завантаження таблиці дескриптора є привілейованими.

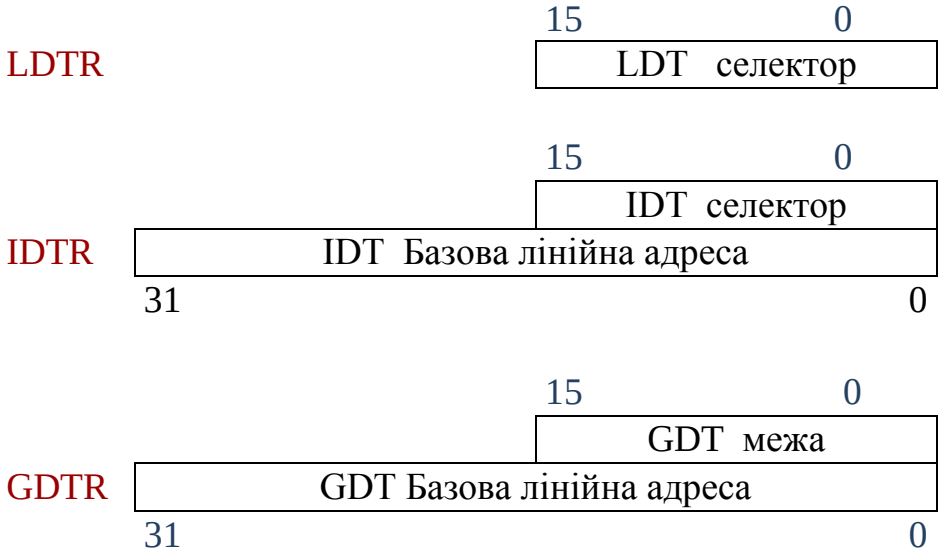


Рис. 5.15. Таблиці декрипторів і відповідні регістри

Таблиця глобального дескриптора (GDT) містить дескриптори, доступні для всіх задач у системі. У GDT може бути будь-який тип дескриптора сегмента, крім дескрипторів, які використовують для обслуговування переривань (тобто дескриптори переривання і пастки). GDT є в кожній системі. Звичайно GDT містить код і сегменти даних, які використовують

операційні системи, сегменти стану задачі і дескриптори для LDT в системі. Перше гніздо GDT відповідає нульовому селектору, його не застосовують.

Таблиця локального дескриптора (LDT) містить дескриптори, які асоціюються (співвідносяться) з заданою задачею. Звичайно, операційні системи створені так, щоб кожне завдання мало окрему LDT. LDT може містити тільки код, дані, стек, вентиль (логічний елемент) задачі і дескриптори вентиля виклику. Таблиці LDT забезпечують механізм для ізоляції коду і сегмента даних заданої задачі від решти операційної системи, тоді як GDT містить дескриптори, загальні для всіх задач. Завдання не може отримати доступу до сегмента, якщо його дескриптора сегмента нема в одній з поточних LDT або GDT. Це забезпечує ізоляцію і захист для сегмента задачі, даючи змогу водночас розділяти глобальні дані між завданнями. На відміну від 6-байтових регістрів GDT або IDT, які містять базові адреси і межі, видима частина регістра LDT містить тільки 16-бітовий селектор, по якому з GDT автоматично завантажуються програмно недоступні поля базової адреси та сегмента.

Таблиці LDT створюються за необхідності.

Таблиця дескриптора переривань (IDT) містить дескриптори, які відображають адреси 256 векторів переривань. В IDT можуть бути тільки вентилі задач, переривань і пасток. IDT повинна бути не меншою від 256 байт для того, щоб фіксувати дескриптори для 32 визначених переривань. Кожне переривання, яке використовує система, повинно мати елемент (вхід) в IDT. Елементи IDT аналізують команди INT, вектори зовнішніх переривань і винятки.

5.7.3. Опис дескрипторів

Дескриптори – це структури даних, які використовують для означення властивостей програмних елементів (сегментів, вентилів і таблиць). Дескриптор визначає положення елемента у пам'яті, розмір області, яку він займає (тобто межу), його призначення і характеристику захисту. Дескриптори 16- і 32-розрядних процесорів відрізняються розрядністю поля базової адреси (24 і 32 біти), а також трактуванням поля межі.

Детальніше розглянемо структуру дескриптора.

Атрибутивні біти дескриптора. Дескриптори є 8-байтовими величинами, які містять атрибути (ознаки) про задану область простору лінійної адреси (тобто сегмента), на який вказує селектор. Ці атрибути включають 32-бітову базову адресу, лінійну адресу сегмента, 20 бітів довжини і ступеня деталізації сегмента, рівень захисту і привілеїв прочитання, написання або виконання, розмір замовчування операндів (16 бітів або 32 біти), і тип сегмента. Вся атрибутивна нформація про сегмент міститься в 12 бітах у дескрипторі сегмента. Звичайний формат дескриптора показано на рис. 5.16.

Усі сегменти в 32-розрядному процесорі мають три загальні атрибутивні поля: P біт, DPL біт і S біт. Біт P (присутність) дорівнює 1, якщо сегмент завантажується у фізичну пам’ять. Якщо P=0, то будь-які спроби отримати доступ до сегмента викликають виняток неприсутності (виняток 2). Рівень привілеїв дескриптора DPL – це 2-бітове поле, яке визначає рівень захисту від 0 до 3, співвіднесений з сегментом. Процесор має дві головні категорії сегментів: системні і несистемні сегменти (для коду і даних). Біт S дескриптора сегмента визначає, чи є заданий сегмент системним сегментом коду, чи сегментом даних. Якщо S=1, то сегмент є або сегментом коду, або сегментом даних. Якщо S=0, то сегмент є системним.

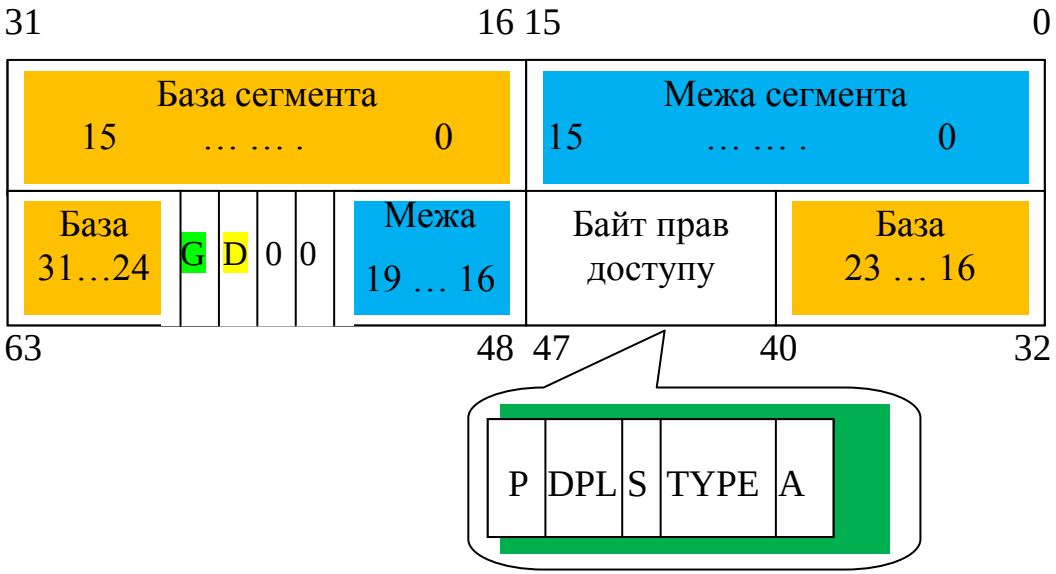


Рис. 5.16. Дескриптори сегмента та розшифрування байта прав доступу

DPL – рівень привілеїв дескриптора; S – сегмент дескриптора; A – біт доступу; P – біт присутності; G – біт гранулярності (G=1 – сторінкова, G=0 – байтова гранулярність); D – біт визначення розміру операнда (D=1– 32 біти, D=0 – 16 біт).

Для реалізації зміни рівнів привілеїв використовуються особливі системні об'єкти – шлюзи, чи вентилі виклику. Вони також мають свої дескриптори.

Дескриптори сегментів коду і даних (S=1). На рис. 5.16 зображено загальний формат дескриптора сегментів коду і даних. У цих дескрипторах є декілька загальних полів. Біт A (Accessed - звертання) задається кожного разу, коли процесор отримує доступ до дескриптора. Біт A використовують операційні системи для підтримки статистики застосування сегмента. Біт G (Granularity), або біт ступеня деталізації, визначає, чи є довжина сегмента деталізованою по байтах або по сторінці.

Сегменти процесора можуть мати довжину 1Мбайт зі ступенем деталізації байта, коли G=0, або 4 Гбайти, зі ступенем деталізації сторінки, коли G=1, тобто 2^{20} сторінок, кожна довжиною 4 К. Ступінь деталізації зовсім не стосується розбиття пам'яті на сторінки. Система процесора може складатися з сегментів зі ступенем деталізації байтів або із ступенем деталізації сторінок, незалежно від наявності або відсутності сторінкової організації пам'яті.

Біт виконання E (Executable) визначає, яким є сегмент – коду, чи даних. Сегмент коду, коли E=1, S=1, може бути тільки виконаний або тільки прочитаний (запис неможливий), що задано бітом R (Readable – прочитання). Сегменти коду виконуються тоді, коли R=0, і зчитуються при R=1.

З а у в а ж е н н я : Сегменти коду можуть бути модифіковані за допомогою псевдонімів. Псевдоніми – це сегменти даних, які можна записати і які займають ту саму область лінійного адресного простору, що і сегмент коду.

Біт D (Default Operation Sise) визначає довжину замовчування для операндів і виконавчих адрес. Якщо D=1, то використовують 32- бітові операнди і 32-бітові режими адресування. Якщо D=0, то застосовуються 16-бітові операнди і 16-бітові режими адресування. Отже, усі наявні сегменти коду i286 будуть працювати в i386 чи i486, якщо біт D=0.

Ще одна ознака сегментів коду визначена бітом C (Conforming – підпорядкованість). Якщо C=1, то код може виконуватися, якщо поточний рівень привілеїв (CPL) не нижчий від рівня привілеїв дескриптора (DPL); якщо C=0, то керування до заданого сегмента може надаватися тільки тоді, якщо CPL=DPL.

Якщо E=0, S=1, то сегменти ідентифіковані як *сегменти даних*, і їх використовують для двох типів сегментів процесора: сегмента стеку і сегмента даних. У цьому випадку біт ED (другий біт поля Type) напряму розширення визначає, в якому напрямі розглядають сегмент:

вниз, тобто сегмент стеку, або вверх – сегмент даних. Якщо сегмент є сегментом стеку, то всі зміщення повинні бути більшими, ніж межа сегмента.

У сегментів даних усі зміщення повинні бути менші або дорівнювати межі. Іншими словами, сегменти стеку починаються з базової лінійної адреси плюс максимальна межа і зростають вниз до базової лінійної адреси плюс межа. З іншого боку, сегменти даних починаються з базової лінійної адреси і розширюються до базової лінійної адреси плюс межа.

Біт W (перший біт поля Type – ознака запису), контролює можливість запису в сегмент. Якщо W=0, то сегменти даних можна тільки читати. Сегмент стеку повинен мати W=1.

Біт B в останньому байті дескриптора визначає розмір регістра вказівника стеку. Якщо B=1, то команди PUSH, POP, CALL використовують 32-бітовий ISP-регістр вказівника стеку, і верхня межа набуває значення FFFFFFFFh. Якщо B=0, то команди, які застосовують стек, використовують 16-бітовий SP-регістр, і верхня межа набуває значення FFFFh.

Формати дескриптора системи. Сегменти системи призначені для зберігання локальних таблиць дескрипторів та дескрипторів стану задач і вентилів. Дескриптори 32-розрядних систем містять 23-бітову базову лінійну адресу і 20-бітову межу сегмента.

Дескриптори 16-розрядних систем мають 24-бітову базову лінійну адресу і 16-бітову межу сегмента. Дескриптори i80286 визначені верхніми 16 бітами, які всі дорівнюють нулю.

- Дескриптори LDT (S=0, TYPE=2) надають інформацію про локальну таблицю дескрипторів. LDT містять таблицю дескрипторів сегмента, які стосуються тільки певної задачі, бо команда завантаження LDTR можлива тільки на рівні привілеїв 0. Поле DPL ігнороване.

Оскільки таблиці LDT не є обов'язковими, і їх створюють за потребою, то дескриптори LDT містяться в глобальній таблиці дескрипторів GDT. Селектор з регістра LDTR вибирає у

таблиці GDT потрібний дескриптор, і атрибути таблиці LDT стають доступними для процесора.

- Дескриптори TSS (S=0, TYPE=1, 3, 8, B) сегмента стану задачі містять інформацію про розташування, розмір і рівні привілеїв сегмента стану задачі (TSS). TSS, відповідно, є спеціальним фіксованим сегментом – інформатором, який містить усю інформацію стану задачі і поле зв'язку для можливості вкладення задач.

Поле TYPE використовують для визначення того, чи є завдання поточним, тобто чи воно в ланцюжку активних задач, чи в TSS. Поле TYPE також визначає, чи містить сегмент 286 або386 TSS. У регістрі задання TR є селектор, який визначає поточний сегмент стану задачі.

- Дескриптори вентилів (шлюзів) (S=0, TYPE=4-7,CF). Вентиль (шлюз) використовують для керування доступом до точок входу в межах сегмента коду. Поле TYPE визначає тип вентиля:
  - вентиль виклику – (4)–80286, (C) –80386;
  - вентиль задачі – (5)–80286, (D)–80386;
  - вентиль переривання – (6)–80286, (E)–80386;
  - вентиль пастки – (7)–80286, (F)–80386.

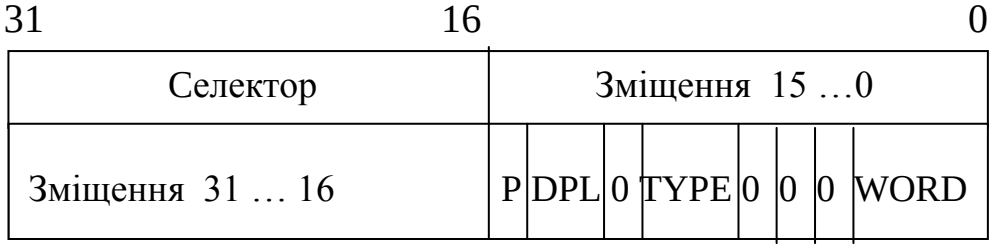


Рис. 5.17. Дескриптори вентилів (32-бітовий формат)

Вентиль забезпечує рівень взаємовідносин між джерелом і адресатом передавання керувань. Це дає змогу процесору автоматично перевіряти захист, а системотехнікам – керувати точками входу в операційних системах. Вентиль виклику застосовують для зміни рівня привілеїв, вентиль задачі – для виконання включення задачі, а вентилі переривання і пастки – для визначення процедури переривання.

Дескриптори вентилів виклику складаються з трьох полів: *байта доступу, довгого вказівника селектора* (позначає початок процедури) і ще *слова* (визначає кількість параметрів, яку необхідно скопіювати зі стеку програми, що викликає, в стек процедури, що викликає).

Поле WORD COUNT (поле рахунку слова) використовують у ключах викликів, коли відбувається зміна в привілейованому рівні. Воно визначає кількість слів зі стеку процесу, що викликає, які автоматично копіюються у стек процедури, яку викликають. Інші типи вентилів ігнорують поле WORD COUNT. Вентилі переривання і пастки застосовують селектор і поле зміщення адресата дескриптора вентиля як вказівник початку процедури опрацювання переривань або пастки. Відмінність між вентилями переривання і пастки полягає в тому, що вентиль переривання унеможливлює переривання: скидає біт IF, тоді як вентиль пастки цього не робить. Вентиль задачі використовують для включення задачі, він може стосуватися тільки сегмента стану задачі.

Формат байта доступу є однаковим для всіх дескрипторів вентиля: P=1 означає, що вміст вентиля є істинним, P=0 – що вміст не є істинним і викликає виняток 11, якщо на нього посилаються. DPL – це рівень привілеїв дескриптора, який визначає, коли заданий дескриптор може бути використаний задачею.

5.7.4. Поля селектора

Селектор у захищеному режимі має три поля:

1) індикатор локальної або глобальної таблиці дескриптора TI;
2) індекс входу дескриптора;
3) рівень привілею ініціатора запиту (селектора).

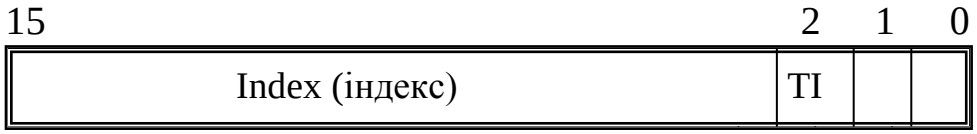


Рис. 5.18. Формат селектора сегмента

Біти TI позначають, з якої таблиці вибрано дескриптор (TI=0 – глобальна таблиця дескрипторів, TI=1 – локальна таблиця дескрипторів). Індекс вибирає один із 8К дескрипторів у відповідній таблиці дескрипторів. Біти RPL рівня привілеїв ініціатора запиту не беруть участі у виборі дескриптора, однак їх застосовують для контролю привілеїв у механізмі захисту.

Кешування дескрипторів сегмента. Крім значення селектора, кожний регістр сегмента має регістр кеш (cache – буфер, склад) дескриптора сегмента, який з ним співвідноситься. Кожного разу, коли значення регістра сегмента змінюється, 8-байтовий дескриптор, співвіднесений з цим селектором, автоматично завантажується. Вміст кеш-дескриптора є невидимими для програміста. Оскільки кеш-дескриптори змінюються тільки тоді, коли змінюється регістр сегмента, то програми, які змінюють таблиці дескрипторів, повинні перезавантажувати відповідні регістри сегмента після зміни значення дескриптора.

Вміст у кеш-дескрипторах сегмента змінюється залежно від режиму, в якому працює процесор. Для сумісності з архітектурою 8086 базу задають у 16 разів більшою від поточного

значення селектора. Межа фіксується на 0000FFFFh, а ознаки – так щоб показати, що сегмент присутній і повністю використовується.

У режимі реального адресування внутрішній рівень завжди фіксується на найвищий рівень привілеїв, тому команди уведення/виведення та інші операційні коди є привілейованими і виконаються завжди. У захищеному режимі кожне з полів визначене відповідно до вмісту дескриптора сегмента, індексованого значенням селектора, що завантажене у регістр сегмента. Віртуальна програма виконується на найнижчому рівні привілеїв (рівні 3) для того, щоб дозволити організацію пасток всіх його команд, чутливих до PL і команд, які виконуються тільки для рівня 0.

На завершення зазначимо, що механізм сегментування має свої переваги і недоліки. До переваг належать:

можливість реалізації віртуальної пам'яті;

побудова операційних систем на базі механізму захисту через привілеї;

виявлення та опрацювання програмних помилок унаслідок порушення меж чи неправильних вказівників.

Головним недоліком механізму сегментування є необхідність виконувати додаткові службові функції у разі завантаження селекторів у сегментні регістри. Для цього потрібно мінімум два цикли шини, тому завантаження сегментного регістра триває значно довше, ніж регістра загального призначення. Детальніше про це див. [2, 3].

5.8. Посторінкова організація пам'яті

Сутність посторінкової організації пам’яті. В англомовній літературі для позначення сторінкового типу організації пам'яті використовують термін "підкачування" (swapping), що означає ще один тип керування пам’яттю з метою організації віртуальної пам’яті багатозадачних ОС.

На відміну від сегментації, яка організовує програми і дані у модулі різного розміру, підкачування ділить програмні модулі на численні сторінки одного і того ж розміру. Підкачування прямо не стосується логічної структури програми чи даних, водночас селектори сегмента можна розглядати як логічні імена модулів програми або структури даних.

Сторінка звичайно відповідає частині модуля або структури даних. З огляду на переваги локальності коду і посилання на дані в оперативній пам'яті у кожний певний момент часу необхідно тримати лише невелику кількість сторінок з кожної активної задачі .

Організація посторінкової пам'яті. Механізм підкачування. Механізм керування сторінками використовує дворівневу таблицю для переведення лінійної адреси (з блоку сегментації) у фізичну (рис. 5.19).

Механізм містить три компоненти:

1) вказівник (каталог) сторінки;

2) таблиця сторінки;

3) самі сторінки (рамки сторінки).

Усі елементи резидентної пам’яті механізму підкачування становлять 4 Кбайти.

Механізм вмикають заданням біта PG=1 у регістрі CR0.

Базовий регістр дескриптора підкачування CR2 є регістром лінійної адреси помилки підкачування. Він містить 32-бітову лінійну адресу, яка викликає визначення помилки в останній сторінці.

CR3 – це регістр фізичної базової адреси вказівника сторінки, який містить фізичну початкову адресу цього вказівника. Молодші 12 біт CR3 завжди дорівнюють 0 для того, щоб забезпечити такий стан, за якого вказівник сторінки завжди вирівняний. Завантажують його командою MOV CR3, REG, яка викликає кеш-уведення таблиці сторінок.

Вказівник сторінки довжиною 4 Кбайта дає змогу розташувати 1024 одиниць вказівника сторінок. Кожна одиниця вказівника сторінки містить адресу наступного рівня таблиці сторінки та інформацію про ці таблиці. Вміст одиниці вказівника сторінки показано на рис. 5.20. Індексом для вибору правильної одиниці вказівника сторінки слугують старші 10 біт лінійної адреси (A22-A31).

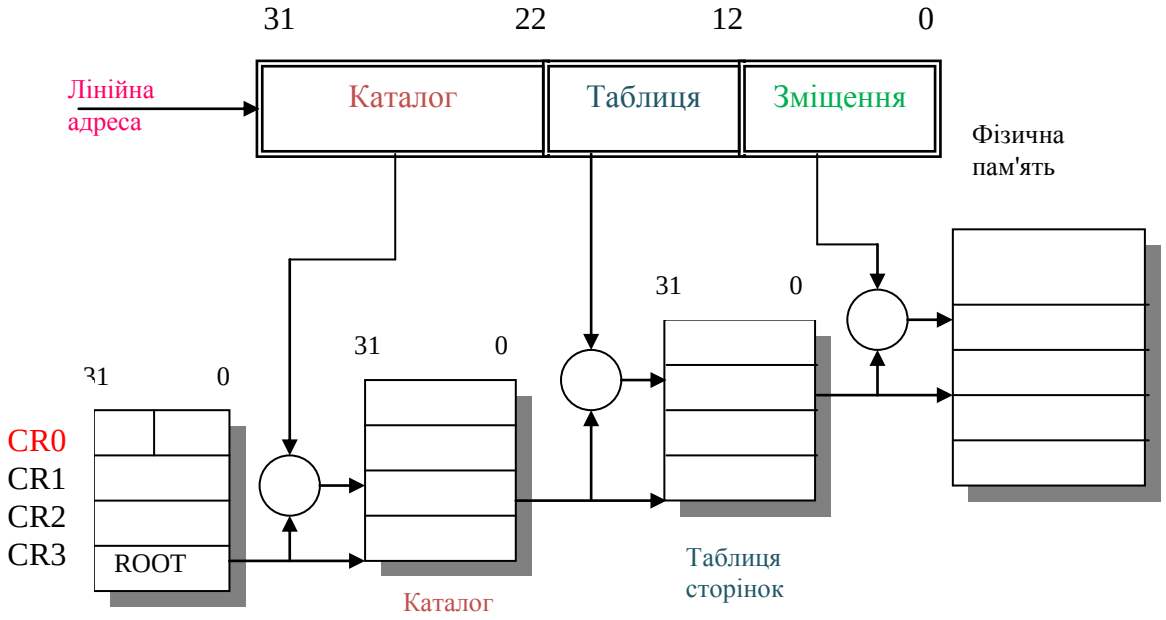


Рис. 5.19. Механізм сторінкової переадресації

Таблиці сторінок. Кожна таблиця сторінки має розмір 4 Кбайти і містить до 1 Кбайта елементів. Елементи таблиці сторінки містять базову фізичну адресу й атрибути самих сторінок. Індексом пошуку у таблиці одного з елементів використовують біти лінійної адреси A12-A21. Фізична адреса формується з адреси сторінки, яку беруть з таблиці, й молодших 12 бітів лінійної адреси (див. рис. 5.19).

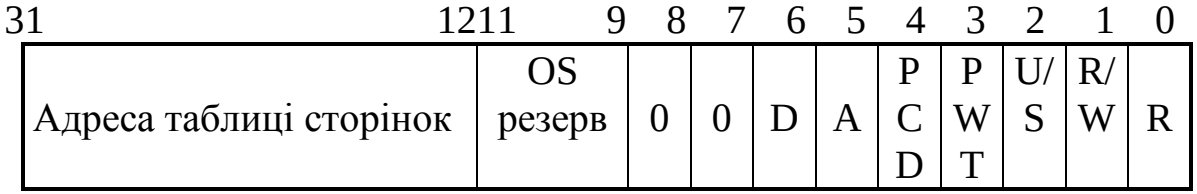


Рис. 5. 20. Рядок каталога сторінки

Таблиці сторінок можуть бути розташовані між завданнями і записані на диск.

5.8.1. Елементи таблиці і вказівник сторінок

Молодші 12 бітів елементів таблиці сторінки й елементів вказівника сторінки містять статичну інформацію про сторінки і таблиці сторінки, відповідно. Біт P визначає, який елемент може бути використаний у передаванні адреси. Якщо P=1, то елемент можна застосувати для адресної трансляції. Якщо P=0, то елемент не може бути використаний для трансляції, і всі решту є в наявності для застосування програмою. Наприклад, 37 бітів, які залишилися, можуть бути використані для задання, де на диску розташована сторінка. Біт A (біт 5, прав доступу)

задається процесором для двох типів елементів, перш ніж відбудеться доступ до прочитання або написання перетвореного елемента.

Біт D (біт 6) дорівнює 1 перед операцією записування за адресою, у перетворенні якої приймає участь рядок таблиці, однак він (біт D) невизначений для елементів вказівників сторінки. Біти A, D, P зазнають змін процесором апаратно у заблокованих шинних циклах, що запобігає виникненню конфліктів з іншими процесорами або додатковими пристроями. Програми, які змінюють ці біти, повинні використати команду LOCK для забезпечення цілістності таблиці сторінки для багатокористувацьких систем.

Поле OS-резерв програмно використовує операційна система під час аналізу того, як довго сторінка була в пам’яті з часу доступу. ОС може виконати алгоритм заміни сторінки як такої, яку останнім часом практично не використовували.

Біт PWT визначає політику записування у випадку кешування, а біт PCD забороняє кешування пам’яті для тих сторінок, що обслуговуються (застосовують у процесорах i486).

Біти U/S (користувач, супервізор) і READ/WRITE (RW) використовують для забезпечення ознак захисту сторінок.

5.8.2.Захист рівня сторінки

Механізм захисту сторінок розрізняє два типи привілеїв: користувач і супервізор. Користувач відповідає рівню привілеїв 3, що ґрунтується на сегметації, а супервізор включає всі рівні захисту: 0, 1 і 2.

Програма, яка виконується на рівні 0, 1 і 2, обминає захист сторінки, хоча захист, що ґрунтується на сегментації і надалі вводиться в дію обладнанням.

5.8.3. Буфер асоціативної трансляції

Обладнання посторінкового механізму призначене для підтримки вимог системи віртуальної пам’яті. Однак робота суттєво погіршиться, якщо процесору необхідно отримати доступ до двох рівнів таблиці для кожного звертання до пам'яті.

Для вирішення цієї проблеми у процесорі вмонтовано асоціативний кеш-буфер найчастіше уживаних сторінок, до яких є доступ. Цей кеш називають TLB-буфером асоціативної трансляції. TLB (Translation Lookaside Buffer) – це чотиридоріжковий набір, що співвідноситься з 32 елементами кеш-таблиці сторінки. Він автоматично зберігає найчастіше використовувані елементи таблиці сторінки в процесорі; 32 елементи TLB співвіднесені з 4 Кбайтовим розміром сторінки. Це дає змогу зберігати інформацію про трансляцію 128К адрес пам’яті, що для багатьох багатозадачних використань TLB досягає коефіцієнта ефективності близько 98%.

5.8.4. Механізм підкачування

Механізм сторінкового переадресування (підкачування) вмикається тоді, коли у регістрі CR0 біт PG=1. Обладнання підкачування після отримання 32-бітної лінійної адреси з блоку сегментації, порівнює старші 20 бітів лінійної адреси з усіма 32 елементами TLB для визначення того, чи підходять вони один одному. Якщо відповідність встановлена, то 32-бітова фізична адреса обчислюється і після того подається на адресну шину. Однак якщо елемент таблиці сторінки не міститься в TLB, то процесор прочитає відповідний елемент вказівника сторінки. Якщо P=1 в елементі вказівника сторінки (а не в елементі таблиці сторінки), то таблиця сторінки (а не сторінка) розміщена в пам’яті, і процесор прочитає відповідний елемент таблиці та задасть біт доступу A.

Якщо P=1 в елементі таблиці сторінки, то сторінка міститься в пам’яті, і процесор поновить біти доступу A і D. Якщо P=0, то незалежно від того, де він дорівнює нулю, чи в

елементі таблиці сторінки чи в елементі вказівника сторінки, мікропроцесор генеруватиме помилку сторінки (Виняток 14), опрацювання якого спричинить завантаження сторінки у пам'ять. Помилку сторінки (Виняток 14) може виробити й процесор, якщо посилання на пам’ять порушує ознаки захисту сторінки (біти U/S або R/W). Такою помилкою є, наприклад, спроба писати на сторінку, яку можна тільки читати. CR2 міститиме лінійну адресу, яка викликала помилку сторінки. Оскільки Виняток 14 класифіковано як помилку CS:IP, то він визначить команду, яка викликала помилку сторінки. Для ідентифікації причини помилки у стек завантажують 16-бітову адресу помилки. На рис. 5.21 показано формат коду помилки сторінки та інтерпретацію бітів.

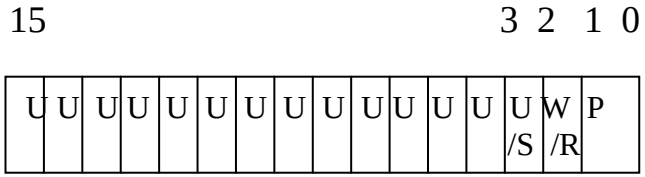


Рис. 5.21. Формат коду помилки у разі відмови сторінки:

біт P – причина відмови (P=0 – порушення захисту, P=1 – відсутність сторінки у пам'яті), W/R=0 – читання, W/R=1 – записування, U/S=1 – користувач, U/S=0 – супервізор, біти U не використовують.

Операційна система повинна коректно генерувати початкові таблиці сторінок і опрацьовувати будь-які помилки сторінки, зокрема, занулити TLB, коли відбуваються будь-які зміни з елементом в таблиці сторінки через перезавантаження CR3. Очищення TLB відбувається також у випадку зміни значень біта P у будь-яких таблицях.

Процесори Pentium і подальші модифікації дають змогу оперувати сторінками з розміром 4 Мбайт. Це забезпечує ефективніше використання коштовного обладнання механізму сторінкової адресації.

Стосовно переваг і недоліків механізму посторінкового адресування насамперед зазначимо, що цей механізм, на відміну від сегментування, діє вибірково, тобто його задають програмним способом. До переваг належать:

факт фіксованого розміру сторінки. Це дає змогу ліпше організовувати обмін з накопичувачами на дисках (підганяючи розмір сторінки до розміру сектора чи кластера);

відсутність проблеми фрагментації. Невикористаних місць між сторінками немає;

великий об'єкт може бути розташований не обов'язково у сусідніх сторінках, а у довільних;

сторінкова організація невидима для програміста і не потребує знання бази сегмента і зміщення.

Деякі недоліки:

наявність внутрішньої фрагментації, тобто програмний об'єкт може точно не вкладатися у сторінку чи кілька сторінок;

звертання до каталогу сторінок і таблиці сторінок необхідні під час кожного звертання до пам'яті.

Ці недоліки можна значно компенсувати завдяки використанню кеш-буфера [2].

5.9. Захист та привілеї

Функція захисту процесора полягає у забороні несанкціонованого виконання критичних команд, зокрема, команди HLT, яка зупиняє процесор, а також команд, що впливають на сегменти коду і даних. Розрізняють три групи механізмів захисту:

- обмеження використання сегментів;
- обмеження доступу до сегментів завдяки правилам привілеїв;
- наявність привілейованих команд або операцій, які можуть бути виконані тільки за певних рівнів CPL та IOPL.

На відміну від традиційних систем, які використовують у мікропроцесорах для захисту, 80386 забезпечує захист як частину інтегрального блоку керування пам’яттю і не використовує складного зовнішнього обладнання та програмного забезпечення. 32-розрядні процесори пропонують також додатковий вид захисту, що грунтується на сторінковій організації пам’яті.

Тридцятидворозрядні процесори у захищеному режимі мають чотири рівні привілеїв, які оптимізуються для підтримки потреб багатозадачних ОС, ізолюють і захищають програми користувача одну від одної і від ОС.

Рівні привілеїв керують використанням привілейованих команд, а також командами уведення/виведення і доступом до сегмента і дескриптора сегмента. У разі виконання майже кожної машинної команди відбувається перевіряння захисту.

Рівні привілеїв пронумеровані від 0 до 3. Рівень 0 є найбільше привілейованим (рис. 5.21).

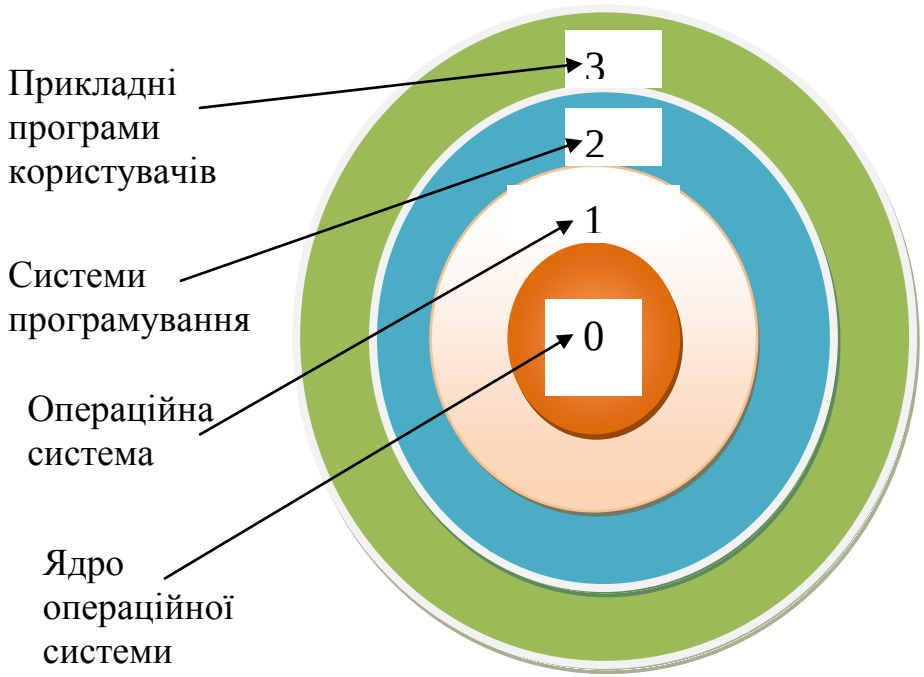


Рис. 5.22. Рівні привілеїв, або кільця захисту

Традиційні системи супервізор/користувач працюють з двома кільцями захисту. Переважно супервізору (тобто операційній системі) відведено кільце 0, а користувачеві – кільце 3 (наприклад, ОС UNIX).

5.9.1. Правила привілеїв

Процесор керує доступом як до даних, так і до процедур між рівнями привілеїв задач згідно з такими правилами: дані, які поміщені в сегмент з рівнем привілеїв P, можуть бути доступні тільки з кодом, що працює з рівнем привілеїв у крайньому разі з таким же ступенем привілеїв, як P.

Сегмент, або процедура коду, з рівнем привілеїв P, може бути викликана задачею, що працює на тому ж або менш привілейованому рівні, ніж P.

5.9.2. *Рівні привілеїв*

Привілеї задач. В будь-який момент часу 32-розрядний процесор виконує завдання у одному з чотирьох рівнів привілеїв. Поточний рівень привілеїв CPL (встановлюється двома молодшими бітами регістра CS) визначає рівень привілеїв задачі. CPL задачі можна змінити тільки процедурою керування через дескриптори вентиля до кодового сегмента з іншим рівнем привілеїв. Отже, програма, яка виконується на рівні привілеїв 3, може викликати програму ОС на рівні привілеїв 1 засобами вентиля. Задача, яка виконується на нульовому рівні привілеїв, має доступ до всіх сегментів, які описані у GDT, і є найбільш привілейованою.

Привілеї селектора RPL. Рівень привілеїв селектора визначений двобітовим полем RPL селектора, і його використовують тільки для задання меншого рівня привілеїв, у порівнянні з поточним рівнем привілеїв для заданого сегменту.

Цей рівень називають ефективним рівнем привілеїв задачі EPL і визначають як найменш привілейований рівень CPL задачі і RPL селектора, тобто якщо RPL селектора дорівнює 0, то CPL завжди визначає рівень привілеїв для можливого доступу використання селектора. Якщо ж RPL=3, то доступ до сегмента на рівні 3 відбудеться незалежно від CPL задачі.

Селектор, що його формує програма, може обмежити будь-яке значення RPL. Це забезпечує команда налагодження RPL – ARPL, яка змушує втручатися в біти RPL, CPL постановника.

Привілеї уведення/виведення. Рівень привілеїв уведення/ виведення IOPL дає змогу коду операційної системи, що працює при CPL=0, обмежити менш привілейований рівень, за якого можуть бути використані команди уведення/виведення.

Виняток 13 – порушення загального захисту – виникає тоді, коли застосовують команду уведення/виведення, а CPL задача менш привілейована, ніж IOPL (пригадаємо, що IOPL розташовується в бітах 13, 14 регістра EFLAGS).

Команди INT, INC, OUT викликають виняток 13, якщо CPL більше, ніж IOPL.

Для того, щоб задачі "оминали" спрацьовування захисту, процесор має декілька спеціальних команд перевірки вказівника, що допомагає підтримувати цілісність системи.

Команда ARPL {операнди, селектор, регістр, функція} налагоджує потрібний рівень привілеїв: APL селектора, числовий максимум поточного селектора, значення RPL і значення RPL у регістрі; встановлює нульову ознаку ZF=1, якщо селектор RPL був змінений.

Команда VERR {операнди, селектор, функція} – перевірка для читання (верифікація читання) – задає нульову ознаку, якщо сегмент, до якого належить селектор, може бути прочитаний.

Команда VERW {операнд, селектор} – перевірка для запису (верифікація запису) – задає нульову ознаку, якщо сегмент, на який посилається селектор, може бути записаний.

Команда LSL {операнди, регістри, селектор, функція} завантажує межу сегмента, зчитуючи її в регістри, якщо правило привілеїв і тип дескриптора це дозволяють.

Команда LAR {операнд, регістр, селектор, функція} завантаження прав доступу. Зчитує байт права доступу дескриптора в регістр, якщо дозволяють правила привілеїв. Ознака IF=1 у разі успішного завершення обох останніх команд.

Процесор виконує перевірку захисту. Спочатку він перевіряє, чи сегмент не є нульовим. Неправильний сегмент виробляє виняток 11, після того перевіряє, чи належить селектор до правильного типу сегмента.

Контроль доступу до сегментів даних здійснюється у разі виконання команд завантаження селекторів у регістри SS, DS, ES, FS, CS. Ці команди, окрім завантаження у SS, змушені робити посилання на дескриптори сегментів даних або сегментів команд. Винятком з цих правил є дозволений для читання підпорядкований кодовий сегмент, який може бути прочитаний з будь-яким значенням CPL. Після того в решті решт виконується перевірка правильності привілеїв. CPL порівнюється з EPL і, якщо EPL більше привілейований, ніж в CPL, то виникає виняток 13.

Правила, які стосуються сегмента стеку, дещо відрізняються від тих, які утягують сегменти даних. Команда, яка завантажує селектор SS, повинна посилатися на дескриптор сегмента даних, що дозволяє записування. Для цього DPL і RPL повинні дорівнювати CPL. Порушення цієї умови або порушення правил привілеїв вироблятимуть виняток 13.

Помилка – стек не присутній – також виробляє виняток 13, а для для неприсутнього коду або сегменту даних – характерний виняток 11.

Передавання рівня привілеїв. Передавання керування всередині сегмента відбувається тоді, коли селектор завантажується в регістр CS. Для звичайних систем більшість з цих передавань є просто результатом виклику або переходу до інших програм.

Є п'ять типів передавання керувань. Частина з них приводить до передавання рівня привілеїв. Зміна рівня привілеїв виконується тільки внаслідок передавання керування з використанням вентилів включення задачі і переривання або вентилів пасток. Передавання керувань може відбутися тоді, коли селектор співвідноситься з правильним типом дескриптора. Будь-яке порушення використання правил дескрипторів спричинить виняток 13.

Для безпеки системи всі передавання керувань підпорядковані таким правилам привілеїв:

1) переходи рівня привілеїв відбуваються тільки через вентилі (шлюзи);
2) команди JMP і CALL можуть здійснювати посилання або на підпорядкований сегмент коду з DPL, який є більшим або дорівнює CPL, або на непідпорядкований сегмент з DPL, який дорівнює CPL.
3) переривання, опрацювання яких відбувається в межах задачі, підпорядковані також правилам привілеїв, як і CALL;
4) сегменти підтвердження коду доступні для рівнів привілеїв, які є такими ж або менш привілейованими, ніж CS сегмента підтверджувального коду. Потрібний рівень привілею RPL в селекторі, що визначає вентиль і CPL задачі, повинні дорівнювати або мати більший привілей, ніж CPL вентиля;
6) переривання, які можуть змінити рівень привілеїв, можуть передавати керування сегменту коду з рівнем привілеїв, таким же або більш привілейованим, ніж CPL тільки через вибраний вентиль з таким же або більш привілейованим, ніж CPL задачі;
7) команди повернення, які не перемикають задачі, можуть повернути керування тільки до сегмента коду з таким же або меншим привілеєм.

Зміна рівня привілеїв, яка відбувається у разі передавання керувань, призводить до автоматичного перевизначення стеку. Увімкнення задачі повинно бути виконано командами CALL, JMP або INT, які належать або до вентиля задачі, або до сегмента стану задачі, чиї DPL менші або мають такий самий привілей, як і CPL попередньої задачі. У разі повернення на попередній рівень привілеїв його стек відновиться як частина IRET- або RET-команди.

5.9.3. Вентилі (шлюзи) виклику

Одна із головних функцій використання вентилів (шлюзів) полягає в тому, щоб забезпечити захищений метод передавання привілеїв усередині задачі.

Вентиль має рівень привілеїв, що відрізняється від рівня привілеїв коду сегмента, який викликають. Як системний об'єкт, шлюз має свій власний дескриптор, який, однак, не визначає жодного адресного простору.

Вентилі підлягають правилу передавання керування привілеїв і таким способом можуть передавати контроль тільки більш привілейованому рівню. Вентилі виклику доступні тільки через команду CALL і синтаксично ідентичні до виклику нормальної підпрограми, а вентилі переривання і вентилі пасток працюють точно так само, як і вентилі виклику, проте в цьому разі не відбувається копіювання параметрів.

Єдина відмінність між вентилями пасток і переривань полягає в тому, що передавання керувань через вентиль переривання блокує подальше переривання, тобто біт IF дорівнює нулю, а вентилі пастки залишають статус переривання не зміненим (детальніше див. [2]).

5.9.4. Перемикання задач

Дуже важливою ознакою в будь-якій багатозадачній операційній системі є її змога швидкого перемикання виконуваних процесів. Операція перемикання задачі процесора зберігає цілісний стан машини (усі регістри, адресний простір і зв'язок з попередньою задачею), завантажує новий стан керування, контролює перевіряння захисту і починає виконання нової задачі. (Усе це відбувається, наприклад, для 80386 приблизно за 17 мкс). Перемикання задачі відбувається через виконання внутрішньосегментної команди JMP або CALL, які посилаються на сегмент стану задачі TSS, або дескриптор вентиля задачі в GDT, або LDT.

Команда INT винятку пастки або вентиля переривання може також викликати операцію перемикання задачі, якщо існує дескриптор вентиля задачі, співвіднесений до області дескриптора IDT.

Дескриптор вентиля задачі містить TSS селектор. Процесор 80386 підтримує і 286, і 386 стиль TSS. Формати TSS для 16-розрядних і 32-розрядних процесорів різні.

Межа 80386 TSS повинна бути більшою, ніж 0064h (002Bh для 286 TSS), а останнім елементом 80386 TSS – байт 0FFh.

У додатковому просторі TSS операційна система розташовує таку додаткову інформацію: чому завдання не активне; час, протягом якого це завдання виконували; відкриває файли, які належать до завдання. Кожне завдання повинно мати співвіднесені з нею TSS.

Поточний TSS визначений спеціальним регістром процесора, який називають регістром стану задачі TR. Дескриптор TSS позначає сегмент, який містить повний стан процесора, а дескриптор вентиля задачі містить селектор, який вказує на дескриптор TSS.

Регістри бази і межі, співвіднесені з TR і TA, завантажуються тоді, коли TA завантажується в новий селектор. Повернення з завдання виконує команда IRET. Коли IRET виконана, то керування передається до задачі, яка була першою. Ознака вкладення задач NT у регістрі ознак керує функцією команди. При NT=0 команда IRET виконує регулярне повернення, залишаючись у поточній задачі, а коли NT=1, то IRET виконує операцію перемикання до попередньої задачі.

Коли команди CALL або INT ініціалізують увімкнення задачі, то новий TSS буде позначений зайнятим. Ці ж команди задають у новій задачі біт NT, ініціалізуючи ввімкнення задачі. Переривання, яке не викликає перемикання задачі, очистить NT (біт NT буде заново відновлено після виконання програми опрацювання переривання).

Сегмент стану задачі процесора позначають зайнятим шляхом зміни поля типу дескриптора. Використання селектора, який належить до сегмента стану зайнятої задачі, викликає виняток 13.

У разі перемикання задач зміна контексту співпроцесора не відбувається автоматично, оскільки нова задача може не використовувати співпроцесор. Коли процесор перемикає задачу, то він задає біт TS. Процесор виявить перше використання команди розширення після перемикання задачі і виробить виняток 7 (нема співпроцесора). Опрацьовувач цього винятку сам визначить, чи необхідна зміна контексту.

Виняток 7 трапиться також тоді, коли буде зроблена спроба виконати команди ESC або WAIT, якщо TS=1 і MP=1.

5.10. Віртуальна конфігурація 8086

Тридцятидворозрядний процесор дає змогу виконувати програми 8086 як у реальному, так і у віртуальному режимах. Віртуальний режим 32-розрядного процесора (V86) для користувача є гнучкішим і привабливішим, оскільки можна скористатися всіма перевагами цього процесора і, крім того, виконувати в ньому програ-ми, як і в реальному режимі. Механізм захисту процесора забезпечує одночасне виконання операційної системи і програм 8086, а операційна система мікропроцесора 80386 – також програм 80286, 80386.

Усі програми, які виконуються у V86, мають рівень привілеїв 3. Адресування відбувається так само, як і у реальному режимі. Увійти у режим V86 (біт регістра ознак VM=1) можна двома способами:

виконати команду IRET у 32-розрядному режимі, коли копія регістра ознак збережена у стеку з заданим бітом VM (при CPL=0);

перемкнути задачу на 32-розрядну, у якої в TSS копія регістра ознак має заданий біт VM.

Вихід з режиму V86 можливий тільки в разі опрацювання переривання. Новий режим (захищений чи V86) буде задано відповідно до TSS нової задачі.

5.10.1. Механізм адресування у віртуальному 8086 режимі

Однією з головних відмінностей адресування віртуального режиму від решти (реального і захищеного) є те, як інтерпретовано селектори.

У віртуальному 8086 режимі, як і в реальному, для отримання лінійної адреси вміст сегментних регістрів зсувається ліворуч на 4 біти і додається зміщення. Завдяки посторінковій організації пам'яті адресний простір задачі в 1 Мбайт у віртуальному режимі може бути відображений у будь-якому місці 4-гігабайтового простору лінійної адреси 32-розрядного процесора.

У віртуальному режимі адреси, які перевищують 1 Мбайт, будуть виробляти виняток 13 (як і в реальному режимі), однак ці обмеження не є важливими, оскільки більшість задач, які виконуються у віртуальному 8086 режимі, є одиничними програмами мікропроцесора 8086 (а їхній розмір не перевищує 1 Мбайт).

5.10.2. Підкачування у віртуальному режимі

Механізм підкачування дає змогу одночасно опрацьовувати численні програми і забезпечує захист та ізоляцію операційної системи. Завдяки обладнанню підкачування можна 20-бітову лінійну адресу, яку виробляє програма у віртуальному режимі, розбити на 256 сторінок. Одна із сторінок може бути розташована у будь-якому місці в межах 4-гігабайтового простору фізичної адреси 32-розрядного процесора. Крім того, оскільки CR3 (базовий регістр вказівника сторінки) завантажується у разі перемикання задачі, то кожне завдання у віртуальному режимі може використати різні схеми для того, щоб відобразити сторінки на різні місця в пам’яті. І, нарешті, обладнання підкачування дає змогу розділити код ОС 8086 між численними аплікаціями 8086.

5.11. Режим системного керування SMM

Сучасні моделі 32-розрядних процесорів (починаючи з деяких модифікацій 386 і 486-го) мають додатковий режим системного керування SMM (System Management Mode). Головно його використовують для реалізації системи керування енергоспоживанням. Відразу після входження у режим процесор зберігає свій стан (контекст) – майже всі регістри – у спеціальній ділянці фізичної пам'яті SMRAM (розмір від 32 Кбайт до 4 Гбайт). Згодом процесор переходить до виконання опрацьовувача SMI, який міститься також у SMRAM. У цьому випадку заборонено всі типи переривань і не генеруються винятки. Процедура опрацьовувача завершується командою RSM, внаслідок виконання якої процесор відновлює свій стан з копії, яка зберігалася у SMRAM. Якщо не задано спеціального режиму, то регістри стеку і кодів операцій мають 16-бітове значення. Режим SMM має такі властивості:

- обчислення адрес відбувається так само, як і реальному режимі;
- межа не перевищує значення 4 Гбайти;
- ознака переривань IF=0;
- немасковане переривання NMI – заборонене;
- ознака TF у регістрі EFLAGS не задана (покроковий режим забороненний);
- регістр DR7 занулений, пастки для налагодження заборонені;
- може бути виконана команда RSM (у інших режимах вона викликає неправильний код операції).

Значення базової адреси SMBASE (переважно 3000h) задає процесор у випадку скидання (сигнал RESET), її можна змінити програмним способом унаслідок дії опрацьовувача SMI. Для цього достатньо змінити значення базової адреси у ділянці збереженого контексту за адресою SMBASE+FEF8h і тоді після виконання команди RSM під час опрацювання сигналу SMI буде використана нова ділянка пам’яті.

Для керування рестартом команд уведення/виведення застосовують слово за адресою SMBASE+FF00h. Рестарт команди буде виконано у тому випадку, коли опрацьовувач запише у слово код 0FFh, а якщо нульове значення слова не буде змінено, то після виходу з SMM процесор виконуватиме наступну команду. Можливий рестарт команди HALT (зупинка процесора), якщо додано нульовий біт слова за адресою SMBASE+FF02h. Цей біт встановлюється в одиничний стан у випадку появи переривання під час виконання команди HALT. Якщо опрацьовувач SMI збереже це значення, то після завершення SMM процесор повторно виконає команду HALT, якщо не збереже, то процесор виконає команду, яка є наступною після HALT (детальніше див., напр. [3]).

5.12. Особливості мікропроцесора i80486. Кеш-пам'ять

Архітектура МП Intel 80486 дуже подібна до архітектури i80386. Однак вирізняє її два суттєві вирішення – організація внутрішньої кеш-пам'яті та пристрою з плаваючою комою.

Тут дані і адреси передаються по внутрішніх 32-розрядних двонапрямлених шинах, а для покращення обміну даними з кешом запроваджено режим пакетного передавання даних (Burst Mode). Запроваджено елементи RISK архітектури, що дало змогу найужи-ваніші команди виконувати протягом одного такту. Крім того:

1) додано нову ознаку AC (Alignment Check) контролю вирівнювання (біт18);
2) додано нові біти у регістрах керування CRO (NE, WP, AM, CD) та CR3 (PCD, PWT). Біт CD (CasheDisable) – керування кешом, біт NW (NotWrite-through) – наскрізним записом у зовнішню пам'ять;
3) додано нові команди (три прикладні – BSWAP (Byte SWAP) обміну байтами, XADD

(eXchange and ADD) обміну і додавання, CMPXCHG (CoMPare and eXCHanGe) порівняння і обміну) і три системні команди для керування кеш-пам'яттю і буфером TLB);

4) уведено функції контролю вирівнювання операндів;
5) чергу команд збільшено до 16;
6) розширено засоби тестування (рег. TR3, TR4, TR5 для тестування внутрішнього кешу);
7) застосовано множення тактової частоти системної плати (внутрішня частота DX2 дорівнює подвоєній зовнішній, а у DX4 – кратність може бути 2, 2,5 і 3).

Спрощена внутрішня архітектура процесора 80486 показана на рис. 5.23.

Кеш-пам'ять розділяє дві 32-бітові шини даних з пристроєм сегментації, цілочисловим пристроєм та пристроєм з плаваючою комою. Ці дві шини можна використати сумісно як 64-бітову шину для передавання між пристроями. У цьому випадку 64-бітові дескриптори сегментів передаються з кеш-пам'яті у пристрій сегментації, 32 біти прямо передаються по одній шині даних, а інші 32 – через цілочисловий пристрій, отже, всі 64 біти досягають пристою сегментації одночасно.

У МП 80486, як і у 80386, застосовано конвеєризацію команд: одночасно виконуються операції, пов'язані з попереднім вибиранням команд, дешифруванням команд, виконанням мікрокоду, а також цілочислові операції, операції з плавачою комою, сегментації, сторінкового перетворення, керування кеш-пам'яттю і шинним інтерфейсом.

Рис. 5.23. Спрощена схема архітектури МП80486

Робота кеш-пам'яті (внутрішньої, або першого рівня).

З метою збільшення швидкодії під час звертання процесора до оперативної пам’яті у архітектурі 32-розрядних процесорів зреалізована така ієрархія пам’яті, яка передбачає наявність відносно великої ємності і меншої швидкодії динамічної **DRAM** та меншої ємності і більшої швидкодії кеш-пам’яті, яка будується на мікросхемах статичної пам’яті **SRAM** (**Static RAM**). Термін “кеш” (**cache**) у нашому випадку відповідає значенню “схованка” і секрет цієї схованки полягає у тому, що спеціальний контролер КЕШу може передбачати використання процесором певної частини оперативної пам’яті і наперед завантажувати її у швидкодіючу кеш-пам’ять. Спочатку кеш-пам’ять розташовували на системній платі, а згодом стали переносити у мікропроцесор. Розрізняють кеш першого рівня (**L1 Cache**), другого (**L2 Cache**) та третього рівнів (використовують у робочих станціях і серверах). У сучасних мікропроцесорах, як правило, використовують кеш-пам’ять рівнів **L1** і **L2**. Кеш-пам’ять третього рівня може вбудовуватись у мікропроцесор, або ж встановлюватись на системній платі.

Кеш-пам'ять i486 розділяє дві 32-бітові шини даних з пристроєм сегментації, цілочисловим пристроєм та пристроєм з плаваючою комою. Ці дві шини можна використати сумісно як 64-бітову шину для передавання між пристроями. У цьому випадку 64-бітові дескриптори сегментів передаються з кеш-пам'яті у пристрій сегментації, 32 біти прямо передаються по одній шині даних, а інші 32 – через цілочисловий пристрій, отже, всі 64 біти досягають пристою сегментації одночасно.

За означенням кеш-пам’ять є меншої ємності, ніж оперативна, отже не може зберігати копію всієї оперативної пам’яті. Цей тип пам’яті зберігає лише обмежену кількість інформації і таблицю (список) відповідності даних ділянкам основної пам’яті. Крім того не вся оперативна пам’ять, яка доступна процесору, може кешуватися. Головною причиною у цьому є можливості контролера кешу.

Ефективність роботи кеш-пам’яті очевидно залежить від того, наскільки вдало розташовані у ній дані, до котрих відбуваються звернення процесора. Розрізняють два випадки:

1. якщо в результаті звернення процесора до кешу знайдено відповідні дані, попередньо прочитані з основної пам’яті, то вважають, що відбулося *кеш-попадання* (cache hit);
2. якщо в результаті звернення процесора даних у кеші не виявилося, то вважають що відбувся *кеш-промах* (cache miss). У цьому випадку процесор повинен прочитати дані з основної пам’яті.

Відношення кількості кеш-попадань до загальної кількості звернень називають коефіцієнтом збігу, або успіху.

Відсоток вдалих попадань головно залежить від алгоритму кешування блоків даних з основної пам’яті до кешу.

Контролер кешу забезпечує передавання рядків даних певної довжини (cache line). Кожному рядку кешу відповідає певний блок даних основної пам’яті та інформація про адресу скопійованих у нього даних і про її стан. Якщо в поточний момент часу у рядку відображена достовірна інформація, то такий рядок називають дійсним (valid), у протилежному випадку – недійсним. Інформація про адресу блоку даних чи номер сторінки та стан рядка називають тегом (tag) і зберігають у пов’язаній з даним рядком комірці спеціальної пам’яті тегів (tag RAM). Можливий варіант секторованого кешу, в якому один рядок містить інформацію про кілька суміжних комірок (секторів).

Розрізняють дві політики чи стратегії запису даних з кешу в оперативну пам'ять; **наскрізний запис WT(Write Through) і зворотний запис WB(Write Back).** Наскрізний запис передбачає виконання кожної операції запису одночасно в рядок кеша і оперативну пам'ять. Ця стратегія застосована у перших процесорах і486. У сучасних процесорах переважає стратегія

зворотнього запису, суть якої полягає у зменшенні кількості операцій запису на системній шині основної пам'яті. Докладніше про це див. у [Таненбаум, Локазюк].

Залежно від способу відображення блоку основної пам’яті на рядок кешу розрізняють три типи архітектур кеш-пам’яті:

- кеш прямого відображення (direct-mapped cache);
- повністю асоціативний кеш (fully associative cache);
- набірно-асоціативний кеш (set-associative cache).

Кеш прямого відображення передбачає, що адреса пам’яті, за якою відбувається звернення до кеша, однозначно визначає рядок кеша, де може знаходитись відповідний блок. Кеш цього типу застосовується у вторинному кеші більшості системних плат сучасних ПЕОМ. У цьому типі архітектури легко обчислити ємність кешованої основної пам’яті $M_{кеш}$

$$\mathbf{M_{кеш} = V_C * 2^n},$$

де Vc – ємність кеш-пам’яті , n – розрядність пам’яті тегів.

Наприклад, якщо на системній платі встановлено кеш ємністю 256 Кбайт і розміром рядка 32 байти, то ємність кешованої пам’яті дорівнює 64 Мбайти.

У повністю асоціативному кеші будь-який його рядок може відображати будь-який блок основної пам’яті. Зреалізований для обмеженого числа кешів першого рівня.

Набірно-асоціативний кеш містить кілька паралельних і погоджено працюючих каналів прямого відображення. Ця архітектура широко застосовується для первинного кешу сучасних ПЕОМ. Ємність кешованої пам’яті визначається так само, як і у випадку прямого відображення, однак тут розрахунок ведеться для одного блоку а не всього кешу.

Розглянемо роботу чотириканальної набірно-асоціативної кеш-пам'яті на прикладі **внутрішньої кеш-пам’яті i486.** КП несекторована, тобто кожний біт достовірності стосується цілого рядка. Якщо адреса потрапляє у кеш, то з нею порівнюють усі теги і у випадку зрівняння одного з тегів процесор запитує інформацію за адресою, що асоціюється з тегом (потрапляння). У цьому випадку цикл шини не потрібний. Якщо інформації у КП нема (випадок промаху), то виконується заповнення рядка КП за одне або декілька 16-байтових передавань даних (для процесорів Pentium – 32-байтові). Схема організації внутрішньої кеш-пам’яті показана на рис. 5.24..

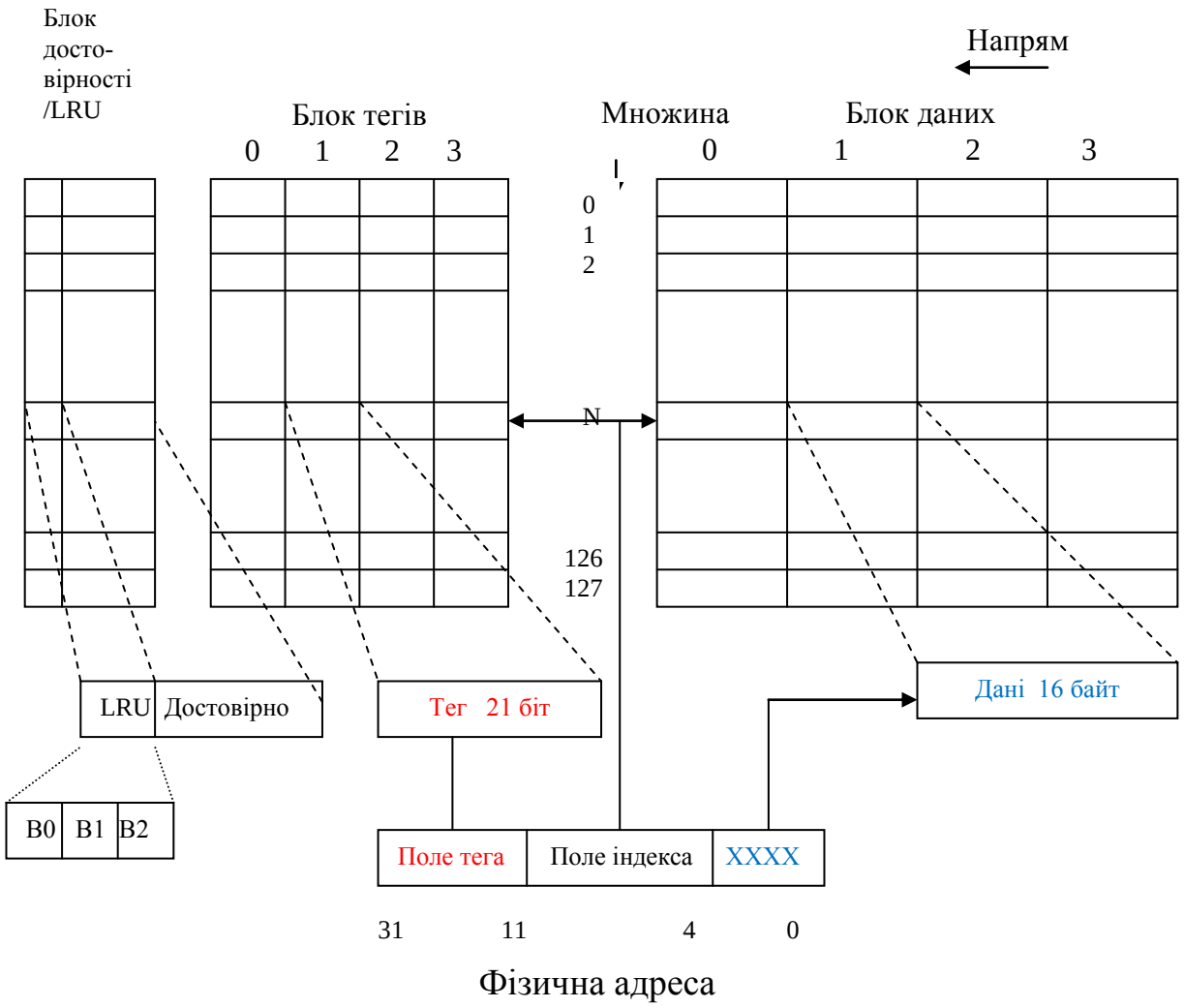


Рис. 5.24. Схема організації внутрішньої кеш-пам'яті

Адресування у КП відбувається шляхом розділення старших 28 бітів фізичної адреси на три частини: 7 бітів поля індекса визначають номер множини (із 128 множин). Старші 21 бітів є полем ознаки, або тегу. Ці біти порівнюються з тегами кожного рядка в індексованій множині і відображають, чи зберігається 16-байтовий рядок КП за заданою фізичною адресою. Молодші 4 біти фізичної адреси вибирають байт усередині рядка КП.

КП функціонально розділена на три блоки: блок даних, блок тегів та блок достовірності і LRU (Least Recently Used – той, що використовувався найдавніше). У цьому випадку застосовано алгоритм псевдо LRU, який описується згодом.

Блок даних містить до 8 Кбайт і розділений за чотирма напрямами, кожний з яких має 128 16-байтових множин, або рядків КП. Такий вибір є компромісом між швидкодією КП з прямим відображенням у разі потрапляння і великим коефіцієнтом потраплянь повністю асоціативної пам'яті.

Режим роботи КП визначений програмно (CR0, біт CD). Унаслідок очищення КП всі біти достовірності дорівнюють нулю.

Алгоритм псевдо LRU: біти LRU оновлюються під час кожного "потрапляння" в КП або заповнення рядка. Біти достовірності в разі очищення КП дорівнюють нулю. Якщо у циклі виявиться "промах" в КП і потрібно передати з пам'яті певний рядок, то для заповнення буде вибрано один з чотирьох рядків множини. Якщо у множині є недостовірний рядок, то власне він і заповниться. Якщо ж усі рядки достовірні, то замінюваний рядок буде вибрано за допомогою бітів з LRU.

Схема вибору заміни рядка. Якщо всі рядки в множині достовірні, то вибір рядка для заміни відбувається за такою схемою:

B0	B1	B2		
0	0	x	змінюється рядок	L_0
0	1	x	змінюється рядок	L_1
1	x	0	змінюється рядок	L_2
1	x	1	змінюється рядок	L_3

Модифікація бітів LRU відбувається так:

- якщо останнє звертання в множині було до рядка L0 або L1, то біт B0 =1, а в разі звернення до рядка L2 або L3 біт B0=0;
- якщо останнє звернення в парі L0-L1 було до рядка L0, то B1=1, а в разі звернення до L1 – B1=0;
- якщо останнє звернення в парі L2-L3 було до рядка L2, то B2=1, а в разі звернення до L3 – B2=0.

У випадку потраплянь КП працює з такою ж швидкістю, як регістри процесора.

У МП 80486 та наступних процесорах за допомогою пакетних циклів (в одному пакеті довжиною до 16 байт можна передавати тільки вирівняні блоки даних за суміжними адресами) подвійне слово з пам'яті зчитується за кожний такт. Для пересилання одного рядка кешу потрібно чотири 32-розрядні шинні цикли. Завдяки цьому дуже швидко заповнюється внутрішня КП і буфер попереднього вибирання команд. У результаті використання КП близько 90-95% запитів на читання задовольняє власне вона і тільки 5-10% припадає на промахи. Про пакетний режим передавання даних можна довідатись детальніше у [3].

Крім внутрішньої КП, застосовується також зовнішня (можливі ємності 64, 128 Кбайт з нарощенням до 512 Кбайт). Вмикають цю пам’ять між шиною процесора та системною шиною. У сучасних процесорах це кеш-пам'ять 2 і 3 рівнів.

Найшвидшою пам'яттю є кеш першого рівня - L1-cache. По суті, вона є невід'ємною частиною процесора, оскільки розташована на одному з ним кристалі і входить до складу функціональних блоків. Складається з кеша команд і кеша даних. Деякі процесори без L1 кешу не можуть функціонувати. На інших його можна відключити, але тоді значно падає продуктивність процесора. L1 кеш працює на частоті процесора, і, в загальному випадку, звернення до нього може проводитися кожен такт. Обсяг зазвичай невеликий - не більше 128 Кбайт.

Другим за швидкодією є L2-cache - кеш другого рівня. Зазвичай він розташований або на кристалі, як і L1, або в безпосередній близькості від ядра, наприклад, в процесорному картриджі (тільки в слотові процесорах). У старих процесорах - набір мікросхем на системній платі. Об'єм L2 кешу від 128 Кбайт до 1-12 Мбайт. У сучасних багатоядерних процесорах кеш другого рівня, перебуваючи на тому ж кристалі, є пам'яттю роздільного користування - при загальному обсязі кеша в nM Мбайт на кожне ядро припадає по nM / nC Мбайта, де nC кількість ядер процесора. Зазвичай латентність L2 кешу, розташованого на кристалі ядра, становить від 8 до 20 тактів ядра. На відміну від L1 кешу, його відключення може не вплинути на продуктивність системи. Однак, у завданнях, пов'язаних з численними зверненнями до обмеженої області пам'яті, наприклад, СУБД, продуктивність може впасти в десятки разів.

Кеш третього рівня найменш швидкодіючий і зазвичай розташований окремо від ядра ЦП, але він може бути дуже значного розміру - більше 32 Мбайт. L3 кеш повільніше попередніх кешів, але все одно значно швидше, ніж оперативна пам'ять. У багатопроцесорних системах знаходиться в загальному користуванні. Застосування кешу третього рівня виправдано в дуже вузькому колі завдань і може не лише не дати збільшення продуктивності, але і привести до загального зниження продуктивності системи.

Наявність кеша другого і третього рівнів найбільш корисно в математичних задачах, наприклад, при обрахунку полігонів, коли обсяг даних менше розміру кешу. У цьому випадку, можна завантажити всі дані відразу в кеш, а потім проводити їх обробку.

5.13. Розширення MMX

Розширення MMX застосовують для організації мультимедійної роботи та опрацювання 2D і 3D-графічних файлів. Головна його ідея полягає у використанні технології ОКБД (див. тему 2).

У систему команд уведено 57 нових команд для одночасного опрацювання кількох одиниць даних. Для роботи застосовано нові типи упакованих 64-бітових даних, а саме:

- упаковані байти (вісім байт);
- упаковані слова (чотири слова);
- упаковані подвійні слова (два подвійних слова);
- почетверенне слово (одне слово).

Ці типи даних можна опрацьовувати у регістрах MMX0–MMX7, які є молодшими бітами стеку 80-бітових регістрів співпроцесора. Збіг регістрів для опрацювання кодів MMX чи FPU є особливістю MMX розширення і потребує від програміста уваги у використанні цих регістрів. Часта зміна використання кодів MMX чи FPU може знизити продуктивність процесора внаслідок необхідності збереження і відновлення стану FPU.

Технологія MMX має ще одну особливість, яка полягає у підтримці команд *арифметики з насиченням*. Суть такої арифметики полягає у тому, що замість переповнення чи антипереповнення фіксується максимально чи мінімально можливе значення величини. Це дуже важливо, наприклад, під час роботи з графічними об'єктами, де відбувається обчислення кольору.

Додаткові команди можна розділити на такі групи:

- арифметичні (+, - , x, +x);
- логічні (І, І-НЕ, АБО, АБО-НЕ);
- порівняння;
- перетворення форматів;
- зсуви;
- пересилання даних;
- очищення ММХ (у слові тегів).

Команди ММХ доступні з будь-якого режиму процесора. Вони не породжують нових винятків і не впливають на стан регістра ознак.

Після запровадження розширення ММХ (для паралельної обробки цілих чисел) виявилося, що існує безліч завдань, які інтенсивно працюють з дійсною арифметикою, причому кількість операцій досить велика, а самі операції прості. Спеціально для подібного роду завдань фірма Intel придумала розширення SSE. Це розширення потрібно для виконання паралельних операцій над упакованими дійсними числами. Формат упакованих дійсних чисел такий: в одній 128-бітній змінній знаходиться 4 числа типу single (32-бітові дійсні числа). Наведемо приклад роботи таких команд:

```
var
   x, y, z: array [0..3] of single;
begin
   x [0]: = ...; x [1]: = ...; x [2]: = ...; x [3]: = ...;
   y [0]: = ...; y [1]: = ...; y [2]: = ...; y [3]: = ...;
   asm
     movups XMM0, x; {в регістрі XMM0 знаходиться масив x}
```

```
        movups XMM1, y; {в регістрі XMM1 знаходиться масив y}
        addps XMM0, XMM1;        {тепер в XMM0 чотири суми, обчислені паралельно}
        movups z, XMM0; {тепер в z знаходиться значення XMM0}
      end;
  end;
```

Після цього фрагмента z [0] = x [0] + y [0], z [1] = x [1] + y [1], z [2] = x [2] + y [2], z [3] = x [3] + y [3]

Список літератури до теми

1. *Д.Брамм, П.Брамм.* Микропроцессор 80386 и его программирование. М., 1990.
2. *Григорьев В.Л.* Микропроцессор i486. Архитектура и программирование. Кн.1. – М., 1993.
3. *Гук М.* Процессоры Intel: от 8086 до PentiumII. – СПб, 1997.
4. *Смит Б.Э., Джонсон М.Т.* Архитектура и программирование микропроцессора Intel 80386. – М., 1992.
5. *Micael Edelhart.* Intel's official quide to 386 Computing. Osborne McGraw-Hill, 1991.
6. 386™SX Microprocessor programmer's reference manual. Intel Corporation, 1989.
7. i486™ Processor programmer's reference manual. Intel Corporation, 1990.

Контрольні запитання до теми

1. Які головні відмінності між 32 та 16-розрядними мікропроцесорами ?
2. За якими адресами (молодшими чи старшими) зберігається байт молодшого порядку у подвійному слові?

3. Скільки є типів сегментів?
4. Чи підтримує процесор одночасну роботу зі сторінками і сегментами?
5. Скільки 8-бітових портів може мати 32-розрядний процесор?
6. Яка відмінність між перериванням і винятком?
7. Як називають виняток, який виявляється й опрацьовується відразу після виконання команди з помилкою?
8. Який регістр зберігається у стеку перед опрацюванням переривання?
9. З якою метою переривання поділяють за пріоритетами?
10. Скільки і які компоненти використовують для формування логічної адреси?
11. Які типи адресних просторів Ви знаєте?
12. Що таке дескриптор?
13. Який біт дескриптора і у який стан (0 чи 1) його потрібно задати, щоб дескриптор був системним?
14. Для чого слугують біти D і P дескриптора?
15. Який розмір сторінки і сегмента 32-розрядного процесора?
16. Скільки рівнів захисту має 32-розрядний процесор?
17. Як можна змінити рівень привілеїв?
18. Як працює механізм посторінкової організації пам'яті?
19. Як влаштована кеш-пам'ять процесора?
20. Як працює мехамізм LRU?
21. Що означає віртуальний 8086-режим роботи 32-розрядного процесора?
22. З якою метою у процесорі запроваджено розширення MMX?