МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВІННИЦЬКИЙ ДЕРЖАВНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

А.М. Пєтух, В.П. Майданюк, Є. Л. Ольшевський

АРХІТЕКТУРА І ПРОГРАМУВАННЯ 32-Х РОЗРЯДНИХ МІКРОПРОЦЕСОРІВ

Затверджено на засіданні Ученої ради Вінницького державного технічного університету як навчальний посібник для студентів спеціальності "Програмне забезпечення автоматизованих систем". Протокол № 5 від 30 грудня 1999 р.

УДК 681.3.06
Архітектура і програмування 32-х розрядних мікропроцесорів: Навчальний посібник/ А.М. Пєтух, В.П. Майданюк, Є.Л. Ольшевський - Вінниця: ВДТУ, 2000. - 130 с. Укр. мовою/

У навчальному посібнику розглянуті процесори сімейства x86, а саме їх 32-розрядні нащадки (починаючи з 80386), приведено опис їхньої програмної моделі, режими роботи, система команд, відмінності. Відомості наведені в начальному посібнику можуть використовуватись як при самостійній роботі студентів, так і при виконанні курсових і дипломних робіт.

Навчальний посібник призначений для студентів спеціальності "Програмне забезпечення автоматизованих систем" та інших споріднених спеціальностей

Бібліогр. 10 назв, іл , табл.

Рецензенти:
д.т.н., проф. В.П. Кожемяко
к.т.н., доц. В.Є. Качуровський
к.т.н., доц. В.І. Мєсюра

Вступ

Безсумнівно, що сьогодні найбільшою популярністю користуються процесори сімейства x86. Але водночас лише деякі користувачі можуть коротенько пригадати їхню історію, відмінності а також охарактеризувати сучасні моделі процесорів.

Даний навчальний посібник присвячений процесорам сімейства x86, а саме їх 32-розрядним нащадкам (починаючи з 80386), опису їхньої програмної моделі, режимів роботи, відмінностям.

У першому розділі приведений стислий історичний екскурс в історію МП сімейства x86, у другому розділі характеризується загальна модель 32-розрядного x86 - сумісного процесора, у третьому докладно описана робота в захищеному режимі, четвертий розділ містить опис додаткових можливостей цих процесорів,. У додатках приведений опис команд процесора та призначення виводів деяких процесорів.

Наприкінці кожної глави подаються контрольні питання. Після ознайомлення з темою рекомендується відповісти на контрольні питання. Матеріал рекомендується вивчати послідовно, проте ті, хто вже знайомий з темою, може, відповівши на контрольні питання, перейти до наступної теми.

На думку авторів наведені в навчальному посібнику відомості можуть бути використані студентами, а також магістрами та аспірантами в їх самостійній роботі.

Матеріал навчального посібника підготували А.М. Петух - вступ, розділ 1; В.П. Майданюк - розділи 1,2,3, контрольні питання до всіх розділів; Є. Ольшевський – розділи 1, 4.

Автори висловлюють щиру подяку рецензентам, доброзичливе ставлення та поради яких сприяли покращенню змісту та структури навчального посібника, а також інженеру-програмісту Кашпруку О. А.

РОЗДІЛ 1. ЕВОЛЮЦІЯ ПРОЦЕСОРІВ ФІРМИ INTEL СІМЕЙСТВА X86

У цій главі розглянуті процесори від 8086 до Pentium III. Оскільки практичний інтерес до процесорів від 8086 до 80386 згасає, вони розглянуті менш докладно. Процесори i486 розглядаються більш докладно, оскільки їхньої потужності вистачає для багатьох сучасних додатків. Крім цього, приведений опис мікропроцессорів виробників, відмінних від Intel

1.1 Стислий історичний екскурс

Історія мікропроцесорів почалася в 1971 році, коли фірма Intel випустила перший мікропроцесор **i4004**. Він мав розрядність даних 4 біти, спроможність адресувати 640 байт пам'яті, тактову частоту 108 кГц і продуктивність 0,06 MIPS. Такий процесор уже міг працювати в якості обчислювального ядра калькулятора. Він містив 2300 транзисторів і був виконаний за технологією з роздільною здатністю 10 мкм. Через рік з'явився його 8-розрядний «родич» - **i8008, що** міг адресувати вже 16 Кбайт пам'яті.

У 1974 році з'явився *8-розрядний* процесор **i8080, що** став дуже популярним пристроєм. Він мав частоту 2 МГц, адресував 64 Кбайт пам'яті і містив 6000 транзисторів завдяки технології 6 мкм. Процесор вимагав трьох джерел живлення (+5, +12 і -5 V) і складної двотактної синхронізації. На цьому процесорі будувалися різноманітні термінали, контролери і навіть перший персональний комп'ютер - Altair. У нашій країні запізнілим «эхом» 8080 стали процесори К580ИК80 і КР580ВМ80, на базі яких на початку і в середині 80-х років було розроблено багато «саморобних» персональних комп'ютерів (ПК) [1,2].

Наступним етапом став процесор **i8085** (5 МГц, 0,37 MIPS, 6500 транзисторів, технологія 3 мкм, вітчизняний аналог К1821 ВМ85). Він зберіг популярну регістрову архітектуру 8080 і програмну сумісність, але в нього додали порт послідовного інтерфейса, скасували спеціальні інтегральні схеми підтримки (тактовий генератор і системний контролер) і дещо змінили зовнішній інтерфейс. Головним подарунком розроблювачам апаратури став перехід на єдину напругу живлення - +5 В.

Процесор **Z80** фірми Zilog являвся варіацією на тему 8080 і 8085. Зберігши програмну сумісність із 8080, у нього ввели додаткові регістри, що дозволило істотно підвищити продуктивність. Результат виявився вражаючим - ще недавно популярні комп'ютери Sinclair, побудовані на Z80, демонстрували на іграх графіку, що не поступається PC на 16-розрядному процесорі 80286.

Перший *16-розрядний* процесор **i8086** (вітчизняний аналог К1810 ВМ86 [3]) фірма Intel випустила в 1978 році. Частота - 5 МГц, продуктивність - 0,33 MIPS, але інструкції вже з 16-бітовими операндами (пізніше з'явилися процесори з тактовими частотами 8 і 10 МГц). Технологія 3 мкм, 29 000 транзисторів, адресний простір пам'яті - 1 Мбайт. Регістрова архітектура і система команд істотно відрізнялася від 8080, але, природно, прослідкуються загальні ідеї. Через рік з'явився i8088 той же процесор, але з 8-розрядною шиною даних. З нього почалася історія IBM PC, нерозривно пов'язана з усім подальшим розвитком процесорів Intel. Масове поширення і відкритість архітектури IBM PC призвели до лавиноподібних темпів появи нового програмного забезпечення, розроблюваного великими, середніми і дрібними фірмами, а також ентузіастами-одинаками. Технічний прогрес і тоді, і зараз був не мислимий без розвитку процесорів, але з урахуванням величезного обсягу програмного забезпечення, що вже існує для PC, уже тоді виник принцип оберненої програмної сумісності - старі програми повинні працювати на нових процесорах. Таким чином, усі нововведення в архітектурі наступних процесорів повинні були пристроюватися до ядра, що існує. На процесорах відбивалися й особливості архітектури PC. Взяти, наприклад, вектори переривань. Фірма Intel зарезервувала перші 32 вектори «для службового користування», проте на них наклались переривання BIOS PC. У результаті з'явився додатковий засіб опрацювання винятків сопроцессора, застосовуваний у старших моделях процесорів.

Процесор **i80286**, що знаменує наступний етап розвитку архітектури, з'явився тільки в 1982 році. Він уже мав 134 000 транзисторів (технологія 1,5 мкм) і адресував до 16 Мбайт фізичної пам'яті. Його принципові нововведення - захищений режим і віртуальна пам'ять розміром до 1 Гбайт - не знайшли масового застосування; процесор здебільшого використовувався як дуже швидкий процесор 8086 [4].

Народження 32-розрядних процесорів ознаменувалося в 1985 році моделлю **i80386** (275 000 транзисторів, 1,5 мкм). Розрядність шини даних (як і внутрішніх регістрів) досягла 32 біта, що дає можливість адресувати фізичну пам'ять ємністю 4 Гбайт. З'явилися нові регістри, нові 32-бітові операції, істотно доопрацьований захищений режим, були введені режим V86 і сторінкове керування пам'яттю. Процесор знайшов широке застосування в PC; на його благодатному грунті стала розростатися операційна система Microsoft Windows із додатками.

Історія процесора 80386 повторила долю 8086/8088: першу модель з 32-розрядною шиною даних (згодом названу **386DX**) змінив **386SX** із 16-розрядною шиною даних. Він досить легко

вписувався в архітектуру PC AT, яка раніше базувалася на процесорі 80286.

Процесор **Intel486DX** [5,6] з'явився в 1989 році. Транзисторів - 1,2 млн., технологія 1 мкм. Від процесора 80386 істотно відрізняється розміщенням на кристалі первинного кеша й умонтованого арифметичного співпроцесора – FPU (Floating Point Unit) (попередні процесори використовували зовнішні співпроцесори x87). Крім того для підвищення продуктивності в цьому CISC-процесорі (як і в наступних) застосоване RISC-ядро. CISC – Complete Instruction Set Computer – процесори з повним набором інструкцій, RISC – Reduced Instruction Set Computer – процесори з скороченою системою команд. Далі з'явилися його різновиди, що відрізняються наявністю або відсутністю сопроцесора, застосуванням внутрішнього множення частоти, способами кешування і т. ін. Тоді ж Intel зайнялася енергозбереженням, що відбилося й у лінії 386- з'явився процесор **Intel386SL**.

У 1993 році з'явилися перші процесори **Pentium** [7,8] із частотою 60 і 66 МГц - 32-розрядні процесори з 64-розрядною шиною даних. Транзисторів - 3,1 млн, технологія 0,8 мкм, напруга живлення - 5 В. Його від 486 принципово відрізняє суперскалярна архітектура - спроможністю за один такт випускати з конвеєрів до двох інструкцій (що, звичайно, не означає можливості проходження інструкції через процесор за півтакта або один такт). Інтерес до процесора з боку виробників і покупців PC стримувався його дуже високою ціною. Крім того, виник скандал із помилкою в співпроцесорі. Хоча фірма Intel математично обгрунтувала невисоку можливість її появи (раз у декілька років), вона все-таки пішла на безкоштовну заміну вже проданих процесорів на нові, полагоджені.

Процесори Pentium із частотою 75, 90 і 100 МГц, що з'явилися в 1994 рік представляли друге покоління процесорів Pentium. При майже тому ж числі транзисторів вони виконувалися за технологією 0,6 мкм, що дозволило знизити потужність споживання. Від першого покоління вони відрізнялися внутрішнім множенням частоти, підтримкою мультипроцесорних конфігурацій і іншим типом корпуса. З'явилися версії (75 МГц у мініатюрному корпусі) для мобільних застосувань (блокнотних PC). Процесори Pentium другого покоління стали дуже популярними в PC. У 1995 році були випущені процесори на 120 і 133 МГц, виконані вже за технологією 0,35 мкм (перші процесори на 120 МГц робилися за технологією 0,6 мкм). 1996-й називають роком Pentium-з'явилися процесори на 150,166 і 200 МГц, і Pentium став рядовим процесором у масових PC.

Паралельно с Pentium розвивався і **Pentium Pro**, що відрізнявся «динамічним виконанням», спрямованим на збільшення числа паралельно виконуваних інструкцій. Крім того, у його корпусі

розмістили вторинний кеш, що працює на частоті ядра, об'ємом від 256 Кбайт. Процесор містить 5,5 млн. транзисторів ядра і 15,5 млн. транзисторів для вторинного кеша об'ємом 256 Кбайт.

На початку 1997 року був випущений **Pentium MMX**. Цей процесор відрізняється від Pentium подвоєним об'ємом первинного кеша і наявністю деяких елементів архітектури, запозичених у Pentium Pro, а також наявністю технології MMX (MultiMedia eXtensions) – розширення системи команд процесора для мультимедійних додатків. Ці процесори мають 4,5 млн. транзисторів.

У травні 1997 року був випущений **Pentium II**, що являє собою трохи урізаний варіант Pentium Pro із більш високою внутрішньою тактовою частотою і з підтримкою MMX. Вторинний кеш цих процесорів розміром від 512 Кб до 2 Мб працює на половині частоти ядра.

Дешевим варіантом цього процесора став **Celeron**, молодші моделі якого взагалі не мають вторинного кеша, а старші мають 128 Кбайт, при цьому кеш розташований на кристалі ядра і працює на частоті ядра.

У 1998 році була представлена технологія 3D Now! фірми AMD, яка реалізована у процесорі AMD K6-2. Функції технології 3D Now! ті ж, що і в MMX, але є можливість виконання розрахунків з плаваючою точкою[8].

На початку 1999 року фірмою INTEL був представлений процесор **Pentium III**, що відрізняється від Pentium II підтримкою інструкцій ISSE а також наявністю власного унікального серійного номера. Нова технологія ISSE (Internet Streaming SIMD Extension) застосовується замість MMX і на відмінність від MMX дозволяє виконувати розрахунки з числами з плаваючою точкою [9].

1.2 Процесори i8088/ i8086/ i80186/ i80286

Це перші процесори із сімейства x86, що призначалися для виконання 8/16 розрядних операцій :

- Логічних
- Арифметичних
- Ланцюгових операцій
- Введення/виведення

Процесори мають 20 бітову шину адреси, що дозволяє адресувати до 1 Мбайта пам'яті. Адресація пам'яті - сегментна. Шина даних у **i8086** - 16 розрядна, у **i8088-** 8 розрядна. Для прискорення виконання операцій з плаваючою точкою сумісно з цими процесороми використовується арифметичний співпроцесор i8087.

i80186 - це подальший розвиток i8086, проте поширення одержав в основному в різноманітних контролерах.

Процесор **i80286** ознаменував новий етап у розвитку сімейства. У ньому реалізована 24-х бітна шина адреси, що дозволило використовувати до 16 Мбайт пам'яті. Крім цього введений захищений режим процесора. Внутрішня архітектура – 16-и розрядна. Цей процесор разом із Intel випускала і фірма AMD. Для нього розроблений математичний співпроцесор **i80287**.

1.3 Процесор i80386

Цей процесор символізує черговий ривок вперед, зроблений фірмою Intel. І навіть тепер багато популярних операційних систем, у тому числі і Windows 95, можуть виконуватися на цьому процесорі. Головні відмінні риси МП 80386 це режими адресації до пам'яті і об’єм пам'яті, до якої можна адресуватися. МП 80386 може адресувати до 4 Гбайт пам'яті і використовує сторінкову адресацію пам'яті. Внутрішня архітектура - 32 розрядна. Крім захищеного режиму 80286 він має також 32 бітовий захищений режим і режим V86.

Починаючи з цього процесора, починають з'являтися клони - DX і SX. SX і DX позначає "полегшену" і повну версію того самого процесора. Для 386-го процесора варіант SX був розроблений з 16-розрядним інтерфейсом, що дозволяло заощаджувати на обв'язуванні і встановлювати пам'ять по два SIMM, а не по чотири, як для DX. При роботі з 16-розрядними додатками 386SX майже не відстає від 386DX на тій ж частоті, проте на 32-розрядних додатках він працює значно повільніше через поділення кожного 32-розрядного запиту до пам'яті на два 16-розрядних. В дійсності більшість комп'ютерів з процесором 386DX працюють швидше комп'ютерів з процесором 386SX навіть на 16-розрядних додатках - завдяки тому, що на платах з 386DX частіше усього встановлений апаратний кеш, котрого немає на більшості плат з SX. Внутрішня архітектура 386SX - цілком 32-розрядна, і програмно виявити різницю між SX і DX без запиту коду процесора, виміру швидкості роботи магістралі або розміру буфера попередньої виборки в загальному випадку неможливо.

Цей процесор не містить вмонтованого арифметичного сопроцесора, тому розроблений сопроцесор i80387, причому він різниться для DX і SX.

1.4 Сімейство 80486

Це сімейство, у зв'язку з зростанням популярності процесорів x86, містить значне число моделей.

Вони на 100% програмно сумісні з мікропроцессорами 386 DX і SX і містять один мільйон транзисторів об'єднаної кеш-пам’яті (надшвидкої оперативної пам'яті), разом з апаратурою для виконання операцій з плаваючою точкою і керуванням пам'яттю на одній мікросхемі, проте підтримують програмну сумісність з попередніми моделями сімейства процесорів архітектури x86 [2].

Операції, які часто використовуються виконуються за один цикл, що можна порівняти зі швидкістю виконання RISC-команд. Вісьми кілобайтний уніфікований кеш для коду і даних, сполучений із шиною пакетного обміну даними зі швидкістю 80/106 Мбайт/сек при частоті 25/33 МГц гарантують високу продуктивність системи навіть із недорогими дисками (DRAM).

Нові можливості розширюють багатозадачність систем. Нові операції збільшують швидкість роботи із семафорами в пам'яті. Устаткування на мікросхемі гарантує безконфліктну роботу кеш-пам’яті і підтримує засоби для реалізації багаторівневого кешування. Вмонтована система тестування перевіряє мікросхемну логіку, кеш-пам’ять і мікросхемне посторінкове перетворення адрес пам'яті. Можливості налагодження містять у собі установку пасток контрольних точок у виконуваному коді і при доступі до даних.

Основні характеристики процесора i486 такі:

- Повна програмна сумісність із ЦПУ 386 DX, 386 SX, процесорами 80286, 8086 і 8088;
- Модуль виконання команд розроблений так, щоб виконувати операції, що часто зустрічаються, за один цикл;
- 32-розрядний процесор для виконання арифметичних і логічних операцій;
- Вмонтований модуль опрацювання арифметичних операцій із плаваючою точкою для підтримки 32, 64, і 80-розрядних форматів, заданих у відповідність із стандартом IEEE 754 (объектно сумісному з 387 DX і 387 SX арифметичними співпроцесорами);
- Внутрішня 8 Кбайтна кеш-пам’ять, що забезпечує швидкий доступ до часто використовуваних даних і операцій;
- Сигнали керування шиною для підтримки безконфліктності кеша в багатозадачних системах;
- Сегментація, одна з форм керування пам'яттю для створення незалежних, захищених адресних просторів;
- Посторінкова розбивка, одна з форм керування пам'яттю, що забезпечує доступ до структур даних, більших, ніж доступний об’єм пам'яті, шляхом збереження даних частково в пам'яті, частково на диску;
- Оператори, що дозволяють перезапуск програми після винятку

(необхідні для підтримки посторінкового доступу до віртуальної пам'яті);
- Конвеєрне виконання команд, що перекривається за часом з інтепретацією інших команд;
- Регістри налагодження для апаратної підтримки контрольних точок у командах і даних.

Для процесорів сімейства i486 SX позначає варіант без вмонтованого співпроцесора. Ранні моделі являли собою просто відбракування від DX із несправним співпроцесором (співпроцесор у них був заблокований). Для установки такого процесора замість DX потрібно було переналагоджувати системну плату. Більш пізні версії випускалися самостійно, і могли установлюватися замість DX без зміни настроювань плати. Крім відсутності співпроцесора і ідентифікаційних кодів, моделі SX також нічим не відрізняються від відповідних моделей DX, і їх ідентифікація неможлива. SX2, DX2 і DX4 - варіанти відповідних процесорів із внутрішнім подвоєнням або потроєнням частоти. Наприклад, апаратне настроювання плати для DX2-66 виконується, як для DX33, і на вхід подається частота 33 МГц, проте в програмних настроюваннях може знадобитися збільшення затримок при звертанні до пам'яті для компенсації збільшення швидкості роботи процесора. Всі внутрішні операції в процесорах виконуються відповідно в два і три рази швидше, проте обмін по зовнішній магістралі визначається зовнішньою тактовою частотою. За рахунок цього DX4-100 працює втроє швидше DX33 тільки на тих ділянках програм, що цілком поміщаються в його внутрішній кеш, на великих фрагментах це відношення може впасти до двох з половиною і менше.

До сімейства 80486 належать також такі процесори як AMD 5x86 (133 і 160 Мгц), Cyrix 5x86 (100 і 133Мгц відповідно), а також такий процесор як UMC U5. Останній відрізняється насамперед оптимізованим мікрокодом, за рахунок чого команди, які часто використовуються, виконуються за менше число тактів, у порівнянні з процесорами Intel, AMD, Cyrix та іншими. Процесори U5 не мають внутрішнього множення частоти, а результати вимірювання швидкодії ідентичні процесорам з тактовою частотою 65 МГц, які одержують за допомогою деяких вимірювальних програм, утворюються тому, що для визначення частоти програмі необхідно правильно розпізнати процесор - точніше, число тактів, за який він виконає тестову послідовність, а більшість поширених програм не вміють правильно розпізнавати U5.

Cyrix 5x86 (m1sc) і AMD 5x86 - процесори, сумісні по виводах із Intel P24D (i486DX4-100 останніх моделей), містять елементи архітектури P5 (Pentium): 16-кілобайтовий внутрішній кеш із

відкладеним записом, загальний для команд і даних, передбачення переходів, оптимізація виконання команд. Cyrix 5x86 має 64-розрядну внутрішню шину даних і систему розпаралелювання операцій. Процесори Cyrix 5x86 можуть працювати в режимах подвоєння і потроєння частоти (є також можливість програмного відключення множення), а процесори AMD 5x86 - у режимах потроєння й учетверення. Cyrix 5x86 на частоті 120 МГц по тестах WinStone і WinBench приблизно прирівнюється до Intel P5-90, а AMD 5x86 на частоті 133 МГц - до Intel P5-75. За іншими тестами результати можуть значно різнитися в обидві сторони за рахунок того, що внутрішня швидкість виконання деяких послідовностей команд у цих процесорів вище прирівняних до них P5, проте швидкість обміну з зовнішнім кешем і пам'яттю в них істотно нижче. Крім цього, P5 має значно більш потужний співпроцесор, і по швидкості арифметики з плаваючою точкою процесори 5x86 сильно від нього відстають.

Крім цього клони 80486 випускали також фірми Texas Instruments (TI 80486) і IBM.

1.5 Сімейство P54C (або ж сімейство Pentium).

Процесори *Pentium фірми* Intel представляють п'яте покоління процесорів сімейства x86. За базовою регістровою архітектурою і системою команд вони сумісні із вищеописаними 32-розрядними процесорами, але мають 64-бітову шину даних, завдяки чому їх іноді помилково називають 64-розрядними. У порівнянні з попередніми поколіннями процесори Pentium мають такі якісні відмінності:

- Суперскалярна архітектура: процесор має два конвеєри обробки, що працюють паралельно (U-конвеєр із повним набором інструкцій і V-конвеєр із дещо обмеженим набором), завдяки чому він спроможний одночасно виконувати дві інструкції. Проте переваги цієї архітектури цілком реалізуються тільки при виконанні спеціальним чином скомпільованих програм. Технологія динамічного передбачення переходів разом із виділеним внутрішнім кешем команд обсягом 8 Кбайт, забезпечують максимальну загрузку конвеєрів.
- Внутрішній (Level 1) кеш даних об’ємом 8 Кбайт працює з відкладеним (до звільнення зовнішньої шини) записом і настроюється на режим наскрізного або оберненого запису, підтримуючи протокол MESI,
- Зовнішня шина даних з метою підвищення продуктивності виконана 64-розрядною, що потребує відповідної організації пам'яті.
- Вмонтований арифметичний сопроцесор за рахунок архітектурних поліпшень (конвеєризації) у 2-10 разів перевершує FPU-486 за

продуктивністю.

- Введено декілька нових інструкцій, у тому числі розпізнавання сімейства і моделі CPU (CPUID).
- Застосовано виявлення помилок внутрішніх пристроїв (внутрішній контроль паритету) і зовнішнього інтерфейсу шини, контролюється паритет шини адреси.
- З'явилася можливість побудови двопроцесорної системи з надлишковим контролем функціональності.
- Реалізовано інтерфейс побудови двопроцесорних систем із симетричною архітектурою (починаючи з другого покоління Pentium).
- Введено засоби керування енергоживленням.
- Застосована конвеєрна адресація шинних циклів.
- Скорочено час (число тактів) виконання інструкцій.
- Введено трасування переходів і моніторинг продуктивності.
- Розширено можливості віртуального режиму - введена віртуалізація прапорця переривань.
- Введена можливість оперування сторінками розміром 4 Мбайт у режимі сторінкової переадресації (Paging). Всі Pentium-процесори мають засоби режиму системного керування SMM (System Menegment Mode), можливості якого розширювалися в міру появи нових моделей.

Це сімейство містить у собі такі процесори як Cyrix 6x86 (M1) і AMD 5x86 (SSA/5, K5) а також Intel Pentium. Вони сумісні по виводах із Intel P5. За рахунок більш кращої внутрішньої оптимізації ці процесори з цілочислової арифметики дещо швидші Intel P5 на тих же частотах, проте - відстають при виконанні операцій з плаваючою точкою.

Тут також слід зазначити той факт, що процесори Intel Pentium 60-75 Мгц не сумісні по виводах із процесорами старших моделей.

Основна відмінність цього сімейства в тому, що в ньому є 64-розрядна магістраль, що значно прискорює обмін із зовнішнім кешем і пам'яттю. Суперскалярна архітектура: один виконавчий пристрій замінений на два - U і V, кажний - із своїм власним конвеєром; обидва паралельно ведуть вибірку, розшифровування і виконання команд. Пристрій U є основним і може виконувати всі команди, а пристрій V - допоміжним і виконує тільки найбільш типові команди. Внутрішній кеш розділений на кеш команд і кеш даних. Є також система передбачення переходів шляхом опереджувального перегляду, що дозволяє у випадку вірного передбачення виконати перехід за один такт. Поліпшений у порівнянні з i486 математичний співпроцесор.

Крім цього, варто мати на увазі, що при маркуванні клонів використовується так званий P-rating - зразкова відповідність продуктивності процесора на додатках загального характера (поширені ОС, типові офісні програми, ігри середньої складності) процесору Intel Pentium із зазначеною тактовою частотою. Наприклад , AMD 5x86-133 приблизно відповідає Pentium-75 і має позначення -P75.

1.6 Сімейство P55C (або сімейство Pentium MMX).

У це сімейство будемо включати тільки ті з процесорів, що сумісні з роз'ємом Socket 7. Тоді в це сімейство потрапляють такі процесора як:

- Pentium MMX (166,200,233,266 МГц)
- AMD K6 (166,200,233,266,300 МГц)
- Cyrix 6x86MX (PR166+,PR200+,PR233+,PR266+), Cyrix M2(233,266,300,333)
- IDT Winchip, Winchip 2
- AMD K6-2(200,233,266,300,333,350,400),K6-3(400,450,500)

Відмінною рисою цих процесорів є підтримка команд MMX-технології (MultiMedia eXtension) - додаткових можливостей, орієнтованих на обробку цифрового зображення і звука, анонсованих Intel у процесорах P55C. MMX містить у собі 57 нових команд і вісім додаткових регістрів, призначених для обробки звукових і відеосигналів; команди можуть використовуватися в режимі SIMD (Single Instruction, Multiply Data - одна команда, багато даних - ОКМД), коли одною командою одночасно обробляється декілька елементів даних. Процесори з MMX-технологією мають також подвоєний (32 Кбайти) об'єм внутрішнього (L1) кеша.

Розширення MMX реалізовані у виді додаткового режиму, у який процесор може перемикатися зі звичайного режиму роботи. У режимі MMX набір регістрів співпроцесора (FPU) використовується для збереження даних MMX-команд - це гарантує сумісність з операційними системами, що не підтримують MMX напрямку. Проте таке суміщення може знизити ефективність роботи у випадку попеременного використання звичайних обчислень із плаваючою точкою і роботи в режимі MMX.

Використання MMX дозволить перенести основне навантаження з обробки зображення і звука на центральний процесор, залишивши відео- і звуковим адаптерам тільки перетворення аналог - цифра. Інакше кажучи, із зростанням потужності центральних процесорів стає вигідніше виконувати на них ту роботу, що декілька років тому була віддана периферійним

відео- і звуковим процесорам через недостатню потужність центрального; зараз знову відбувається повернення до централізованої обробки.

1.7 Процесори класу P55C інших виробників

Процесори AMD K6-2 (K6 3D),K6-3 відрізняються підтримкою технології 3DNow! Технологія 3DNow! , запропонована AMD у своєму новому процесорі K6-2 (кодове ім'я було K6 3D), являє собою розвиток технології MMX. MMX - це додаткові 57 інструкцій процесора і 8 додаткових регістрів, що покликані збільшити продуктивність мультимедійних додатків. Якщо програма використовує ці можливості, то це вносить чималий внесок у швидкість її виконання. Технічні відомості і нові рішення, застосовані в процесорі AMD-K6 MMX Enhanced:

- Продуктивність шостого покоління, що суперничає з Pentium II
- Висока продуктивність при роботі з операційними системами Windows 95 і Windows NT
- Новітня шестиступенева конвеєрна суперскалярна мікроархітектура RISC86®.
- Сім паралельних виконавчих блоків
- Рівнобіжні складні декодери інструкцій x86 у RISC86
- Новітній дворівневий алгоритм обчислення передбачення переходів
- Упереджене виконання команд
- Цілком асинхронне виконання команд
- Перейменування регістрів і передача даних
- Декілька великих блоків (64-Кбайт) кеш-пам’яті L1
- Кеш-пам’ять команд ємністю 32 Кбайт, а також кеш-пам’ять перекодування
- Двопортовий кеш даних об’ємом 32 Кбайт з оберненим записом
- Протокол MESI з автоматичним відновленням кеша
- Високопродуктивний пристрій для обчислень із плаваючою точкою (FPU), сумісний зі стандартом IEEE 754
- Високопродуктивна реалізація команд мультимедійних розширень (MMX(TM))
- Цілком сумісний режим керування системою (SMM)
- Сумісність із стандартом Socket 7, що дозволяє виробникам і дистриб’юторам ПК використовувати наявну інфраструктуру і конструктивні рішення. Socket-7-процесор, сумісний по виводах із P55C (Pentium MMX). Кристал виконаний за технологією 0.25 мкм і працює при напрузі ядра 2.2 В. Чіп підтримує

системну шину 100 МГц і з нею показує кращі результати, тобто для його використання рекомендується застосування архітектури Super 7.

Тепер розглянемо процесор AMD K6-3 [9] . Його технічні дані такі:

- Чіп, виготовлений за технологією 0.25 мкм;
- Ядро CXT, що являє собою звичайне ядро K6-2 із можливістю пакетного запису;

- Працює в Socket-7-системних платах, але потребує відновлення BIOS;
- Кеш першого рівня - 64 Кбайти, по 32 Кбайти на код і дані;
- Має вмонтований кеш другого рівня обсягом 256 Кбайт;
- Кеш материнської плати працює як кеш третього рівня;
- Напруга живлення 2.3-2.5 В;
- Набір із 21 SIMD-команди 3DNow! Є 2 конвеєри, що оперують із двома парами речовинних чисел одинарної точності;
- Частоти - 350, 400, 450 і 475 МГц. Системна шина 100 МГц (для моделі 475 МГц - 95 МГц). Можлива робота і на 66 МГц системній шині;
- 3DNow! підтримується бібліотекою графіки Windows–95 DirectX 6.0 і вище.

Як очевидно зі специфікації, AMD K6-III - це AMD K6-2 плюс 256 Кбайт кеша другого рівня, що інтегрований в ядро і працює на його частоті.

Розглянемо тепер процесори Cyrix 6x86MX і MII. Нова в процесорі MII тільки назва. Насправді M II-300 являє собою Cyrix 6x86MX із внутрішньою частотою 233 MHz. Чому тоді 300 і чому нова назва? Справа в тому, що Cyrix використовує для маркування своїх процесорів так званий Performance Rating (PR), що являє собою тактову частоту порівнянних за продуктивністю (на тесті Ziff-Davis Winstone) процесорів. Компанія спеціально підкреслює, що це не Pentium Rating, хоча 6x86MX прирівнювався до Pentium MMX, тобто Cyrix 6x86MX PR200 прирівнюється за продуктивністю до Pentium MMX 200. Справа в тому, що тепер Cyrix порівнює свій процесор не з Pentium MMX, а з Pentium II, звідси і нова назва.

На деяких бізнес-додатках M II-300 на 25% перевершує Intel Celeron 266 MHz при однаковій конфігурації комп'ютерів і має однакову продуктивність з Pentium II 300 MHz.

Ці результати досягнуті завдяки тому, що M II включає 64 КВ уніфікованої кеш-пам'яті L1 (загальної для команд і даних), проти 16+16 КВ у процесорів Intel, а розмір кеш-пам'яті для бізнесів-додатків є критичним. У іншому ж, як випливає з табл.1.1, в архітектурі M II присутні всі переваги процесорів Intel, включаючи перейменування регістрів, передбачення переходів і т. і. (табл. 1.1).

Таблиця 1. 1- Порівняльні характеристики процесорів **M II та Pentium II**

	Можливості M II	**Pentium II**
Підтримка MMX	I	I
Суперскалярність	I	I
Суперконвеєрність	I	I
Перейменування регістрів	I	I
Незалежне завантаження даних	I	I
Багаторівневе передбачення переходів	I	I
Умовне виконання	I	I
Позачергове виконання	I	I
80-бітний арифметичний сопроцесор	I	I
Кеш L1	64 КВ загальний	16 + 16 КВ

1.8 Процесори Intel шостого покоління (P6)

Відлік шостого покоління процесорів почався з процесора Pentium Pro, випущеного в 1995 році. Зараз до цього покоління відносяться Pentium II (1997 р.), Celeron, Xeon (1998 р.) і, нарешті, Pentium III (1999 р.). Основні характеристики цих процесорів наведені в табл. 1.2. Від попереднього покоління ці процесори головним чином відрізняє застосування "динамічного виконання" (зміни порядку виконання інструкцій) і архітектура подвійної незалежної шини. Тут вторинному кешу, введеному в процесор (але не у всі моделі), виділяється окрема високошвидкісна магістраль. У ході еволюції покоління до системи команд Pentium Pro, розширеної щодо Pentium із метою скорочення умовних переходів, було додане розширення MMX - так з'явилося Pentium II. Тепер ідею MMX - одночасне виконання однієї інструкції над групою операндів - поширили і на інструкції з плаваючою точкою: SSE (Streaming SIMD Extensions) - основний козир Pentium III. Правда, дещо раніше те ж саме (але в меншому обсязі) було зроблено фірмою AMD - розширення 3DNow! було реалізовано вже в процесорах K6-2 для Socet 7.

Процесори *Pentium Pro* випускалися в модифікованих корпусах SPGA (Staggered Pin Grid Array) із матрицею штиркових виводів, частина з яких розташована в шаховому порядку. У однім корпусі (мікросхемі) встановлено 2 кристали - ядро процесора і вторинний кеш власного (Intel'овского) виготовлення. Цей кеш працював на частоті ядра процесора, що за всю історію Pentium Pro із початкових 150 МГц піднявся усього тільки до 200 МГц. Об’єм кеша в різних модифікаціях був від 512 Кбайт до 2 Мбайт, для підвищення надійності застосовувався ECC-контроль. Для цих процесорів призначений *сокет 8* із 387 виводами. Інтерфейс дозволяє безпосередньо об'єднувати до 4 процесорів для симетричної мультипроцесорної обробки (SMP). Можливо і парне вмикання процесорів для функціонально-надлишкового контролю (FRC), при якому один процесор тільки перевіряє дії іншого.

Процесори *Pentium II* сполучають архітектуру Pentium Pro із технологією MMX. У порівнянні з Pentium Pro розмір первинного кеша (16+16 Кбайт) подвоєний, розмір вторинного кеша варіюється від 0 до 2 Мбайт. У процесорі використовується нова технологія виготовлення корпусів - картридж із друкарським крайовим рознімання, на який виведена системна шина (Single Edge Contact Cartridge - SECC). На картриджі розміром 14 x 6,2 x 1,6 см установлена мікросхема ядра процесора (CPU Core), декілька мікросхем, що реалізують вторинний кеш, і допоміжні дискретні елементи (резистори і конденсатори). Зняття вторинного кеша з мікросхеми процесора дозволяє використовувати для кеш-пам’яті і пам'яті тегів мікросхеми сторонніх виробників, що спеціалізуються на випуску надшвидкодіючої пам'яті. Об’єм вторинного кеша визначається ємністю і числом встановлених мікросхем пам'яті. У той же час зберігається незалежність шини вторинної кеш-пам’яті, що тісно пов'язана з ядром процесора власною локальною шиною.

Перші процесори Pentium II (до випуску вони мали кодову назву *Klamath*), що з'явилися навесні 1997 року, нараховували біля 7,5 млн транзисторів тільки в процесорному ядрі і виконувалися за технологією 0,35 мкм. Вони мали тактові частоти ядра 233, 266 і 300 МГц при частоті системної шини 66,6 МГц. При цьому вторинний кеш працював на половинній частоті ядра і кешував тільки перші 512 Мбайт простору пам'яті. Для цих процесорів був розроблений *слот 1*, який за складом сигналів дуже нагадує сокет 8 для Pentium Pro. Проте слот 1 дозволяє об'єднувати лише пару процесорів для реалізації симетричної мультипроцесорної системи або системи з надлишковим контролем функціональності (FRC). Так що цей

процесор являє собою більш швидкий Pentium Pro із підтримкою MMX, але з урізаною підтримкою мультипроцесування.

Наступна модифікація Pentium II, що мала кодову назву *Deshutes*, з'явилася в 1998 році і виконувалася вже за технологією 0,25 мкм. Це дозволило підняти тактову частоту (чим менші елементи, тим менше вони розсіюють потужність, що особливо критично на високих частотах). Процесор на 333 МГц має частоту шини 66,6 МГц, а процесори на 350 МГц і більше вже мають частоту системної шини 100 МГц. Для роботи на такій частоті ефективна оперативна пам'ять на мікросхемах SDRAM (синхронна динамічна пам'ять), у якої в середині пакетного циклу дані передаються в кожному такті. Ці процесори також встановлюються в слот 1 (не більше двох в системі). Починаючи з процесорів 350 МГц об’єм пам'яті, яка кешується на L2, збільшився до 4 Гбайт. Призначення сигналів системної шини процесорів Pentium II і Pentium Pro приведено в додатку 1.

Для найпростіших комп'ютерів за тією ж 0,25 мкм-технологією випустили полегшений варіант процесора Pentium II, названий *Celeron*. Перші процесори Celeron мали частоту ядра 266 і 300 МГц (частота шини - 66 МГц). Вторинний кеш відсутній, що помітно відбилося на продуктивності (системні плати для слота 1 вторинного кеша, природно, не мають). При падінні цін на системні плати і дешевизні самого процесора - Celeron стає дійсно недорогим. Сучасні процесори *Celeron*, починаючи з моделі *Celeron 300A* (із частотою 300 МГц), мають невеликий (128 Кбайт) вторинний кеш, встановлений на кристалі ядра і працюючий уже на повній частоті ядра. Ці процесори відомі також за назвою *Mendocino*. Крім широко відомих особливостей вторинного кеша (або його немає, або 128 К), процесор Celeron має такі відмінності від Pentium II:

- Розрядність шини адреси скорочена з 36 до 32 біт (пам'ять що адресується - 4 Гбайт).
- Контроль паритету шини адреси і шини запиту, ECC-контроль шини даних і контроль невиправних помилок шини, а також сигнал ініціалізації шини - відсутні.
- Процесори призначені тільки для одиничних конфігурацій: для функціонально-надлишкового контролю не вистачає сигналу *FRCERR#*, а із сигналів запиту шини залишився тільки *BR0#*, що не дозволяє використовувати симетричні двопроцесорні конфігурації. Правда, умільці знайшли сигнал *BR1#* і на кристалі ядра в упаковці SEPP, і в корпусі PPGA (тут його дістати зовсім просто), що дозволяє використовувати Celeron у двопроцесорних системах.
- Коефіцієнти множення частоти, принаймні офіційно, фіксовані -

сигнали *LINT[0:1]#*, *A20M#* і *IGNNE#* у якості задатчиків коефіцієнта множення частоти під час дії *RESET#* в інформаційному листку не фігурують.

Для потужних комп'ютерів призначене сімейство *Xeon*. Для них ввели новий *слот 2*, що (разом з інтерфейсом нового процесора) дозволяє будувати як надлишкові системи з FRC, так і симетричні 1-, 2-, 4- і навіть 8-процесорні системи. Частота шини - 100 МГц, частота ядра - 400 МГц і більше, вторинний кеш, як і в Pentium Pro, працює на частоті ядра. Об’єм вторинного кеша - 512 Кбайт, 1 або 2 Мбайт при кешуванні до 64 Гбайт (весь адресний простір при 36-бітовій адресації). Процесори Xeon відрізняються не тільки більшою потужністю, але і великими габаритними розмірами - 15,2x12,7x1,9 см.

Процесори Xeon мають нові засоби збереження системної інформації. Постійна (тільки для читання) пам'ять процесорної інформації *PIROM* (Processor Information ROM) береже такі дані, як електричні специфікації ядра процесора і кеш-пам’яті (діапазони частот і напруг живлення), S-специфікацію і серійний 64-бітовий номер процесора. За інструкцією ідентифікації *CPUID* така інформація недоступна. Енергонезалежна пам'ять *Scratch EEPROM* призначена для занесення системної інформації постачальником процесора (або комп'ютера з цим процесором) і може бути захищена від наступного запису. Процесор обладнаний термодатчиком (термодіод на кристалі ядра) з програмованим пристроєм контролю температури. Цей пристрій має аналого-цифровий перетворювач, що калібрується по термодіоду конкретного процесора на етапі тестування картриджа. Константа настроювання термометра заноситься в PIROM. Пристрій термоконтролю програмується - задається частота перетворень і пороги температури, після досягнення яких формується сигнал переривання. Для взаємодії з PIROM, Scratch EEPROM і пристроєм термоконтролю процесор має додаткову послідовну шину *SMBus* (System Management Bus), яка грунтується на інтерфейсі I2C.

Новинка 1999 року - процесори *Pentium III* - є подальшим розвитком Pentium II. Їхньою головною відмінністю є розширення набору SIMD-інструкцій - SSE (Streaming SIMD Extensions), що досягається завдяки новому блоку 128-розрядних регістрів. Крім того, у них розширена інструкція CPUID, яка дає можливість одержати унікальний 64-бітовий ідентифікатор процесора (той, що в Xeon можна також було прочитати по SMBus). "Прості" Pentium III встановлюються в слот 1, Pentium III Xeon - у слот 2. За характеристиками вторинного кеша і можливостями мультипроцесорних конфігурацій ці процесори аналогічні своїм

попередникам Pentium II і Pentium II Xeon. Частота системної шини - 100 МГц.

Таблиця 1.2 - Основні характеристики процесорів шостого покоління

П р оцесор	Частота			Пам'ять			Інструкції SIMD
	Ядра	Шини	Кеша L2	Адресована	Кеш L2	Кешована	
Pentium Pro	150–200	60–66	F	64 Гб	256К – 1М	?*	–
Pentium II	233–333	66	F/2	64 Гб	256К – 2М	512 Мб	MMX
Pentium II	350–450	66, 100	F/2	64 Гб	256К – 2М	4 Гб	MMX
Pentium II OverDrive	333	66	F	64 Гб	512К	64 Гб	MMX
Pentium II Xeon	400–450	100	F	64 Гб	512К – 2 М	64 Гб	MMX
Celeron	266–300	66	–	4 Гб	–	–	MMX
Celeron	300–433	66	F	4 Гб	128 К	4 Гб?*	MMX
Pentium III	450–500	100	F/2	64 Гб	512К	4 Гб	MMX, SSE
Pentium III Xeon	500–550	100	F	64 Гб	512К – 2 М	64 Гб	MMX, SSE
Mobile Pentium II	233–300	66	F/2	4 Гб	256 – 512К	256–512 Мб	MMX
Mobile Pentium II	266–333	66	F	4 Гб	256 К	4 Гб	MMX
Mobile Celeron	266–300	66	F	4 Гб	128К	4 Гб	MMX

1.9 Принципи кешування пам'яті

Архітектура сучасних 32-розрядних процесорів включає ряд засобів кешування пам'яті: два рівні кеша інструкцій і даних (LI Cache і L2 Cache), буфери асоціативної трансляції (TLB) блока сторінкової переадресації і буфери запису. Ці засоби в різних варіаціях (на кристалі, картриджі процесора або на системній платі) подані в системах із процесорами 486, Pentium і P6. У процесорі 80386 (Intel) був тільки TLB, а кеш-пам’ять, установлювалась на системній платі і не мала підтримки з боку процесора. Основна пам'ять комп'ютерів реалізується на відносно повільній динамічній пам'яті (DRAM), звертання до неї призводить до простою процесора - з'являються такти очікування (wait states). Статична пам'ять

(SRAM), побудована, як і процесор, на тригерних комірках, за своєю природою спроможна наздогнати сучасні процесори за швидкодією і зробити непотрібними такти очікування (або хоча б скоротити їхню кількість). Розумною поступкою для побудови економічних і продуктивних систем явився ієрархічний спосіб організації оперативної пам'яті. Ідея полягає в сполученні основної пам'яті великого об’єму на DRAM із відносно невеличкою кеш-памятью на швидкодіючих мікросхемах SRAM.

У перекладі слово *кеш (cache)* означає «таємний склад», «схованка». Таємниця цього складу полягає в його «прозорості» - для програми він не являє собою додаткової області пам'яті, що адресується. Кеш є додатковим швидкодіючим сховищем копій блоків інформації з основної пам'яті, можливість звертання до яких найближчим часом велика. Кеш не може берегти копію всієї основної пам'яті, оскільки його об’єм у багато разів менший основної пам'яті. Він береже лише обмежену кількість блоків даних і *каталог* (cache directory) - список їх поточної відповідності областям основної пам'яті.

При кожному звертанні до пам'яті контролер кеш-памя’ті за каталогом перевіряє, чи є дійсна копія викликаних даних у кеші Якщо вона там є, те це випадок *кеш-попадання* (cache hit), і дані беруться з кеш-памя’ті. Якщо дійсної копії там немає, це випадок *кеш-промаху* (cache miss), і дані беруться з основної пам'яті, відповідно до алгоритму кешування, блок даних, прочитаний з основної пам'яті, за певних умов замістить один із блоків кеша. Від інтелектуальності алгоритму заміщення залежить відсоток влучень і, отже, ефективність кешування. Пошук блока в списку повинний виконуватись досить швидко, щоб не звести нанівець виграш від застосування швидкодіючої пам'яті. Звертання до основної пам'яті може починатися одночасно з пошуком у каталозі, а у випадку влучення - перериватися (архітектура Look aside). Це заощаджує час, але зайві звертання до основної пам'яті ведуть до збільшення енергоспоживання. Інший варіант: звертання до зовнішньої пам'яті починається тільки після фіксації промаху (архітектура Look Through), при цьому губиться принаймні один такт процесора, зато економиться енергія.

У сучасних комп'ютерах кеш звичайно будується за дворівневою схемою. *Первинний кеш* (LI Cache) вмонтований у всі процесори класу 486 і старше; це *внутрішній* кеш. Об’єм цього кеша невеликий (8-32 Кбайт). Для підвищення продуктивності, для даних і команд часто використовується роздільний кеш (так звана Гарвардска архітектура - протилежність Прінстонській, що використовує загальну пам'ять для команд і даних). *Вторинний кеш* (L2 Cache) для процесорів 486 і Pentium є *зовнішнім*

(встановлюється на системній платі), а в P6 розташовується в одній упаковці з ядром і під'єднується до спеціальної внутрішньої шини процесора.

Кеш-контролер повинний забезпечувати *когерентність* (coherency) - узгодженість даних кеш-пам'яті обох рівнів із даними в основній пам'яті, при тій умові, що звертання до цих даних може виконуватися не тільки процесором, але й іншими активними (busmaster) адаптерами, які під'єднані до шин (PCI, VLB, ISA і т.д.). Необхідно також врахувати, що процесорів може бути декілька, і в кожного може бути свій внутрішній кеш. Контролер кеша оперує *рядками* (cache line) фіксованої довжини. Рядок зберігає копію блока основної пам'яті, розмір якого, збігається з довжиною рядка. З кожним рядком кеша пов'язана інформація про адресу скопійованого в нього блока основної пам'яті і про його стан. Рядок може бути *дійсним* (valid) - це означає, що в даний момент часу він вірогідно відображає відповідний блок основної пам'яті, або *недійсним.* Інформація про те, який саме блок займає даний рядок (тобто старша частина адреси або номер сторінки), і про її стан називається *тегом* (tag) і зберігається в пов'язаній з даним рядком комірці спеціальної *пам'яті тегів* (tag RAM). У операціях обміну з основною пам'яттю, звичайно, бере участь цілий рядок (несекторований кеш), для процесорів 486 і старших довжина рядка збігається з обсягом даних що передаються за один пакетний цикл (для 486 це 4x4=16 байт, для Pentium 4x8=32 байти). Можливий і варіант секторованого (sectored) кеша, при якому один рядок містить декілька суміжних комірок - *секторів,* розмір яких відповідає мінімальній порції обміну даних кеша з основною пам'яттю. При цьому в записі каталога, що відповідає кожному рядку, повинні зберігатися біти дійсності для кожного сектора даного рядка. Секторування дозволяє заощаджувати пам'ять, необхідну для збереження каталога при збільшенні об'єму кеша, оскільки велика кількість біт каталога виділяється під тег і вигідніше використовувати додаткові біти дійсності, чим збільшувати глибину індексу (кількість елементів) каталога.

Рядки кеша під відображення блока пам'яті виділяються при промахах операцій читання, у P6 рядки заповнюються і при запису. Запис блока, що не має копії в кеші, виконується в основну пам'ять (для підвищення швидкодії запис може виконуватись через буфер відкладеного запису). Поводження кеш-контролера при операції запису в пам'ять, коли копія викликаної області знаходиться в деякому рядку кеша, визначається його алгоритмом, або *політикою запису* (Write Policy). Існують дві основні політики запису даних із кеша в основну пам'ять: *наскрізний запис WT* (Write Through) і *зворотний запис WB* (Write Back).

Політика WT передбачає виконання кожної операції запису (навіть однобайтової), що потрапляє в кешований блок, одночасно у рядок кеша і в основну пам'ять. При цьому процесору при кожній операції запису необхідно виконувати відносно тривалий запис в основну пам'ять. Алгоритм достатньо простий у реалізації і легко забезпечує цілісність даних за рахунок постійного збігу копій даних у кеші й основній пам'яті. Для нього не потрібно берегти ознаки присутності і модифікації - цілком достатньо тільки інформації тега. Але ця простота обертається малою ефективністю запису. Існують варіанти цього алгоритму з застосуванням відкладеного буферованого запису, при якому дані в основну пам'ять переписуються через FIFO-буфер під час вільних тактів шини.
Політика WB дозволяє зменшити кількість операцій запису на шині основної пам'яті. Якщо блок пам'яті, у який повинний провадитися запис, відображений у кеші, то фізичний запис спочатку буде зроблено в цей дійсний рядок кеша, і він буде відзначений як *брудний* (dirty), або модифікований, тобто такий, який потребує запису в основну пам'ять. Тільки після запису в основну пам'ять рядок стане *чистим* (clean), і його можна буде використовувати для кешування інших блоків без втрати цілісності даних. У основну пам'ять дані переписуються тільки цілим рядком. Запис в основну пам'ять може відкладатися контролером до настання крайньої необхідності (звертання до кеш-пам'яті іншого абонента, заміщення в кеші новими даними) або виконуватися у вільний час після модифікації всього рядка. Даний алгоритм складніший в реалізації, але істотно ефективніший, ніж WT.

У залежності від способу визначення взаємної відповідності рядка кеша й області основної пам'яті розрізняють три архітектури кеш-пам’яті: *кеш прямого відображення* (direct-mapped cache), *цілком асоціативний кеш* (fully associative cache) і їхня комбінація - частково- або *набірно-асоціативний кеш* (set-associative cache).

1.10 Технологія динамічного виконання

Швидкодія суперскалярного процесора Pentium Pro (і старших моделей) досягається не тільки за рахунок суперскалярної конвеєрної обробки (декілька конвеєрів – має 12-ступеневі незалежні конвеєри), але і застосуванням нової технології – технології динамічного виконання. Новий підхід не має обмежень прямої залежності традиційного слідування стадій підвантаження і виконання інструкцій, що відкриває широкі можливості обробки інструкцій з використанням їх об’єднання.

Для реалізації технології динамічного виконання необхідна заміна фундаментальної стадії “виконання” незв’язаними стадіями

“планування-виконання” і “вивантаження”. Це дозволяє інструкціям (мікроінструкціям - μops) розпочинатися в будь-якому порядку, а закінчуватись у відповідності з їх початковим порядком в програмі. Процесор Pentium Pro реалізований у вигляді трьох незалежних пристроїв [10] під’єднаних до пулу мікроінструкцій (рис. 1.1):

- Пристрій виборки і декодування (Feth/Decoder Unit), на вхід якого поступає потік інструкцій програми користувача із кеш інструкцій, що розшифровуються в ряд мікрооперацій (μops), які поступають в пул мікроінструкцій.
- Пристрій планування-виконання (Dispatch/Execute Unit) виконує μops з урахуванням взаємозв’язку даних і готовності пристроїв і тимчасово зберігає результати віртуального виконання.
- Пристрій вивантаження (Retire Unit) слідкує за порядком вивантаження віртуальних результатів.

Крім того, в його склад входить пристрій шинного інтерфейсу, який відповідає за під’єднання цих трьох модулів до шин процесора.

Пристрій виборки і декодування містить три паралельних декодери. Більшість інструкцій декодуються безпосередньо в один μops, деякі в чотири μops, а складні інструкції вимагають більшої кількості. Пристрій планування вибирає μops з пула мікроінструкцій. За один такт процесор Pentium Pro може планувати до 5-и μops за такт. Основна суть цього процесу планування є виконання інструкцій не за порядком; μops передаються на виконавчі пристрої безвідносно порядку в початковій програмі.

Розглянемо фрагмент коду на рис. 1.2. Нехай, наприклад, перша мікроінструкція завантаження регістра r1 генерує сигнал *MISS* (дані відсутні в кеш – промах), тоді традиційне ядро процесора повинно очікувати поки модуль інтерфейсу шини не прочитає дані із основної пам’яті і не відтранслює їх перед переміщенням до інструкції 2. Такий метод призупиняє процесор для очікування даних і таким чином він не довикористовується. Необхідно пам’ятати також, що швидкість пристроїв пам’яті зросла тільки на 60 % за останні 10 років, в той час як швидкість мікропроцесорів збільшилась в десятки раз.

Щоб зменшити час очікування пам’яті, процесор Pentium Pro продивляється на декілька кроків вперед мікроінструкції і якщо можливо виконує їх. В прикладі на рис. 1.2. інструкція 2 не може бути виконана, оскільки вона залежить від результату інструкції 1, а інструкції 3, 4 можуть бути виконані. Якщо для виконання доступно декілька μops одночасно, то їх виборка із пула інструкцій

виконується згідно з однією з модифікацій стратегії *FIFO* (first-come-first-served). Результати виконання повертаються в пул інструкцій.

Пристрій вивантаження перевіряє стан μops в пулі інструкцій –

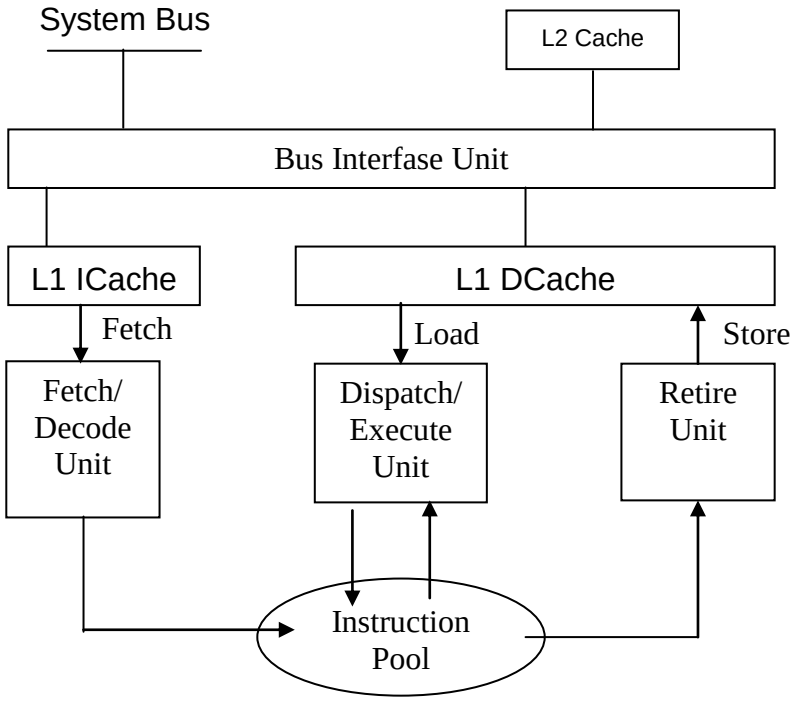


Рис. 1.1 - Архітектура процесора Pentium Pro. L1 ICache – первинний кеш інструкцій (команд) ; L1 DCache – первинний кеш даних; L2 Cache – вторинний кеш ; Fetch/Decode Unit – пристрій вибірки і декодування; Dispatch/Execute Unit – пристрій планування і виконання; Retire Unit – пристрій вивантаження; Bus Interfase Unit – шинний інтерфейс; Instruction Pool – пул мікроінструкцій

```
r1<= mem [r0]     /*Instuction1*/
r2<= r1 + r2      /*Instuction2*/
r3<= r5 + 1       /*Instuction3*/
r4<= r6 - r3      /*Instuction4*/
```

Рис. 1.2 - Типовий фрагмент коду

виконується пошук інструкцій, які були виконані і можуть бути видаленими. При цьому пристрій вивантаження не тільки вибирає виконані μops, але і відновлює порядок їх розташування в початковій програмі.

Додаткову інформацію відносно 32-х розрядних процесорів можливо отримати в INTERNET за такими адресами:

INTEL: http://www.intel.com; http://www.intel.ru;
AMD: http://www.amd.com;
CYRIX: http://www.cyrix.com

Контрольні питання

1. Який процесор є базовим для процесорів сімейства x86?
2. Які принципові особливості процесора i80286?
3. З якої моделі мікропроцесора починається історія 32-х розрядних процесорів та їх характеристики?
4. Охарактеризуйте 32-х розрядний процесор i486.
5. Які особливості процесорів Pentium?
6. Що таке технологія MMX, її призначення?
7. Дайте характеристику процесорів Pentium Pro та Pentium 11?
8. Дайте порівняльну характеристику процесорів Pentium 11 та Pentium 111?
9. Охарактеризуйте 32-х розрядні процесори, виробників відмінних від INTEL.
10. Поясніть основні методи кешування пам'яті.
11. Що таке технологія динамічного виконання?

РОЗДІЛ 2. ПРОГРАМНА МОДЕЛЬ 32-РОЗРЯДНИХ ПРОЦЕСОРІВ

Історія 32-розрядних процесорів Intel почалася з процесора Intel386. Він увібрав у себе усі властивості своїх 16-розрядних попередників 8086/88 : 80286 для забезпечення сумісності з величезним об'ємом ПО, що існувало на момент його появи. Проте в процесорах 80386 відсутнє жорстке обмеження на довжину безперервного сегмента пам'яті - 64 Кбайт. У захищеному режимі 32-розрядних процесорів воно відсунулося до 4 Гбайт - межа фізично адресованої пам'яті, що тоді можна було вважати майже «нескінченністю». Ці процесори мають підтримку віртуальної пам'яті об'ємом до 64 Тбайт, вмонтований блок керування пам'яттю підтримує механізми сегментації і сторінкової трансляції адрес (Paging). Процесори забезпечують чотирирівневу систему захисту просторів пам'яті і введення/виведення, а також переключення задач. Система команд розширена при збереженні всіх команд 8086, 80286. Процесор може працювати в одному з двох режимів і перемикатися між ними достатньо швидко як у ту, так і в іншу сторону:

Real Address Mode - режим реальної адресації (або просто реальний режим - Real Mode), цілком сумісний з 8086. У цьому режимі можлива адресація до 1 Мбайта фізичної пам'яті (насправді, як і в 80286, майже на 64 Кбайт більше).

Protected Virtual Address Mode - захищений режим віртуальної адресації (або просто захищений режим - Protected Mode). У цьому режимі процесор дозволяє адресувати до 4 Гбайт фізичної пам'яті, через які при використанні механізму сторінкової адресації можуть відображатися до 64 Тбайт віртуальної пам'яті кожної задачі.

Істотним доповненням є **Virtual 8086 Mode** - режим віртуального процесора 8086. Цей режим є особливим станом задачі захищеного режиму, у якому процесор функціонує як 8086. На одному процесорі в такому режимі можуть паралельно виконуватися декілька задач з ізольованими один від одного ресурсами. При цьому використання фізичного адресного простору пам'яті управляється механізмами сегментації і трансляції сторінок. Спроби виконання неприпустимих команд, виходу за рамки відведеного простору пам'яті і дозволеної області введення/виведення контролюються системою захисту.

Процесори, починаючи з Pentium і деяких моделей 486, мають і особливий режим системного керування *System Management Mode (SMM),* у якому процесор виходить в ізольований від інших режимів

простір пам'яті. Цей режим використовується з службовою метою, а також для налагодження програм.

Процесори можуть оперувати з 8, 16 і 32-бітовими операндами, рядками байтів, слів і подвійних слів, а також із бітами, бітовими полями і рядками бітів. У архітектуру процесорів введені засоби налагодження і тестування. У цій главі ми розглянемо базову програмну модель, загальну для всіх існуючих на даний момент 32-розрядних процесорів: 80386, 486, Pentium, Pentium Pro, Pentium II, Pentium III, Celeron а також сумісних з ними. Ця модель охоплює набір регістрів процесора, організацію пам'яті і введення/виведення, типи даних, систему команд, переривання і винятки. Роботі процесора в захищеному режимі (сегментації, захисту, сторінковій переадресації) присвячено розділ 4.

2.1 Регістри процесора

Процесори мають регістри, що підрозділяються на такі категорії (рис. 2.1 – рис. 2.4):

- регістри загального призначення,
- покажчик інструкцій,
- регістр прапорців,
- регістри сегментів,
- системні адресні регістри,
- керуючі регістри,
- регістри налагодження,
- регістри тестування і модельно-специфічні регістри.

Набір регістрів загального призначення (рис. 2.1) включає відповідні регістри процесорів 8086 і 80286. Всі ці регістри мають розрядність 32 біти. До старого позначення їхніх імен додалася приставка E (Extended - розширений). Відсутність приставки в імені означає посилання на молодші 16 біт розширених регістрів. Як і в 8086, можливо незалежне звертання до молодшого і старшого байтів регістрів AX, BX, CX і DX. Розрядність операнда даних може складати 1,8, 16,32 або 64 біти, бітового поля 1-32 біти, адреси - 16 або 32 біти.

Покажчик інструкцій EIP містить зсув наступної інструкції, що виконується щодо бази сегмента коду. При 16-бітовій адресації використовуються тільки молодші 16 біт (IP).

Регістр прапорців EFLAGS (рис. 2.2) також розширений до 32 бітів. Біти, визначені для 8086 і 80286, мають старе призначення. У порівнянні з 80286 з'явилися біти VM і PF, ряд прапорців додався з появою процесорів 4-го і 5-го поколінь.

Призначення бітів регістра *EFLAGS* таке*:*

- ID (Id Flag) - прапорець доступності команди ідентифікації CPUID

(P5+ і деякі 486).

- VIP (Virtual Interrupt Pending) - віртуальний запит переривання (P5+)
- VIF (Virtual Interrupt Flag) - віртуальна версія прапорця IF (дозволу на переривання) для багатозадачних систем (P5+).
- AC (Alignment Check) - прапорець контролю вирівнювання. При виконанні програм із рівнем привілеїв 3 (три) у випадку звертання до операнду, невирівняного по відповідній межі (2, 4, 8 байт), і при встановленому прапорці AC формується виняток #AC із нульовим кодом помилки. На рівнях привілеїв 0, 1, контроль вирівнювання не виконується (486+).
- VM (Virtual 8086 Mode) - у захищеному режимі включає режим віртуального процесора 8086. Спроба використання привілейованих інструкцій у цьому режимі викликає виняток #GP. Біт може встановлюватись тільки в захищеному режимі - інструкцією IRET на нульовому рівні привілеїв або переключенням задач на будь-якому рівні привілеїв. На біт не діють інструкції POPF, а PUSHF у цей біт завжди заносить 0. Його одиничне значення може зберегтися тільки в образі EFLAGS, що зберігається при перериванні, перемиканні задач або переході в режим SMM.
- RF (Resume Flag) - прапорець поновлення, використовується разом із регістрами точок зупинок.
- IOPL (Input/Output Privilege Level) - рівень привілеїв введення/виведення.
- NT (Nested Task Flag) - прапорець вкладеної задачі.
- OF (Overflow Flag) - прапорець переповнення. Встановлюється, якщо результат арифметичної операції не поміщається в операнді призначення.
- DF (Direction Flag) - прапорець керування напрямком в операціях з рядками. При одиничному значенні індексні регістри, що беруть участь в операціях з рядками, автоматично декрементуються на кількість байтів операнда, при нульовому - інкрементуються.
- IF (Interrrupt-enable Flag) - прапорець керування перериваннями. При одиничному значенні дозволяється виконання маскованих апаратних переривань.
- TF (Trap Flag) - прапорець трасування (покрокового режиму). При його установці після виконання кожної команди викликається внутрішнє переривання типу 1 (#DB).
- SF (Sign Flag) - прапорець знаку. Вказує на стан старшого біта результату. Одиничне значення старшого біта результату - ознака від'ємного числа.
- ZF (Zero Flag) - прапорець нульового результату.

- AF (Auxiliary Flag) - прапорець додаткового перенесення (позики) у зошиті для десяткової арифметики. Встановлюється при наявності перенесення з молодшої тетради результату в старшу.
- PF (Parity Flag) - прапорець паритету, встановлюється при парному числі одиниць у результаті.
- CF (Carry Flag) - прапорець перенесення (позики) старшого біта в арифметичних операціях.

Регістри сегментів містять 16-бітові покажчики (у реальному режимі) або селектори дескрипторів (у захищеному режимі) сегментів. CS (Code Segment) - сегмент кодів команд, SS (Stack Segment) - сегмент стека, DS (Data Segment) - сегмент даних, ES, FS і GS - додаткові сегменти даних. Використання сегментних регістрів визначається типом звертання до пам'яті. Для багатьох типів звертань можливо застосування альтернативних сегментних регістрів, що вводиться префіксами команд CS:, SS:, DS:, ES: FS: або GS:. З кожним із шести сегментних регістрів пов'язані програмно-недоступні сховані регістри дескрипторів (їх ще називають Segment Descriptor Cache - кеш сегментних регістрів), які автоматично завантажуються при завантаженні відповідних сегментних регістрів. У захищеному режимі в регістри дескрипторів із таблиці дескрипторів завантажується 32-бітова базова адреса, 32-бітовий ліміт і атрибути сегментів. За вмістом цих схованих регістрів при кожному звертанні до пам'яті виконується обчислення лінійної адреси і перевірка захисту, причому саме ці регістри задають властивості сегментів і в захищеному й у реальному режимах. Образ цих 96-бітових регістрів доступний у режимі SMM і внутрішньосхемній емуляції, формати специфічні для кожної моделі процесорів. У реальному режимі, у який процесор входить після апаратного скидання, ліміт (розмір сегмента) фіксований - 64 Кбайт, атрибути не використовуються, а в якості базової адреси заноситься значення сегментного регістра, зсунуте на 4 біти вліво. У захищеному режимі ліміт може задаватися в межах 1 байт - 4 Гбайт. Позаштатним перемиканням із захищеного в реальний режим можна домогтися і нестандартного розміру сегментів.

Керуючі регістри (Control Registers) **CR0**, **CR1**, **CR2**, **CR3** призначені для збереження ознак стану процесора, загальні для всіх задач (рис. 2.3).

Регістр CRO містить у собі біти регістра *MSW* (Machine State Word) процесора 80286. Для забезпечення програмної сумісності команди *LMSW* і *SMSW* торкаються тільки цих молодших 4-х біт. Призначення біт регістра CRO таке:

- PE (Protection Enable) - дозвіл захисту. Встановлення цього прапорця інструкцією LMSW або LOAD CRO переводить процесор

у захищений режим, скидання прапорця (повернення в реальний режим) можливо тільки за інструкцією LOAD CRO. Скидання біта PE є частиною досить довгої послідовності інструкцій, які підготовлюють коректне перемикання в реальний режим.

- MP (Monitor Processor Extension) - моніторинг співпроцесора. Дозволяє викликати виняток #NM за кожною командою WAIT при TS=1. При виконанні програм для процесорів 286/287 і 386/387 на процесорах 486DX і старших прапорець MP повинний бути встановлений.
- EM (Processor Extension Emulated) - емуляція співпроцесора. Установка цього прапорця викликає появу винятку #NM при кожній команді, що відноситься до співпроцесора, що дозволяє прозоро здійснювати його програмну емуляцію.
- TS (Task Switch) - перемикання задач. При встановленні цього прапорця наступна команда, яка відноситься до співпроцесора, викликає виняток #NM, що дозволяє програмно визначити, чи відноситься контекст співпроцесора до поточної задачі. Цей біт скидається інструкцією CLTS.
- ET (Extension Type) - індикатор підтримки інструкцій математичного співпроцесора. Використовується в процесорах 486+, для 486SX - ET=0, для інших процесорів - ET=1.
- NE (Numeric Error) - дозвіл стандартного (для Intel, але не для PC) механізму повідомлення про помилку FPU через генерацію винятку (486+).
- WP (Write Protect) - дозвіл захисту сторінок пам'яті.
- AM (Alignment Mask) - дозвіл контролю вирівнювання (контроль вирівнювання виконується тільки на рівні привілеїв 3 при AM=1 і прапорці AC=1).
- NW (Not Writethrough) - заборона наскрізного запису кеша і циклів анулювання.
- CD (Cache Disable) - заборона заповнення кеша (влучення в раніше заповнені рядки при цьому, обслуговуються кешем).
- PG (Paging Enable) - вмикання механізму сторінкової переадресації пам'яті.

Сполучення MP=0 EM=0 TS=0, яке встановлюється за апаратним скиданням, забезпечує повну сумісність із 8086/88 (виняток #NM не формується). Сполучення MP=1, EM=0 використовується при наявності співпроцесора, а MP=0, EM=1 при його програмній емуляції.

Регістр CR1 не використовується.

Регістр CR2 (Page Fault Linear Address) призначений для зберігання 32-бітової лінійної адреси, за якою була отримана остання відмова сторінки пам'яті.

Регістр CR3 (Page Directory Base Register) у старших 20 бітах зберігає фізичну базову адресу таблиці каталога сторінок. З молодших 12 бітів у процесорах 486+ використовуються такі:

- PCD (Page-Level Cache Disable) - заборона кешування сторінки (одне із джерел апаратного сигналу PCD для керування зовнішнім кешем).
- PWT (Page-Level Writes Trough) - кешування сторінки з наскрізним записом (одне із джерел апаратного сигналу PWT для керування зовнішнім кешем).

Регістр CR4 (присутній у процесорах Pentium і старших) містить біти дозволу архітектурних розширень. Призначення розрядів регістра CR4 таке:

- VME (Virtual-8086 Mode Extensions) - дозвіл використання віртуального прапорця переривань у режимі V86, що дозволяє підвищити продуктивність за рахунок скорочення викликів монітора віртуальних машин.
- PVI (Protected-Mode Virtual Interrupts) - дозвіл використання віртуального прапорця переривань у захищеному режимі.
- TSD (Time Stamp Disable) - перетворення інструкції RDTSC (читання лічильника міток реального часу) у привілейовану.
- DE (Debugging Extensions) - розширення налагодження (дозвіл точок останова на інструкціях звертання до заданих портів введення/виведення).
- PSE (Page Size Extension) - розширення розміру сторінки (4 Кбайт і 4 Мбайт).
- PAE (Physical Addres Extension) - розширення фізичної адреси (сторінки 4 Кбайт і 2 Мбайт, 36-бітова адресація).
- MCE (Machine-Check Enable) - дозвіл машинного контролю (формування винятку #MC за машинною помилкою, P5+).
- PGE (Paging Global Extensions) - дозвіл глобальності в сторінковій переадресації. При PGE=1 за командою MOV CR3 у TLB очищаються тільки входження з невстановленим бітом глобальності G (P6+).
- PCE (Performance-monitoring Counter Enable) - дозвіл звертання до лічильників подій (інструкція RDPMC) на будь-якому рівні привілеїв. Системні адресні регістри призначені для посилань на сегменти і таблиці в захищеному режимі.

Регістри GDTR (Global Descriptor Table Register) і IDTR (Interrupt Descriptor Table Register) програмно завантажуються 6- байтовими операндами, що включають 32-бітову лінійну базову адресу і 16-бітовий ліміт глобальної таблиці дескрипторів і таблиці дескрипторів переривань (рис.4). У регістр задачі TR (Task Register) і регістр селектора локальної таблиці дескрипторів LDTR (Local Descriptor

Рис. 2.1 - Основні регістри 32-разрядних процесорів

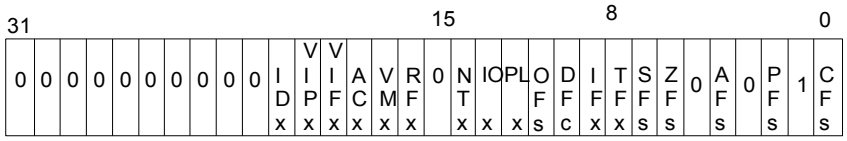


Рис. 2.2 - Регістр прапорців EFLAGS (X-системний прапорець, S- прапорець стану, C - керуючий прапорець)

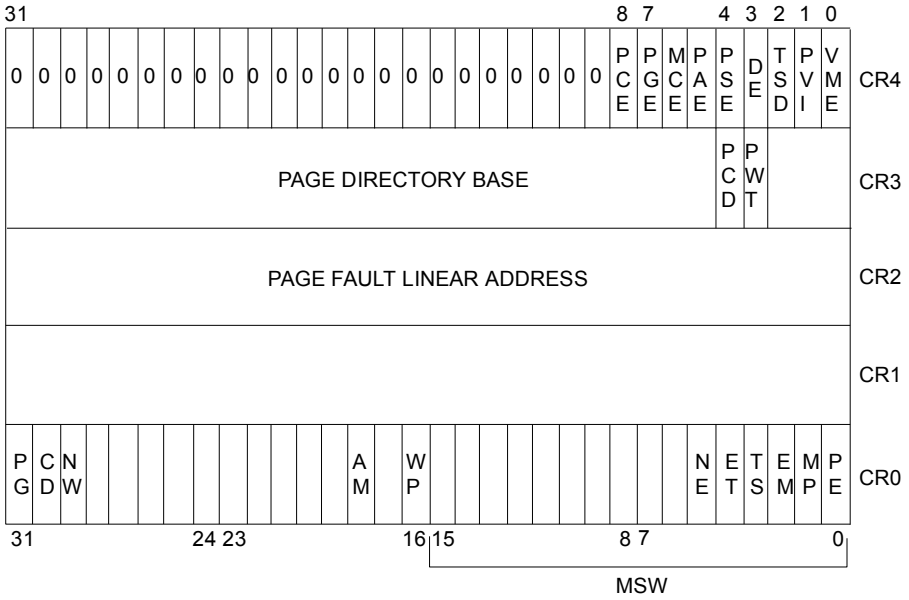


Рис. 2.3 - Керуючі регістри CR0-CR4

Рис. 2.4 - Системні адресні регістри

Table Register) завантажуються 16-бітові селектори дескрипторів сегмента стана задачі TSS і локальної таблиці дескрипторів LDT. Це завантаження викликає автоматичне завантаження самих дескрипторів, що містять 32-бітові поля лінійної базової адреси і ліміту, а також полів атрибутів у пов'язані з ними невидимі регістри дескрипторів.

Регістри налагодження (Debug Register) призначені для завдання і керування налагоджуваними точками зупинок. Регістри DRO...DR3 (Linear Breakpoint Address 0...3) зберігають 32-бітові лінійні адреси точок зупинок (див. главу 4).

Регістри DR4, DR5 у 80386 і 486 не використовуються, звертання до них еквівалентно звертанню до регістрів DR6, DR7. У процесорі Pentium при включеному розширенні налагодження звертання до цих регістрів викликає винятки неприпустимого коду операції (#UD).

Регістр DR6 (Breakpoint Status) відображає стан контрольної точки. Регістр DR7 (Breakpoint Control) управляє установкою контрольних точок. Склад регістрів тестування (Test Register) варіюється в залежності від типу процесора. Процесори 80386 мали тільки два регістри, призначених для тестування кеша сторінкової переадресації - TR6 і TR7, для процесора 486 склад регістрів розширений: TR3 - регістр даних внутрішнього кеша, TR4 - тестовий регістр стану кеша, TR5 - керуючий регістр тестування кеша, TR6 (Test Control) - керуючий регістр для теста кешування сторінок, TR7 (Test Status) - регістр даних для теста кешування сторінок.

У процесорах Pentium і старших моделях тестові регістри входять у групу модельно-специфічних регістрів MSR. Для цих процесорів звертання до регістрів TRx викликає виняток #UD неприпустимого коду операції.

Модельно-специфічні регістри MSR (Model-Specific Registers) призначені для керування розширеннями налагодження, моніторингом продуктивності, машинним контролем, кешування областей фізичної пам'яті й інших функцій. Їхнє призначення прив'язується до архітектури конкретного процесора, склад змінюється від моделі до моделі, доступ привілейований. Інструкції обміну з цими 64-бітовими регістрами припускають, що дані знаходяться в парі EDX:EAX, а номер вказується в регістрі ECX, що дозволяє необмежено (до 4 мільярдів) збільшувати число цих регістрів.

Доступність регістрів різних груп залежить від режиму роботи процесора і рівня привілеїв задачі (табл. 2.1).

Таблиця 2.1 - Доступність регістрів 32-разрядних процесорів

Режим	Реальний		Захищений		Віртуальний 8086	
Регістри	Заванта-ження	Зберігання	Заванта-ження	Зберігання	Заванта-ження	Зберігання
Загального призначенн	Так	Так	Так	Так	Так	Так
Сегментів	Так	Так	Так	Так	Так	Так
Прапорців	Так	Так	Так	Так	IOPL[1]	IOPL[1]
Керуючі	Так	Так	PL=0	PL=0	Ні	Так
GDTR, IDTR	Так	Так	PL=0	Так	Ні	Так
LDTR, TR	Ні	Ні	PL=0	Так	Ні	Ні
Налагодження	Так	Так	PL=0	PL=0	Ні	Ні
Тестування	Так	Так	PL=0	PL=0	Ні	Ні
MSR	PL=0	PL=0	PL=0	PL=0	PL=0	PL=0

[1] *PUSHF* і *POPF* залежать від рівня привілеїв.

2.2.Організація пам'яті

Пам'ять для 32-розрядних процесорів 80x86 підрозділяється на байти (8 бітів), слова (16 бітів), подвійні слова (32 біти). Слова записуються в двох суміжних байтах, починаючи з молодшого. Адресою слова є адреса його молодшого байта. Подвійні слова записуються в чотирьох суміжних байтах, починаючи з молодшого байта, адреса котрого і є адресою подвійного слова.

Більш значними одиницями є сторінки і сегменти. Пам'ять може логічно організовуватися у виді одного або множини сегментів перемінної довжини (у реальному режимі - фіксованої). Сегменти можуть вивантажуватися на диск і в міру необхідності з нього підкачуватися у фізичну пам'ять. Крім сегментації, у захищеному режимі можлива розбивка (Paging) логічної пам'яті на сторінки розміром 4 Кбайт, кожна з який може відображатися на будь-яку область фізичної пам'яті. Починаючи з 5-го покоління, з'явилася можливість збільшення розміру сторінки до 4 Мбайт. Сегментація і розбивка на сторінки можуть застосовуватися в будь-яких сполученнях. Сегментація є засобом організації логічної пам'яті, який використовується на прикладному рівні. Розбивка на сторінки застосовується на системному рівні для керування фізичною пам'яттю.

Стосовно до пам'яті розрізняють три адресних простори: логічний, лінійний і фізичний. Основним режимом роботи 32-розрядних процесорів вважається захищений режим, у якому працюють усі механізми перетворення адресних просторів (рис. 2.5).

Логічна адреса (віртуальна), складається із селектора сегмента (у реальному режимі - просто адреси сегмента) і ефективної адреси

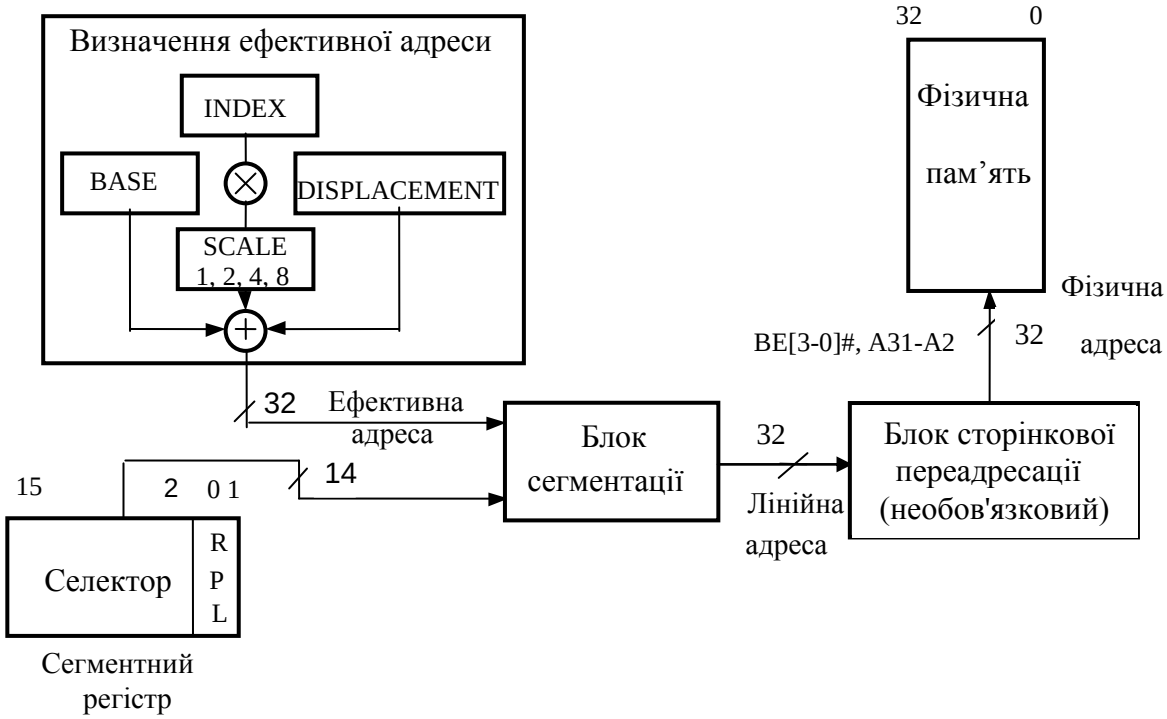


Рис. 2.5 – Формування адреси пам’яті 32-х розрядних процесорів у захищеному режимі

- зміщення (offset). Ефективна адреса формується додаванням компонентів base, index, displacement з урахуванням масштабу scale. Оскільки кожна задача може мати до 16 К селекторів, а зміщення, обмежене розміром сегменту, може досягати 4 Гбайт, логічний адресний простір для кожної задачі може досягати 64 Тбайт. Весь цей простір віртуальної пам'яті в принципі доступний програмісту (за умови підтримки з боку операційної системи).

Блок сегментації транслює логічний адресний простір у 32-бітовий простір лінійних адрес. Лінійна адреса утвориться додаванням базової адреси сегмента з ефективною адресою. Базова адреса сегмента в реальному режимі утворюється множенням вмісту поточного сегментного регістра на 16 (як і в 8086). У захищеному режимі базова адреса завантажується з дескриптора, що зберігається в таблиці, по селектору, завантаженому у сегментний регістр, що використовується.

32-бітова фізична адреса пам'яті утворюється після перетворення лінійної адреси блоком сторінкової переадресації. Вона виводиться на зовнішню шину адреси процесора. У найпростішому випадку (при вимкненому блоці сторінкової переадресації) фізична адреса збігається з лінійною. Ввімкнутий блок сторінкової переадресації здійснює трансляцію лінійної адреси у фізичну сторінками розміром 4 Кбайт (для старших поколінь процесорів також можливі сторінки розміром 2 або 4 Мбайт). Цей блок забезпечує розширення розрядності фізичної адреси

процесорів шостого покоління до 36 біт. Блок переадресації може вмикатися тільки в захищеному режимі.

Для звертання до пам'яті процесор (разом із зовнішньою схемою) формує шинні сигнали MEMWR# (Memory Write) і MEMRD# (Memory Read) для операцій запису і читання відповідно. Шина адреси розрядністю 32 біти дозволяє адресувати 4 Гбайт фізичної пам'яті, але в реальному режимі доступний тільки 1 Мбайт, що починається з молодших адрес.

У реальному режимі адресації пам'яті забезпечується сумісність із процесором 8086, що своєю 16-розрядною адресною шиною охоплює простір фізичної пам'яті в 1 Мбайт. Для сумісності з 80286 32-розрядні процесори повторюють його помилку, пов'язану з перенесенням, що виникає при додаванні адреси сегмента з ефективною адресою. При обчисленні фізичної адреси можливо виникнення перенесення, що викликає появу одиниці на лінії A20 шини адреси. Максимальне значення адреси в реальному режимі 10FFEF досягається при Seg=FFFFh і EA=FFFFh. Для забезпечення повної програмної сумісності з 8086 у PC використовується вентиль Gate A20, що примусово занулює біт A20 системної шини адреси. Вентиль у PC управляється через програмно-керований біт контролера клавіатури 8042 або більш швидким засобом (Gate A20 Fast Control), обумовленим чипсетом системної плати.

У реальному режимі розмір сегмента фіксований , як і в 8086, він складає 64 Кбайт (FFFFh). Спроба використання ефективної адреси, що виходить за межу сегмента, при 32-бітовій адресації формує виняток #GP. При 16-бітовій адресації при обчисленні ефективної адреси можливе перенесення у розряд A16 ігнорується і сегмент «сгортається в кільце» (як і в 8086). Засоби контролю стежать і за переходом через межу сегмента під час звертання за «прикордонною» адресою. При спробі адресації до слова, що має зсув FFFFh, або до подвійного слова зі зсувом FFFDh-FFFFh (їхні старші байти виходять за межу сегмента), або виконання інструкції, хоча б один байт якої не вміщається в даному сегменті, процесор формує виняток #GP. При спробі виконання інструкції співпроцесора *(ESCAPE)* з операндом пам'яті, що не вміщається в сегменті, формується виняток *9* - Processor Extension Segment Overrun Interrupt (тільки для 80386).

Система команд 32-розрядних процесорів (див. додаток) передбачає 11 режимів адресації. При цьому тільки в двох випадках операнди не пов'язані з пам'яттю. Це операнд-вміст регістра, що береться з будь-якого 8-, 16- або 32-бітового регістра процесора, і безпосередній операнд (8, 16 або 32 бітовий), який міститься в самій команді. Інші дев'ять режимів так чи інакше звертаються до пам'яті.

При звертанні до пам'яті ефективна адреса обчислюється з

використанням таких компонентів:

- Зсув (зміщення) (Displacement або Disp) - 8-, 16- або 32-бітне число, включене в команду.
- База (Base) - вміст базового регістра. Звичайно використовується для вказівки на початок деякого масиву.
- Індекс (index) - вміст індексного регістра. Звичайно використовується для вибору елемента масиву.
- Масштаб (Scale) - множник (1, 2, 4 або 8), указаний у коді інструкції. Цей елемент використовується для вказівки розміру елемента масиву, доступний тільки при 32-бітовій адресації. Ефективна адреса обчислюється згідно з формулою:

$$EA = Base + Index * Scale + Disp.$$

Окремі доданки в цій формулі можуть бути відсутні. Можливі режими адресації приведені в табл. 2.2. Процесор може працювати з 32- або 16-бітовою адресацією, 16-бітова адресація працює так само, як і в процесорах 8086 і 80286, при цьому в якості компонентів адреси використовуються молодші 16 біт відповідних регістрів. При 32-бітовій адресації використовуються розширені 32-розрядні регістри і додаткові режими з масштабуванням індексу. Відмінності 16- і 32-бітових режимів адресації приведені в табл. 2.3.

У реальному режимі за замовчуванням використовується 16-бітова адресація, але за допомогою префікса зміни розрядності адреси (Address Length Prefix) для поточної інструкції можна перемикатися на 32-бітову. При цьому з'являються додаткові можливості адресації (масштабування), але значення ефективної адреси все рівно не може бути більшим 64-кілобайт - при спробі звернутися за ефективною адресою більшою за це значення генерується виняток #GP (General Protection Fault).

У захищеному режимі адресація за замовчуванням визначається бітом D дескриптора кодового сегмента, що використовується: при D=0 - 16 біт, при D=1 - 32 біти. Префікс розрядності адреси перемикає розрядність для поточної інструкції на протилежну.

При звертаннях до пам'яті використання сегментних регістрів за замовчуванням визначається типом звертання (табл. 2.4). Для більшості типів звертання на час поточної інструкції при необхідності можливо використання альтернативного сегментного регістра, на що вказує *префікс заміни сегмента* (CS:, *DS:, ES:,* SS:, FS: або *GS:)* перед кодом інструкції.

2.3 Введення/виведення

Як і процесор 8086, 32-розрядні процесори дозволяють адресувати до 64 К однобайтових або 32 К двобайтових регістрів у просторі, окремому від пам'яті. Додатково є можливість звертання до 32-бітових портів. При операціях введення/виведення виводи

Таблиця 2.2- Режими адресації пам’яті 32-розрядних процесорів

Режим	Адрес
Пряма адресація (Direct Mode)	*EA=Disp*
Непряма регістрова адресація (Register *Indirect Mode)*	*EA=Base*
Базова адресація (Based Mode)	*EA=Base+Disp*
Індексна адресація (Index Mode)	*EA=Index+Disp*
Масштабована індексна адресація (Scaled Index Mode)	*EA=Scale x Index + Disp*[1]
Базово-індексна адресація (Based Index Mode)	*EA=Base+Index*
Масштабована базово-індексна адресація (Based Scaled Index Mode)	*EA=Base+Scale x Index*[1]
Базово-індексна адресація із зсувом (Based Index Mode with Displacement)	*EA=Base+Index+Disp*
Масштабована базово-індексна адресація із зсувом (Based Scaled Index Mode with Displacement)	*EA=Base+Scale x Index+Disp*[1]

[1] Масштабування індексу можливо тільки при 32-бітовій адресації.

Таблиця 2.3- Відмінності режимів адресації

Компонент	16-бітова адресація	32-бітова адресація
Базовий регістр	BX або BP	Будь-який 32-бітовий регістр загального призначення
Індексний регістр	SI або DI	Будь-який 32-бітовий регістр загального призначення, крім ESP
Маштаб	Hi (завжди «1»)	1, 2, 4 або 8
Зсув	0, 8 або16 біт	0, 8 або 32 біти

Таблиця 2.4- Використання сегментних регістрів при звертанні до пам’яті

Тип звертання до пам’яті	Сегментний регістр	
	за замовчуванням	Альтернативний
Вибірка команд	*CS*	*Hi*
Стекові операції	*SS*	*Hi*
Рядок-приймач	*ES*	*Hi*
Будь-які інші посилання на пам’ять, крім тих, що використовують в якості базового регістра BP, EBP або ESP	*DS*	*CS, ES, SS, FS. GS*
Посилання на пам’ять, що використовують в якості базового регістра BP, EBP або ESP	*SS*	*CS, DS, ES. FS. GS*

A[16:31] не використовуються. Адреса пристрою задається або в команді (тільки молодший байт, старший - нульовий), або береться з регістра DX (повна 16-бітова адреса). Команди введення/виведення викликають шинні цикли з активними сигналами IORD#, IOWR# (читання порта введення/виведення і запис в порт введення/ виведення). Рядкові (ланцюгові) команди забезпечують блокове введення/виведення із швидкістю, що перевищує аналогічні операції зі стандартним контролером DMA. У адресному просторі введення/виведення область OF8-OFF зарезервована для використання співпроцесором (при звертанні до співпроцесора процесор 80386 виставляє одиницю на найстаршій лінії шини адреси, що використовується для спрощення дешифрації адрес).

У захищеному режимі інструкції введення/виведення є привілейованими. Це означає, що вони можуть виконуватися задачами тільки з визначеним рівнем привілеїв, обумовленим полем IOPL регістра прапорців або бітовою картою дозволу введення/виведення (I/O Permission Bitmap), що зберігається в сегменті стану задачі. Несанкціонована спроба виконання цих інструкцій викликає виняток 13 (#GP) - порушення захисту (знамените повідомлення «General Protection Error»).

2.4 Переривання і винятки

Переривання і винятки порушують нормальний хід виконання програми для обробки зовнішніх подій або повідомлень про виникнення особливих умов або помилок.

Переривання підрозділяються на апаратні (масковані і не масковані), які викликаються електричними сигналами на входах процесора, і програмні, які виконуються за командою INT xx. Програмні переривання, строго кажучи, перериваннями не є - це лише своєрідний засіб виклику процедур, але процесором вони опрацьовуються як різновид переривань.

Апаратні переривання підрозділяються на ті що маскуються і на немасковані. Процесор може сприймати переривання після виконання кожної команди, довгі ланцюгові команди мають для сприйняття переривань спеціальне вікно. Апаратні переривання визиваються сигналами на входах INTR і NMI, а для процесорів старших поколінь вони можуть приходити по шині APIC.

Переривання, що маскуються, викликаються переходом у високий рівень сигналу на вході INTR (Interrupt Request) при встановленому прапорці дозволу переривань (IF=l). У цьому випадку процесор зберігає в стеку регістр прапорців, скидає прапорець IF і формує два цикли підтвердження переривання, у яких генеруються керуючі сигнали INTA# (Interrupt Acknowledge). Високий рівень

сигналу INTR повинний зберігатися принаймні до підтвердження переривання. Перший цикл підтвердження переривання - холостий, по другому імпульсу INTA# зовнішній контролер переривань передає по шині номер вектора, що обслуговує даний тип апаратного переривання. Переривання з отриманим номером вектора виконується процесором так само, як і програмне. Обробка поточного переривання може бути у свою чергу перервана немаскованим перериванням, а якщо оброблювач переривання установить прапорець IF, то й іншим маскованим апаратним перериванням.

Немасковані переривання виконуються незалежно від стану прапорця IF за сигналом NMI (Non Mascable Interrupt). Високий рівень на цьому вході викликає переривання з типом (вектором) 2, що виконується так само, як і масковане. Його обробка не може перериватися під дією сигналу на вході NMI до виконання команди IRET.

Винятки (Exceptions), або особливі випадки, підрозділяються на відмови, пастки й аварійні завершення.

Відмова (fault) - це виняток, що виявляється й обслуговується *до* виконання інструкції, що викликає помилку. Після обслуговування цього винятку керування повертається знову на ту ж інструкцію (включаючи всі префікси), що викликала відмову. Відмови, що використовуються в системі віртуальної пам'яті, дозволяють, наприклад, підкачати з диска в оперативну пам'ять необхідну сторінку або сегмент.

Пастка (trap) - це виняток, що виявляється й обслуговується *після* виконання інструкції, що його викликає. Після обслуговування цього винятку керування повертається на наступну інструкцію. До класу пасток відносяться і програмні переривання.

Аварійне завершення (abort) - це виняток, що не дозволяє точно встановити інструкцію, що його викликала. Воно використовується для повідомлення про серйозну помилку, таку як апаратна помилка або ушкодження системних таблиць.

Набір і обробка винятків реального і захищеного режимів різні. Під винятки Intel резервує вектори 0-31 у таблиці переривань, проте в PC частина з них перекривається системними перериваннями BIOS і DOS.

Процедура, що обслуговує переривання або виняток, визначається за таблицею переривань за допомогою номера восьмибітового покажчика (вектори) переривання. Номер для програмних переривань задається командою, для маскованих апаратних переривань вводиться від зовнішнього контролера в другому циклі INTA# (або по шині APIC). Немасковане переривання

має фіксований номер, а винятки генерують і передають номер вектора всередині процесора.

Кожному номеру (0-255) переривання або винятку відповідає елемент у таблиці дескрипторів переривань IDT (Interrupt Descriptor Table). У реальному режимі таблиця переривань містить *дальні адреси* (подвійне слово) обслуговуючих процедур і після скидання розташовується, починаючи з нульових адрес. Командою LIDT можна змінювати положення таблиці в межах першого мегабайта, а розмір (0-3FFh) може бути зменшений до 0-07Fh. При спробі обслуговування переривання з номером, що виходить за розмір таблиці, генерується виняток #DF. У захищеному режимі таблиця IDT містить 8-байтові **дескриптори переривань** і може мати розмір від 32 до 256 дескрипторів і розташовуватися в будь-якому місці фізичної пам'яті.

Аналіз умов обслуговування переривань і винятків виконується в такому порядку (за спаданням пріоритету):

1. Перевірка на виняток-пастку налагодження (#DB) за виконаною інструкцією (покроковий режим через прапорець TF або точку зупинки за даними через регістри налагодження).
2. Перевірка на виняток-відмову налагодження (#DB) за наступною інструкцією (точка зупинки за інструкцією через регістр налагодження).
3. Немасковане переривання (апаратне за входом NMI).
4. Масковане переривання (апаратне за входом INTR при IF=I).
5. Перевірка на виняток-відмову сегментації (#NP або #GP) при вибірці наступної інструкції.
6. Перевірка на виняток-відмову сторінки (#PF) при вибірці наступної інструкції.
7. Перевірка на відмову декодування наступної інструкції (#UD або #GP).
8. Для операції WAIT перевірка TS і *MP* (виняток #NM, якщо TS=1 і MP=1*).*
9. Для операції ESCAPE (інструкція математичного співпроцесора) перевірка EM і TS (виняток #NM, якщо EM=1 або TS=1).
10. Для операції *WAIT* або *ESCAPE* перевірка на виняток #MF від співпроцесора.
11. Перевірка на відмову сегментації (#NP, #SS, #GP) або відмову сторінки (#PF) для операндів, що використовуються в інструкції.

Подвійна відмова (Double Fault) - виняток #DF - виникає, коли при обробці винятку, пов'язаного із сегментацією (#TS, #NP, #SS або #GP), процесор виявляє виняток, відмінний від відмови сторінки

(#PF). Також подвійна відмова виникає, якщо при обробці винятку відмови сторінки #PF виявляється виняток іншого типу. У цьому випадку теж виконується виняток #DF.

Таблиця 2.5- Переривання і винятки захищеного режиму

Номер	Функція	Мнемоніка	Тип
0	Переповнення при ділені на «0»	*#DE*	*Fault*
1	Виняток налагодження	*#DB*	*Fault/Trap*
2	Немасковане переривання (NMI)	-	*Interrupt*
3	Виняток налагодження (INT 3)	*#BP*	*Trap*
4	Виняток по переповненню (INTO)	*#OF*	*Trap*
5	Переривання по контролю діапазону (BOUND)	*#BR*	*Fault*
6	Недопустимий код операції	*#UD*	*Fault*
7	Співпроцесор недоступний або перемикалась задача	*#NM*	*Fault*
8[1]	Подвійна відмова	*#DF*	*Abort*
9	Порушення границі сегмента співпроцесором (тільки 386/387)	-	*Fault*
10[2]	Неприпустимий сегмент стану задачі	*#TS*	*Fault*
II[2]	Сегмент відсутній	*#NP*	*Fault*
12[3]	Порушення границі сегменту стека або сегмент стека відсутній	*#SS*	*Fault*
13[4]	Загальне порушення захисту	*#GP*	*Fault*
14[2]	Відмова сторінки	*#PF*	*Fault*
15	Зарезервовано	-	-
16	Виняток співпроцесора	*#MF*	*Fault*
17[2]	Контроль вирівнювання (486+)	*#AC*	*Fault*
18[2]	Машинний контроль (Р5+)	*#MC*	*Abort*
19-31	Зарезервовано	—	—
0-255	*Апаратні (масковані) і програмні переривання INT n*	—	*Trap*

[1]В реальному режимі — вектор переривання не попадає в таблицю.

[2]В реальному режимі не виникають, але можливі в V86.

[3]В реальному режимі —порушення границі сегменту стека.

[4]В реальному режимі —порушення границі сегменту даних або кода.

Якщо під час обслуговування винятку відмови сторінки відбудеться ще одна відмова сторінки, то виконується **аварійна зупинка** (Shutdown) процесора. Під час аварійної зупинки ніякі нові інструкції не виконуються. З цього стану процесор можна вивести тільки апаратно сигналом NMI, залишаючи його в захищеному

режимі, або сигналом RESET, який переводить процесор у реальний режим.

Переривання і винятки захищеного режиму процесора приведені в табл. 2.5. При обробці винятків процесор записує в стек слово коду помилки (Error Code) – “0” – помилка відсутня, інакше селектор дескриптора з яким пов’язана помилка.

2.5. Математичний співпроцесор

Математичний співпроцесор призначений для розширення

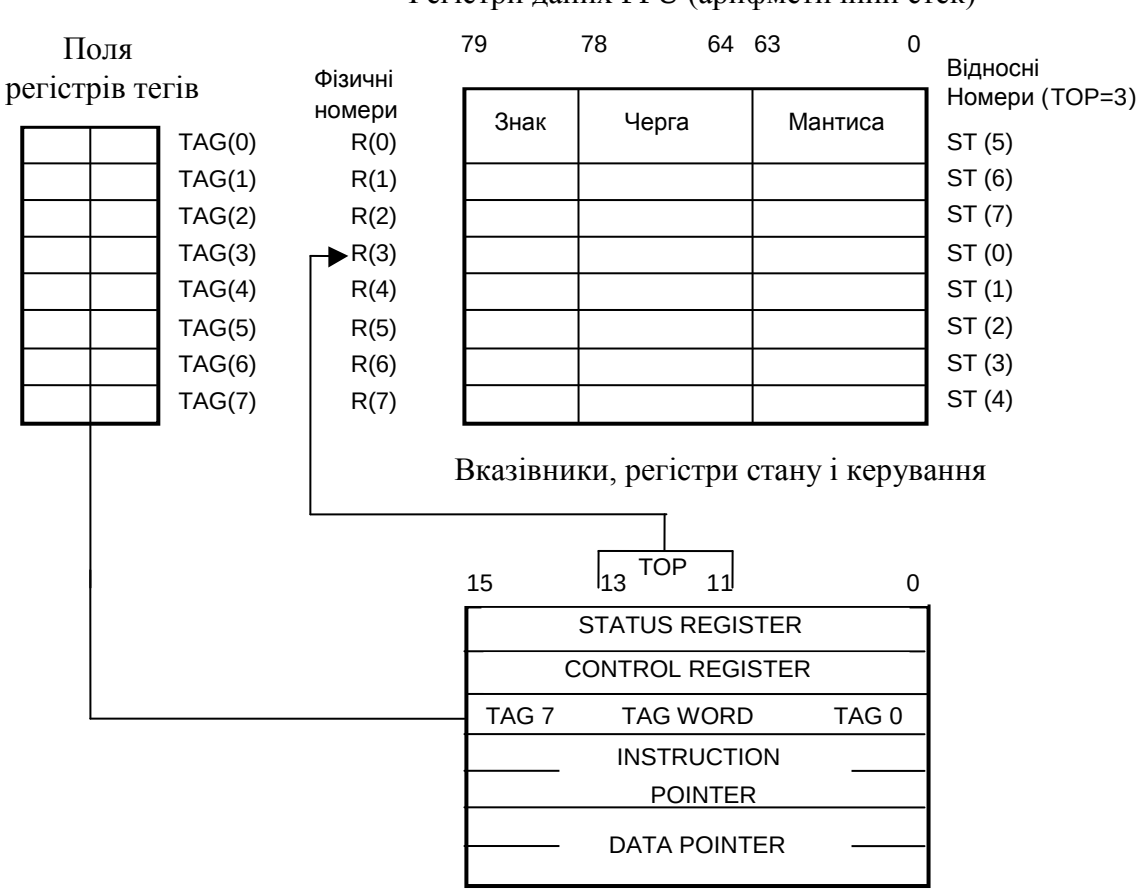


Рис. 2.6 - Регістри математичного співпроцесора

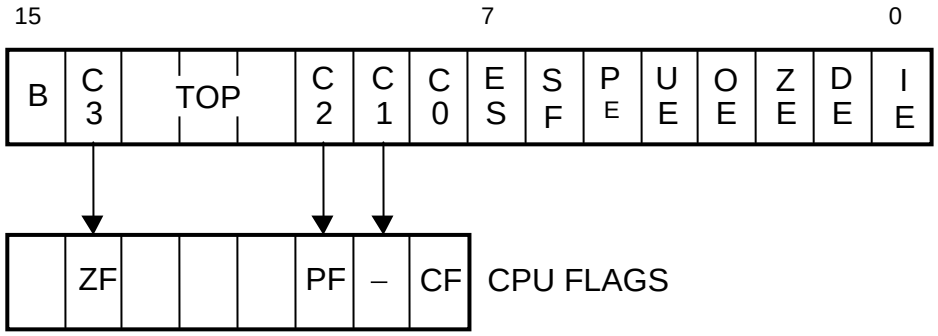


Рис. 2.7 - Слово стану співпроцесора

обчислювальных можливостей центрального процесора - виконання арифметичних операцій, обчислення основних математичних функцій (тригонометричних, експоненти, логарифма) і т.д. У різних поколіннях процесорів він називався по-різному - FPU (Floating Point Unit - процесор чисел із плаваючою точкою) або NPX (Numeric Processor extension - числове розширення процесора). Співпроцесор підтримує сім типів даних; 16-, 32-, 64-бітові цілі числа; 32-, 64-, 80-бітові числа з плаваючою точкою і 18-розрядне число в двійково-десятковому форматі. Формат чисел із плаваючою точкою відповідає стандартам IEEE 754 і 854. Застосування співпроцесора підвищує продуктивність обчислень у сотні раз. З програмної точки зору співпроцесор і процесор виглядають як єдине ціле. Фізично співпроцесор може бути окремою мікросхемою (387 і молодше), що під'єднується до локальної шини основного процесора, або розташовуватися прямо на кристалі центрального процесора (486+). У будь-якому випадку співпроцесор виконує тільки свої специфічні команди, а всю роботу з декодування інструкцій і доставці даних здійснює CPU. Співпроцесор може виконувати обчислення паралельно з центральним процесором, не звертаючи увагу і на переключення задач у захищеному режимі. Як і основний процесор, співпроцесор може працювати в реальному або захищеному режимі і переключати розрядність - 16 або 32. Переключення режимів впливає на формат відображення регістрів співпроцесора в оперативній пам'яті, при цьому формат використовуваних внутрішніх регістрів не змінюється.

З програмної точки зору співпроцесор містить блок регістрів даних, регістр керування і групу регістрів стану і покажчиків (рис. 2.6).

Регістри даних R0-R7 розрядністю 80 біт організовані в *стек.* Номер регістра, що є поточною вершиною стека, зберігається в полі *TOP* керуючого регістра. Операція *push* зменшує *TOP* на 1 і поміщає дані в регістр, що є новою вершиною стека. Операція *pop* записує дані з вершини стека в пам'ять і інкрементує покажчик. Інструкції адресують регістри або явно, або неявно. Неявна адресація має на увазі операнд, що знаходиться у вершині стека. Явна адресація вказує зсув регістра щодо вершини стека (на рис. 2.6 - номера STi). Внутрішня шина має 84 розряди для прискорення виконання операцій, але в програмній моделі важлива зовнішня логічна структура. З кожним регістром пов'язане двобітове *поле тегів* (tag field), призначене для швидкого аналізу стану регістра. Комбінація біт 00 (Valid) указує просто на наявність операнда в регістрі, 01 (Zero) - на його нульове значення, 10 (Special) - спеціальне призначення, 11 (Empty) - регістр порожній. Поля тегів об'єднані в одне *слово тегів* (Tag Word). Теги модифікуються тільки

співпроцесором - при відновленні їх з образа регістрів, збереженого в пам'яті, використовується тільки інформація «порожній» (11) або «непустий» (00, 01, 10). Значення тега для непустого регістра визначається співпроцесором за фактичними даними.

Слово стану (Status Word) відбиває загальний стан співпроцесора (мал. 2.7). Одиничне значення біта 8 (Busy) указує на зайнятість виконанням операції або наявність необслуженого запиту на переривання винятку (цей біт збережений винятково для забезпечення сумісності з 8087). Поле TOP указує на вершину стека. Біти *C[0:3]* визначають код умови (Condition Code), що інтепретується в залежності від виконаної інструкції. Якщо після виконання інструкції *FSTSWAX, що* пересилає слово стану NPX у регістр *AX,* виконати інструкцію *SAHF,* біти коду умови переносяться в регістри прапорців процесора, що дозволяє використовувати їх у інструкціях умовних переходів. Нові інструкції FPU P6 *(FCOMI, FCOMIP, FUCOMI* і *FUCOMIP)* змінюють прапори регістра *EFLAGS* безпосередньо, а інструкції *FCMOVEcc* дозволяють виконувати умовні пересилки в залежності від стану цих прапорців. У такий спосіб із програмного коду виключаються зайві інструкції і розгалуження.

Біт *ES* (Error Summary Status) встановлюється при виникненні немаскованого винятку, при цьому формується і сигнал на виході FERR# (ERROR# у 80287/387).

Біт SF (Stack Flag) встановлюється при некоректній операції зі стеком (переповнювання зверху або знизу).

Біти 0-5 установлюються при виникненні відповідних винятків. Слово стану 8087 мало деякі відмінності: біт 7 із тим же призначенням називався *IR* (Interrupt Request), а прапорець SF був відсутнім (біт 6 не використовувався).

Керуюче слово (Control Word.) служить для вибору опцій виконуваних операцій (рис. 2.8).

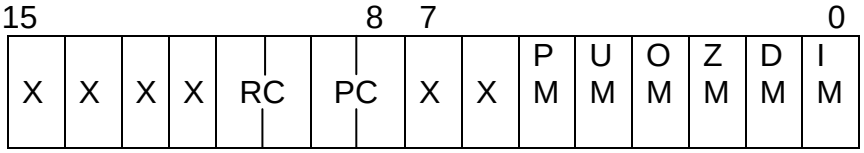


Рис. 2.8. *Керуюче слово співпроцесора*

Одиничні значення біт 0-5 маскують окремі винятки. Поле *PC* (Precision Control) задає точність: 00 - 24 біти (одинарна), 10 - 53 біта (подвійна), 11-64 біта (розширена), 01 - зарезервовано. Поле *RC* (Rounding Control) визначає засіб округлення: 00 - до найближчого значення, 01 - у напрямку до -∞, 10 - у напрямку до +∞, 11- у напрямку до 0. Співпроцесор 8087 мав біт загальної маски

переривання (біт 7), а біт 12 - *IC* (Infinity Control) - управляв представленням нескінечнності: 0 - афінне, 1 - проекційне представлення. У співпроцесорах 80287+ представлення нескінечнності завжди афінне.

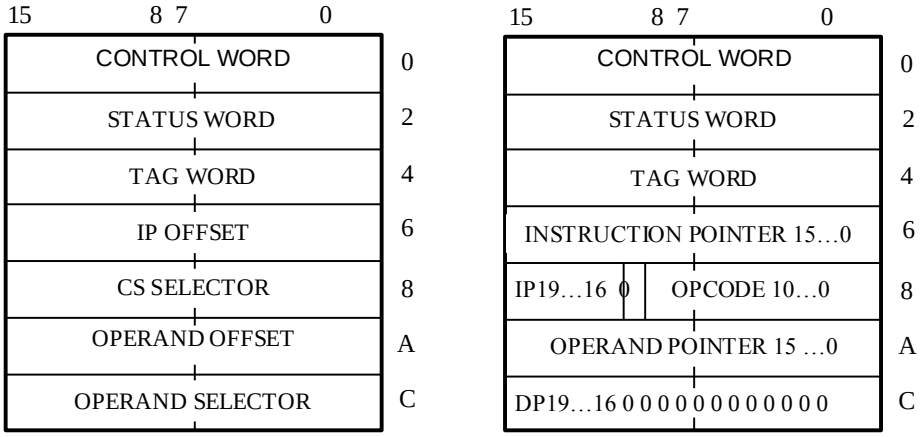
Покажчики інструкцій і даних служать для зберігання фізичної адреси, коду виконуваної операції і фізичної адреси операнда. Від двобайтового коду операції зберігаються тільки 11 молодших біт (5 старших завжди рівні 11011). У реальному і захищеному режимах роботи співпроцесора покажчики мають різне призначення. Формати також залежать і від розрядності операндів. Формати образів покажчиків у 16-бітовому реальному і захищеному режимах приведені на рис. 2.9, *а, б.* 32-бітові формати приведені на рис. 2.10. Співпроцесори 80287+ зберігають у покажчику інструкцій адресу, що вказує на всі префікси інструкції, у той час як у 8087 адреса вказувала на саму інструкцію.

У процесі виконання інструкцій співпроцесора можуть виникати особливі випадки - ***винятки.*** Кожному типу винятків відповідає прапорець регістра стану і біт маски в керуючому регістрі (див. табл. 2.6). За допомогою масок можна забороняти генерацію винятків за різноманітними класами умов (одиничне значення біта маскує виняток). При виникненні умови винятку встановлюється прапорець даного винятку в регістрі стану. Прапорці накопичуються в регістрі до явного скидання за інструкцією *FCLEX* або *FNCLEX.* Незамаскований виняток викликає й установку загального прапорця *ES,* а також апаратний сигнал помилки на виході співпроцесора.

Таблиця 2.6-Винятки співпроцесора

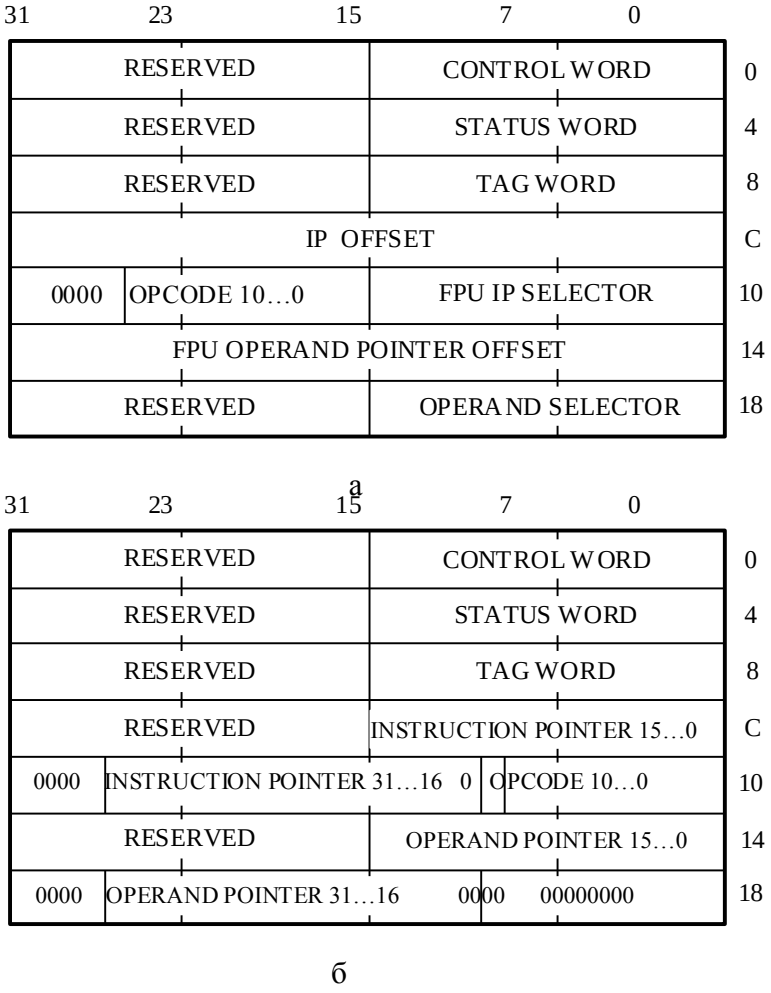
Прапорець	Маска	Виняток
IE	*IM*	Invalid Operation – недійсна операція: витяг даних із порожнього регістра, невизначений результат (0/0 або ∞/∞), квадратний корінь з негативного числа, операція з нечисловыми перемінними і т.п. Якщо встановлений прапорець *SF*, то виняток пов'язаний із переповненням стека зверху або знизу
DE	*DM*	Denormalized Operand – денормалізований хоча б один з операндів
ZE	*ZM*	Zero Divide – ділення на нуль ненульового операнда
OE	*OM*	Overflow – переповнення зверху
UE	*UM*	Underflow – переповнення знизу (зникнення порядку)
PE	*PM*	Precision – результат не може бути точно поданий у заданому форматі, при цьому виконується округлення

Виникнення замаскованого винятку тільки фіксується в регістрі стану, не викликаючи переривання, а співпроцесор виконує відповідну даному типу ***внутрішню*** процедуру опрацювання винятку



а б

Рис. 2.9. Образи покажчиків сопроцесора в 16-бітному форматі: а – реальний режим; б – захищений режим.



а

б

Рис. 2.10. Образи покажчиків сопроцесора в 32-бітному форматі: а – реальний режим; б – захищений режим.

для продовження роботи. Незамаскований виняток опрацьовується *зовнішньою* процедурою. Зовнішня процедура опрацювання винятків виконується центральним процесором за сигналом від співпроцесора і може зчитати в пам'ять вміст регістрів покажчиків разом із словами керування, стану і тегів. Процесор аналізує стан прапорця ES при виконанні наступної інструкції *WAIT/FWAIT* або іншої інструкції FPU, що *очікує* (waiting). Інструкції, у яких мнемоніка має префікс «F» є інструкціями, що очікують; для ряду інструкцій FPU є версії з префіксом «FN», при їхньому виконанні прапорець ES не аналізується.

Опрацювання винятків зовнішніми процедурами для процесорів 486+ можливо в двох режимах. Природний для процесора режим включається установкою прапорця *NE* регістра CR0. При цьому незамаскований виняток співпроцесора викликає виняток процесора *#MF* - відразу після виникнення, перед такою інструкцією FPU, що очікує або перед інструкцією MMX. Виняток викликає відповідну процедуру опрацювання *#MF* за вектором 16 (10h).

Спосіб, природний для IBM PC, вибирається скиданням прапорця *NE.* У цьому випадку виняток *#MF* процесором внутрішньо не формується, а зовнішній оброблювач може бути викликаний за апаратним перериванням від сигналу FERR#.

Тепер поводження процесора у випадку виявлення співпроцесором умови немаскованого винятку буде визначатися вхідним сигналом IGNNE#. Якщо він пасивний, то процесор призупиниться (відразу або перед наступною інструкцією FPU) для реакції на апаратне переривання за сигналом FERR#. Якщо сигнал IGNNE# активний, то припинення не буде (але сигнал FERR# формується). Цей спосіб іде коренями ще в PC AT-286, де апаратний сигнал помилки співпроцесора заводився на вхід контролера переривань IRQ13, викликаючи у випадку винятку співпроцесора *переривання 117* (75h). Таке рішення, очевидно, було прийнято щоб уникнути конфлікту з відеосервісом *(BIOS INT 10h),* що разом з іншими сервісами BIOS розроблювачі PC визначили на зарезервовану (але майже невикористану в 8086) область векторів переривань. З погляду програмування обчислень цей спосіб гірше, оскільки масковане переривання може вироблятися *асинхронно* стосовно процесу, виконуваному FPU. Виняток же формується *синхронно* наступною інструкцією співпроцесора, що полегшує визначення пов'язаного з ним контексту. Проте з погляду програмування більш частих, чим обчислення, операцій з екраном (відеосервіс BIOS *INT 10h* саме перекриває виняток *#MF, що* має номер 16=10h) використовувати штатні винятки *#MF* невигідно,

оскільки оброблювач, розбираючись із джерелом переривання або винятку, буде витрачати занадто багато часу.

З інструкціями, що відносяться до співпроцесора, також можуть бути пов'язані винятки 7 (#NM - емуляція співпроцесора або перемикання задачі), *9* (порушення межі сегмента) і *13* (#GP - порушення захисту).

Інтерфейс взаємодії співпроцесора з основним процесором істотно змінювався двічі.

Співпроцесор 8087 для 8086/8088 і 80186/80188 під'єднується паралельно практично до всіх интерфейсних сигналів центрального процесора. Відслідковуючи сигнали стану CPU, співпроцесор разом із ним переглядає і декодує інструкції, "виловлюючи" із них свої. Якщо команда виконує обмін даними з пам'яттю, CPU «допомагає» співпроцесору в обчисленні адреси, звільняючи його від усіх хитростей формування фізичної адреси, прийнятих у 8086. Після відпрацьовування всіх циклів передачі процесор переходить до виконання наступної інструкції, а співпроцесор починає обчислення. Після обчислювальної інструкції, що виконується співпроцесором, повинна слідувати команда *WAIT* (або *FWAIT,* що те саме), за якою процесор чекає низького рівня сигналу на вході TEST#, пов'язаного з виходом BUSY 8087. Сигнал зайнятості BUSY виробляється співпроцесором на час виконання обчислення, чим і забезпечується його синхронізація. Команду *WAIT* доцільно вводити не відразу після команди обчислення, а безпосередньо перед тим, як будуть потрібні результати обчислень - тоді процесор і співпроцесор можуть якийсь час працювати паралельно. Для сигналізації про винятки використовується сигнал переривання від співпроцесора INT. У IBM PC сигнал переривання від сопроцесора через логічну схему надходить на вхід NMI CPU і викликає немасковане переривання (вектор 2).

Співпроцесори 80287 і *80387* із процесорами 80286 і 80386 обмінюються даними через шинні цикли введення/виведення, що автоматично генеруються CPU. При цьому в просторі введення/виведення використовується область адрес OOF8-OOFOh (386 при цьому виставляє "1"на старшій лінії адреси). Синхронізація, як і раніше, організується за допомогою сигналу BUSY#. Тепер операція *WAIT* вмонтована у всі інструкції, що відносяться до співпроцесора, тому необхідність у застосуванні окремої інструкції *WAIT* перед кожною командою відпала. Незамаскований виняток викликає активний сигнал на виході ERROR#. Цей вихід, за задумом розроблювачів, повинний з'єднуватися з однойменним входом процесора, і тоді при наступній інструкції співпроцесора буде генеруватись виняток #MF (10h). Проте в IBM PC AT цей вихід

заводиться не на процесор, а на вхід вторинного контролера переривань IRQ13, що викликає переривання 117 (75h).

Процесори 486 і старші можуть мати лише *вмонтований співпроцесор,* інтерфейс із зовнішнім співпроцесором у них не передбачений. Тепер для формування винятків не потрібен зовнішній апаратний інтерфейс. Проте для сумісності з архітектурою IBM PC процесори 486+ мають у регістрі *CRO* прапорець *NE* і пару сигналів FERR# і IGNNE#. Сигнал помилки співпроцесора FERR# аналогічний сигналу ERROR# співпроцесора 287/387 - він формується у випадку виникнення немаскованого винятку. Вхідний сигнал IGNNE# змушує процесор ігнорувати помилки співпроцесора (не призупинятися на них).

2.6. Технологія MMX

Технологія MMX орієнтована на додатки мультимедіа, 2D/3D-графіку і комунікації. Це розширення базової архітектури з'явилося тільки після другого покоління процесорів Pentium. Основна ідея MMX полягає в одночасному опрацюванні декількох елементів даних за одну інструкцію - так звана технологія SIMD (Single Instruction - Multiple Data). Розширення MMX використовує нові типи упакованных 64-бітових цілочислених даних:

- упаковані байти (Packed byte) - вісім байтів;
- упаковані слова (Packed word) - чотири слова;
- упаковані подвійні слова (Packed doubleword) - два подвійних слова;
- учетверене слово (Ouadword) - одне слово.

Ці типи даних можуть спеціальним чином опрацьовуватися в регістрах MMXO-MMX7, які є молодшими бітами стеку 80-бітових регістрів FPU. Як і регістри FPU, ці регістри не можуть використовуватися для адресації пам'яті. Збіг регістрів MMX і FPU накладає обмеження на чергування кодів FPU і MMX - турбота про це лежить на програмісті додатків із MMX.

Ще одна особливість технології MMX - підтримка *арифметики з насиченням* (saturating arithmetic). Її відмінність від звичайної арифметики з циклічним переповнюванням (wraparound mode) полягає в тому, що при виникненні переповнення в результаті фіксується максимально можливе значення для даного типу даних, а перенесення ігнорується. У випадку переповнення знизу в результаті фіксується мінімально можливе значення. Граничні значення визначаються типом (знаковий або беззнаковий) і розрядністю

перемінних. Такий режим обчислень зручний, наприклад, для задач цифрової обробки зображення та мовних сигналів.

У систему команд введено 57 додаткових інструкцій для одночасного опрацювання декількох одиниць даних. Одночасно обробляється 64-бітове слово, яке може містити як одну одиницю для обробки, так і 8 однобайтових, 4 двобайтових або 2 чотирибайтових операнди. Нові інструкції включають такі групи:

- арифметичні (Arithmetic Instructions), що включають додавання і віднімання в різних режимах, множення і комбінацію множення і додавання;
- порівняння (Comparison Instructions) елементів даних на рівність або за величиною;
- перетворення форматів (Conversion Instructions);
- логічні (Logical Instructions) - І, І-НІ, АБО і додавання за модулем “2”, які виконуються над 64-бітовими операндами;
- зсуви (Shift Instructions) - логічні й арифметичні;
- пересилки даних (Data Transfer Instructions) між регістрами ММХ і цілочисленими регістрами або пам'яттю;
- очищення ММХ (Empty MMX State) - установка ознак порожніх регістрів у слові тегів.

Інструкції ММХ не впливають на прапорці умов.

Регістри ММХ на відміну від регістрів FPU адресуються фізично, а не щодо значення покажчика стека *TOP.* Більш того, будь-яка інструкція ММХ занулює поле TOP регістра стана FPU. У слові тегів вільному регістру відповідає комбінація 11, інші комбінації вказують тільки на зайнятість регістра. Після кожної операції ММХ біти тегів, використовуваного регістра призначення, занулюються. Біти [79:64] регістрів FPU (не використовуються ММХ) заповнюються одиницями, так що помилкове використання даних ММХ інструкцією FPU призведе до винятку.

Інструкції ММХ не породжують нових винятків. Винятки при виконанні інструкцій ММХ можуть виникати тільки у випадку порушення меж в звертаннях до пам'яті (як при обміні даними, так і по вибірці інструкції). Проте якщо попередня інструкція FPU породила умову винятку, то він відбудеться при виконанні інструкції ММХ. Після його опрацювання інструкція ММХ може бути благополучно виконана.

Інструкції ММХ доступні з будь-якого режиму процесора. При переключенні задач необхідно стежити за коректністю зберігання контексту, як і при роботі з FPU.

Часте чергування кодів FPU і ММХ може знизити продуктивність за рахунок необхідності зберігання і відновлення дуже об'ємного контексту FPU.

Наявність підтримки MMX визначається за бітом 23 регістра *EDX* після виклику інструкції *CPUID (1).*

Технологія *3DNow! ,* введена фірмою AMD у процесорах K6-2, розширює можливості MMX. Вона дозволяє оперувати з новим типом даних - парою упакованих чисел у форматі з плаваючою точкою. Ці числа займають по подвійному слову в 64-бітових регістрах MMX. Процесор K6-2 має два виконавчих блоки, що спроможні одночасно виконувати операції з плаваючою точкою над своїми регістрами. Кожна така операція займає усього два такти, операції цілком конвейєризовані. Таким чином, із конвеєрів процесора за кожний такт можуть сходити чотири результати операцій із плаваючою точкою. У систему команд додана 21 нова інструкція MMX, велика частина яких призначена для опрацювання упакованих чисел із плаваючою точкою. Є і нова цілочислена MMX-інструкція усереднення восьми пар 8-бітових чисел, призначена для декодерів MPEG-2. Крім того, є команда швидкого переключення FPU-MMX і попередньої вибірки в первинний кеш даних. Технологія 3DNow! дає помітний результат при опрацюванні графіки, хоча не претендує на витиснення графічних акселераторів, а покликана служити їхнім потужним доповненням. При цьому зберігається програмна сумісність із старими процесорами й операційними системами (оскільки регістри MMX відображаються на регістри FPU, для них працюють механізми зберігання контексту в багатозадачних ОС). Підтримка 3DNow! визначається за бітом 31 регістра *EDX* після виклику *CPUID(8000_0001h).*

2.7. Типи даних

32-розрядні процесори безпосередньо підтримують (використовують у якості операндів) знакові і безнакові цілі числа, рядки байтів, цифр і символів, бітові рядки, покажчики і числа з плаваючою точкою (рис. 2.11). Розглянемо ці типи докладніше:

- *Біт* (Bit) - одиниця інформації. Біт у пам'яті задається *базою* (адресою слова) і *зсувом* (номером біта в слові).
- *Бітове поле* (Bit Field) - група до 32 суміжних бітів, що розташовуються не більше ніж у 4 байтах.
- *Бітовий рядок* (Bit String) - набір суміжних бітів довжиною до 4 Гбіт.
- *Байт* (Byte) - 8 бітів.
- *Числа без знака:* байт/слово/подвійне/учетверене слово (Unsigned Byte/Word/Double Word/Quade Word), 8/16/32/64 біти.
- *Ціле число зі знаком:* байт/слово/подвійне/учетверене слово (Integer Byte/Word/Double Word/Quade Word). Одиничне значення

найстаршого біта (знак) є ознакою негативного числа, що зберігається в додатковому коді.

- *Дійсні числа у форматі з плаваючою точкою* (опрацьовуються співпроцесором FPU):
  - одиночної точності (Single Precision), 32 біта - 23 біти мантиса, *8 бітів порядок;*
  - подвійної точності (Double Precision), 64 біти - 52 біти мантиса, 11 біт порядок;
  - підвищеної точності (Extended Precision), 80 біт - 64 біти мантиса, 15 біт порядок.
- *Двійково-десяткове число* (BCD Binary Coded Decimal):
  - 8-розрядні упаковані (Packed BCD), що містять два десяткових розряди в однім байті;
  - 8-розрядні неупаковані (Unpacked BCD), що містять один десятковий розряд у байті;
  - 80-розрядні упаковані (опрацьовуються тільки співпроцесором).
- *Рядки* байтів, слів і подвійних слів (Bit String, Byte String, Word String, Double Word String).
- *Покажчики:*
  - довгий покажчик (48 біт) - 16-бітовий селектор (або сегмент) і 32-бітовий зсув;
  - короткий покажчик - 32-бітовий зсув;
  - просто покажчик (32 біти, єдиний тип покажчика для 8086 і 80286) - 16-бітовий селектор (або сегмент) і 16-бітовий зсув.

Числа у форматі з плаваючою точкою й упаковані 80-бітовими BCD-числами опрацьовуються співпроцесорами 8087/287/387 і умонтованими FPU процесорів класу 486 і вище. Процесори MMX опрацьовують і декілька інших типів упакованих 64-бітових даних (див. п. 2.6). 16-розрядні процесори з приведених типів даних не підтримують учетверені слова всіх типів, бітові поля і рядки, рядки подвійних слів, короткі і довгі покажчики.

У сімействі 80x86 прийнято, що слова записуються в двох суміжних байтах пам'яті, починаючи з молодшого. *Адресою слова* є адреса його молодшого байта. Подвійні слова записуються в чотирьох суміжних байтах, починаючи з молодшого байта, адреса якого і є адресою подвійного слова. Цей порядок називається Little-Endian Memory Format. У інших сімействах процесорів застосовують і обернений порядок - Big-Endian Memory Format, у якому адресою слова (подвійного слова) є адреса його старшого байта, а молодші байти розташовуються в наступних адресах. Для взаємного перетворення форматів слова є інструкція *XCHG,* подвійного слова - *BSWAP* (486+).

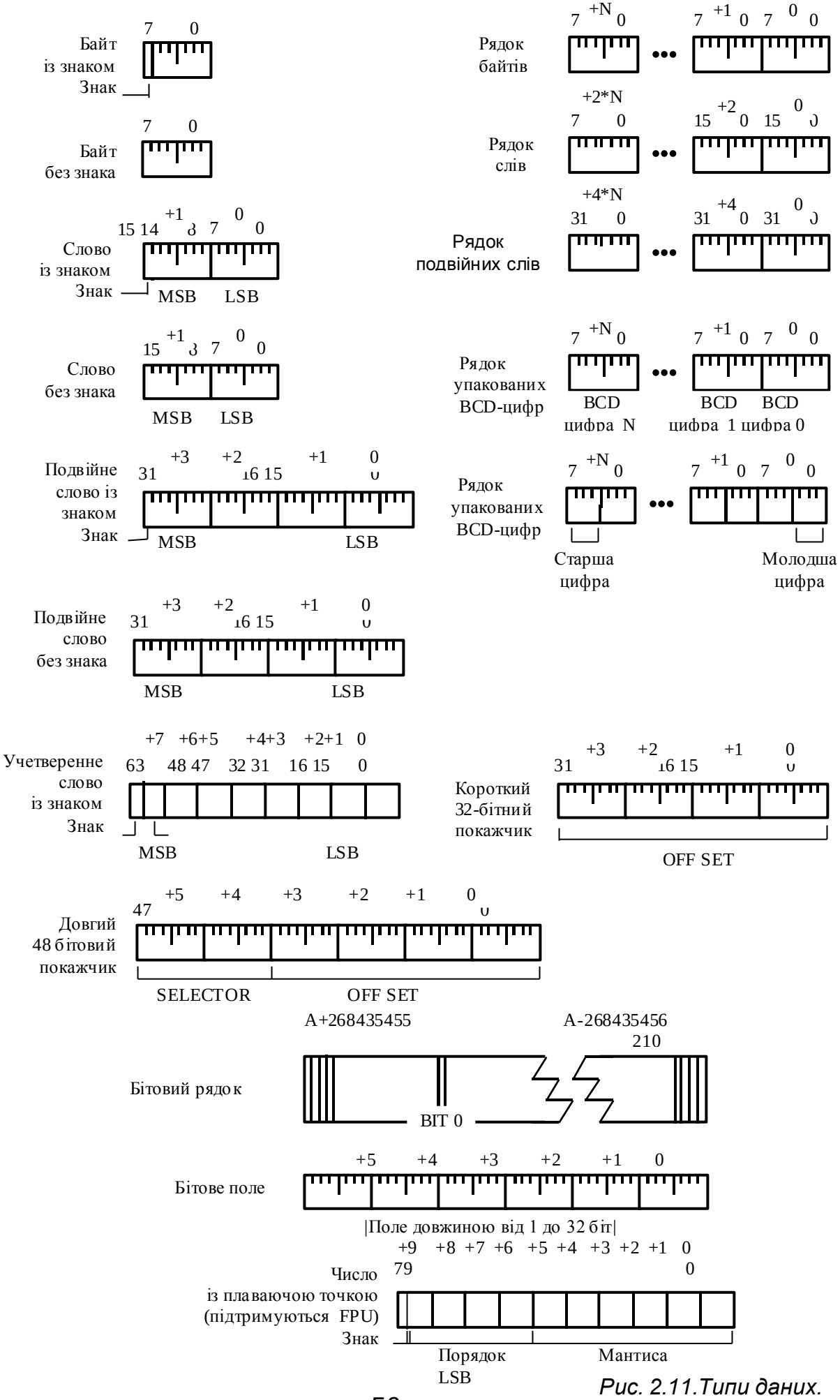


Рис. 2.11.Типи даних.

2.8. Система команд

Система команд 32-розрядних процесорів є істотно розширеною системою команд 8086/80286. Розширення стосуються збільшення розрядності адрес і операндів, більш гнучкої системи адресації, появи принципово нових типів даних (бітові рядки і поля) і команд.

Команди (інструкції) містять одно- або двобайтовий код інструкції, за яким можуть слідувати декілька байтів, що визначають режим виконання команди й операнди. Команди можуть використовувати до трьох операндів (або ні одного). Результат виконання записується в місце, визначене інструкцією, або (і) у місце, задане операндом призначення. Операнди можуть знаходитися в пам'яті, регістрах процесора або безпосередньо в команді. Для 32-розрядних процесорів розрядність «слова» (word) за замовчанням може складати 32 біти, а не 16. Це поширюється на багато інструкцій, включаючи і рядкові. У реальному режимі і режимі віртуального 8086 за замовчуванням використовується 16-бітова адресація і 16-бітові операнди-слова. У захищеному режимі режим адресації і розрядність слова за замовчуванням визначаються дескриптором кодового сегмента. Перед будь-якою інструкцією може бути використаний *префікс переключення розрядності адреси* або *слова.* При адресації пам'яті використання сегментного регістра, передбаченого командою, у ряді інструкцій може змінюватися *префіксом зміни сегмента* (Segment Override). Призначення префіксів розкрите в табл. 2.7. Префікси можуть використовуватися в будь-якому сполученні (не більш одного з кожної групи), їхня дія поширюється тільки на одну інструкцію.

Таблиця 2.7- Призначення префіксів інструкцій

Префікс	Призначення
SIZ	Зміна розрядності слова даних (386+)
ADDRSIZ	Зміна розрядності адреси (386+)
CS: SS: DS: ES: FS: GS:	Зміна сегментного регістра (FS. і GS: тільки для 386+)
LOCK	Захоплення локальної шини на час виконання інструкції
REP REPB/REPZ REPNE/REPNZ	Префікси повтору рядкових операцій до онулення *CX (CX* декрементується на кожному повторі) – безумовного й умовного (при *ZF=1* і ZP=0 відповідно)

Інструкції пересилки даних (табл. 2.8, додаток 2) дозволяють передавати константи або перемінні між регістрами і пам'яттю, а також портами введення/виведення в різноманітних комбінаціях,

але в пам'яті може знаходитися не більш одного операнда. У цю групу віднесені й інструкції перетворення форматів - розширень і перестановки байтів. Операції зі стеком виконуються словами з розрядністю, обумовленою поточним режимом. При запису в стек слова покажчик стека SP зменшується на число байтів слова (2 або 4), при читанні - збільшується. 8086 інструкції пересилки не впливають на вміст регістра прапорців. Інструкції пересилки за результатами порівняння *(CMPXCHG)* модифікують прапорець ZF. Нові інструкції умовної пересилки *(CMOVxx)* дозволяють скоротити число розгалужень у програмі.

Інструкції двійкової арифметики (табл. 2.9, додаток 2) виконують всі арифметичні дії з байтами, словами і подвійними словами, що кодують знакові або беззнакові цілі числа. Множення і ділення для 8086 можливі тільки з акумулятором, результат для 16-бітових операндів розширюється в *DX.* Для 286+ можливо дво- і триадресне множення з розширенням тільки в старший байт (два байти для 386+).

Інструкції десяткової арифметики є доповненням до попередніх. Вони дозволяють оперувати з неупакованими (біти [7:4]=0, біти [3:0] містять десяткову цифру 0-9) або упакованими (біти [7:4] містять старшу, біти [3:0] - молодшу десяткову цифру 0-9) двійково-десятковими числами. Арифметичні операції над цими числами потребують застосування інструкцій корекції форматів (табл. 2.10, додаток 2).

Інструкції логічних операцій (табл. 2.11, додаток 2) виконують усі функції булевої алгебри над байтами, словами або подвійними словами.

Зсуви й обертання (циклічні зсуви) (табл. 2.12, додаток 2) виконуються над регістром або операндом у пам'яті. Число позицій, на які виконується зсув, береться з безпосереднього операнда або регістра *CL* за модулем 8 для однобайтового операнда і за модулем 16 або 32 для операнда-слова, у залежності від розрядності даних (32 тільки для 386+). Біти, що виштовхуються при зсувах, потрапляють у прапорець *CF.* При зсувах вліво і простому зсуві вправо біти , що звільняються , заповнюються нулями *(*інструкції *SAL* і SHL - синоніми). При арифметичному зсуві вправо старший біт (знак) зберігає своє значення. При циклічних зсувах біти, що виштовхуються потрапляють і в прапорець CF, і в позиції , що звільняються. У зсувах можуть брати участь і два операнди (інструкції *SHLD* і *SHRD).*

Інструкції опрацювання біт і байт (табл. 2.13, додаток 2) дозволяють перевіряти (копіювати в CF) і встановлювати значення указаного операнда, а також шукати встановлений біт. Бітові операції виконуються над 16- або 32-бітовим словом в пам'яті або

регістром. Інструкції BSF, *BSR* і *BT* не змінюють значення слова; BTC, *BTR* і BTS впливають на зазначений біт слова r/m. Номер біта береться з операнда за модулем 16 або 32, у залежності від розрядності.

Передача керування (табл. 2.14, додаток 2) здійснюється за допомогою інструкцій безумовних і умовних переходів, викликів процедур і переривань (винятків). *Безумовний перехід JMP)* може бути як внутрішньосегментним (ближнім або коротким), так і міжсегментним (дальнім). Адреса переходу може безпосередньо вказуватися в команді, а при непрямій адресації адреса переходу знаходиться в регістрі або пам'яті і може мати додаткові доданки. *Короткий* перехід (short) може передавати керування тільки на адресу призначення, віддалену від поточного в межах -128... +127 байт, *ближній* (near) - у межах сегмента. При *дальньому* (far) переході адреса призначення (безпосередня або непряма) включає нове значення покажчика інструкцій і значення (або селектор) сегмента коду, забезпечуючи доступ до будь-якої точки пам'яті (у межах, дозволених захистом).

Умовні переходи в 8086 і 80286 можливі тільки короткі (8-байтовий зсув), процесори 386+ припускають перехід у межах 16- або 32-байтового зсуву, у залежності від режиму адресації. Умовні переходи виконуються за станом прапорців і/або вмісту регістра CX (ECX). Інструкції циклів комбінують умовний перехід із декрементом регістра *CX(ECX).*

Інструкція *виклику процедури (CALL)* передає керування в точку переходу, зберігаючи адресу наступної за нею інструкції в стеку. За інструкцією повернення *(RET)* ця адреса відновиться в покажчик інструкцій (і в CS при дальньому виклику). Як і безумовний перехід, виклик і повернення можуть бути як внутрішньосегментними (ближніми), так і міжсегментними, допускаючи ті ж режими адресації. Міжсегментні виклики і повернення в захищеному режимі здійснюються через вентилі викликів.

Рядкові операції (табл. 2.15, додаток 2) виконуються з операндами в пам'яті, які адресуються регістрами DS:SI *(DS:ESI)* для джерела і *ES:DI (ES:EDI)* для приймача. Операції можуть використовуватися з префіксами умовного або безумовного повтору. Після кожної пересилки або порівняння індексні регістри (SI, DI або обидва) автоматично інкрементуються або декрементуються на кількість байтів, що беруть участь в операції (1,2 або 4). Напрямок модифікації визначається прапорцем *DF: DF=0* - інкремент, DF=1 - декремент. Рядкові інструкції введення/виведення з префіксами повтору дозволяють досягати

високих швидкостей обміну з портами, правда при повному завантаженні процесора.

Операції з прапорцями (табл. 2.16, додаток 2) дозволяють змінювати значення окремих прапорців, а також зберігати їхнє значення в стеку (або регістрі *AH)* і відновлювати збережені значення.

Інструкції завантаження покажчиків (табл. 2.17, додаток 2) дозволяють завантажувати дальні покажчики з пам'яті в регістр загального призначення і відповідний сегментний регістр. Крім того, із сегментними регістрами можливі стекові операції *PUSH* і POP, а також обмін через регістри загального призначення *(MOV).*

У табл. 2.18 (додаток 2) приведені різні інструкції, що не входять у перераховані вище класи.

Інструкція *ENTER* служить для підготування викликів процедур (із підтримкою вкладеності) у мовах високого рівня. Вона виділяє область перемінних у стеку. Обернена інструкція *LEAVE* відновлює вихідне значення покажчика стека.

Інструкції MMX (табл. 2.19, додаток 2) мають складну мнемоніку, що включає такі елементи:

- префікс *P* (Packed), що вказує на обробку упакованих форматів;
- мнемоніку операції (наприклад, ADD, CMP або XOR);
- суфікс, що вказує тип насичення: *US* (Unsigned Saturation) - насичення беззнакове, S (Signed saturation) - насичення знакове;
- суфікс, що вказує тип даних: B - упаковані байти, *W* - упаковані слова, D - упаковані подвійні слова, Q - учетверене слово.

Інструкції, у яких типи вхідних і вихідних даних різняться (наприклад, перетворення), мають два суфікси.

Для інструкцій пересилки даних операнд джерела і призначення можуть знаходитися в пам'яті *(m32* або *m64),* цілочислених регістрах *(ir32)* або регістрах MMX *(mm).* Для інших інструкцій, крім перерахованих вище, операнд-джерело може бути і безпосереднім, а операнд призначення завжди є регістром MMX. Для операндів, що знаходяться в пам'яті, застосовні всі режими адресації, що існують.

Інструкції розширення 3DNowl (табл. 2.20, додаток 2) з'явилися з процесорами AMD K6-2. Ці інструкції стають дуже популярними, хоча процесори Intel, яким присвячена дана книга, їх не підтримують.

Інструкції математичного співпроцесора (FPU) (табл. 2.21, додаток 2), мають свою специфіку завдання операндів. Перемінна *st(0)* знаходиться на вершині стека співпроцесора, *st(i)* зсунута від вершини на *i.* Загрузка даних починається з декремента покажчика стека співпроцесора (поле TOP) - переміщення вершини. Якщо нова вершина не порожня (по полю *TAG)* або стек вичерпаний,

визивається виняток із вказівкою причини. Після завантаження поле *TAG* установлюється відповідно до завантаженого числа. При зчитуванні зі стека виконується інкремент TOP, а в поле *TAG* старої вершини встановлюється ознака порожньої комірки. Спроба використання порожнього регістра в операціях або для зберігання результатів у пам'яті викликає виняток. Інструкції з префіксом «F» попередньо перевіряють прапорець винятку *ES* (вони називаються інструкціями , що очікують), інструкції з префіксом «FN» прапорець винятку не перевіряють (інструкції , що неочікують). Ряд інструкцій не викликає винятки у випадку, якщо виявляються операнди не-числа (NaN).

До системних інструкцій (табл. 2.22, додаток 2) відносяться інструкції керування захистом - завантаження і зберігання регістрів дескрипторів і регістра задачі; перевірки і вирівнювання привілеїв; обміну з керуючими, налагоджувальними і модельно-специфічними (включаючи тестові) регістрами; керування кешуванням, захоплення шини і зупинки процесора.

У табл. 2. 8-2. 21 прийняті такі позначення:

- *reg* ~ 16/32-бітовий регістр;
- *mem* - 16/32-бітова комірка пам'яті, що адресується регістрами процесора;
- *r8, ri6, r32* - тільки 8-, 16- або 32-бітовий регістр;
- *m8, m16, m32, m64* - тільки 8-, 16-, 32- або 64-бітова комірка пам'яті, що адресується регістрами;
- *r/m* - 8/16/32-бітовий регістр або комірка пам'яті, що адресується регістрами процесора;
- *r/m/i* - 8/16/32-бітовий регістр, комірка пам'яті, що адресується регістрами процесора, або безпосередній операнд;
- *i* - безпосередній операнд 8, 16, 32 або 64 біт;
- *i8, i16* - безпосередній операнд 8, 16 бітів;
- *moffs8, moffs16, moffs32* - 8-, 16- або 32-бітова комірка, що адресується в команді (не через регістри процесора);
- *ptr16:16, ptr16:32* - операнд-покажчик: 16-бітовий селектор і 16/32-бітний зсув;
- *m16:16, m16:32* - комірка пам'яті, що містить покажчик;
- *rel8, rel16, rel32* - зсув короткий (127 байт) або ближній всередині сегмента коду щодо адреси наступної інструкції;
- *st(0)* - верхівка стека FPU;
- *st(i)* - регістр стека FPU, зсунутий від верхівки на і;
- *m80bcd* - упаковане 10-байтове двійково-десяткове число в пам'яті, яке використовується FPU;
- *m16int, m33int, m64int* - цілочислені значення в пам'яті, що використовуються FPU;

- *m32real, m64real, m80real* - речовинні числа в пам'яті, що використовуються FPU;
- *m94/108byte* - образ регістрів (контексту) FPU у пам'яті, включаючи стек;
- *m14/28byte* - образ стану (environment) FPU у пам'яті;
- *m16&32* - 6-байтова структура в пам'яті (база і ліміт GDT і IDT).

Для деяких інструкцій у таблицях через косу риску зазначені синоніми, операнд призначення завжди стоїть першим.

Контрольні питання

1. Дайте характериктеристику режимів роботи 32-х розрядних процесорів INTEL.
2. Призначення основних регістрів 32-х розрядних процесорів INTEL.
3. Приведіть формат регістру прапорців і призначення його розрядів.
4. Дайте характеристику керуючих регістрів.
5. Яка організація пам'яті в системах з 32-х розрядними процесорами?
6. Які механізми перетворення адресних просторів використовуються в 32-х розрядних процесорах?
7. Порядок обчислення ефективної адреси при звертанні до пам'яті.
8. Охарактеризуйте механізми введення/виведення 32-х розрядних процесорів.
9. Охарактеризуйте систему переривань 32-х розрядних процесорів.
10. В чому різниця між апаратними та програмними перериваннями?
11. Наведіть типи винятків в 32-х розрядних процесорах INTEL.
12. Які доступні регістри у математичного співпроцесора?
13. Основні типи даних, які використовують 32-х розрядні процесори?
14. Які особливості арифметики використовуються в технології MMX?
15. Охарактеризуйте систему команд 32-х розрядних процесорів.
16. Які особливості використання команд MMX?

3. Захищений режим

Захищений режим Protected Virtual Address Mode є основним (найбільш природним) режимом роботи 32-розрядних процесорів. У цьому режимі процесор дозволяє адресувати до 4 Гбайт (до 64 Гбайт) фізичної пам'яті, через які при використанні механізму сегментації можуть відображатися до 64 Тбайт віртуальної пам'яті кожної задачі. Режим віртуального процесора 8086 - Virtual 8086 Mode або V86 - є особливим станом задачі захищеного режиму, у якому процесор функціонує як 8086 із можливістю використання 32-розрядних адрес операндів.

Захищений режим з'явився ще в процесорі 80286, але мав не всі можливості, доступні в 32-розрядних процесорах. Приведений нижче опис захищеного режиму можливо застосувати і до процесора 80286, але з урахуванням таких обмежень:

- регістри CRn, DRn і TRn відсутні;
- вхід у захищений режим здійснюється тільки після завантаження MSW із PC=1, вихід у реальний - тільки після апаратного скидання;
- режим віртуального процесора 8086 V86 відсутній;
- формати дескрипторів мають 16-бітові поля ліміту і 24-бітові поля базової адреси, що обмежує розмір сегмента до 64 Кбайт, обсяг фізичної пам'яті не перевершує 16 Мбайт, віртуальної - 1 Гбайт; звертання до пам'яті за межею 16 Мбайт призводить до кільцевого «згортання» адрес: звертання до 17-го мегабайту фізично адресує перший (що справедливо і для 386SX);
- режими 32-бітових адрес і даних відсутні;
- блок сторінкової переадресації відсутній (фізична адреса пам'яті еквівалентна лінійній);
- обмеження на операції введення/виведення накладаються тільки через IOPL, бітова карта дозволу введення/виведення відсутня.

3.1. Основні поняття захищеного режиму

Захищений режим призначений для забезпечення незалежності виконання декількох задач, що передбачає захист ресурсів однієї задачі від можливого впливу іншої (задачами будемо вважати як додатки, так і виконання задач операційної системи).

Основним ресурсом, який захищається є пам'ять, у якій зберігаються коди, дані і різноманітні системні таблиці (наприклад, таблиця переривань) . Захищати потрібно і спільно використовувану апаратуру, звертання до якої звичайно виконується через операції введення/виведення і переривання. У захищеному режимі процесор

апаратно реалізує багато функцій захисту, необхідних для побудови супервізора багатозадачної ОС, у тому числі механізм віртуальної пам'яті.

Захист пам'яті грунтується на сегментації. **Сегмент** - це блок простору пам'яті визначеного призначення. До елементів сегмента можливо звертання за допомогою інструкцій процесора, що використовують різні режими адресації для формування адреси в межах сегмента. Максимальний розмір сегмента - 4 Гбайт (для процесорів 8086 і 80286 межа була усього 64 Кбайт). Сегменти пам'яті виділяються задачам операційною системою, але в реальному режимі будь-яка задача може перевизначити значення сегментних регістрів, що задають положення сегмента в просторі пам'яті, і «залізти» у чужу область даних або коду. У захищеному режимі сегменти теж розподіляються операційною системою, але прикладна програма зможе використовувати тільки дозволені для неї сегменти пам'яті,обираючи їх за допомогою селекторів із попередньо сформованих таблиць дескрипторів сегментів.

Процесор може звертатися тільки до тих сегментів пам'яті, для яких є дескриптори в таблицях. Механізм сегментації формує лінійну адресу згідно з схемою приведеною на рис. 3.1. Дескриптори вибираються за допомогою 16-бітових селекторів, які програмно завантажуються в сегментні регістри (рис. 3.2). Поле Index сумісно з індикатором таблиці TI дозволяє вибрати дескриптор з локальної (TI=0) або з глобальної (TI=0) таблиці дескрипторів. Поле RPL (Requested Privilege Level) указує необхідний рівень привілеїв (див. нижче). Для сегментних регістрів, що не використовуються, призначений нульовий селектор сегмента, який формально адресується до найпершого елемента глобальної таблиці. Спроба звертання до пам'яті по такому сегментному регістрі викликає виняток. Виняток виникне і при спробі завантаження нульового селектора в регістр CS або SS.

Дескриптори являють собою 8-байтові структури даних, які використовуються для визначення властивостей програмних елементів (сегментів, вентилів і таблиць). Дескриптор визначає положення елемента в пам'яті, розмір області (ліміт), яку він займає, його призначення і характеристики захисту. Всі дескриптори зберігаються в таблицях, звертання до яких підтримується процесором апаратно.

Захист пам'яті за допомогою сегментації не дозволяє:

- використовувати сегменти не за призначенням (наприклад, намагатися трактувати область даних як коди інструкцій);

- порушувати права доступу (намагатися модифікувати сегмент, призначений тільки для читання, звертатися до сегмента, не маючи достатніх привілеїв, і т.п.);
- адресуватися до елементів, що виходять за ліміт (межу) сегмента:
- змінювати вміст таблиць дескрипторів (тобто параметри сегментів), не маючи достатніх привілеїв.

Захищений режим надає засоби переключення задач. Стан кожної задачі (значення всіх пов'язаних із нею регістрів процесора) може бути збережений в спеціальному сегменті стану задачі TSS (Task Status Segment), на який указує селектор у регістрі задачі TR. При переключенні задач достатньо завантажити новий селектор у регістр задачі, і стан поточної задачі автоматично збережеться в її TSS, а в процесор завантажиться стан нової (можливо, раніше перерваної) задачі, і почнеться (продовжиться) її виконання.

Чотирирівнева ієрархічна система привілеїв призначена для керування використанням привілейованих інструкцій і доступом до дескрипторів. Рівні привілеїв нумеруються від 0 до 3, нульовий рівень відповідає максимальним (необмеженим) можливостям доступу і відводитьсять для ядра операційної системи. Рівень 3 має найобмеженіші права і звичайно надається прикладним задачам. Систему захисту звичайно зображують у виді концентричних кіл, що відповідають рівням привілеїв (рис. 3.3), а самі рівні привілеїв іноді називають колами захисту. Послуги, надані задачам, можуть знаходитися в різних колах захисту. Передача керування між задачами контролюється вентилями (Gate), що називаються також шлюзами, які перевіряють правила використання рівнів привілеїв. Через вентилі, задачі можуть одержати доступ тільки до дозволених їм сервісів (послуг) інших сегментів.

Рівні привілеїв відносяться до дескрипторів, селекторів і задач. Крім того, у регістрі прапорців є поле привілеїв введення/виведення, за допомогою якого забезпечується керування доступом до інструкцій введення/виведення і керування прапорцем переривань. Дескриптори і привілеї є основою системи захисту: дескриптори визначають структури програмних елементів (без яких ці елементи неможливо використовувати), а привілеї визначають можливість доступу до дескрипторів і виконання привілейованих інструкцій. Будь-яке порушення захисту призводить до виникнення спеціальних винятків, які обробляються ядром операційної системи.

Механізм віртуальної пам'яті дозволяє будь-якій задачі використовувати логічний адресний простір розміром до 64 Тбайт (16К сегментів по 4 Гбайт). Для цього кожний сегмент у своєму дескрипторі має спеціальний біт, що вказує на присутність даного

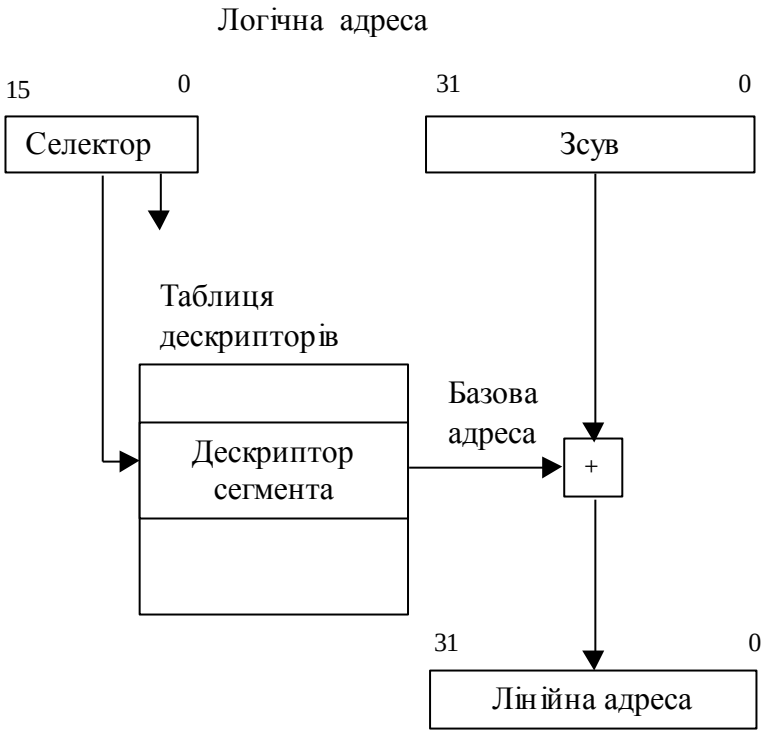


Рис. 3.1. Формування лінійної адреси в захищеному режимі

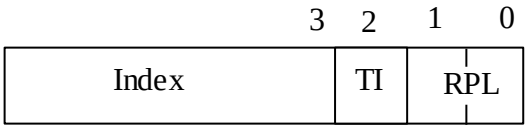


Рис. 3.2. Формат селектора.

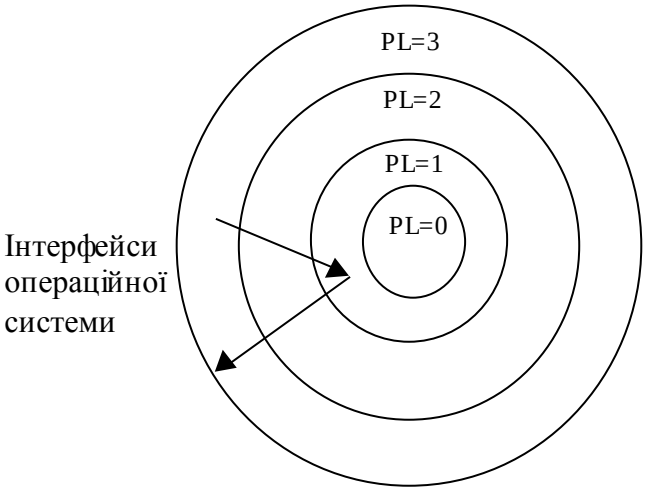


Рис. 3.3. Рівні привілеїв. PL=0 - максимальні привілеї, ядро ОС; PL=1 – системні сервіси; PL=2 – розширення ОС; PL=3 – додатки.

сегмента в оперативній пам'яті в даний момент часу. Сегмент, який не використовується може бути вивантажений з оперативної пам'яті в зовнішню (наприклад, дискову), про що робиться позначка в його дескрипторі. На місце, що звільнилося, із зовнішньої пам'яті може завантажуватися вміст іншого сегмента (цей процес називається свопінгом або підкачуванням), і в його дескрипторі робиться позначка про присутність у пам'яті. При звертанні задачі до відсутнього сегмента процесор виробляє відповідний виняток, оброблювач якого і керує віртуальною пам'яттю в операційній системі. Механізм сторінкової переадресації забезпечує віртуалізацію пам'яті, що адресується логічною адресою, на рівні сторінок фіксованого розміру. Після підкачування сегмента (сторінки) виконання задачі продовжується, так що віртуалізація пам'яті для прикладних задач прозора (якщо не брати до уваги затримку, викликану підкачуванням).

Процесор надає тільки необхідні апаратні засоби підтримки захисту і віртуальної пам'яті, а їхнє реальне використання й усталеність роботи програм і самої операційної системи в захищеному режиму, звісно ж, залежать від коректності побудови ОС і завбачливості її розробників. Добре спроектована операційна система захищеного режиму може забезпечити усталеність ОС навіть при некоректному поводженні прикладних задач.

3.2. Дескриптори і таблиці

Існують три типи таблиць дескрипторів - локальна таблиця дескрипторів LDT (Local Descriptor Table), глобальна таблиця дескрипторів GDT (Global Descriptor Table) і таблиця дескрипторів переривань IDT (Interrupt Descriptor Table).

Розміри таблиць можуть знаходитися в межах 8 байт-64 Кбайт, що відповідає числу елементів у таблиці від 1 до 8 К. З кожною із цих таблиць пов'язаний відповідний регістр процесора. Регістри GDTR і IDTR мають програмно-доступне 16-бітове поле ліміту, що задає розмір таблиці, і 32-бітове (у 80286 - 24-бітове) поле базової адреси, що визначає положення таблиці в просторі лінійних (у 80286 - фізичних) адрес пам'яті. У регістра LDTR програмно доступно тільки 16-бітове поле селектора, по якому з GDT автоматично завантажуються програмно недоступні і невидимі поля базової адреси і ліміту.

Команди завантаження регістрів таблиць LGDT, LIDT і LLDT є привілейованими (виконуються тільки на рівні привілеїв 0). Команди LGDT і LIDT завантажують із пам'яті 6-байтове поле, що містить базову адресу і ліміт локальної таблиці. Команда LLDT завантажує

тільки селектор, що посилається на дескриптор, який містить базову адресу і ліміт локальної таблиці дескрипторів.

Глобальна таблиця (GDT) містить доступні всім задачам дескриптори будь-яких типів, крім дескрипторів переривань і пасток. Нульовий елемент цієї таблиці процесором не використовується. Локальна таблиця (LDT) може бути власною для кожної задачі і містить тільки дескриптори сегментів, вентилів задачі і викликів. Сегмент недоступний задачі, якщо його дескриптора немає в даний момент ні в GDT, ні в LDT. Вибір таблиці (локальна або глобальна) визначається за значенням біта TI селектора (рис. 3.2), а положення (номер) дескриптора задається 13-бітовим полем INDEX селектора. При посиланні на дескриптор, що виходить за ліміт таблиці, виникає виняток #GP.

Таблиця дескрипторів переривань, яка використовується в захищеному режимі, може містити описи до 256 переривань. Таблиця може містити тільки вентилі задач, переривань і пасток. Базова адреса і ліміт таблиці завантажується привілейованою командою LIDT (аналогічно LGDT). Розмір IDT повинен бути не менше 256 байт, для того щоб у неї помістилися всі зарезервовані переривання процесора. Посилання на елементи IDT виконується програмно по командах INT, по апаратних перериваннях і винятках процесора. При виникненні переривання або винятку, дескриптор якого виходить за ліміт таблиці, формується виняток #DF.

Дескриптори мають 8-байтовий формат як для 16-розрядних (80286), так і для 32-розрядних процесорів. Призначення дескриптора визначається полями байта керування доступом (Access Rights Byte)-байта зі зсувом 5. Дескриптори 16- і 32-розрядних процесорів відрізняються розрядністю поля базової адреси (24 і 32 біти) і трактуванням поля ліміту, яке повинно забезпечувати розмір сегмента до 64 Кбайт або 4 Гбайт відповідно. Два старших байти в дескрипторів 80286 завжди нульові (через вимоги сумісності з наступними процесорами, оголошеними при випуску 80286), що дозволяє їх відрізняти і коректно використовувати, виконуючи 16-бітові додатки захищеного режиму на 32-розрядних процесорах. Два старших байти дескрипторів 32-розрядних процесорів містять розширення полів BASE і LIMIT, біт дробності G (Granularity), що визначає, у яких одиницях заданий ліміт: G=0-в байтах, G=1-в сторінках по 4 Кбайт (це і забезпечує максимальну довжину в 4 Гбайт). У дескрипторах сегментів, відсутніх у фізичній пам'яті (P=0), процесор цікавиться тільки байтом керування доступом. Інші байти можуть використовуватися на розсуд ОС.

Дескриптори сегментів коду і даних визначають базову адресу, розмір сегмента, права доступу (читання, читання/запис, тільки

виконання коду або виконання/читання), а для систем із віртуальною пам'яттю ще і присутність сегмента у фізичній пам'яті (рис. 3.4, рис. 3.5).

У байті керування доступом поля мають таке призначення:

- Біт 7 - P (Present) - присутність у пам'яті. При P=1 сегмент відображений у фізичній пам'яті, при P=0 відображення немає і поля базової адреси і ліміту не використовуються.
- Біти 6, 5 - DPL (Descriptor Privilege Level) - атрибути привілеїв сегмента.
- Біт 4 - A (Accessed) - звертання. A=0 - до сегмента не було звертанн, A=1 -селектор даного сегмента завантажувався в регістр сегмента або для нього виконувалася команда тестування.

Для дескриптора сегмента даних (включаючи і стек):

- Біт 2 - E (Expand Down) - контрольований напрямок розширення: E=0 - розширюваний нагору (зсув не повинен перевищувати значення ліміту); E=1 – розширення вниз (стек, в якого зсув повинен перевищувати значення ліміту).

Для сегментів даних (включаючи і стек), розширюваних вниз (E=1), біт B в передостанньому байті дескриптора визначає верхню межу сегмента: при B=0 - FFFFh (максимальний розмір сегмента - 64 Кбайт), при B=1 - FFFFFFFFh (максимальний розмір сегмента - 4 Гбайти). Цей же біт у дескрипторі сегмента стека визначає і розрядність використовуваного покажчика стека: B=0 - використовується 16-бітовий SP (покажчик вершини стеку), розрядність даних при операціях PUSH, POP -16 біт; при B=1 використовується 32-бітовий ESP, розрядність даних при операціях PUSH, POP- 32 біти.

У сегмент коду запис неможливий, ліміт указує на його останній байт, а біти типу мають таке призначення:

- Біт 2 - C (Conforming, конформність або підпорядкованість): при C=1 код може виконуватися, якщо поточний рівень привілеїв (CPL) не нижче рівня привілеїв дескриптора (DPL); при C=0 (не підлеглий сегмент) керування до даного сегмента може передаватися, тільки якщо CPL=DPL.
- Біт 1 – R (Readable) - дозвіл (R=0) або заборона (R=1) читання сегмента.

Запис у сегмент коду можлива тільки через псевдонім (Alias) - сегмент даних із дозволеним записом, що має ті ж значення бази і ліміту. Біт D (Default Operation Size) у передостанньому байті дескриптора сегмента коду визначає розрядність адрес і операндів. За замовчуванням D=0 - 16 біт, а при D=1-32 біти.

Системні сегменти призначені для збереження локальних таблиць дескрипторів LDT (Local Descriptor Table) і стану задач TSS (Task State Segment). Їхні дескриптори визначають базову адресу, ліміт сегмента (1-64 Кбайт), права доступу (читання, читання/запис, тільки виконання коду або виконання/читання) і присутність сегмента у фізичній пам'яті (рис. 3.6). У цих дескрипторів біт P в байті керування доступом визначає дійсність (P=1) або недійсність (P=0) вмісту сегмента. Поле рівня привілеїв DPL використовується тільки в дескрипторах сегментів стану задач. Поле Type (1-3, 9-B) визначає тип сегмента:

- 0,8 - неприпустимі значення;
- 1 - доступний сегмент стану задачі 80286 (Available TSS-286);
- 2 - таблиця локальних дескрипторів (LDT);
- 3 - зайнятий сегмент стану задачі 80286 (Busy TSS-286);
- 9 - доступний сегмент стану задачі 386+ (Available TSS-386);
- A - не визначено (зарезервовано);
- B - зайнятий сегмент стана задачі 386+ (Busy TSS-386).

Міжсегментна передача керування безпосередньо (командами JMP, CALL, INT, RET і IRET) можлива тільки до сегментів коду з тим же рівнем привілеїв або до підпорядкованих сегментів, рівень привілеїв яких вище, ніж CPL (при цьому CPL не змінюється). Для переходів із зміною рівня привілеїв використовуються вентилі (Gate) різних типів, які іноді називають шлюзами. Для кожного способу непрямої міжсегментної передачі керування є відповідні вентилі, їхнє використання дозволяє процесору автоматично виконувати контроль захисту. Вентилі виклику (Call Gates) використовуються для викликів процедур із зміною рівня привілеїв, вентилі задач (Task Gates) використовуються для переключення задач, вентилі переривань (Interrupt Gates) і пасток (Trap Gates) визначають процедури обслуговування переривань. Вентилі виклику дозволяють автоматично копіювати задане число слів із старого стека в новий. Вентилі переривань відрізняються від вентилів пасток тільки тим, що вони забороняють переривання (скидають прапорець IF), а вентилі пасток - ні. Для кожного типу вентилів використовуються відповідні дескриптори вентилів (Gate Descriptors - рис. 3.7) .

У байті керування доступом у цих дескрипторів біт P визначає дійсність (P=1) або недійсність (P=0) вмісту сегмента. Поле DPL задає рівень привілеїв. kПоле Type визначає тип вентиля:

- 4 - вентиль виклику 80286 (Call Gate);
- 5 - вентиль задачі 80286 (Task Gate);
- 6 - вентиль переривання 80286 (Interrupt Gate);
- 7 - вентиль пастки 80286 (Trap Gate);

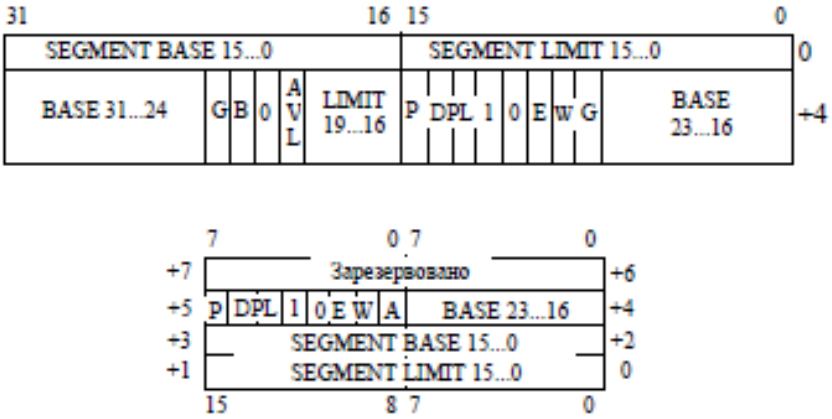


Рис. 3.4 - Дескриптор сегменту даних: а – 32-бітовий формат; б – 16-бітовий формат 80286

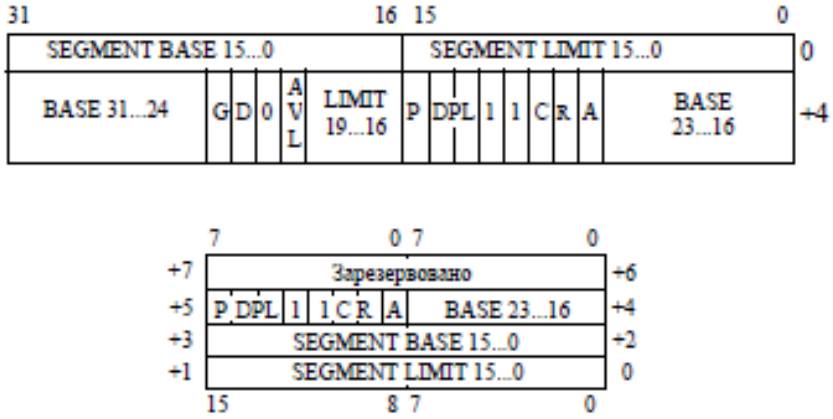


Рис. 3.5 - Дескриптор сегменту коду: а – 32-бітовий формат; б- 16-бітовий формат.

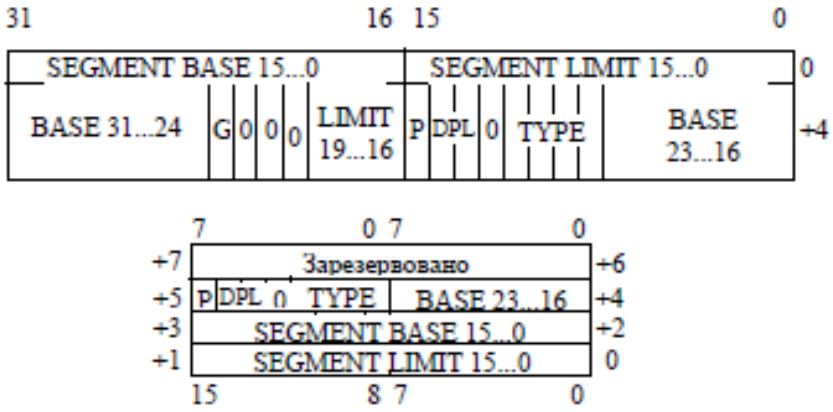


Рис. 3.6 - Дескриптор системних сегментів: а: 32-бітовий формат; 16-бітовий формат

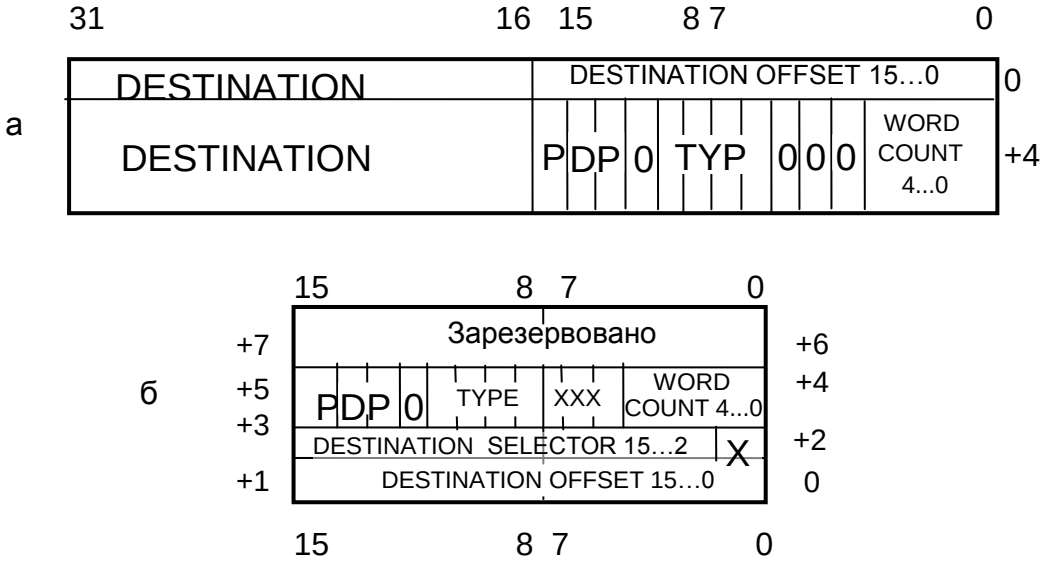


Рис. 3.7 - Дескриптори вентилів: а – 32-бітовий формат; б – 16-бітовий формат

- C - вентиль виклику 386+ (Call Gate):
- D - вентиль задачі 386+ (Task Gate);
- E - вентиль переривання 386+ (Interrupt Gate);
- F - вентиль пастки 386+ (Trap Gate).

Поле Word Count використовується тільки у вентилях викликів і визначає число слів, які автоматично копіюються із стека процесу, що викликає, в стек процедури, що викликається. Для сегментів 80286 слова 16-бітові, для 386+ - 32-бітові.

Слово Destination Selector для вентилів виклику, переривань і пасток задає селектор цільового сегмента коду, а для вентиля задачі - селектор цільового TSS. Слово Destination Offset задає зсув (адресу) точки входу в цільовому сегменті. При використанні вентилів може виникнути виняток #GP, який означає, що селектор указує на некоректний тип дескриптора. При спробі використання недійсного вентиля (P=0) виникає виняток #NP.

3.3. Привілеї

У захищеному режимі процесор має чотирирівневу систему привілеїв, що керує використанням привілейованих інструкцій і доступом до дескрипторів (і пов'язаних із ними сегментах). Рівні привілеїв нумеруються від 0 до 3, вищому рівню привілеїв відповідає нульовий рівень. Рівні привілеїв забезпечують захист задач, які ізолюються одна від одної локальними таблицями дескрипторів. Сервіси операційної системи, обробники переривань і інше системне забезпечення можуть включатися у віртуальний адресний простір кожної задачі і захищатися системою привілеїв. Кожна частина системи працює на своєму рівні привілеїв. Задачі, дескриптори і селектори мають свої атрибути привілеїв.

Привілеї задач (Task Privilege) впливають на виконання інструкцій і використання дескрипторів. Поточний рівень привілею задачі CPL (Current Privilege Level) визначається двома молодшими бітами регістра CS. CPL задачі може змінюватися тільки при передачі керування до нового сегмента через дескриптор вентиля. Задача починає виконуватися з рівня CPL, вказаного селектором кодового сегмента всередині TSS у тому випадку, коли задача ініціюється за допомогою операції переключення задач. Задача, що виконується на нульовому рівні привілеїв, має доступ до всіх сегментів, описаних у GDT, і є найпривілейованішою. Задача, що виконується на рівні 3, має найменші права.

Поточний рівень привілею може змінюватися тільки при передачі керування через вентилі. Привілеї дескриптора (Descriptor Privilege) задаються полем DPL байта керування доступом. DPL визначає найбільший номер рівня привілеїв (фактично, найменші привілеї), з яким можливий доступ до даного дескриптора. У найзахищенішого дескриптора DPL=0, до нього мають доступ тільки задачі з CPL=0. У дескриптора з найменшим рівнем привілеїв DPL=3, його можуть використовувати задачі з CPL=0, 1, 2, 3. Це правило застосовується до всіх дескрипторів, за винятком дескриптора LDT. Привілеї селектора (Selector Privilege) задаються полем RPL (Requested Privilege Level) - двома молодшими бітами селектора. За допомогою RPL можна урізати ефективний рівень привілеїв EPL (Effective Privilege Level), що визначається як максимальне зі значень CPL і RPL. Селектор із RPL=0 не вводить додаткових обмежень.

Контроль доступу до сегментів даних виконується при виконанні команд, що завантажують селектори в SS, DS, ES, FS і GS. Команди завантаження DS, ES, FS і GS повинні посилатися на дескриптори сегментів даних або сегментів кодів, які дозволяють читання. Для одержання доступу ефективний рівень привілеїв EPL повинний бути рівним або меншим (арифметично) рівня привілеїв DPL дескриптора.

Винятком із цього правила є читання підпорядкованого сегмента коду, що може бути прочитаний задачею з будь-яким CPL. Якщо ефективний рівень привілеїв не дозволяє доступ або посилання виконується на некоректний тип дескриптора (на дескриптор вентиля або на дескриптор кодового сегмента, який тільки виконується), формуєтьсяється виняток #GP. При посиланні на неіснуючий дескриптор формується виняток #NP. Команди завантаження регістра SS повинні посилатися на дескриптор сегмента даних, що припускає запис. При цьому DPL і RPL повинні бути рівні CPL. Порушення цієї умови і посилання на дескриптор

іншого типу породжують виняток #GP, при посиланні на неіснуючий дескриптор виробляється виняток #SS.

Контроль типів і привілеїв при передачі керування виконується при завантаженні селектора в регістр CS. Тип дескриптора, на який посилається даний селектор, повинен відповідати виконуваній інструкції. Порушення типу (наприклад, посилання інструкції JMP на вентиль виклику) породжують виняток #GP.

При передачі керування діють такі правила привілеїв, порушення яких також призводить до винятку #GP:

- команди JMP або CALL можуть посилатися або на підпорядкований сегмент коду з DPL, більшим або рівним CPL, або на непідпорядкований сегмент із DPL рівним CPL;
- переривання усередині задачі або виклики, що можуть змінити рівень привілеїв, можуть передавати керування кодовому сегменту з рівнем привілеїв, рівним або більшим рівня привілеїв CPL, тільки через вентилі з тим же або меншим чим CPL рівнем привілеїв;
- інструкції повернення, що не переключають задачі, можуть передати керування тільки кодовому сегменту з таким же або меншим рівнем привілеїв;
- переключення задач може виконуватися за допомогою виклику, переходу або переривання, що посилаються на вентиль задачі або сегмент стану задачі (TSS) із тим же або меншим рівнем привілеїв. Зміна рівня привілеїв, що відбувається при передачі керування, автоматично викликає перевизначення стека. Початкове значення покажчика стека SS:SP для рівня привілеїв 0, 1, 2 міститься в TSS. При передачі керування по командах JMP або CALL у SS:SP завантажується нове значення покажчика стека, а старі значення поміщаються в новий стек. При поверненні на старий рівень привілеїв його стек відновлюється (як частина інструкції RET або IRET).

Для викликів підпрограм із передачею параметрів через стек і зміною рівня привілеїв, із попереднього стека в новий копіюється фіксоване число слів, задане у вентилі. Команда міжсегментного повернення RET із вирівнюванням покажчика стека при поверненні коректно відновить значення попереднього покажчика.

Привілеї і бітова карта дозволу введення/виведення контролюють можливість виконання операцій введення/виведення і керування прапорцем переривань IF. Рівень привілеїв введення/виведення визначається полемо IOPL (Input/Output Privelege Level) регістра прапорів. Значення IOPL можна змінити тільки при CPL=0. При CPL=IOPL на операції введення/виведення і керування прапорцем ніяких обмежень не накладається. При CPL >

IOPL спроба введення/виведення, виконана задачею з TSS класу 80286, викликає виняток #GP (відмова). Якщо CPL > IOPL, а з задачею пов'язаний TSS 386+, інструкції введення/виведення можуть виконуватися тільки по адресах портів, для яких установлені нульові біти в карті дозволу введення/виведення, що є в TSS. Спроби звертання до портів, яким відповідають одиничні біти карти або які не потрапили в карту (її розмір може урізатися), викликають виняток #GP. При CPL > IOPL спроба виконання інструкцій CLI і STI викликає виняток #GP. Неявне керування прапорцем переривань інструкціями завантаження або відновлення регістра прапорців блокується без генерації винятків.

3.4. Захист

Для надійної роботи багатозадачних систем необхідний захист задач одна від одної. Захист призначений для запобігання несанкціонованого доступу до пам'яті і виконання критичних інструкцій - команди HLT, що зупиняє процесор, команд введення/виведення, керування прапорцем дозволу переривань і команд, що впливають на сегменти коду і даних. Механізми захисту вводить такі обмеження:

- обмеження використання сегментів (наприклад, заборона запису в сегменти даних, що тільки читаються або спроба виконання даних як коду). Для використання доступні тільки сегменти, дескриптори яких описані в GDT і LDT;
- обмеження доступу до сегментів через правила привілеїв;
- обмеження набору інструкцій - виділення привілейованих інструкцій або операцій, що можна виконувати тільки при визначених рівнях CPL і IOPL;
- обмеження можливості міжсегментних викликів і передач керування.

У захищеному режимі при виконанні інструкцій процесор виконує перевірки таких умов, що породжують винятки:

Перевірка при завантаженні сегментних регістрів

- Перевищення ліміту таблиці дескрипторів - #GP.
- Неіснуючий дескриптор сегмента - #NP або #SS.
- Порушення привілеїв - #GP.
- Завантаження невірного дескриптора або типу сегмента - #GP:
    - Завантаження в SS сегмента коду або сегмента даних тільки для читання.
    - Завантаження керуючих дескрипторів у DS, ES або SS.
    - Завантаження сегментів, що тільки виконуються у DS, ES або SS.

- Завантаження сегмента даних у CS.

Перевірка посилань операндів

- Запис у сегмент коду або сегмент даних тільки для читання - #GP.
- Читання з сегмента кодів, що тільки що виконується - #GP.
- Перевищення ліміту сегмента - #SS або #GP.

Перевірка привілеїв інструкцій

- CPL<>0 при виконанні інструкцій LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT, INVD, INVLPG, WBINVD, операції з регістрами DRn, TRn, CRn - #GP.
- CPL > IOPL при виконанні інструкцій STI, CLI, а для 80286 ще й інструкції LOCK - #GP.
- CPL > IOPL при виконанні інструкцій IN, INS, OUT, OUTS із портами, не дозволеними бітовою картою введення/виведення - #GP.

При виконанні команд IRET і POPF із недостатнім рівнем привілеїв, біти IF і IOPL у регістрі прапорців не змінюються, винятки не породжуються:

- IF не змінюється, при CPL > IOPL;
- IOPL не змінюється, якщо CPL > 0.

Перевірки при передачі керування за інструкціями JMP, CALL, RET, INT і IRET включають як перевірку посилань за лімітом (у «ближніх» формах JMP, CALL і RET виконуються тільки ці перевірки), так і перевірку правил привілеїв при міжсегментних передачах через вентилі.

Для того щоб задача не викликала спрацьовування захисту, у систему команд введені спеціальні інструкції тестування покажчиків. Вони дозволяють швидко засвідчити можливість використання селектора або сегмента без ризику породження винятку:

- ARPL - вирівнювання RPL. При її виконанні RPL селектора прирівнюється максимальному значенню з поточного RPL селектора і поля RPL в указаному регістрі. Якщо при цьому RPL змінився, установлюється ZF=1;
- VERR - перевірка можливості читання: якщо сегмент, на який указує селектор, припускає читання, установлюється ZF=1;
- VERW - перевірка можливості запису: якщо сегмент, на який указує селектор, припускає запис, установлюється ZF=1,
- LSL - читання ліміту сегмента в регістр, якщо дозволяють привілеї. При успіху встановлюється ZF=1;

- LAR - читання байта доступу дескриптора в регістр, якщо дозволяють привілеї. При успіху встановлюється ZF=1.

Деякі функції захисту виконуються і механізмом сторінкової переадресації, проте, на відміну від сегментного захисту, існують засоби обходу сторінкового захисту на рівні користувача (CPL=3).

3.5. Переключення задач

Для багатозадачних і багатокористувацьких операційних систем важлива спроможність процесора до швидкого перемикання з задачі на задачу. Операція перемикання задач процесора (Task Switch Operation) зберігає стан процесора і зв'язок із попередньою задачею, завантажує стан нової задачі і починає її виконання. Перемикання задач виконується по інструкції міжсегментного переходу (JMP) або виклику (CALL), яка посилається на сегмент стану задачі TSS (Task State Segment) або дескриптор вентиля задачі в GDT або LDT. Перемикання задач може виконуватись також по апаратних і програмних перериваннях і винятках, якщо відповідний елемент у IDT є дескриптором вентиля задачі. Дескриптор TSS указує на сегмент, що містить повний стан процесора, а дескриптор вентиля задачі містить селектор, що вказує на дескриптор TSS. Кожна задача повинна мати пов'язаний із нею TSS (рис. 3.8, 3.9). 32-розрядні процесори припускають і 16-бітний (у стилі 80286) формат TSS. Обидва типи сегментів містять образи регістрів процесора, роздільні покажчики стеків для рівнів привілеїв 0, 1 і 2, а також обернене посилання на селектор TSS задачі, що викликала. Вільне поле TSS може використовуватися на розсуді ОС. TSS для процесорів 386+ містить елементи, відсутні в 80286: бітові карти дозволу введення/виведення і перенаправлення переривань, а також біт відлагоджувальної пастки T (при T=1 перемикання в дану задачу викликає виняток налагодження). Останнім елементом TSS-386+ повинний бути байт OFFh. Карту перенаправлення переривань підтримують тільки процесори з розширенням VME (Virtual Mode Extenssion – біт в регістрі CR4, який дозволяє велючення режиму EV86, що відрізняється від режиму V86 можливістю віртуалізації переривань). Значення поля ліміту дескриптора для TSS-286 повинно перевищувати 002Bh, а для TSS-386+ - 0064h.

Карта дозволу введення/виведення (I/O Permission Bit Map), розташована наприкінці TSS-386+, має по одному біту на кожну адресу портів введення/виведення. Дозволові звертання відповідає нульове значення біта. Максимальний розмір таблиці (2000h), що відповідає всім 64 К адресам, може бути урізаний лімітом TSS, але байт-термінатор OFFh повинний обов'язково вписуватися в ліміт TSS.

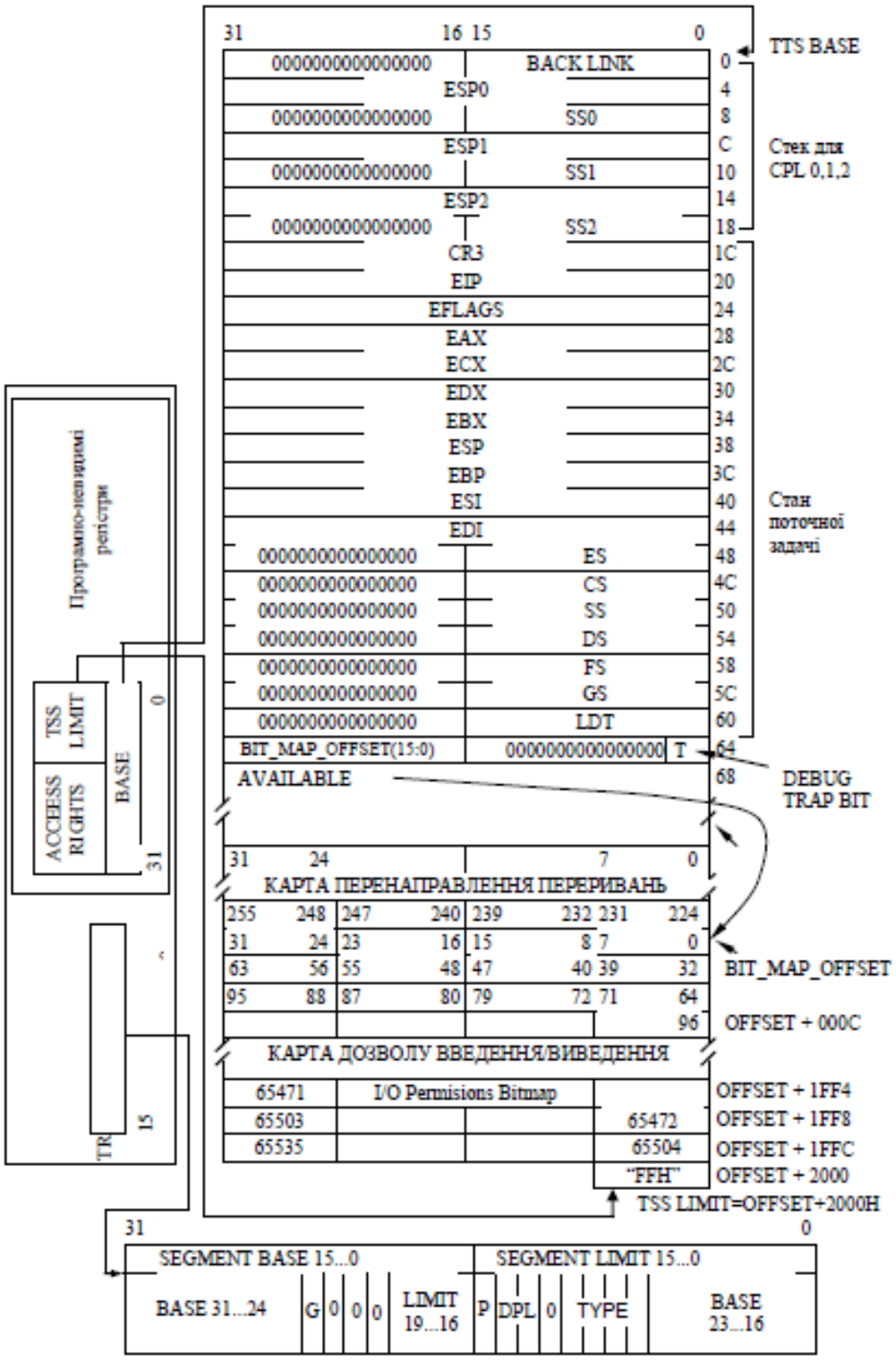


Рис. 3.8 - Сегмент стану задачі 386+

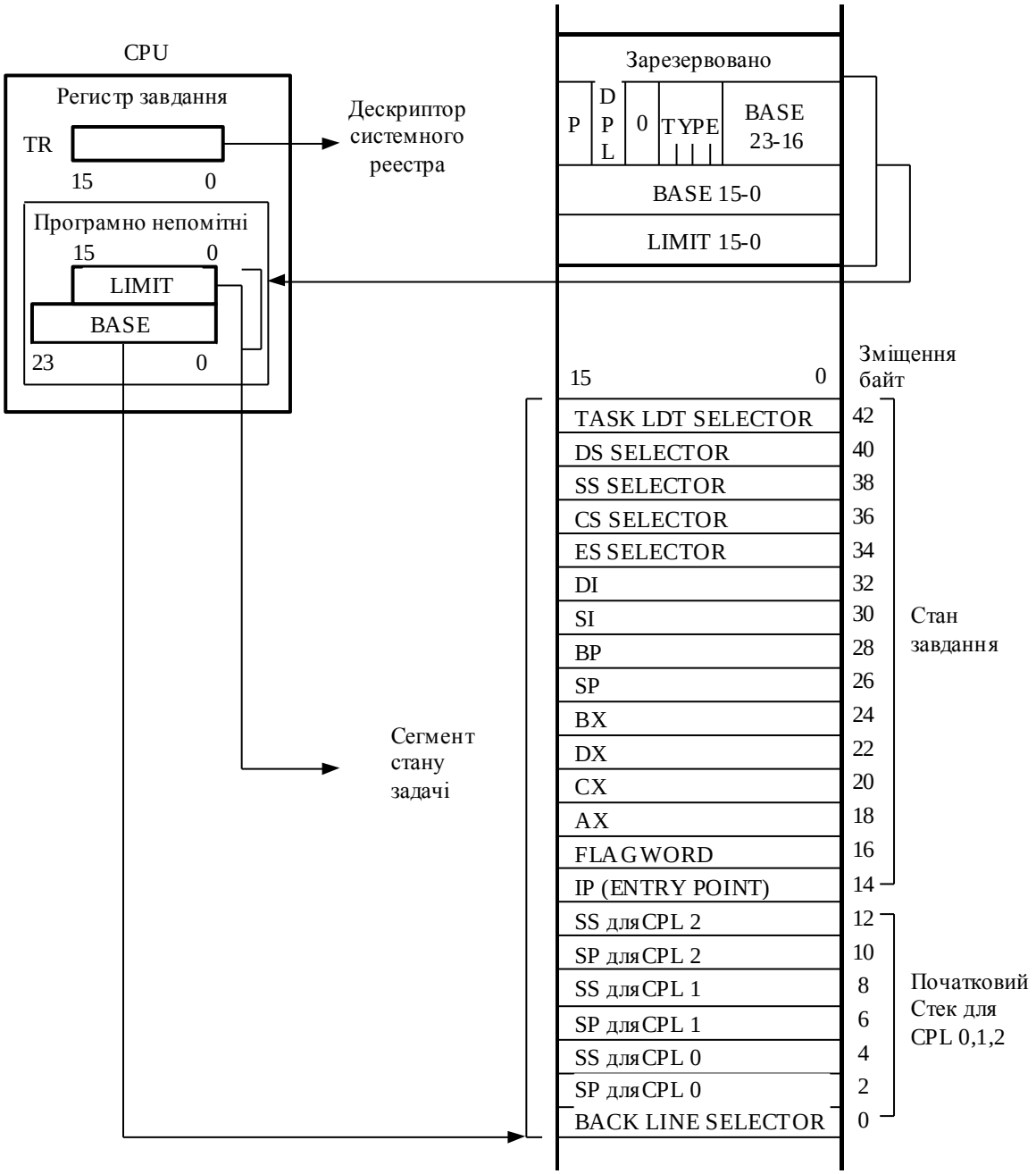


Рис. 3.9. Сегмент стану задачі 80286.

Порти з адресами, що не потрапили в усічену таблицю, вважаються недоступними.

Поточний TSS ідентифікується спеціальним регістром задачі TR (Task Register). Цей регістр містить селектор, що посилається на дескриптор поточного TSS. Програмно-невидимі базові адреси і ліміт, пов’язані з TR, завантажуються в TR нового селектора.

Для повернення керування задачі, що викликала поточну задачу використовується інструкція IRET. В регістрі прапорців є прапорець вкладеної задачі NT (Nested Task), який керує функцією інструкції IRET. При NT=0 IRET працює звичайним чином,

залишаючись в поточній задачі. При NT=1 (поточна задача вкладена) IRET виконує перемикання на попередню задачу.

Коли інструкції CALL, JMP, INT виконують перемикання задач, старий (крім випадку JMP) і новий TSS відмічаються занятими (міняється значення поля TYPE в їх дескрипторах) і в поле оберненого посилання в новому TSS встановлюється значення селектора старого TSS. Інструкції CALL і INT, які перемикають задачі, встановлюють в новій задачі біт NT. Переривання, які викликають перемикання задач скидають біт NT. Цей біт може також встановлюватись або скидатись інструкціями POPF і IRET.

Зміна контексту співпроцесора при перемиканні задач автоматично не виконується, оскільки новій задачі співпроцесор може бути непотрібний. Але процесор визначає перше використання співпроцесора після перемикання задач і формується виняток #NM. Оброблювач цього винятку сам визначить, чи необхідна зміна контексту. Кожний раз при перемиканні задач процесор установлює біт TS (Task Switched) у MSW. Це вказує на те, що контекст процесора може відноситися до іншої задачі. При виконанні інструкцій ESC або WAIT, якщо TS=1 і MP=1 (співпроцесор присутній), формується виняток #NM (відсутній співпроцесор).

3.6. Сторінкове керування пам'яттю

Сторінкове керування (Paging) є засобом організації віртуальної пам'яті з підкачуванням сторінок по запиту (Demand-Paged Virtual Memory). На відміну від сегментації, що організує програми і дані в модулі різного розміру, сторінкова організація оперує з пам'яттю, як із набором сторінок однакового розміру.

У момент звертання сторінка може бути присутня у фізичній оперативній пам'яті, а може бути вивантаженою на зовнішню (дискову) пам'ять. При звертанні до вивантаженої сторінки пам'яті процесор формує виняток #PF - відмова сторінки, а програмний оброблювач винятку (частина ОС) одержить необхідну інформацію для свопінгу - «підкачування» відсутньої сторінки з диска. Сторінки не мають прямого зв'язку з логічною структурою даних або програм. У той час, коли селектори можна розглядати як логічні імена модулів кодів і даних, сторінки представляють частини цих модулів. З огляду на звичайну властивість локальності (близького розташування необхідних комірок пам'яті) коду і посилань на дані, в оперативній пам'яті в будь-який момент часу варто берегти тільки невеличку область сегментів, необхідних активним задачам. Цю можливість (а отже, і збільшення припустимого числа одночасно виконуваних задач при обмеженому об’ємі оперативної пам'яті) саме і забезпечує сторінкове керування пам'яттю. У перших 32-розрядних процесорах

(починаючи з 80386) розмір сторінки складав 4 Кбайт. Починаючи з Pentium, з'явилася можливість збільшення розміру сторінки до 4 Мбайт, одночасно з використанням сторінок розміром 4 Кбайт. У P6 є можливість розширення фізичної адреси до 36 біт (64 Гбайт), при якій можуть використовуватися сторінки розміром 4 Кбайт і 2 Мбайт.

Базовий механізм сторінкового керування використовує дворівневу табличну трансляцію лінійної адреси у фізичну (рис. 3.10). Механізм має три частини: каталог сторінок (Page Directory), таблиці сторінок (Page Table) і власне сторінки (Page Frame). Механізм включається установкою біта PG=1 у регістрі CRO. Регістр CR2 зберігає лінійну адресу відмови (Page Fault Linear Address) - адреса пам'яті, за якою була виявлена остання відмова сторінки. Регістр CR3 призначений для збереження фізичної адреси каталога сторінок (Page Directory Physical Base Address). Його молодші 12 біт завжди нульові (каталог вирівнюється по границі сторінки).

Каталог сторінок розміром 4 Кбайт містить 1024 32-бітових рядки PDE (Page Directory Entry). Кожний рядок (рис. 3.11 а) містить 20 старших біт адреси таблиці наступного рівня (молодші біти цієї адреси завжди нульові) і ознаки (атрибути) цієї таблиці. Індексом пошуку в каталозі сторінок є 10-ть старших біт лінійної адреси (A22-A31). Кожна таблиця сторінок також має 1024 рядки PTE (Page Table Entry) аналогічного формата (рис. 3.11 б), але ці рядки містять базову фізичну адресу (Page Frame Adress) і атрибути самих сторінок. Індексом пошуку в таблиці є біти A12-A21 лінійної адреси. Фізична адреса утворюється з адреси сторінки, узятої з таблиці, і молодших 12 біт лінійної адреси.

Рядки каталога і таблиць мають такі біти атрибутів:

- P (Present) - біт присутності. P=1 означає можливість використання даного рядка для трансляції адреси. Біт присутності входжень у таблиці, які використовуються поточним кодом, що виконується , повинний бути встановлений. Програмний код не повинний його змінювати «на ходу». Якщо P=0, то всі інші біти доступні операційній системі і можуть використовуватися для одержання інформації про місцезнаходження даної сторінки.
- A (Accessed) - ознака доступу, що встановлюється перед будь-яким читанням або записом за адресою, у перетворенні якої бере участь даний рядок.
- D (Dirty) - ознака, що встановлюється перед операцією запису за адресою, у перетворенні якої бере участь даний рядок. У такий спосіб позначається використана - «брудна» сторінка, яку у випадку заміщення необхідно вивантажити на диск.

Біти P, A, D модифікуються процесором апаратно в заблокованих шинних циклах. При їхній програмній модифікації в

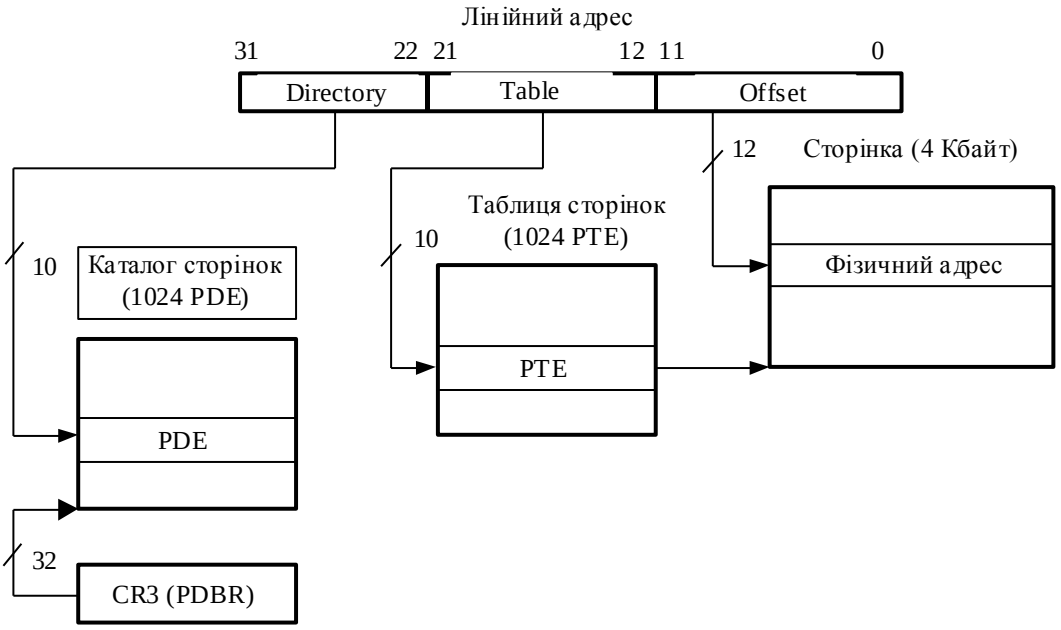


Рис. 3.10. Базовий механізм сторінкової переадресації.

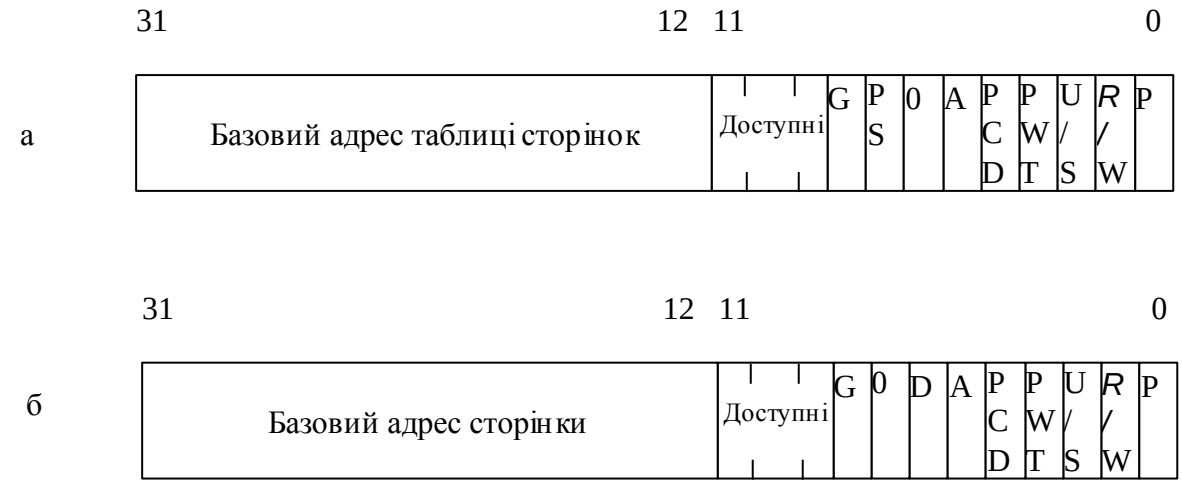


Рис. 3.11. Структура 32-бітних елементів сторінкового перетворення: а - рядок каталогу; б – рядок таблиці.

багатопроцесорних системах повинний використовуватися префікс LOCK, що гарантує збереження цілісності даних.

Поле OS Reserved (біти 9-11) програмно використовується на розсуд ОС і може зберігати, наприклад, інформацію про «вік» сторінки, необхідну для реалізації заміщення за алгоритмом LRU

(Least Recently Used – сторінка, що найдовше не використовувалася, заміщується першою).

- Біт PWT (Page Write Through) визначає режими запису при кешуванні, а біт PCD (Page Cache Disable) забороняє кешування пам'яті для сторінок або таблиць (використовуються на процесорах 486+), що обслуговуються ;
- Біт PS (Page Size) задає розмір сторінки (тільки в PDE). При PS=0 сторінка має розмір 4 Кбайт, PS=1 використовується в режимах розширення розміру сторінки PSE (Page Size Extension) і розширення фізичної адреси PAE (Physical Address Extensions), які мають процесори Pentium і старші. В цих режимах розмір сторінки може бути збільшений до 4 Мбайт.

Механізм захисту сторінок розрізняє два рівні привілеїв: користувач (User) і супервізор (Supervisor). Користувачу відповідає рівень привілеїв 3, супервізору - рівні 0, 1 і 2. Рядки таблиць мають атрибути захисту сторінок - біти U (User) і W (Writable - можливий запис); у деяких описах ті ж біти називаються U/S (User/Supervisor) і R/W (Read/Write). Ці атрибути в рядку каталога сторінок відносяться до всіх сторінок, на які посилається даний рядок через таблицю другого рівня. Атрибути захисту в рядку таблиці сторінок відносяться до конкретної сторінки пам'яті, яку він обслуговує. Захист на рівні сторінок включається установкою біта WP (Write Protect) у регістрі CRO. За апаратним скиданням він занулюється.

Захист на рівні сторінок пояснює табл. 3.1.

Таблиця 3.1- Захист на рівні сторінок

U (U/S)	W(R/W)	Дозволено при PL=3	Дозволено при PL=0,1,2
0	0	Ні	Читання/запис
0	1	Ні	Читання/запис
1	0	Тільки читати	Читання/запис
1	1	Читання/запис	Читання/запис

- Біт G (Global), що з'явився в P6, визначає глобальність сторінки. Він аналізується тільки в рядку, що вказує на сторінку фізичної пам'яті (у PTE для сторінок у 4 Кбайт, у PDE - для сторінок 2 Мбайти або 4 Мбайти). Цей біт, керований тільки програмно, дозволяє позначити сторінки глобального використання (наприклад, ядра ОС). При G=1 і PGE=1 (регістр CR4) рядки з покажчиками на глобальні таблиці не будут занулятися в TLB при завантаженні регістра CR3 або перемиканні задач, що знижує затрати на обслуговування віртуальної пам’яті.

Механізм сторінкового керування при звертанні до пам'яті може породжувати виняток #PF, Він виникає при звертанні до відсутньої сторінки і при порушенні прав доступу, обумовлених рівнем привілеїв і бітами U і W. Для ідентифікації причини відмови в стек поміщається 16-бітовий код помилки. Хоча назви біт і їх положення в цьому коді збігаються з атрибутами рядків в PDE і PTE, їхнє призначення відрізняється:

- Біт U/S указує на рівень привілеїв, при якому відбулася відмова (1 користувач, 0 - супервізор).
- Біт W/R вказує на операцію, при якій відбулася відмова (0 - читання, 1 - запис).
- Біт P вказує на причину відмови (P=1 - відсутність сторінки в пам'яті, P=0 - порушення захисту).

Перевірка захисту на рівні сторінок виконується після перевірок захисту сегментів. Якщо при спробі доступу до пам'яті спрацював захист сегментів, то перевірка на рівні сторінок уже не виконується.

Звертання при кожній операції доступу до пам'яті до двох таблиць, розташованих у пам'яті, істотно знижує продуктивність. Для запобігання цьому в процесор введений буфер асоціативної трансляції TLB (Translation Look aside Buffer) для збереження інтенсивно використовуваних рядків таблиць. У процесорах 80386 і 486 буфер являє собою чотириканальний набірно-асоціативний кеш на 32 рядки таблиць трансляції.

У процесорах Pentium і старших розширення розміру сторінки

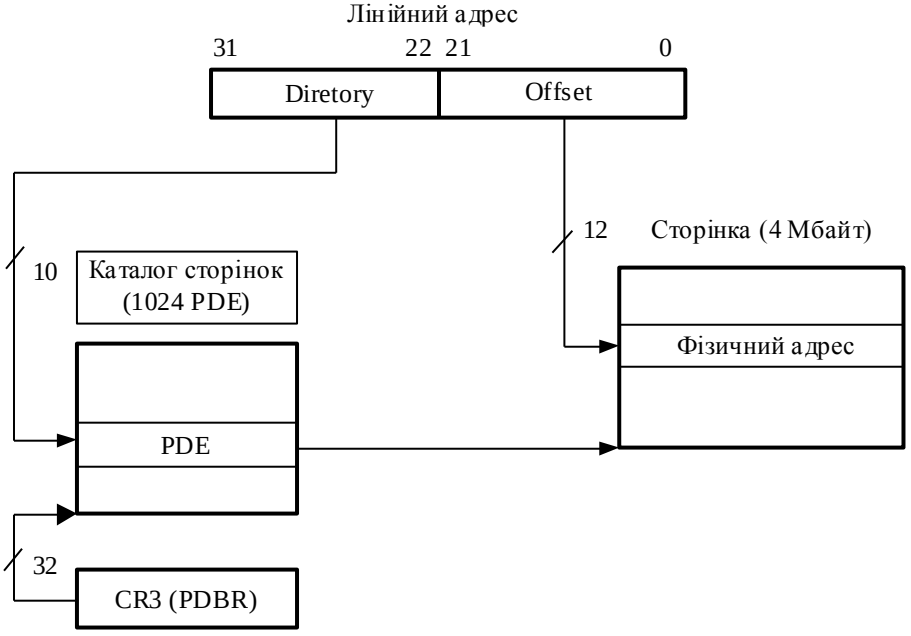


Рис. 3.12. Сторінкова переадресація в режимі PSE.

(PSE) дозволяється установленням біта PSE=1 в регістрі CR4. Якщо

CR4.PSE=1 і PDE.PS=1, то біти 31-22 рядка катологa сторінок є базовою фізичною адресою сторінки розміром 4 Мбайта (рис. 3.12)

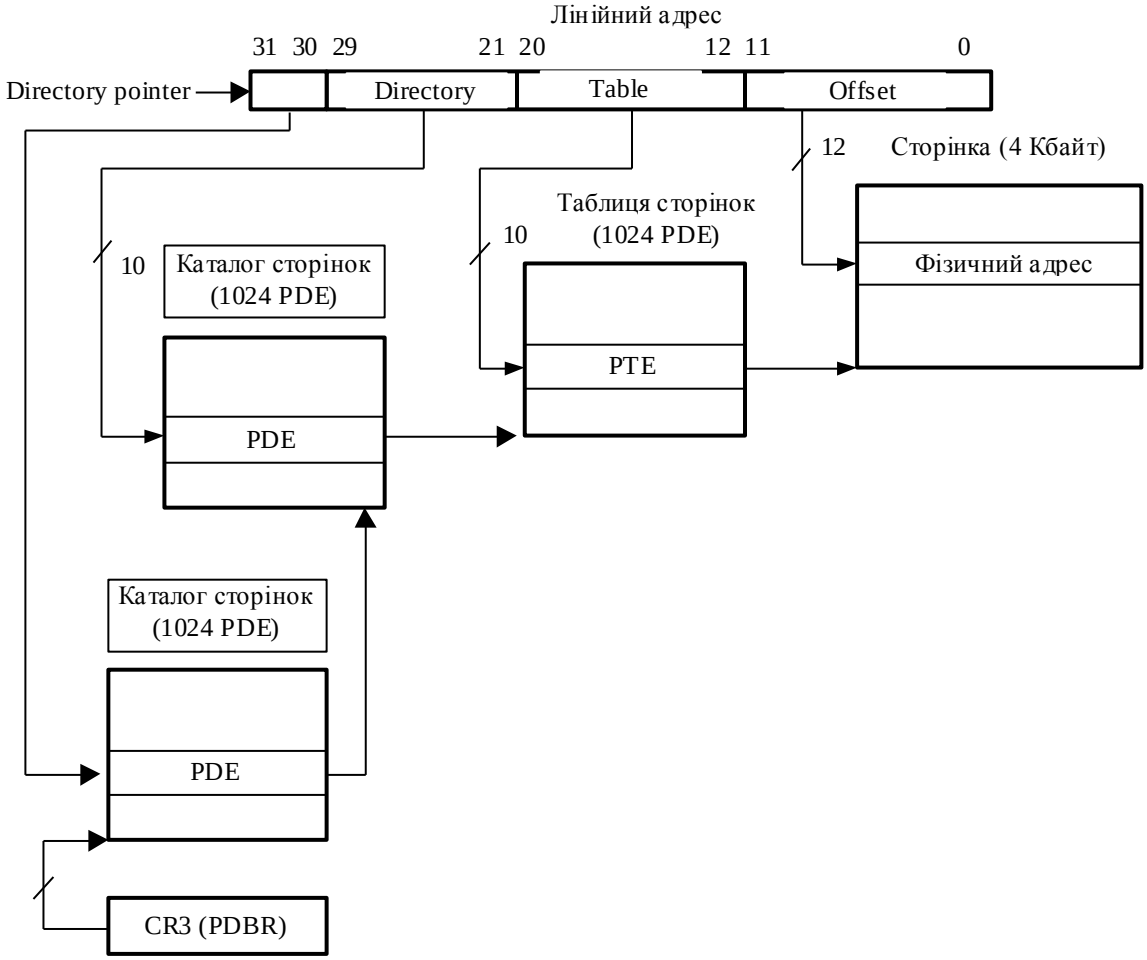


Рис. 3.13. Сторінкова переадресація в режимі PAE для сторінок розміром 4 Кбайти.

Процесор P6 також підтримує розширення фізичної адреси (PAE) до 64 Гбайт. Вмикається установленням біта PAE в регістрі CR4, при цьому біт PSE ігнорується (розширення недоступне). Оскільки у процесорів 6-го покоління розрядність зовнішньої шини адреси складає 36 біт, то при CR4.PAE=0 , біти адреси A[35:32] встановлені в нуль. При CR4.PAE=1 старші 4 біта появляються тільки в результаті роботи блока сторінкової переадресації. Причому при PS=0 вибираються сторінки розміром 4 Кбайти (рис. 3.13), а при PS=1 сторінки розміром 2 Мбайти (рис. 3.14). У режимі PAE, блок сторінкової переадресації оперує з 64-бітовими елементами (рис. 3.15). Ще однією відмінністю від звичайних режимів сторінкової переадресації є наявність маленької таблиці 64-бітових покажчиків.

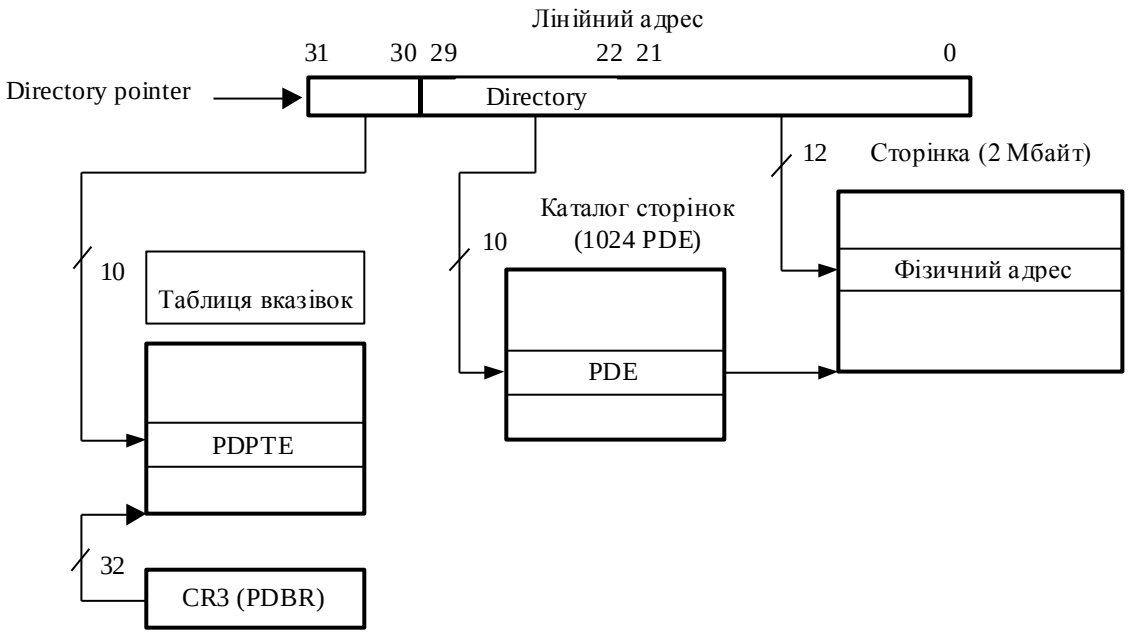


Рис. 3.14. Сторінкова переадресація в режимі PAE для сторінок розміром 2 Мбайти.

Наявність у процесора розширень розмірів сторінок і фізичної адреси можна виявити за допомогою інструкції CPUID. Якщо процесор цю інструкцію не підтримує, то у нього немає і цих засобів.

3.7. Віртуалізація переривань

Відповідно до базової архітектури 32-розрядних процесорів, у захищеному режимі інструкції CLI і STI, які керують прапорцем переривання IF, є чутливимими до рівня привілеїв. Якщо рівень привілеїв задачі дозволяє (CPL ≤ IOPL), вони будуть впливати на прапорець переривань, як і інструкції, що впливають на цей прапорець неявно. Спроба їхнього використання при CPL>IOPL викликає виняток-відмову #GP. Інструкції, що управляють прапорцем переривання неявно, винятків не викликають, але і не змінюють стан прапорця.

Прикладні програми повинні мати можливість захистити себе від впливу зовнішніх переривань. Для захисту критичних ділянок коду звичайно застосовують інструкції CLI і STI, що можуть зустрічатися в прикладних задачах досить часто. Відпрацьовування винятків щораз при їхній появі значно знижує продуктивність обчислювального процесу. У той же час дозволяти прикладним програмам напряму управляти прапорцем переривання процесора далеко не завжди припустимо, оскільки це може призводити до втрати керування операційною системою. У багатозадачній системі з розділюваними пристроями, зовнішні переривання спочатку опрацьовуються операційною системою, яка визначає, до якої задачі

а

б

в

г

Рис. 3.15. Структура 64-бітних елементів сторінкового перетворення: а – рядок таблиці покажчиків на каталоги; б – рядок каталога для сторінки розміром 4 Кбайти; в – рядок таблиці для сторінки розміром 4 Кбайти; г – рядок каталогу для сторінки розміром 2 Мбайти.

(задач) відноситься кожне конкретне переривання. ОС повинна мати можливість повідомити відповідній задачі (або усім) про його виникнення, але тільки в той момент, коли задача має стан із дозволеними перериваннями.

Задача, що повинна опрацьовувати апаратні переривання, може одержувати їх як у реальному виді (як зовнішні переривання), так і у віртуальному. Коли додатку передаються реальні переривання, для нього стане проблематичним використання віртуальної пам'яті з підкачуванням сторінок по запиту. Якщо сторінка з оброблювачем виявиться вивантаженою, то при виникненні апаратного переривання буде потрібно підкачування сторінки, що займає значний час. Для оброблювачів переривань такий час реакції неприйнятний.

Щоб уникнути підкачування сторінок по апаратних перериваннях, оброблювачі переривань повинні розташовуватися в ядрі ОС, що постійно присутнє у пам'яті. Якщо задача-приймач переривання знаходиться в стані, який можливо переривати , ОС сигналізує їй про виникнення переривання відразу. Якщо задача знаходиться в стані, який неможливо перервати, ОС сигналізує їй про переривання тільки після переходу в стан, що переривається . Таким чином, задачі передаються віртуальні переривання. У цьому випадку можливо використання віртуальної пам'яті, оскільки оброблювачі, що виконують критичні за часом дії, завжди знаходяться в пам'яті.

Віртуалізація переривань на основі вищезгаданих базових властивостей процесора виконується ОС чисто програмно, проте опрацювання потоку частих винятків дуже неефективне. У процесорах Pentium і останніх моделях 486 з'явилися нові апаратні засоби віртуалізації переривань. У регістрі EFLAGS з'явилися прапорці віртуального дозволу переривання VIF і переривання, що очікує, VIP. У регістрі CR4 біт PVI (Protected-Mode Virtual Interrupt) дозволяє віртуалізацію прапорця переривань у захищеному режимі для задач із CPL=3. При VIP=0, а також при VIP=1 і CPL<3 інструкції CLI і STI будуть поводитися вищеописаним способом - або управляти прапорцем IF, або викликати виняток #GP. При VIP=1 і CPL=3 їхнє поводження змінюється:

- Якщо IOPL=3, то CLI і STI перемикають прапорець IF.
- Якщо IOPL<3 і VIP=0 (немає переривання, що очікує), то CLI/STI просто управляють прапорцем VP.
- Якщо IOPL<3 і VIP=1 (є присутнім переривання, що очікує,), то спроба дозволу переривання (установки VIF), використовуючи STI, викликає виняток #GP. Його оброблювач і донесе задачі звістку про зовнішнє переривання, що мало місце.

Отже, прапорець віртуального дозволу переривання VIF указує на стан задачі, що переривається або що не переривається, і виконується при CPL=3; цей прапорець не торкається всього процесора. Прапорець переривання, що очікує, VIP встановлюється в образі регістрів задачі операційною системою, якщо зовнішнє переривання прийшло в момент, коли задача знаходилася в стані, який неможливо перервати. Моніторинг переходу задачі в стан із дозволеними перериваннями здійснюється процесором автоматично без зайвих витрат часу. Прапорці, що беруть участь у віртуалізації переривань:

- VI P - модифікується інструкцією IRETD, якщо CPL=0.
- VIF - модифікується інструкцією IRETD, якщо CPL=0; інструкціями CLI і STI як було показано вище.
- IOPL - модифікується інструкцією IRET(D) і POPF(D), якщо CPL=0.
- IF - модифікується інструкцією IRET(D) і POPF(D), якщо CPL $\leq$ IOPL, інструкціями CLI і STI як було показано вище.

Наявність у процесора засобів віртуалізації переривань можна виявити за допомогою інструкції CPUID. Якщо процесор цю інструкцію не підтримує, то в нього немає і цих засобів.

3.8. Режим віртуального 8086 (V86 і EV86)

Прикладні програми для 8086 можуть здійснюватися на 32-розрядних процесорах як у реальному режимі, так і в режимі віртуального 8086 (V86), що є особливим станом задачі захищеного режиму. Призначення цього режиму - формування віртуальної машини, що емулює процесор 8086.

Віртуальна машина формується програмними засобами операційної системи - монітором V86, що підтримується спеціальними апаратними засобами процесора. Режим V86 дозволяє користуватися апаратними засобами підтримки багатозадачності. У цьому режимі працює захист і механізм сторінкової переадресації, що дозволяє адресуватися до будь-якої області 4-Гігабайтового простору фізичної пам'яті. Виконання додатків 8086 у середовищі V86 можливо паралельно з додатками захищеного режиму. ОС захищеного режиму емулює оточення звичайної машини 8086, включаючи переривання і перехват звертань до портів. Однак при IOPL<3 (забезпечує стійкість системи з повною віртуалізацією), продуктивність віртуальної машини буде низькою.

Проблему віртуалізації переривань вирішує розширений режим EV-86 (появився в останніх моделях 486 – вмикається при CR4.VME=1). В цьому режимі сегмент стану задачі має 32-байтову

структуру – карту перенаправлення переривань (interrupt redirection bitmap). Кожний біт цієї карти відповідає одному з 256 (32x8) програмних переривань, що викликаються інструкцією INT n. На програмні переривання, що викликаються іншим чином, винятки та апарпатні переривання карта перенаправлення не діє. Якщо відповідний біт в карті встановлений, то переривання викличе виняток-відмову з виходом із EV-86, а якщо біт скинутий, то переривання обробляється без виходу з EV-86.

3.9. Перемикання «реальний-захищений режим»

Перемикання процесора в захищений режим із реального здійснюється завантаженням у CR0 слова з одиничним значенням біта PE (Protect Enable). Для сумісності з ПО для 80286 біт PE може бути встановлений також інструкцією LMSW. До перемикання в пам'яті повинні бути проініціалізовані необхідні таблиці дескрипторів LDT і GDT. Відразу після вмикання захищеного режиму процесор має CPL=0. Для всіх 32-розрядних процесорів рекомендується виконувати таку послідовність дій для перемикання в захищений режим.

1. Заборонити переривання, що маскуються скиданням прапорця IF, а виникнення немаскованих переривань блокувати зовнішньою логікою. Програмний код на час «перехідного періоду» повинний гарантувати відсутність винятків і не використовувати програмних переривань. Ця вимога викликана зміною механізму виклику оброблювачів переривань.
2. Завантажити в регістр GDTR базову адресу GDT (інструкцією LGDT).
3. Інструкцією MOV CRO установити прапорець PE, а якщо необхідне сторінкове керування пам’яттю, то і прапорець PG.
4. Відразу після цього повинна виконуватися команда міжсегментного переходу (JMP Far) або виклику (CALL Far) для очищення черги інструкцій, декодованих у реальному режимі, і виконання серіалізації процесора. Якщо вмикається сторінкове перетворення, то коди інструкцій MOV CR0 і JMP або CALL повинні знаходитися в сторінці, для якої фізична адреса збігається з логічною (для коду, якому передається керування, ця вимога не ставиться).
5. Якщо планується використання локальної таблиці дескрипторів, інструкцією LLDT завантажити селектор сегмента для LDT у регістр LDTR,
6. Інструкцією LTR завантажити в регістр задач (TR) селектор TSS для початкової задачі захищеного режиму.

7. Перезавантажити сегментні регістри (крім CS), вміст яких ще відноситься до реального режиму або виконати перехід, або виклик іншої задачі (при цьому перезавантаження регістрів відбудеться автоматично). У сегментні регістри, що не використовуються, завантажується нульове значення селектора.
8. Інструкцією LIDT завантажити в регістр IDTR адресу і ліміт IDT - таблиці дескрипторів переривань захищеного режиму.
9. Дозволитити масковані і немасковані апаратні переривання.

Перемикання процесора з захищеного режиму в реальний можливо не тільки через апаратне скидання, як це було в 80286, але і за допомогою скидання біта PE в CRO. При цьому для коректного переходу, відповідно до документації на процесори, повинні виконуватися такі дії:

1. Заборонити масковані переривання скиданням прапорця IF, а немасковані - зовнішньою схемою.
2. Якщо ввімкнена сторінкова трансляція адрес, то необхідно забезпечити рівність лінійних і фізичних адрес для поточного виконуваного коду (перейти на таку сторінку), а також для таблиць GDT і IDT, занулити біт PG в регістрі CRO і завантажити нулі в CR3 для очищення кеш-буфера TLB.
3. Передати керування сегменту, що читається з лімітом 64 Кбайт.
4. Завантажити в сегментні регістри SS, DS, ES, FS і GS селектор дескриптора (ненульовий), у якому встановлений ліміт 64 Кбайт, байтова дробність (G=0), розширюваність нагору (E=0), доступність запису (W=1) і присутність (P=1). Якщо сегментні регістри не перезавантажувати, виконання буде продовжуватися з атрибутами, успадкованими від захищеного режиму.
5. Ініціалізувати таблицю векторів переривань реального режиму (у межах першого мегабайта) і указати на неї інструкцією LIDT.
6. Скинути біт PE для переходу в реальний режим.
7. Виконати дальній перехід на програму реального режиму, що скине чергу інструкцій, декодованих у захищеному режимі, і завантажить відповідні права доступу до сегмента коду.
8. Завантажити коректне значення в сегментні регістри і покажчик стеку.
9. Дозволити переривання.

Після цього завантажуються інші регістри. Процесор тепер працює в реальному режимі, за замовчуванням з 16-розрядними адресами і даними.

Кроки 3 і 4 призначені для завантаження програмно-недоступних регістрів дескрипторів сегментів параметрами стандартного реального режиму. Проте замість них можна створити і «нереальний» (Unreal, по Р. Коллінзу) режим, що відрізняється від

реального можливістю доступу до сегментів великого (до 4 Гбайт) розміру. Правда, у процесорів 80286 і 80386 ліміт кодового сегмента примусово обмежується розміром 64 Кбайт, але в старших процесорів великий розмір допустимий для всіх сегментів. «Нереальний режим» часто використовується менеджерами пам'яті для DOS і ігровими програмами, що потребують великого об’єму пам'яті.

Контрольні питання

1. Які відмінності захищеного режиму 32-х розрядних процесорів від захищеного режиму i286?
2. Як формується лінійна адреса в захищеному режимі?
3. Які рівні захисту пам'яті використовуються в захищеному режимі?
4. Які типи таблиць дескрипторів використовуються в захищеному режимі?
5. Формати селекторів та дескрипторів сегментів?
6. Поясніть принципи організації віртуальної пам'яті.
7. Які основні правила привілеїв діють в захищеному режимі?
8. Поясніть принципи перемикання задач в захищеному режимі?
9. Сторінкове керування пам'яттю і його особливості.
10. Як виконується захист на рівні сторінок?
11. Дайте пояснення поняття "віртуалізація переривань".
12. Охарактеризуйте режими віртуального процесора 8086 - V86 і EV86.
13. Приведіть послідовність дій для перемикання із реального режиму в захищений.
14. Як виконується перемикання процесора з захищеного в реальний режим?

Глава 4. Додаткові можливості 32-х розрядних процесорів

В цьому розділі розглядаються режими «нерегулярної» роботи процесорів, що залишаються «за кадром» для звичайного користувача. Опис режиму SMM разом з запуском і налагодженням може показатися дивним, але насправді це є цілком виправданим. Цей режим з окремим адресним простором пам'яті стоїть окремо від реального, захищеного і V86 режимів і може використовуватися навіть із метою налагодження.

4.1. Початкове скидання і тестування

Апаратне скидання (hardware reset) виконується процесором при ввімкнені живлення і за сигналом RESET. За високим рівнем сигналу на вході RESET (за низьким рівнем сигналу RESET* у P6) процесор припиняє виконання інструкцій і перестає керувати локальною шиною. Після закінчення сигналу RESET процесор зчитує конфігураційну інформацію з деяких вхідних ліній інтерфейсу (у залежності від типу процесора) і починає функціонування. У такий спосіб установлюються: коефіцієнт множення тактової частоти, режим (WB/WT) роботи кеша, роль процесора в багатопроцесорних системах і спосіб подання сигналів переривань (для процесорів, що мають APIC).

Якщо під час закінчення сигналу RESET на певному вході процесора утримувати низький рівень сигналу, процесор почне виконувати внутрішній тест BIST (Built-in Self-Test). Тестуванню піддається більша частина обладнання процесора, що для 80386 і 486 потребує приблизно мільйона тактів, а для P6 - біля 5,5 млн. тактів, що займає десятки мілісекунд. Після закінчення самотестування процесор починає роботу як після звичайного скидання, а регістр EAX містить сигнатуру результату тестування. Про успішне виконання тесту свідчить нульове значення сигнатури.

Скидання переводить процесор у реальний режим і встановлює ряд регістрів у визначений стан. Зокрема:

- FLAGS = 0002h, біти VM і RF його розширення занулюються;
- у регістрі CR0 занулюються біти PG, TS, EM, MP і PE;
- CS = F000h;
- EIP = 0000FFF0h;
- DS = ES = SS = FS = GS = 0000h;
- Регістр DX містить інформацію про тип процесора.

Апаратне скидання анулює рядки кеш-пам'яті, буфери трансляції (TLB) і таблиць переходів (BTB), а також встановлює

регістри FPU і MMX у певний стан (але не такий, як за інструкцією FINIT). При такому сполученні регістрів CS:IP процесор, знаходячись у реальному режимі, починає виконання інструкції, зчитаної за фізичною адресою FFFFFFF0h. Це відбувається тому, що в прихованих регістрах кодового сегмента встановлюється база FFFF0000h і ліміт 0FFFFh, а саме вони і визначають реальне положення сегмента. Таке визначення діє до першої зміни CS при команді міжсегментного переходу або при обслуговуванні переривання чи винятку. Програмний код, що виконується, повинен забезпечити ініціалізацію системи (регістрів процесора, структур даних у пам'яті).

Отже, після скидання і до першої команди міжсегментного переходу або виклику на шині адреси в реальному режимі біти A[20:31] у циклах вибірки команд мають одиничне значення. З цього випливає, що принаймні на початковий період часу після сигналу RESET комп'ютер повинен мати образ BIOS в адресах FFFFFFF0-FFFFFFFFh, у той час як у PC на базі 8086/88 ROM BIOS розташовувалася під межею першого мегабайта (FFFFFh), а AT-286 мала її образ і під межею 16-го мегабайта (FFFFFFh). Видалення BIOS із першого мегабайта пам'яті в режимі нормальної роботи неможливе, оскільки вектори переривань, що посилаються на сервіси BIOS, у реальному режимі можуть адресуватися тільки до пам'яті в діапазоні адрес 0-0FFFFFh (0-10FFEF при відкритому вентилі Gate A20).

Процесори Pentium і старші мають додатковий вхід INIT. Дії за цим сигналом відрізняються від апаратного скидання такими моментами:

- тест BIST не запускається;
- внутрішня кеш - пам'ять не очищається (але TLB і BTB скидаються);
- значення регістрів MSR (включаючи MTRR) не змінюються;
- стан FPU не змінюється.

Цей сигнал може використовуватися для переведення процесора в реальний режим (у стилі 80286) із зберіганням даних у кеші. Таке ж "м'яке" скидання можливе і за повідомленням, одержаним процесором по шині APIC (Advanset Peripheral Interrupt Controller – удосконалений контролер переривань). Процесори P6 і Pentium орієнтовані на мультипроцесорні системи. Для них початкове скидання має додаткові нюанси:

- Pentium: роль процесора (первинний/вторинний) визначається апаратними сигналами в момент спаду RESET. Програмний код ініціалізації виконує тільки первинний процесор, вторинний

процесор зупинено. Запуск вторинного процесора виконається за повідомленням, прийнятим від первинного процесора по шині APIC. Це повідомлення видається під керуванням програми ініціалізації, що виконується.

- P6: усі процесори на шині одночасно виконують протокол мультипроцесорної ініціалізації на шині APIC, визначаючи завантажувальний (BSP - bootstrap processor). Далі програмний код ініціалізації виконує тільки BSP, інші процесори зупинені. Їх запуск відбудеться за повідомленнями, переданими по шині APIC від BSP під керуванням програми, що виконується.

Тестування апаратних засобів процесора можливо не тільки по апаратному скиданню (тест BIST). Починаючи з процесорів 80386, стали застосовуватися тестові регістри TRn, за допомогою яких виконується тестування буферів TLB, а в 486 - і внутрішнього кеша. Починаючи з Pentium, тестові регістри, значно змінені за складом, були введені до складу регістрів MSR.

Починаючи з деяких моделей 486, у процесори стали вводити підтримку тестового інтерфейсу JTAG. Для його використання необхідно зовнішнє (стосовно системи, що тестується) устаткування (комп'ютер), що під’єднується до процесора усього чотирма сигнальними лініями. Для під’єднання по JTAG можливе використання перехідних колодок між сокетом і процесором. Сигнали інтерфейсу JTAG не пов'язані з основним інтерфейсом процесора. По інтерфейсу JTAG можливий запуск тесту BIST, а також вплив на входи і контроль вихідних сигналів процесора відповідно до сценарію програми тестування. Інтерфейс JTAG може застосовуватися також для цілей налагодження.

Процесори, починаючи з Pentium, мають вбудовані засоби перевірки функціонування апаратури під час роботи. У випадку виявлення апаратної помилки ці процесори формують виняток #MC (Machine Check Exception - виняток машинного контролю). Відомості про виявлену помилку зберігаються в спеціальних регістрах із складу MSR. За їхнім вмістом оброблювач #MC може дати повідомлення операційній системі і визначити можливість рестарта інструкції, під час якої помилка була виявлена. Можливості машинного контролю Pentium обмежені перевіркою паритету шини даних при операціях зчитування й успішності завершення шинних циклів обміну. Для обслуговування #MC призначені два регістри P5_MC_TYPE (тип помилки) і P5_MC_ADDR (адреса). У P6 застосовується архітектура MCA (Machine Check Architecture) із розширеними можливостями: контролюються паритет на шині адреси, а також помилки ECC-контролю, кеш-пам'яті і буферів TLB. Склад регістрів розширений. Є глобальні регістри опису можливостей контролю (MCG_CAP), стану (MCG_STATUS) і

керування (MCG_CTL, у Pentium Pro відсутній), а також банк регістрів повідомлень про помилки. Кожному апаратному блоку, що тестується, в банку відповідає набір із регістрів керування (MC i_CTL), стану (MC i_STATUS), адреси (MC i_ADDR) і змішаного призначення (MC i_MISC). Наявність засобів машинного контролю визначається за ознакою MCE, а на розширену архітектуру вказує ознака MCA в списку властивостей, що повідомляються по інструкції CPUID(1). Тестування внутрішніх вузлів сильно залежить від моделі процесора.

4.2. Програмні засоби налагодження

Процесори мають внутрішні засоби, призначені для полегшення процесу налагодження програмного забезпечення. Ці засоби дозволяють передати керування програмі - дебугеру при настанні якоїсь заздалегідь визначеної події, наприклад, виконанню інструкції з заданої адреси, звертанню до комірки пам'яті або до порту введення/виведення, виконанню чергової інструкції. Така програма дозволяє проаналізувати, а якщо треба, то і модифікувати стан процесора, пам'яті і портів у точці зупинки і продовжити виконання програми, можливо, встановивши нові умови зупинки. Дебугер одержує керування через оброблювачі таких винятків:

- Виняток 1 - #DS (Debug exception) - подія налагодження.
- Виняток 3 - #BP (Breakpoint exception) - передача керування по інструкції INT 3.

Для задання подій налагодження використовується ряд засобів:

- Ознака TF (trap) у регістрі EFLAGS - генерація винятку #DB після виконання будь-якої інструкції.
- Ознака T (trap) у TSS - генерація винятку #DB при переключенні на задачу з встановленим прапорцем (286+).
- Регістри налагодження DR0-DR3, DR6, DR7 - генерація винятку #DB по заданих адресах пам'яті (386+) і введення/виведення (Pentium+).

Однобайтова інструкція INT3, що використовується програмними дебугерами, викликає виняток #BP. Від двобайтових інструкцій INT n ця інструкція, крім компактності, відрізняється також тим, що вона нечуттєва до IOPL як у захищеному режимі, так і в режимі V86. Дебугер (або програміст під час налагодження) розставляє ці інструкції (код CCh) замість кодів операцій у необхідних точках. Такі точки зупинки можуть бути поставлені тільки в ОЗП, що не дозволяє користуватись цим методом при налагодженні програмного коду, записаного в ПЗП. Перевагою

даного способу зупинки є необмеженість числа можливих точок зупинки.

Виняток #DB визивається за подіями налагодження або після виконання кожної інструкції при встановленому прапорці покрокового режиму TF. Коли в образі регістра ознак дебугером встановлюється TF=1, при поверненні з оброблювача точки зупинки інструкція IRET встановить прапорець TF, і після виконання такої інструкції спрацює пастка.

Регістри налагодження, що прийшли в архітектуру x86 із процесором 80386, надають більш прогресивні засоби, що дозволяють розставляти точки зупинки навіть у ПЗП і перехоплювати звертання до даних. Розширення налагодження, впроваджені в архітектуру Pentium, дозволяють зупинятися і по звертаннях до портів. Структура регістрів приведена на рис. 4.1.

Рис. 4.1 - Регістри налагодження

Процесори мають чотири регістри для збереження 32-бітових лінійних адрес контрольних точок DR0... DR3.

Регістр DR7 (Debug Control Register) використовується для керування налагодженням. Поля LENi визначають розмір області, що адресується регістрами DR0...DR3, влучення в який викликає спрацьовування пастки:

- 00 - байт;
- 01 - слово (2 байти);
- 10 - невизначено;
- 11 - подвійне слово.

Поля RWi задають тип звертання, що перехоплюється:

- 00 - вибірка інструкції;
- 01 - тільки запис даних у пам'ять;
- 10 - у 80386 і 486 не використовується; у процесорах P5+ при вмиканні розширень налагодження (біт DE=1 у регістрі CR4) - звертання до портів вводу/виводу;
- 11 - читання або запис даних у пам'ять.

За типом винятку, перехоплення вибірки інструкції класифікується як відмова (fault, обробляється до виконання інструкції), а звертання до даних - як пастка (trap, опрацьовується після передачі даних).

Біт GD (Global Debug Register Access Detect), доступний тільки в реальному режимі або в захищеному на рівні CPL=0, дозволяє відслідковувати будь-які спроби звернення до регістрів налагодження. При GD=1 будь-яка спроба звернення викликає виняток #DB (відмова).

Біти GE і LE (Global і Local Exact data breakpoint match) визначають, чи буде генеруватись виняток відразу після завершення операції обміну при ввімкнутій пастці на область даних або воно відбудеться трохи пізніше (або ніколи). Біт LE автоматично скидається при переключенні задач, на біт GE переключення не діє. Пастка на інструкції спрацьовує завжди відразу.

Біти Gi і Li (Global і Local breakpoint enable) дозволяють спрацьовування пасток по контрольних точках. Біти Li автоматично скидаються при перемиканні задач, на біти Gi перемикання не діє. Автоматичне скидання біта Li запобігає зайвим спрацьовуванням при перемиканні задач.

Регістр стана налагодження DR6 (Debug Status Register) дозволяє оброблювачу #DB легко визначити його причину. Причин може бути декілька, їх ідентифікують біти DR6:

- Bi - спрацьовування контрольної точки за регістром DRi;
- BS - пастка покрокового режиму;
- BT - пастка перемикання задач (за бітом T в TSS);
- BD - відмова в спробі доступу до регістрів налагодження при GD=1.

Ознаки Bi мають достовірне значення тільки для контрольних точок із установленими бітами Li або (і) Gi.

Ознака RF (Resume Flag) у регістрі EFLAGS дозволяє заборонити генерацію повторних винятків (після виходу з оброблювача) по одній контрольній точці.

Процесори P6 мають засоби реєстрації останнього переходу, переривання або винятку. Для цього до складу MSR введені п'ять 32-бітових регістрів.

У регістрі DebugCtlMSR визначені такі поля:

- Біт 0 - LBR (Last Branch/Interrupt/Exception) - дозвіл реєстрації вихідної і цільової адреси переходу/переривання/винятку. За винятком налагодження ознака автоматично скидається.
- Біт 1 - BTF (Single-step on Branches) - встановлення покрокового режиму по виявленні переходу. Ознака як і TF автоматично скидається після входу в оброблювач винятку налагодження;
- Біти [2:5] - PBi (Performance monitoring/Breakpoint pins) - дозвіл індикації (імпульсами) проходження точок зупинки, визначених регістрами DRO... DR3 на виводи BPO#... BP3#. При PBi=0 виводяться події монітора продуктивності.
- Біт 6 - TR (Trace message enable) - дозвіл виводу вихідної і цільової адреси переходів/переривання/винятки на системну шину у вигляді повідомлень трасування переходів (при цьому в регістрах трасування інформація буде невизначеною).

У регістр LastBranchFromIP процесор заносить значення IP, яке відповідає інструкції розгалуження, а в LastBranchToIP - цільову адресу переходу. При виникненні переривання або винятку (крім винятку налагодження) в цих регістрах зберігається адреса інструкції, під час якої відбувся виняток (переривання) і адреса оброблювача. Попередньо вміст цих регістрів копіюється в LastExceptionFromIP і LastExceptionToIP.

Вищеописані засоби розуміють наявність дебугера, під керуванням якого виконується програма, що налагоджується. Дебугер (тобто програма, яка призначена для налагодження) звичайно функціонує під управлінням операційної системи. З цього слідує, що операційна система повинна бути роботоздатною. Крім того, програма, що налагоджується, може випадково або навмисно (наприклад, із метою захисту від зламування) «зіпсувати» дебугер або блокувати його дії. Для таких випадків, а також для налагодження системних засобів (ОС або BIOS) вимагаються «незалежні» засоби налагодження. Такими засобами є внутрішньосхемні емулятори (ICE - In-Circuit Emulator), що представляють собою дуже дорогі спеціалізовані програмно-апаратні комплекси. Засоби підтримки внутрішньосхемного емулятора вбудовувалися в процесори, починаючи з 80386, проте не висвітлювалися в загальнодоступній документації. У цих процесорах є вищеописані регістри налагодження, а влучення в точку зупинки, замість винятку #BP, може привести до переходу в режим ICE. Для такого перенаправлення призначена інструкція

ICEBP. У режимі ICE процесор звертається до шини за альтернативними сигналами, і замість оперативної пам'яті комп'ютера, що налагоджується, процесору підставляється внутрішня пам'ять ICE. Для шини комп'ютера, що налагоджується, процесор здається відключеним (керуючі сигнали шинних циклів не формуються). У режимі ICE можливі доступ і до звичайної пам'яті - для цього існують інструкції UMOV (аналогічні звичайним MOV). Для виходу в нормальний режим служить інструкція LOADALL, що попередньо завантажує усі (включаючи невидимі) регістри процесора з образу, що зберігається в пам'яті емулятора (ця інструкція існує тільки в 80286 і 80386, причому коди і формати образа для цих двох процесорів не збігаються). Вхід у режим ICE можливий також за апаратним сигналом від емулятора.

Для цілей налагодження можна використовувати режим SMM (якщо налагоджують не його оброблювач): за апаратним перериванням SMI процесор виходить у «паралельний» простір пам'яті, попередньо вивантаживши в неї вміст всіх видимих і невидимих регістрів процесора. Знаючи структуру їхнього образу (модельно-специфічну), можна проаналізувати стан процесора і навіть обминути захист (змінивши в образі значення CPL на нульове й отримавши необмежені привілеї). Режим SMM дуже нагадує ICE - як за ідеєю, так і за структурою даних що зберігаються в образі. Проте в SMM не можна ввійти за подіями налагодження, і процесор у цьому режимі не має вбудованих засобів доступу до звичайної пам'яті.

Більш дешевий і потужний варіант зовнішнього дебугера використовує зондовий режим (probe mode),що забезпечується засобами JTAG в процесорах, починаючи з Pentium.

4.3. Режим зондового налагодження

Можливість зондового налагодження з'явилася в процесорах Pentium. Зондовий режим надає можливості налагодження, що перевершують внутрішньосхемну емуляцію (ICE), доступну для попередніх процесорів. Раніше процесор виводився з нормального режиму роботи в ізольований адресний простір, що дозволяло цілком ізолювати програмне забезпечення емуляції від середовища, що налагоджується. Це середовище втрачало управління процесором - він йому представлявся неактивним. Аналогічний «відхід» здійснюється й у режимі SMM, що також може використовуватися в цілях налагодження. Проте процесор у режимі ICE або SMM підтримував активність на шині, що забезпечувало йому вибірку: декодування і виконання програмного коду емулятора.

Зондовий режим - Probe Mode - забезпечує можливість дій налагодження при зовнішньо, дійсно статичному, стані процесора. Інструкції налагодження і результати їх виконання передаються між процесором системи, що налагоджується і комп'ютером, що тестує, по інтерфейсу JTAG, ізольованому від системної шини. Цикли звертання в пам'яті або в портах з'являться на системній шині тільки в тому випадку, коли їх зажадає дебугер для аналізу і (або) модифікації вмісту.

Зондовий режим організований як розширення набору інструкцій JTAG, що використовують декілька нових регістрів boundary scan і пару додаткових сигналів R/S# і PRDY. Нові інструкції забезпечують вхід у режим і вихід із нього, побудова зондової інструкції в регістрі PIR (Probe Instruction Register) і її виконання, доступ до регістра даних зонда PDR (Probe Data Register) і деякі інші функції. У зондовому режимі процесор неактивний: вибірка інструкцій не провадиться, переривання (включаючи NMI і SMI) і винятки залишаються не обслугованими до виходу з режиму. Цикли спостереження і заповнення рядків кеша в зондовому режимі можуть проводитися, оскільки операції з пам'яттю можуть виконуватися і при дозволеному кешуванні.

У зондовий режим можна ввійти трьома способами:

- За інструкцією Begin Probe Mode. Її прийом по JTAG зупиняє процесор на найближчій межі інструкцій. Установкою сигналу PRDY процесор сигналізує про готовність до прийому зондової інструкції з JTAG. Вихід із зондового режиму можливий за інструкцією End Probe Mode або сигналом R/S# (його потрібно переключити з високого рівня в низький і навпаки).
- За переходом в низький рівень сигналу R/S#. Про готовність процесор також повідомляє сигналом PRDY. Вийти з режиму в цьому випадку можна тільки по зміні R/S# у високий рівень.
- За винятком налагодження #SP. Для цього в регістрі PMCR (Probe Mode Control Register) повинний бути встановлений біт 1BP (біт 0), що штатно здійснюється інструкцією ICEBP, переданою по JTAG. У цьому випадку за будь-якою подією налагодження (див. п. 3.2) замість генерації винятку процесор переходить у зондовий режим, про що сигналізує високим рівнем PRDY. Вихід із режиму можливий як за інструкцією End Probe Mode, так і за сигналом R/S#.

Зондові інструкції збираються в регістрі PIR. Цей регістр має повне керування усіма виконавчими блоками процесора (для Pentium це обидва конвеєри і ядро FPU), тому зондові інструкції дуже прив'язані до моделі процесора. Інструкції забезпечують доступ (читання і запис) до усіх внутрішніх регістрів процесора,

включаючи MSR і сховані регістри. На дії цих інструкцій немає ніяких обмежень захисту.

4.4. Режим системного керування SMM

Сучасні моделі 32-розрядних процесорів (починаючи з деяких модифікацій 486 і 386SL), крім звичайних режимів - реального, захищеного і V86, - мають додатковий режим системного керування SMM (System Management Mode). Цей режим призначений для виконання ряду дій із повною ізоляцією їх від прикладного програмного забезпечення і навіть операційної системи. Головним чином, цей режим призначений для реалізації системи керування енергоспоживання, хоча може використовуватися й в інших цілях.

У режим SMM процесор може ввійти тільки за сигналом на вході SMI# (System Management Interrupt), процесори P5+ можуть ввійти в SMM і по прийомі відповідного повідомлення по шині APIC. Сигнал SMI# для процесора є перериванням, що не маскується із найвищим пріоритетом. Виявивши активний сигнал SMI# (низький рівень), процесор після завершення поточної інструкції і розвантаженні буферів запису переключається в режим SMM. Для процесорів 386SL, 486 і Pentium про перехід у режим SMM свідчить вихідний сигнал SMIACT#, процесори P6 у SMM кожний цикл системної шини відзначають ідентифікатором EXF4. Відразу після входу в SMM процесор зберігає свій контекст - майже всі регістри - у спеціальній пам'яті SMRAM. Ця пам'ять є виділеною областю фізичної пам'яті, доступ до якої дозволяється зовнішніми (стосовно процесора) схемами в шинних циклах звертання до пам'яті тільки при наявності сигналу SMIACT# (EXF4 у P6). Після зберігання контексту процесор переходить до виконання оброблювача SMI, що розташований у тій ж пам'яті SMRAM. Оброблювач являє собою послідовність звичайних інструкцій, що виконуються процесором у режимі, що нагадує реальний. При вході в режим SMM автоматично забороняються апаратні переривання (включаючи і такі, що не маскуються) і не генерують винятки, так що дії процесора однозначно визначаються програмою оброблювача SMI. Процедура оброблювача завершується інструкцією RSM, за якою процесор відновлює свій контекст з образу, що зберігається у SMRAM, і повертається в звичайний режим роботи.

При поверненні з SMM можливі деякі варіанти в залежності від дій оброблювача (у межах можливості SMM даного процесора). По-перше, оброблювач може програмно внести зміни в образ контексту процесора, і при його відновленні процесор повернеться не в той стан, у якому відбулося SMI. По-друге, можливий вибір варіанта для випадку, коли переривання SMI виникнуло під час зупинки

процесора за інструкцією HALT: можна повернутися знову на інструкцію зупинки або перейти до виконання наступної за нею інструкції. По-третє, процесори, починаючи з Pentium другого покоління (і Enhanced 486 фірми AMD), підтримують можливість рестарту (повторного виконання) інструкції введення/виведення, що передує появі сигналу SMI#.

Можливість рестарту інструкції введення/виведення є розширенням режиму SMM. Її використовують, наприклад, коли прикладна програма (або системний драйвер) намагається звернутися з операцією введення/виведення до периферійного пристрою, що знаходиться в режимі «сну». Системна логіка в цьому випадку повинна сформувати сигнал SMI# раніше сигналу RDY#, що завершує шинний цикл і рестартує інструкції введення/виведення. Оброблювач SMI «розбудить» пристрій, після чого операція введення/виведення рестартує і прикладне ПЗ (або драйвер) «не помітить», що пристрій перебував у режимі “сну”. Таким чином, керування енергоспоживанням може бути організоване на рівні BIOS способом, цілком прозорим для програмного забезпечення (у тому числі й ОС). Прозорість SMM забезпечується такими властивостями цього режиму:

- можливістю тільки апаратного входу в SMM; виконанням коду SMM в окремому адресному просторі;
- повним зберіганням стана перерваної програми в області SMRAM;
- забороною звичайних переривань;
- відновленням стана перерваної задачі по виході з режиму SMM.

Пам'ять SMRAM повинна бути фізично або логічно виділеною областю розміром від 32 Кбайт (мінімальні потреби SMM) до 4 Гбайт. SMRAM розташовується, починаючи з адреси SMIBASE (за замовчуванням 30000h), і розподіляється щодо цієї адреси в такий спосіб:

- FE00h-FFFFh (за замовчуванням 3FE00h-3FFFFh) - область зберігання контексту (розподіляється, починаючи зі старших адрес у напрямку до молодшої, табл. 4.1);
- 8000h (38000h) - точка входу в оброблювач (SMI Handler);
- 0-7FFFh (30000h-37FFFh) - вільна область.

Необережна модифікація полів, заборонених для запису, може призвести до непередбачених результатів. У зарезервованих полях автоматично зберігаються керуючий регістр CR4 і програмно невидимі сховані регістри дескрипторів для CS, DS, ES, FS, GS і SS, але місце розташування і формат їхнього образу залежать від моделі процесора. Фірма Intel їх оголосила такими, «що не

читаються», доступ до них дозволяє обминути сегментний захист. Знаючи їхнє місце розташування, можна, наприклад, написати такий оброблювач SMI, що виведе задачу V86 на CPL=0.

Таблиця 4.1- Карта контексту процесора в SMRAM

Зсув щодо SMBASE	Регістр (поле)	Дозвіл запису
FFFCh	CRO	Немає
FFFSh	CR3	Немає
FFF4h	EFLAGS	I
FFFOh	EIP	I
FFECh	EDI	I
FFESh	ESI	I
FFE4h	EBP	I
FFEOh	ESP	I
FFDCh	EBX	I
FFDSh	EDX	I
FFD4h	ECX	I
FFDOh	EAX	I
FFCCh	DR6	Немає
FFCSh	DR7	Немає
FFC4h	TR1	Немає
FFCOh	LDT Base1	Немає
FFBCh	GS1	Немає
FFB8h	FS1	Немає
FFB4h	DS1	Немає
FFBOh	SS1	Немає
FFACh	CS1	Немає
FFASh	ES1	Немає
FFA7h...	Зарезервовано	Немає
FF94h	IDT Base	Немає
FF93h...	Зарезервовано	Немає
FF88h	GDT Base	Немає
FF87h... FF04h	Зарезервовано	Немає
FF02h	Auto HALT Restart (слово)	I
FFOOh	I/O Instruction Restart (слово)	I
FEFCh	SMM Revision Identifier	Немає
FEF8h	SMBASE (подвійне слово)	I
FEF7h...	Зарезервовано	Немає

* Старші 2 байти зарезервовані

При вході в SMM автоматично не зберігаються:

- Регістри FPU.
- CR2.
- DR0...DR5.
- MSR, включаючи MTRR (для 486 - TR3...TR7).

- Стан контролера JTAG.
- Регістри машинного контролю.
- Регістри APIC.

Якщо режим SMM використовується для від’єднання живлення процесора з можливістю швидкого «пробудження», пам'ять SMRAM, що береже контекст процесора, повинна бути енергонезалежною. Пам'ять SMRAM повинна бути схемотехнічно захищена від доступу прикладних програм. Процесор генерує спеціальний вихідний сигнал SMIACT# (EXF4) під час опрацювання SMI, що і повинний бути ключем доступу до цієї пам'яті. Якщо SMRAM не є енергонезалежною, то системна логіка повинна забезпечити можливість її ініціалізації (запису програмного коду оброблювача) процесором із звичайного режиму роботи до появи сигналу SMI#.

Якщо пам'ять SMRAM «затінює» область 03У, то виникає питання про коректність кешування цієї області (по SMI автоматичне анулювання і розвантаження модифікованих рядків не провадиться). Виключити помилкові кеш-попадання в режимі SMM дозволяє подача сигналу FLUSH# одночасно з SMI# - тоді до входу в SMM відбудеться обернений запис і анулювання рядків (ця операція має більш високий пріоритет). Пасивний стан входу KEN# під час режиму SMI охоронить кешів від заповнення рядків. У P6 аналога сигналу KEN# немає - заборона кешування повинна виконуватися програмуванням MTRR.

- Операційний режим SMM має такі властивості: обчислення адрес аналогічно реальному режиму; ліміт обмежений - 4 Гбайти;
- прапорець переривань IF скинуто;
- переривання NMI заборонені;
- прапорець 7F у регістрі EFLAGS скинуто, покроковий режим заборонено;
- регістр DR7 скинуто, пастки налагодження заборонені;
- інструкція RSM дозволена (в інших режимах вона викликає виняток невірного коду операції);
- за замовчуванням використовується 16-бітовий режим регістрів, стека і кодів операцій.

Стан регістрів процесора при вході в SMM приведено в табл. 4.2. У режимі SMM можливо використання переривань, проте попередньо необхідно подбати про коректну ініціалізацію таблиці переривань (інструкцією LDTR), принаймні для використовуваних векторів. Апаратні переривання, що маскуються, можуть бути дозволені просто установленням прапорця IF, виклик переривання NMI замінюється програмним викликом його оброблювача.

Контекст математичного співпроцесора (і регістри MMX) при SMI автоматично не зберігаються, оскільки операції FPU у режимі SMM навряд чи кому-небудь будуть потрібні. Проте якщо SMI використовується для вимикання процесора, контекст FPU може бути програмно збережений оброблювачем.

Оброблювач SMM може визначити можливості SMM, надані даним процесором, за подвійним словом ідентифікатора SMM (Revision Identifier), збереженим в області контексту за адресою SMBase+FEFCh:

Таблиця 4.2- Стан регістрів при вході в SMM

Регістр	Стан
Загальні регістри	Не визначено
EFLAGS	00000002h
EIP	00008000h
База CS	SMM Base (за замовчуванням 3000h)
Селектор CS	SMM Base, зсунутий на 4 розряди вправо (3000h)
Селектори OS, ES, FS, GS, SS	0000h
Бази DS, ES, FS, GS, SS	000000000h
Ліміти DS, ES, FS, GS, SS	0FFFFFFFFh
CT0	Біти 0, 2, 3 і 31 скинуті (PE, EM, TS і PG); інші не модифіковані
DR6	Не визначено
DR7	00000400h

- біти [15:0] задають номер версії архітектури SMM;
- біт 16 указує на можливість рестарту інструкції введення /виведення;
- біт 17 указує на можливість зміни базової адреси SMRAM і вектора оброблювача SMI.

Для керування рестартом інструкції введення/виведення використовується слово (I/O instruction restart slot) за адресою SMBase+FF00h. Якщо оброблювач запише в це слово значення 0FFh, то після виходу з SMM інструкція, під час якої відбулося переривання SMI, буде рестартована (проте, якщо ця інструкція не була інструкцією введення/виведення, поведінка процесора непередбачена). Якщо оброблювач не змінить нульове значення цього слова, після виходу з SMM процесор перейде до такої інструкції. Рестарт інструкції введення/виведення можливий тільки при вході в SMM за сигналом SMI#. Вхід за повідомленням SMI, що прийшло по шині APIC, не дає можливості рестарту, оскільки це

повідомлення приходить асинхронно стосовно виконання інструкцій процесора.

Значення 3000h базової адреси SMRAM, установлюване за апаратним скиданням (сигналу RESET) процесора, може бути програмно змінене на значення, вирівняне по межі 32 Кбайт, під час виконання оброблювачі SMI. Для цього достатньо змінити значення базової адреси в області збереженого контексту за адресою SMBASE+FEF8h (SMM Base slot), і після виконання інструкції RSM при опрацюванні такого сигналу SMI# буде використовуватися нова область. Природно, необхідно подбати про те, щоб E нової області був присутній код оброблювача. Сигнал INIT процесора Pentium не змінює значення базової адреси SMRAM.

Можливістю рестарту інструкції HALT управляє біт 0 слова за адресою SMBASE+FF02h. Якщо переривання SMI відбулося під час виконання інструкції HALT (зупинка процесора), біт встановлюється в одиничне значення. Якщо оброблювач SMI збереже це значення, після виходу з SMM процесор повторно виконає ту ж інструкцію HALT. Якщо біт скинути, то після виходу з SMM процесор перейде до наступної інструкції.

Для вмикання можливості рестарту інструкції введення/виведення необхідно встановити біт 9 у регістрі TR12 (для процесорів Pentium другого покоління і старше).

Якщо при відновленні контексту за інструкцією RSM процесор виявить, що оброблювач встановив некоректні значення бітів регістрів CR0 або CR4, або невирівнена базова адреса SMRAM, після виходу з режиму SMM процесор перейде в стан Shutdown (аварійна зупинка).

4.5. Мітки реального часу і моніторинг продуктивності

Процесори Pentium і старше мають 64-бітовий лічильник міток реального часу TSC (Time Stamp Counter), що входить до складу MSR. Цей лічильник інкрементується з кожним тактом процесорного ядра; відлік починається з нуля за апаратним скиданням сигналом RESET. Розрядність регістра забезпечує рахунок без переповнювання протягом декількох тисячоріч. Лічильник продовжує рахунок і при виконанні інструкції HLT, і при призупиненні процесора за сигналом STPCLK# (для енергозбереження). Зчитування лічильника забезпечує інструкція RDTSC, установленням біта TSD у CR4 її можна зробити привілейованою (доступною лише при CPL=0). Захищена ОС через оброблювач винятку для задач користувача може емулювати дію інструкції, підставляючи необхідне ОС значення «реального» часу. Читання і запис TSC можливий також за інструкцією звертання до MSR (при CPL=0), причому запис можливий тільки в молодші 32 біти, а старші біти при операції запису

занулюються. Присутність лічильника TSC визначається за інструкцією CPUID (1).

Засоби моніторингу продуктивності (P5+) включають два 40-бітових лічильники, що незалежно програмуються на підрахунок визначених подій або вимір тривалості станів. Враховуватися можуть різноманітні події, пов'язані із шинними операціями, виконанням інструкцій, подіями у внутрішніх вузлах, із роботою конвеєрів, кеша, контролем точок зупинки і т.п. Лічильники програмно керовані - є можливість вказівки що підраховуються подій, читання, запису, запуску і зупинки лічильників, а також дій, виконуваних по їхньому переповнюванні. Крім того, є зовнішні сигнали PM0 і PM1, що програмуються на вказівку факту спрацьовування або переповнювання відповідних лічильників. Оскільки ці сигнали можуть змінювати своє значення з частотою, що не перевищує частоту системної шини, через внутрішнє множення частоти кожна поява сигналів може відбивати декілька (але не більше, ніж коефіцієнт множення) спрацьовувань лічильників. Взаємодія з лічильниками залежить від моделі (лічильники входять до складу MSR), списки що враховуються подій різноманітні для Pentium і P6.

У процесорах Pentium є 3 регістри, доступні як MSR (при CPL=0). Це власне лічильники CTR0 і CTR1 і керуючий регістр CESR (Control And Event Select Register) із такими полями:

- ES0 (біти 5:0) і ES1 (біти 21:16) - вибір подій (event select) для моніторингу (64 варіанти).
- CC0 (біти 8:6) і CC1 (біти 24:22) - керування лічильниками (counter control):
  - 000 - заборона рахунку,
  - 001 - рахунок подій при CPL<3,
  - 010 - рахунок подій при CPL=3,
  - O11 - рахунок подій при будь-якому CPL<3,
  - 100 - заборона рахунку,
  - 101 - рахунок тривалості при CPL<3,
  - 110 - рахунок тривалості при CPL=3,
  - 111 - рахунок тривалості при Будь-якому CPL<3.
- PC0 (біт 9) і PC1 (біт 25) - прапорці керування зовнішніми виводами (Pin Control). При PC<=0 виводи сигналізують про інкремент лічильника, при PC=1 - про переповнювання.

У процесорах P6 із кожним лічильником (PerfCtr і PerfCtr1) пов'язаний власний керуючий регістр: PerfEvtSel і PerfEvtSell відповідно. Всі вони доступні як MSR, але, крім того, для доступу до

лічильників є спеціальна інструкція RDPMC. Установленням в CR4 біта PCE її можна зробити привілейованою.

У регістрах PerfEvtSell визначені такі поля:

- Event select (біти 7:0) - вибір подій.
- Unit mask (біти 15:8) - маска блока, що уточнює тип подій.
- USR (біт 16) - прапорець рахунку при CPL>0 (user mode).
- OS (біт 17) - прапорець рахунку при CPL=0 (operating system mode).
- E (біт 18) - виявлення межі подій (edge detect). Дозволяє визначати середній час перебування у визначеному стані.
- PC (біт 19) - прапорець керування зовнішнім висновком (Pin Control). При PC=1 висновок сигналізує про інкремент лічильника, при PC=0 - про переповнювання
- INT (біт 20) - дозвіл генерації винятку через локальний APIC по переповнюванні лічильника (APIC interrupt enable).
- EN (біт 22, тільки для PerfEvtSel) - прапорець дозволу рахунку (діє на обидва лічильники).
- INV (біт 23) - прапорець інверсії результату порівняння з маскою.
- Counter mask (біти 31:24) - маска лічильника. Якщо за один цикл число подій, що підраховуються, більше (при INV=i - менше) або дорівнює значенню маски, то лічильник інкрементується; інакше - зберігає значення. Використовується для таких подій, як завершення виконання інструкцій (їх може бути декілька за один такт).

Контрольні питання

1. Як виконується апаратне скидання процесора і стан його регістрів після цього?
2. Які особливості скидання процесорів Pentium і старших моделей?
3. Як виконується тестування апаратних засобів процесора?
4. Охарактеризуйте програмні засоби налагодження 32-х розрядних процесорів.
5. Дайте перелік регістрів налагодження та призначення їх розрядів.
6. Охарактеризуйте режим зондового налагодження.
7. Як можна перейти в зондовий режим?
8. Охарактеризуйте режим системного керування SMM.
9. Як виконується перемикання в режим SMM?
10. Що таке мітки реального часу?
11. Які засоби моніторингу продуктивності входять в склад апаратних засобів 32-х розрядних процесорів?

Література

1. Самофалов К.Г., Викторов О.В. Микропроцессоры. - К.: Тэхника, 1989.
2. Мячев А.А., Степанов В.Н. Персональные ЭВМ и микроЭВМ. Основы организации: Справочник/Под ред. А.А.Мячева. - М.: Радио и связь, 1991.
3. Григорьев В.Л. Программирование однокристальных микропроцессоров. - М.: Энергоатомиздат, 1987. - 288 с.
4. Бутурлин А.И. Микропроцессоры Intel 8088/8086, 80286, 80386. - М.: "Крокус-Т", 1992. - 250 с.
5. Фролов А.В., Фролов Г.В. Защищенный режим процессоров Intel 80286, 80386, 80486. Практическое руководство по использованию защищенного режима. - М.: "ДИАЛОГ-МИФИ", 1993. - 240 с.
6. Григорьев В.Л. Микропроцессор i486. Архитектура и программирование(в 4-х книгах). - М.,ГРАНАЛ, 1993. - 382 с.
7. Гук М. Процессоры Pentium 11, Pentium Pro и просто Pentium - СПб: Питер Ком, 1999. - 288 с.
8. Pentium Processor Family. Developer's Manual. Copyright Intel Corporation 1996, 1997.
9. Мюллер С., Зекер К. Модернизация и ремонт ПК: Пер. с англ. - К.: Издательский дом “Вильямс”, 1999. – 992 с.
10. Процессор Pentium Pro. Руководство по микроархитектуре /http://www.intel.ru/contents/procs/ppro/info/p6white/index.htm

Додаток А

Таблиця А.1 - Сигнали системної шини процесорів Pentium Pro і Pentium II

Сигнал	Тип	Призначення
100/66#	I/O	Вибір частоти системної шини: низький рівень – 66 МГц, високий - 100 МГц (у Pentium Pro відсутній)
A[35:3]#	I/O	Address – сигнали шини адреси. Коли сигнал ADS# активний, на шині присутня адреса, коли пасивний – інформація про тип транзакції. Після закінчення дії сигналу RESET# процесор із шини адреси отримує конфігураційну інформацію
A20M#	I	A20 Mask – маскування біта A20 фізичної адреси для емуляції адресного простору 8086 у реальному режимі (його використання в захищеному режимі призведе до непередбачуваних результатів). Під час дії сигналу RESET# використовується для конфігурування множника частоти
ADS#	I/O	Address Strobe - строб адреси, що вводиться ініціатором обміну як індикатор дійсності адреси. За цим сигналом всі агенти шини починають перевірку паритету і протоколу, декодування адреси, внутрішнє спостереження й інші операції пов'язані з новою транзакцією
AERR#	I/O	Address Parity Error – помилка паритету на шині адреси. При відповідному конфігуруванні сигнал може призводити до аварійного припинення транзакції
AP[1:0]#	I/O	Address Parity - біти паритету шини адреси. AP1# ставиться до A[35:24]#, APO# - до A[23:3]#. Сигнал коректного паритету повинен мати низький рівень, якщо низький рівень має непарне число контрольованих сигналом ліній
BCLK	I	Bus Clock – синхронізація шини. Значення всіх синхронних сигналів дійсні за позитивним перепадом BCLK
BERR#	I/O	Bus Error – непоправна помилка шини
BINIT#t	I/O	Bus Initialization – ініціалізація шини. Якщо конфігуруванням використання сигналу дозволено, то він викликає переривання поточної транзакції з втратою даних і скидання у вихідний стан керуючих автоматів всіх агентів шини і їхніх циклічних ідентифікаторів арбітражу
BNR#	I/O	Block Next Request – запит блокування такої транзакції. Вводиться будь-яким агентом шини як запит на припинення, коли він не може сприйняти таку транзакцію
BP[3:2]#	I/O	Breakpoint - сигнали від процесорів, що вказують на попадання в точку зупинки
BPM[1:0]#	I/O	Breakpoint Monitor – сигнали від процесорів, що вказують на влучення в точку зупинки або спрацьовування лічильників, використовуваних для моніторингу продуктивності процесора
BPRI#	I	Bus Priority Request сигнал, використовуваний для арбітражу запитів на володіння шиною
BRO# BR[3:1]#1	I/O I	Bus Request - запит шини. Ці сигнали процесора з'єднуються з лініями шини BREQ[3:0]#
BSEL#	I/O	Bus Select - вибір шини (для майбутніх застосувань). Повинен з'єднуватися із шиною GND
CPUPRES#	O	CPU Present - ознака наявності процесора в сокеті
D[63:0]#	I/O	Data – сигнали 64-розрядної шини даних. Джерело даних при передачі вказує на їхню дійсність сигналом DRDY#
DBSY#	I/O	Data Bus Busy - шина даних зайнята. Використовується агентом, що передає дані, для вказівки на зайнятість шини даних
DEFER*	I	Сигнал, який вказує на те, що вихідний порядок виконання транзакцій не гарантується
DEP[7:0]#	I/O	Data Bus ECC Protection – додаткові сигнали захисту шини даних ECC-кодом
DRDY#	I/O	Data Ready – готовність даних. Встановлюється джерелом даних для вказівки на присутність достовірних даних на шині
FERR#	O	Floating-point Error – помилка FPU (аналогічна сигналу ERROR# співпроцесора Intel387)
FLUSH#	I	Асинхронний сигнал на очищення внутрішнього кеша. За цим сигналом виконуються всі обернені записи й анулюються рядки кеша обох рівнів. Нові

		рядки не виділяються до закінчення дії сигналу. Значення сигналу під час закінчення дії сигналу RESET# використовується для конфігурування процесора
FRCERR	I/O	Functional Redundancy Checking Error – сигнал помилки, виявленої перевірним процесором у системі з надлишковим контролем функціональності master/checker
HIT#, HITM#	I/O	Сигнали результатів операції спостереження за транзакцією. HIT# (Snoop Hit) вказують на кэш-попадання. HITM# (Hit Modified) вказує на попадання в модифікований рядок, забороняючи іншим контролерам шини звертатися до цих даних, до виконання оберненого запису
]ERR#	O	Internal Error - сигнал виявлення внутрішньої помилки. Звичайно з'являється разом із транзакцією *Shutdown.* Сигнал помилки зберігається до його програмного скидання обробловачем NMI або апаратного скидання сигналами RESET#, BINIT# або iNIT#
IGNNE#	I	Ignore Numeric Error – ігнорувати помилки співпроцесора - заборона формування винятку. Використовується для сумісності з AT, де замість винятку формується апаратне переривання. Під час дії сигналу RESET# використовується для конфігурування множника частоти
INIT#	I	Initialization - «м'яка» ініціалізація процесора. Сигнал призводить до скидання загальних регістрів і переходові за вектором, заданим при конфігуруванні по вмиканню. Вміст кеш-пам'яті, буферів запису і регістрів FPU не стосується. Якщо сигнал активний під час закінчення дії сигналу RESET#, процесор виконує BIST
LINT[1:0] (NMI, INTR)	I	Local APIC Interrupt – входи переривань локальних контролерів APIC. Якщо робота APIC заборонена, LINT0 стає сигналом INTR, LINT1 - сигналом NMI. За сигналом RESET# робота APIC дозволяється, і входи працюють у режимі APIC, який може бути скасований програмно. Під час дії сигналу RESET# входи використовуються для конфігурування множника частоти
LOCK#	I/O	Блокування шини на час транзакції
PICCLK	I	APIC Clock - синхронізація шини APIC. У режимі FRC частота PICCLK повинна бути рівною 1/4 BCLK
PICD[1:0]	I/O	APIC Data – двонаправлена послідовна шина обміну повідомленнями APIC
PRDY#	O	Probe Ready - сигнал готовності зонда, використовується апаратними засобами налагодження. Вказує на припинення нормального виконання у відповідь на сигнал R/S# (вхід у зондовий режим)
PREQ#	I	Probe Request - запит зонда на операцію налагодження
PWREN [1:0]	-	Power Enable – виходи, сполучені між собою. Використовуються регулятором напруги для визначення присутності процесора
PWRGOOD	I	Power Good - сигнал справності живлення, який вказує на стабільність напруг живлення і сигналу синхронізації
REQ[4:0]#	I/O	Request Command - запит команди. Вводиться поточним власником шини для визначення типу активної транзакції
RESET#	I	Скидання процесора–конфігурування процесора, ініціалізація регістрів, очищення кеша обох рівнів (без виконання оберненого запису) і перехід до вектора скидання (за замовчуванням 0FFFFFFF0h). Якщо після закінчення дії сигналу активний сигнал INIT#, процесор виконує внутрішній тест BIST (Built-In Self-Test)
RP#	I/O	Request Parity - біт паритету запиту, що захищає лінії ADS# і REQ[4:0]#. Сигнал коректного паритету повинний мати низький рівень, якщо низький рівень має непарнf кількість контрольованих сигналом ліній
RS[2:0]#	I	Response Status - стан відповідача. Сигнали управляються агентом, відповідальним за завершення поточної транзакції
RSP#	I	Response Parity - біт паритету для сигналів RS[2:0]#
SA[2:0]	I	Select Address - ідентифікатор адреси процесора на шині SMBus
SELFSB0	I/O	Вибір частоти системної шини (у Xeon не використовується)
SLOTOCC#	O	Slot Occupied - слот зайнятий. Низький рівень сигналу свідчить про наявність процесора або термінатора в слоті Pentium II. Використовується з комбінацією сигналів VID[4:0] для визначення наявності процесорного ядра в слоті: при VI D [4:01=11111 в слоті – термінатор

SLP#2	I	Sleep – сигнал, що переводить процесор із стану Stop Grant у стан Sleep (режим сну)
SMBALERT#	O	Асинхронна лінія переривання від термодатчика, під'єднаного до шини SMBus
SMBCLK	I	SMBus Clock - синхронізація шини SMBus
SMBDAT	I/O	SMBus Data - дані шини SMBus
SMI#	I	System Management Interrupt - сигнал переривання для входу в режим SMM
STPCLK#	I	Stop Clock – асинхронний сигнал, що переводить процесор у стан Stop Grant із малим живленням
TCK	I	Test Clock – вхід синхронізації шини тестування Test Bus інтерфейсу JTAG
TDI	I	Test Data In - послідовний вхід даних інтерфейсу JTAG
TOO	O	Test Data Out - послідовний вихід даних інтерфейсу JTAG
TESTHI	I	Сигнал, що під'єднується до джерела живлення 2,5 В через резистор 1-100 кОм
TESTLO	I	Сигнал, що підключається до шини GND
THERMDN THERMDP	-	Катод і анод термодіода, що використовується для виміру температури ядра
THERMTRIP#	O	Thermal Trip - сигнал зупинки процесора через перегрів. Якщо внутрішня температура піднімається приблизно до 130°C, процесор зупиняється і формує даний сигнал, що може бути скинутий тільки сигналом RESET# після зниження температури нижче цього порогу
TMS	I	Test Mode State - вибір режиму тестування по шині JTAG
TRDY#	I	Target Ready - сигнал, яким цільовий пристрій указує на готовність прийому даних до запису або неявного оберненого запису
TRST#	I	Test Reset – сигнал скидання логіки TAP (самоскид відбувається автоматично по вмиканню)
VID[4:0][3] VID_CORE[4:0] VID_L2[4:0]	O	Voltage ID – висновки ідентифікації для автоматичного встановлення рівня напруги живлення. Ці виходи можуть бути або вільними, або під'єднуватися до шини GND. Блок живлення повинен встановити відповідну напругу або відімкнутися. VID[4:0] і VIDCORE[4:0] задають живлення ядра, VIDL2[4:0] – живлення вторинного кеша (Xeon)
UP#	O	Upgrade Present - ознака встановлення в сокет 8 процесора OverDrive (не використовується)
WP	I	Write Protect - захист від запису (високим рівнем) Scratch EEPROM

Додаток Б

Таблиця Б.1 - Інструкції пересилки даних

BSWAP r32	Перестановка байтів із порядку молодший-старший (L-H) у порядок старший-молодший (H-L) (486+)
CBW/CWDE	Перетворення байта *AL* у слово *AX* (розширення знака *AL* в *AH: AH* заповнюється бітом *AL.7)* або слова *AX* у подвійне слово *EAX*
CMOVA/CMOVNBE Reg,reg/mem	Пересилка, якщо вище ((CF або ZF)=0) (P6+)
CMOVAE/CMOVNB Reg, reg/mem	Пересилка, якщо не нижче (CF-0) (P6+)
CMOVB/CMOVNAE Reg, reg/mem	Пересилка, якщо нижче (CF=1) (P6+)
CMOVBE/CMOVNA Reg,reg/mem	Пересилка, якщо не вище ((CF або ZF)=1) (P6+)
CMOVC reg, reg/mem	Пересилка, якщо перенесення (CF=l) (P6+)
CMOVE/CMOVZ Reg,reg/mem	Пересилка, якщо дорівнює (ZF=1) (P6+)
CMOVG/CMOVNLE Reg, reg/mem	Пересилка, якщо більше (SF=(OP або ZF)) (P6+)
CMOVGE/CMOVNL Reg, reg/mem	Пересилка, якщо більше або дорівнює *(SF=OF)* (P6+)
CMOVL/CMOVNGE Reg. Reg/mem	Пересилка, якщо менше (ZF!=OF) (P6+)
CMOVLE/CMOVNG Reg. Reg/mem	Пересилка, якщо менше або дорівнює (SF!=OF або ZF=0) (P6+)
CMOVNC reg, reg/mem	Пересилка, якщо немає перенесення (CF=0) (P6+)
CMOVNE/CMOVNZ Reg, Reg/mem	Пересилка, якщо не дорівнює (ZF=0) (P6+)
CMOVNO Reg,reg/mem	Пересилка, якщо немає переповнення (OF=0) (P6+)
CMOVNP/CMOVPO Reg, reg/mem	Пересилка, якщо немає паритету (непарність) (P6+)
CMOVNS reg, reg/mem	Пересилка, якщо результат негативний (SF=0) (P6+)
CMOVO reg,reg/mem	Пересилка, якщо переповнювання (OF=1) (P6+)
CMOVP/CMOVPE Reg, reg/mem	Пересилка, якщо паритет (парність) (P6+)
CMOVS reg, reg/mem	Пересилка, якщо результат негативний (SF=1) (P6+)
CMPXCHG r/m,r	Обмін за результатом порівняння. Порівняння AL *(AX, EAX) із r/m;* якщо дорівнює, то вміст r завантажується в r/m і ZF=1. Інакше вміст r/m завантажується в *AL (AX, EAX), ZF=0* (486+)
CMPXCHG8B m64	Обмін за результатом порівняння. Порівняння *EDX: EAX із* m64; якщо дорівнює, то вміст *ECX:EBX* завантажується в m64 і ZF=1. Інакше вміст m64 завантажується в *EDX:EAX*, ZF=0 (P5+)
CWD/CDQ	Перетворення слова *AX* у подвійне слово *DX:AX* (розширення знака DX заповнюється бітом *AX.* 15) або подвійного слова *EAX* в учетверене *EDX: EAX*
IN AL,i8 ~AX,i8 ~EAX,i8	Введення з порту з адресою i8 в *AL* *~в AX* *~в EAX*
IN AL, DX ~AX, DX	Введення з порту з адресою, що зберігається в *DX*, в *AL* ~в *AX*

~EAX, DX	~в *EAX*
MOV r/m, r ~r, r/m/i ~r,Sreg ~Sreg, r/ml6 ~AL,moffs8 ~AX,moffsl6 ~EAX,moffs32 ~ moffs8, AL ~ moffsl6, AX ~ moffs32,EAX	Пересилка (копіювання) даних із r в r/m ~із r/m/i в r ~із Sreg у r16 ~із r/ml6 у Sreg (крім CS) ~із seg:moffs в *AL* ~із seg:moffs в *AX* ~із seg:moffs у *EAX* ~із *AL* у seg:moffs ~із *AX* у seg:moffs ~із *EAX* у seg:moffs
MOVSX r16, r/m8 ~ r32, r/m8 ~ r32, r/ml6	Копіювання байта зі знаковим розширенням до слова (386+) Копіювання байта зі знаковим розширенням до подвійного слова (386+) Копіювання слова зі знаковим розширенням до подвійного слова (386+)
MOVZX r16, r/m8 ~r32,r/m8 ~r32, r/m16	Копіювання байта з нульовим розширенням до слова (386+) Копіювання байта з нульовим розширенням до подвійного слова (386+) Копіювання слова з нульовим розширенням до подвійного слова (386+)
OUT i8, AL ~i8, AX ~i8, EAX	Виведення у порт з адресою i8 із *AL* ~із *AX* ~із EAX
OUT DX, AL ~DX, AX ~DX, EAX	Виведення у порт з адресою, що зберігається в DX, із *AL* ~ із DX ~ із EAX
POP r/ml6 ~ r/m32 ~Seg	Зчитування слова даних із стека в регістр або пам'ять, *(E)SP* інкрементується ~ у сегментний регістр (крім *CS)*
POPA (POP All)	Зчитування даних із стека в регістри *DI, SI, BP, BX, DX, CX, AX* (286+)
POPAD	Зчитування даних із стека в регістри *EDI, ESI, EBP, EBX, EDX, ECX, EAX* (386+)
PUSH r/ml6 ~ r/m32 ~ il6 ~ i32	Запис слова з регістра або пам'яті в стек після декремента *(E)SP* ~ подвійного слова (386+) ~ безпосереднього операнда-слова (286+) ~ безпосереднього операнда-подвійного слова (386+)
PUSHA (PUSH All)	Запис в стек регістрів *AX, CX, DX, BX, SP* (вихідне значення), *BP, SI, DI* (286+)
PUSHAD	Запис в стек регістрів *EAX, ECX, EDX, EBX, ESP* (вихідне значення), *EBP, ESI, EDI* (386+)
XCHG r/m,r	Обмін даними (взаємний) між регістрами або регістром і пам'яттю

Таблиця Б.2 - Інструкції двійкової арифметики

ADC r/m, r/m/i	Додавання двох операндів із врахуванням перенесення від попередньої операції. Якщо до слова додається восьмирозрядний операнд i8, то він знаком розширюється до слова: r/m=(r/m/i + r/m + *CF)*
ADD r/m, r/m/i	Додавання двох операндів. Якщо до слова додається i8, то останній знаком розширюється до слова: r/m=(r/m + r/m/i)
CMP r/m/, r/m/i	Порівняння (віднімання без збереження результату, установка прапорців)
DEC r/m	Декремент (віднімання 1, але не діє на прапорець *CF):* r/m=(r/m - 1)
DIV *r/m*	Діленння беззнакове: *AX* на r/m8, частка в *AL*, остача в *AH; DX:AX* на r/m16, частка в *AX,* остача в *DX*; *EDX:EAX* на r/m32, частка в *EAX,*

	остача у *EDX;* ділення на 0 або переповнювання (частка не поміщається в регістр) викликає виняток #DE
IDIV r/m8 ~ r/ml6 ~ r/m32	Ділення знакове: *AX* (в *AH* знакове розширення *AL)* на r/m8, частка в *AL*, остача в *AH; DX:AX* (у *DX* знакове розширення *AX)* на r/ml6, частка в *AX*, остача у *DX; EDX:EAX* (у *EDX* знакове розширення *EAX)* на r/m32, частка в *EAX*, остача у EDX; ділення на 0 або переповнювання (частка не поміщається в регістр) викликає виняток *#DE*
IMUL r/m8 ~ r/ml6 ~ r/m32 ~ rl6,r/ml6 ~ r32,r/m32 ~ rl6,r/ml6,i8 ~ r32,r/m32,i8 ~ rl6,i8 ~ r32, i8 ~ rl6, r/ml6,i16 ~ r32, r/m32, i32 ~ rl6, il6 ~ r32, i32	Множення знакове: AX=Al x r/m8 DX:AX=AXx r/rm16 EDX:EAX=EAX x r/m32 (386+) rl6= r16 x r/m16 (286+) r32= r32 x r/m32 (386+) rl6=r/ml6 x Фi8 (286+) r32=r/m32 xi 8 (386+) rl6=rl6 x i8 (286+) r32=r32 x i8 (386+) rl6=r/ml6 x il6 (286+) r32=r/m32 x i32 (386+) rl6=rl6 x il6 (286+) r32=r32 x i32 (386+) Якщо один із співмножників має розрядність меншу, ніж інший, він розширюється знаком. Якщо добуток не поміщається у форматі співмножників (його старша половина не 00/FF, 0000/FFFF, 00000000/FFFFFFFF), то *CF=PF=1,* інакше *CF=PF=0*
INC r/m	Інкремент (додавання 1, але не діє на прапорець CF): r/m=r/m+l
MUL r/m8 ~ r/ml6 ~ r/m32	Беззнакове множення *AL* на r/m8: AX=Al x r/m8 ~ *AX* на r/ml6: DX;AX=AX x r/ml6 ~ *EAX* на r/m32: *EDX:EAX=EAX x r/m32*
NEG r/m	Зміна знака операнда: r/m=0r/m
SBB r/m, r/m/i	Віднімання з позикою: r/m=((r/m - r/m/i) CF). При відніманні від r/ml6 або r/m32 операнд i8 розширюється знаком до 16 або 32 бітів
SUB r/m,r/m/i	Віднімання: r/m=(r/m – r/m/i). При відніманні від r/ml6 або r/m32 операнд i8 розширюється знаком до 16 або 32 бітів
XADD r/m, r	Обмін вмістом і додавання, завантаження суми в r/m (486+)

Таблиця Б.3 - Інструкції десяткової арифметики

AAA	Десяткова корекція *AL* після додавання двох неупакованих чисел інструкцією ADD. У випадку переповнювання інкрементується регістр *AH* і встановлюються прапорці *CF* і *AF*, інакше AF=CF=0
AAD[1]	Десяткова корекція AX перед діленням неупакованого двозначного числа *(AH, AL)* інструкцією DIV для того, щоб результат виявився неупакованим числом. AL=AL+10 x AH, AH=0
AAM[1]	Десяткова корекція *AX* після множення двох неупакованих чисел інструкцією *MUL*
AAS	Десяткова корекція *AL* після віднімання двох неупакованих чисел інструкцією *SUB.* У випадку позики декрементується регістр *AH* і встановлюються прапорці CF і *AF*, інакше AF=CF=0
DAA	Десяткова корекція *AL* після додавання двох упакованих BCD-чисел інструкцією *ADD,* у випадку десяткового перенесення AF=CF=1
DAS	Десяткова корекція AL після віднімання двох упакованих BCD-чисел інструкцією *SUB,* у випадку десяткової позики AF=CF=1

[1] Інструкції AAD і AAM припускають узагальнений формат виклику, при якому корекція виконується за будь-яким модулем (а не тільки за модулем 10).

Таблиця Б.4 - Інструкції логічних операцій

AND r/m, r/m/i	Логічне І (маскування, обнулення тих біт у *r/m,* які нульові в r/m/i)
NOT r/m	Інверсія (переключення всіх біт)
OR r/m, r/m/i	Логічне АБО (установка тих біт в *r/m,* що рівні одиниці в r/m/i)
XOR r/m, r/m/i	Виняткове АБО (переключення тих біт у r/m, що рівні одиниці в r/m/i)

Таблиця Б.5 - Інструкції зсувів

RCL r/m ~r/m,CL ~r/m,i8	Циклічний зсув вліво через біт перенесення *CF* на одну позицію, ~ на *CL* позицій, ~ на i8-позицій (286+)
RCR*p/m* ~r/m. CL ~r/m,i8	Циклічний зсув вправо через біт перенесення CF на одну позицію, ~ на *CL* позицій, ~ на i8-позицій (286+)
ROLp/m -r/m,CL ~r/m,i8	Циклічний зсув вліво на одну позицію, ~ на *CL* позицій, ~ на i8-позицій (286+)
RORr/m ~r/m,CL ~r/m,i8	Циклічний зсув вправо на одну позицію, ~на *CL* позицій, ~на i8-позицій (286+)
SAL r/m ~r/m,CL ~r/m,i8	Зсув арифметичний вліво на одну позицію, ~ на CL позицій, ~ на i8-позицій (286+)
SARr/m ~r/m,CL ~r/m,i8	Зсув арифметичний (із збереженням старшого біта) вправо на одну позицію, ~ на *CL* позицій, ~ на i8-позицій (286+)
SHL r/m ~r/m,CL ~r/m,i8	Зсув вліво на одну позицію, ~ на CL позицій, ~ на i8-позицій (286+)
SHR r/m ~r/m,CL ~r/m,i8	Зсув вправо на одну позицію, ~на *CL* позицій, ~на i8-позицій (286+)
SHLD r/ml6,rl6,i8 ~r/m32,rl6,i8 ~r/ml6,rl6,CL ~r/m32,r32,CL	Зсув вліво r/m16 (r/m32) на i8 біт, вставка даних із r16 (r32) у позиції що звільнилися (386+) те ж, але лічильник зсувів у CL
SHRD r/ml6,rl6,i8 ~r/m32,rl6,i8 ~r/m16,rl6,CL ~r/m32,r32,CL	Зсув вправо r/m16 (r/m32) на i8 біт, вставка даних із r16 (r32) у позиції , що звільнилися (386+) те ж, але лічильник зсувів у CL

Таблиця Б.6 - Інструкції обробки бітів і байтів

BSF reg. Reg/mem	Сканування біт вперед. В reg завантажується індекс (беззнакове зміщення щодо біта 0) наймолодшого одиничного біта reg/mem. Якщо в reg/mem тільки нулі, встановлюється ZF
BSR reg. Reg/mem	Сканування біт назад. В reg завантажується індекс (беззнакове зміщення щодо біта 0) найстаршого одиничного біта reg/mem. Якщо в reg/mem тільки нулі, встановлюється ZF
BT reg/mem,i8 ~reg/mem,reg	Тестування біта – завантаження в *CF* біта з номером i8 або reg (за модулем 16 або 32) із reg/mem

BTC reg/mem,i8 ~ reg/mem,reg	Тестування і зміна значення біта
BTR reg/mem,i8 ~ reg/mem,reg	Тестування і скидання біта
BTS reg/mem,i8 ~ reg/mem,reg	Тестування і встановлення біта
SALC	Встановлення AL=FFh, якщо CF=1, інакше AL=00h (не документовано, код D6h)
SETA/SETNBE r/m8	Встановлення байта в 0lh, якщо вище ((CF АБО ZF)=0), інакше в 00h (P6+)
SETAE /SETNB/SETNC r/m8	Встановлення байта в 0lh, якщо не нижче (CF=0), інакше в 00h (P6+)
SETB/SETNAE/SETC r/m8	Встановлення байта в 0lh, якщо нижче (CF=1), інакше в OOh (P6+)
SETBE/SETNA r/m8	Встановлення байта в 0lh, якщо не вище (CF АБО ZF)=1, інакше в 00h (P6+)
SETE/SETZ r/m8	Встановлення байта в 0lh, якщо дорівнює (ZF=1), інакше в 00h (P6+)
SETG/SETNLE r/m8	Встановлення байта в 0lh, якщо більше (SF=(OF I ZF)), інакше в 00h (P6+)
SETGE/SETNL r/m8	Встановлення байта в 0lh, якщо більше або дорівнює (SF=OF), інакше в 00h (P6+)
SETL/SETNGE r/m8	Встановлення байта в 01h, якщо менше (ZF!=OF), інакше в 00h (P6+)
SETLE/SETNG r/m8	Встановлення байта в 0lh, якщо менше або дорівнює (SF!=OF або ZF=0), інакше в 00h (P6+)
SETNE/SETNZ r/m8	Встановлення байта в 01h, якщо не дорівнює (ZF=0), інакше в 00h (P6+)
SETNO r/m8	Встановлення байта в 0lh, якщо немає переповнення (OF=0), інакше в 00h (P6+)
SETNS r/m8	Встановлення байта в 0lh, якщо невід'ємне (SF=0), інакше в 00h (P6+)
SETO r/m8	Встановлення байта в 0lh, якщо переповнення (OF=1), інакше в 00h (P6+)
SETPE/SETP r/m8	Встановлення байта в 0lh, якщо паритет (парність), інакше в 00h (P6+)
SETPO/SETNY r/m8	Встановлення байта, в 01h, якщо немає паритету (парності), інакше в 00h (P6+)
SETS r/m8 r/m8	Встановлення байта в 0lh, якщо негативне (SF=1), інакше в 00h (P6+)
SETC r/m8	Встановлення байта в 0lh, якщо перенесення (CF=1), інакше в 00h (P6+)
SETNC r/m8	Встановлення байта в 0lh, якщо немає перенесення (CF=0), інакше в 00h (P6+)
TEST r/m,i ~r/m,r	Перевірка біт (логічне I без запису результату, встановлення прапорців)

Таблиця Б.7 - Інструкції передачі керування

JMP ~rel8 ~rel16/32 ~r/m16 ~r/m32 ~ptrl6:16 ~ptrl6:32 ~m16:16 ~ml6:32	Безумовний перехід: Короткий за зсувом відносно наступної інструкції Ближній за зсувом відносно наступної інструкції Ближній за адресою в r/m16 Ближній за адресою в r/m32 Дальній за адресою-операндом Дальній за адресою-операндом Дальній за адресою в m16:16 Дальній за адресою в ml6:32
Виклики процедур і переривань	
BOUND reg,mem	Перевірка на попадання індексу (reg) у межі масиву, задані двома суміжними словами в пам'яті: якщо (reg<DS:[mem]) або (reg<DS:[(mem+2)/4]), виконується *INT 5* (286+)

CALL target	Виклик процедури (8 форм): ближній (за абсолютною непрямою або відносною адресою) або дальній за абсолютною (прямою або непрямою) адресою
INT 3	Виконання програмного переривання 3 - пастки налагодження (однобайтовий код команди)
INT i8	Виконання програмного переривання за номером i8
INTO	Виконання програмного переривання 4, якщо OF=1
IRET IRETD	Повернення з переривання при 16-бітових операндах; те ж при 32-бітових операндах (різні мнемоніки для одного коду)
RET ~i16	Повернення з процедури (ближнє або дальнє) ~із корекцією значення покажчика стеку після повернення - *(E)SP=(E)SP+i16*
Цикли (лічильник циклів: *CX* при 16-бітовій адресації, *ECX* - при 32-бітовій)	
LOOP rel8	*(E)CX=(E)CX-*\ і перехід, якщо *(E)CX=0*
LOOPE/LOOPZ rel8	*(E)CX=(E)CX-1* і перехід, якщо *(E)CX =0* і ZF=1
LOOPNE/LOOPNZ rel8	*(E)CX=(E)CX-i* і перехід, якщо *(E)CX =0* і ZF=0
Умовні переходи (rel16/32 тільки для 386+)	
JC rel8 ~rel16/32	Перехід, якщо перенесення (CF=1)
JE/JZ rel8 ~rell6/32	Перехід, якщо дорівнює (ZF=1)
JNC rel8 ~rel16/32	Перехід, якщо немає перенесення (CF=0)
JNE/JNZ rel8 ~rel16/32	Перехід, якщо не дорівнює (ZF=0)
JNP/JPO rel8 ~rel16/32	Перехід, якщо непарний паритет (PF=0: число одиничних біт непарне)
JP/JPE rel8 ~rel16/32	Перехід, якщо непарний паритет *(PF=1:* число одиничних біт парне)
JCXZ rel8 ~rel16/32	Перехід, якщо CX=0
JECXZ rel8 ~rel16/32	Перехід, якщо *ECX=0* (386+)
Умовні переходи беззнакові (rel16/32 тільки для 386+)	
JA/JNBE rel8 ~rel16/32	Перехід, якщо вище ((CF АБО ZF)=0)
JAE/JNB rel8 ~rel16/32	Перехід, якщо не нижче (CF=0)
JB/JNAE rel8 ~rel16/32	Перехід, якщо нижче (CF=1)
JBE/JNA rel8 ~rel16/32	Перехід, якщо не вище (CF АБО ZF)=1
Умовні переходи знакові (rel16/32 тільки для 386+)	
JG/JNGE rel8 ~rel1б/32	Перехід, якщо більше (SF=(OF і ZF))
JGE/JNL rel8 ~rel16/32	Перехід, якщо більше або дорівнює (SF=OF)
JL/JNGE rel8 ~rel16/32	Перехід, якщо менше (ZF/=OF)
JLE/JNG rel8 ~rel16/32	Перехід, якщо менше або дорівнює (SF!=OF або ZF=0)
JNO rel8 ~rel16/32	Перехід, якщо немає переповнення (OF=0)

JNS rel8 ~rel16/32	Перехід, якщо невід'ємне (SF=0)
JO rel8 ~rell6/32	Перехід, якщо переповнення (OF=1)
JS rel8 ~rel16/32	Перехід, якщо негативно (SF=1)

Таблиця Б.8 - Інструкції операцій з рядками

CMPSB	Порівняння рядків байтів, що адресуються *DS:(E)SI* і *ES:(E)DI із* записом результату порівняння в регістр прапорців
CMPSD	Порівняння рядків подвійних слів, що адресуються *DS:(E)SI* і *ES:(E)DI* із записом результату порівняння в регістр прапорців (386+)
CMPSW	Порівняння рядків слів, що адресуються *DS:(E)SI* і *ES:(E)DI із* записом результату порівняння в регістр прапорців
INSB	Запис байта, введеного з порту, адресованого регістром *DX,* в *ES:(E)DI* (286+)
INSD	Запис подвійного слова, введеного з порту, адресованого регістром DX, у *ES:(E)DI* (386+)
INSW	Запис слова, введеного з порту, адресованого регістром DX, у *ES:(E)DI* (286+)
LODSB	Копіювання байта з *DS:(E)SI* в *AL*
LODSD	Копіювання подвійного слова з *DS:(E)SI* в *EAX* (386+)
LODSW	Копіювання слова з *DS:(E)SI* у ЛХ
MOVSB	Копіювання байта з *DS:(E)SI* у *ES:(E)DI*
MOVSD	Копіювання подвійного слова з *DS:(E)SI* у *ES:(E)DI* (386+)
MOVSW	Копіювання слова з *DS:(E)SI* у *ES:(E)DI*
OUTSB	Виведення байта, зчитаного з *DS:(E)SI* у порт, адресований регістром DX (286+)
OUTSD	Виведення подвійного слова, зчитаного з *DS:(E)Sl* у порт, адресований регістром DX (386+)
OUTSW	Виведення слова, зчитаного з *DS:(E)SI* у порт, адресований регістром DX (286+)
SCASB	Сканування рядка байтів – порівняння *AL із* байтом із рядка *DS:(E)SI* та записом результату порівняння в регістр прапорців
SCASD	Сканування рядка подвійних слів - порівняння *EAX* зі словом із рядка *DS:(E)SI та* записом результату порівняння в регістр прапорців
SCASW	Сканування рядка слів – порівняння *AX* зі словом із рядка *DS:(E)SI та* записом результату порівняння в регістр прапорців
STOSB	Запис байта з *AL* у *ES:(E)DI*
STOSD	Запис подвійного слова з EAX у *ES:(E)DI* (386+)
STOSW	Запис слова з *AX* у *ES:(E)DI*
REP	Префікс повтору рядкових операцій до обнулення *(E)CX, (E)CX* декрементується на кожному повторі
REPE/REPZ	Префікс умовного повтору рядкових операцій виконання *REP* при ZF=1
REPNE/REPNZ	Префікс умовного повтору рядкових операцій виконання *REP* при ZF=0

Таблиця Б.9 - Інструкції роботи з прапорцями

CLC	Скидання прапорця перенесення (CF=0)
CLD	Скидання прапорця напрямку (DF=0 – інкремент *(E)SI, (E)DI)*
CLI	Заборона маскованих апаратних переривань (IF=0)
CMC	Інверсія прапорця перенесення (CF=1-CF)
LAHF	Завантаження прапорців (SF:ZF:0:AF:0:PF:1:CF) у регістр *AH*
POPF (POP Flags)	Вилучення даних із стека в регістр прапорців *(EFLAGS[15:0])*
POPFD	Вилучення даних із стека в розширений регістр прапорців *EFLAGS*
PUSHF (PUSH Flags)	Запис в стек регістра прапорців *(EFLAGS[15:0])*
PUSHFD	Запис в стек розширеного регістра прапорців *EFLAGS*
SAHF	Завантаження прапорців *SF, ZF. AF, PF, CF* з біт 7, 6, 4, 2, 0 регістра *AH*
STC	Встановлення прапорця перенесення (CF=l)
STD	Встановлення прапорця напрямків (DF=1 - декремент *(E)Sl, (E)DI)*
STI	Дозвіл маскованих апаратних перериванні (IF=1)

Таблиця Б.10 - Інструкції завантаження покажчиків

LDS r16,m16:16 ~r32,m16:32	Завантаження дальнього покажчика з пам'яті в DS:rl6 ~ у DS:r32
LES r16,m16:16 ~r32,m16:32	Завантаження дальнього покажчика з пам'яті в ES:rl6 ~ у ES:r32
LFS r16,m16:16 ~r32,m16:32	Завантаження дальнього покажчика з пам'яті в FS:rl6 ~у FS:r32
LGS r16,ml6:l6 - ~r32,m16:32	Завантаження дальнього покажчика з пам'яті в GS:rl6 ~ у GS:r32
LSS rl6,ml6:16 ~r32,ml6:32	Завантаження дальнього покажчика з пам'яті в SS:rl6 ~ у SS:r32

Таблиця Б.11 - Різні інструкції

CPUID	Одержання інформації про процесор в *EAX, EBX, ECX, EBX.* Параметр виклику (номер функції) задається в *EAX* (P5+)
LEA r1б,m ~r32,m	Завантаження ефективної адреси m у регістр r16/32
LAT/XLATB	Трансляція (перекодування) вмісту *AL* в значення з таблиці трансляції, що адресується в *(E)BX: AL=ES:[(E)BX+(AL)]*
NOP	Немає операції
UD2	Невизначені 2-байтові інструкції (викликають виключення *#UD)*
ENTER i16,i8	Виділення блоку параметрів в стекові
LEAVE	Звільнення блоку параметрів в стекові (дії, обернені *ENTER* – відновляються *(E)SP* і *(E)BP)*

Таблиця Б.12 - Інструкції MMX

EMMS	Очищення стека регістрів – установлення всіх бітів в одиницю в слові тегів
Пересилка даних	
MOVD mm, r/m32	Пересилка даних у молодші 32 біти регістра MMX із заповненням старших бітів нулями
MOVD r/m32, mm	Пересилка даних із молодших 32 біт регістра MMX
MOVQ mm/m64,mm	Пересилка даних (64 біти) у регістр MMX
MOVQ mm/m64, mm	Пересилка даних (64 біти) із регістра MMX
Перетворення форматів	
PACKSSDW mm, mm/m64	Упаковування із знаковим насиченням двох подвійних слів, розташованих у mm, і двох подвійних слів mm/m64 в чотири слова, розташованих у mm
PACKSSWB mm, mm/m64	Упаковування зі знаковим насиченням чотирьох слів, розташованих у mm, і чотирьох слів mm/m64 у вісім байтів, розташованих у mm
PACKUSWB mm, mm/m64	Упаковування з насиченням чотирьох знакових слів, розташованих у mm, і чотирьох слів mm/m64 у вісім беззнакових байтів, розташованих у mm
PUNPCKHBW mm, mm/m64	Чергування в регістрі призначення байтів старшої половини операнда-джерела з байтами старшої половини операнда призначення
PUNPCKHWD mm, mm/m64	Чергування в регістрі призначення слів старшої половини операнда-джерела зі словами старшої половини операнда призначення
PUNPCKHDQ mm, mm/m64	Чергування в регістрі призначення подвійного слова старшої половини операнда-джерела з подвійним словом старшої половини операнда призначення
PUNPCKLBW mm, mm/m64	Чергування в регістрі призначення байтів молодшої половини операнда-джерела з байтами молодшої половини операнда призначення
PUNPCKLWD mm, mm/m64	Чергування в регістрі призначення слів молодшої половини операнда-джерела зі словами молодшої половини операнда призначення
PUNPCKLDQ mm, mm/m64	Чергування в регістрі призначення подвійного слова молодшої половини операнда-джерела з подвійним словом молодшої половини операнда призначення

Упакована арифметика

PADDB mm, mm/m64 PADDW mm, mm/m64 PADDD mm, mm/m64	Додавання упакованих байтів (слів або подвійних слів) без насичення (із циклічним переповненням)
PADDSB mm, mm/m64 PADDSW mm, mm/m64	Додавання знакових упакованих байтів (слів) із насиченням
PADDUSB mm,mm/m64 PADDUSW mm, mm/m64	Додавання упакованих беззнакових байтів (слів) із насиченням
PMADDWD mm, mm/m64	Множення чотирьох знакових слів операнда-джерела на чотири знакових слова операнда призначення. Два подвійних слова результатів множення молодших слів підсумовуються і записуються в молодше подвійне слово операнда призначення. Два подвійних слова результатів множення старших слів підсумовуються і записуються в старше подвійне слово операнда призначення
PMULHW mm, mm/m64	Множення упакованих знакових слів із зберіганням тільки старших 16 бітів елементів результату
PMULLW mm, mm/m64	Множення упакованих знакових або беззнакових слів із зберіганням тільки молодших 16 бітів елементів результату
PSUBB mm, mm/m64 PSUBW mm, mm/m64 PSUBD mm, mm/m64	Віднімання упакованих байтів (слів або подвійних слів) без насичення (із циклічним антипереповненням)
PSUBSB mm,mm/m64 PSUBSW mm, mm/m64	Віднімання упакованих знакових байтів (слів) із насиченням
PSUBUSB mm, mm/m64 PSUBUSW mm, mm/m64	Віднімання упакованих беззнакових байтів (слів) із насиченням
Логіка	
PAND mm, mm/m64	Логічне І
PANDN mm, mm/m64	Логічне І mm/m64 і інверсного значення mm
POR mm, mm/m64	Логічне АБО
PXOR mm, mm/m64	Виключне АБО
Порівняння	
PCMPEQB mm, mm/m64 PCMPEQD mm, mm/m64 PCMPEQW mm, mm/m64	Порівняння (на рівність) упакованих байтів (слів, подвійних слів). Всі біти елемента результату будуть одиничними (True) при збігу відповідних елементів (байт, слів або подвійних слів) операндів і нульовими (False) при розбіжності
PCMPGTB mm, mm/m64 PCMPGTD mm, mm/m64 PCMPGTW mm, mm/m64	Порівняння (за розміром) упакованих знакових байтів (слів, подвійних слів). Всі біти елемента результату будуть одиничними (True), якщо відповідний елемент операнда призначення (mm) більше елемента операнда-джерела (mm/m64), і нульовими (False) у протилежному випадку
Зсуви і обертання	
PSLLD mm, mm/m64 ~ mm, i8 PSLLQ mm, mm/m64 ~mm, i8 PSLLW mm, mm/m64 ~mm, i8	Логічний зсув вліво упакованих (подвійних) слів операнда призначення на кількість бітів, зазначених в операнді-джерелі, із заповненням молодших бітів нулями
PSRAD mm, mm/m64 ~mm, i8 PSRAW mm, mm/m64 ~mm, i8	Арифметичний зсув вправо упакованих подвійних знакових слів операнда призначення на кількість бітів, зазначених в операнді-джерелі, із заповненням молодших бітів бітами знакових розрядів
PSRLD mm, mm/m64 ~mm, i8 PSRLQ mm, mm/m64 ~mm, i8 PSRLW mm, mm/m64 ~mm, 18	Логічний зсув вправо упакованих (подвійних,) слів операнда призначення на кількість бітів, зазначених в операнд-джерелі, із заповненням старших бітів нулями

Таблиця Б.13 - Інструкції 3D Now!

FEMMS	Швидкий вхід/вихід у MMX або FPU
PREFETCH mem8	Попередня вибірка рядка даних в первинний кеш
PAVGUSB mm, mm/m64	Знаходження середнього для упакованих 8-бітових беззнакових цілих
PI2FD mm, mm/m64	Перетворення упакованих 32-бітних цілих у формат із плаваючою точкою
PF2ID mm, mm/m64	Перетворення упакованих чисел у форматі з плаваючою точкою в 32-бітові цілі
PMULHRW mm, mm/m64	Множення 16-бітових упакованих цілих з округленням
Операції з упакованими числами у форматі з плаваючою точкою	
PFADD mm, mm/m64	Додавання
PFSUB mm, mm/m64	Віднімання
PFSUBR mm, mm/m64	Зворотне віднімання
PFACC mm, mm/m64	Накопичення
PFMUL mm, mm/m64	Множення
PFCMPGE mm, mm/m64	Порівняння (більше або дорівнює)
PFCMPGT mm, mm/m64	Порівняння (більше)
PFCMPEQ mm, mm/m64	Порівняння (дорівнює)
PFMIN mm, mm/m64	Знаходження мінімуму
PFMAX mm, mm/m64	Знаходження максимуму
PFRCP mm, mm/m64	Наближене обчислення оберненої величини (Reciprocal Approximation)
PFRCPIT1 mm, mm/m64	Перша ітерація при обчисленні оберненої величини (Reciprocal First Iteration Step)
PFRCPIT2 mm, mm/m64	Друга ітерація при обчисленні оберненої величини (Reciprocal Second Iteration Step)
PFRSQRT mm, mm/m64	Наближене обчислення оберненої величини кореня квадратного (Reciprocal Square Root Approximation)
PFRSQIT1 mm, mm/m64	Перша ітерація при обчисленні оберненої величини кореня квадратного (Reciprocal Square Root First Iteration Step)

Таблиця Б.14 - Інструкції FPU

Пересилки даних	
FBLD m80bcd	Перетворення і завантаження (push) упакованого BCD числа з пам'яті в стек
FBSTP m80bcd	Витяг із стека (pop) і запис у пам'ять в упакованому BCD-форматі (10 байт, 18 цифр)
FCMOVB st0,st(i)	Пересилка, якщо нижче (CF=1) (P6+)
FCMOVBE st0,st(i)	Пересилка, якщо не вище (CF АБО ZF)=1 (P6+)
FCMOVE st0,st(i)	Пересилка, якщо дорівнює (ZF=1) (P6+)
FCMOVNB st0,st(i)	Пересилка, якщо не нижче (CF=0) (P6+)
FCMOVNBE st0,st(i)	Пересилка, якщо вище ((CF АБО *ZF)=0)* (P6+)
FCMOVNE st0,st(i)	Пересилка, якщо не дорівнює (ZF=0) (P6+)
FCMOVNU st0,st(i)	Пересилка, якщо не NaN (PF=0) (P6+)
FCMOVU st0,st(i)	Пересилка, якщо NaN (unordered) (PF=0) (P6+)
FILD ml6int ~m32int ~m64int	Завантаження (push) цілого числа з пам'яті в st(0)
FISTml6int ~m32int	Запис st(0) у пам'ять у форматі цілого числа
FISTP ml6int ~m32int ~m64int	Запис st(0) у пам'ять у форматі цілого числа з вилученням (pop)
FLD st(i) ~m32rea1 ~ m64real ~m80real	Завантаження (push) дійсного числа в st(0) із st(i) або з пам'яті

FST m32real ~m64real ~st(i)	Зберігання (копіювання) числа в пам'яті (у дійсному форматі) або регістрі стека
FSTP m32real ~m64real ~m80real ~st(i)	Запис числа в пам'ять (у дійсному форматі) або регістр стека з вилученням (pop): m32/64/80real = st(0); pop st(i) = st (0); pop
FXCH ~st(i)	Обмін значеннями вершини стека і регістра: st(0) і st(l) st(0) і st(i)
Завантаження констант	
FLD1	Завантаження (push) +1.0 у st(0)
FLDL2E	Завантаження (push) log2(e) у st(0)
FLDL2T	Завантаження (push) log2(l0) у st(0)
FLDLG2	Завантаження (push) lg(2) у st(0)
FLDLN2	Завантаження (push) ln(2) у st(0)
FLDPI	Завантаження (push) л у st(0)
FLDZ	Завантаження (push) +0. 0 у st(0)
Базова арифметика	
FABS	Знаходження абсолютного значення: st(0) = ABS(st(O))
FADD m32real ~m64real ~st(0),st(i) ~st(i),st(0)	Додавання дійсних чисел: st(0)=st(0)+m32real st(0) = st(0) + m64real st(0) = st(0) + st(i) st(i) = st(i) + st(0)
FADDP ~st(i),st(0)	Додавання дійсних чисел із вилученням (pop): st(l) = st(l) + st(0); pop st(i) = st(i) + st(0); pop
FCHS	Зміна знака: st(0) = -st(0)
FDIV st(0),st(i) ~st(0,st(0) ~m32real ~m64real	Ділення дійсних чисел: st(0) = st(0)/st(i) st(i) = st(i)/st(0) st(0) = st(0)/m32real st(0) = st(0)/m64real
FDIVP ~st(i),st(0)	Ділення дійсних чисел із вилученням: st(1) = st(1)/st(0);pop st(i)=st(i)/st(0); pop
FDIVR m32real ~m64real ~st(0),st(i) ~st(i),st(0)	Обернене ділення дійсних чисел: st(0) = m32real/st(0) st(0) = m64real/st(0) st(0) = st(i)/st(0) st (i) = st(0)/st(i)
FDIVRP -st(i),st(0)	Обернене ділення дійсних чисел із вилученням: st(1) = st(0)/st(l); pop st(i) = st(0)/st(i); pop
FIADD m32int ~m64int	Додавання з цілим числом: st(0) = st(0) + m32int st(0) = st(0) + m64int
FIDIV m32int ~ml6int	Ділення на ціле число: st(0) = st(0)/m32int st(0) = st(0)/ml6int
PIDIVR m32int ~ml6int	Обернене ділення цілих чисел: st(0) = m32int/st(0) st(0) = ml6int/st(0)
FIMUL m32int ~ml6int	Множення на ціле число: st(0) = st(0) x m32int st(0) =st(0) x ml6int
FISUB m32int ~ml6int	Віднімання цілого числа: st(0) = st(0) - m32int st(0) = st(0) – ml6int
FISUBR m32int ~ml6int	Віднімання від цілого числа: st(0) = m32int - st(0) st(0) = ml6int – st(0)
FMUL st(0),st(i) ~st(i),st(0)	Множення дійсних чисел: st(0) = st(0) x st(i) st(i) = st(0) x st (i)

~m32real ~m64real	st(0) = st(0) x m32real st(0)=st(0) x m64real
FMULP ~st(i),st(0)	Множення дійсних чисел із вилученням: st(l) = st(0) x st(l); pop st(i)=st(0) x st(i); pop
FPREM	Знаходження часткової остачі: st(0) = st(0) MOD st(l). Якщо остача більша дільника (модуля), то встановлюється прапорець C2 і для обчислення "дійсної" остачі потрібно повторне виконання команди (до обнулення C2)
FPREM1	Знаходження часткової остачі в стандарті IEEE (остача не повинна перевищувати половини модуля) (387+)
FRNDINT	Округлення st(0) до найближчого цілого в залежності від прапорця RC: st(0) = INT(st(0))
FSCALE	Масштабування – множення на округлену в сторону нуля степінь числа 2: st(0) =st (0)x2$^{st(1)}$
FSQRT	Добування кореня квадратного : st(0) = ^ st(0)
FSUBst(0),st(i) ~st(i),st(0) ~m32real ~m64real	Віднімання дійсного числа: st(0) = st(0) - st(i) st(i) = st(i) – st(0) st(0) = st(0) - m32real st(0) = st(0) - m64real
FSUBP ~st(i),st(0)	Віднімання дійсних чисел із вилученням: st(l) = st(l) - st(0); pop st(i) = st(i) - st(0); pop
FSUBR m32real ~m64real ~st(0),st(i) ~st(i),st(0)	Обернене віднімання числа: st(0) = m32real – st(0) st(0) = m64real - st(0) st(0) =st(i) - st(0) st(i) = st(0) - st(i)
FSUBRP ~st(i),st(0)	Обернене віднімання з вилученням: st(l) = st(0) - st(l); pop st(i) = st(0) - st (i); pop
FXTRACT	Виділення мантиси і порядку числа з st(0) і декремент TOP, після виконання в st(0) - мантиса, а в st(l) – порядок
Порівняння даних	
FCOM ~st(i) ~m32real ~m64real	Порівняння дійсних чисел: встановлення прапорців, як при st(0) - st(l) ~st(0) - st(i) ~st(0) - m32real ~st(0) - m64real
FCOMI st(i)	Порівняння st(0) із st(i) і відповідне встановлення прапорців у *EFLAGS (ZF, PF, CF)* (P6+)
FCOMIP st(i)	Порівняння st(0) із st(i) і відповідне встановлення прапорців у *EFLAGS (ZF, PF, CF), із* вилученням (P6+)
FCOMP ~st(i) ~m32real ~m64real	Порівняння дійсних чисел із вилученням: встановлення прапорців, як при st(0) - st(1); pop ~st(0) - st(i); pop ~st(0) – m32real; pop ~st(0) - m64real; pop
FCOMPP	Порівняння дійсних чисел st(0) і st(l); подвійне вилучення
FICOM m l6int ~m32int	Порівняння st(0) із цілочисельним операндом із пам'яті: встановлення прапорців, як при st(0) - ml6int або st(0) - m32int
FICOMP ml6int ~m32int	Порівняння st(0) із цілочисельним операндом із пам'яті з вилученням
FTST	Перевірка на нуль – порівняння st(0) із 0.0
FUCOM ~st(i)	Порівняння без генерації винятку у випадку NaN (387+): st(0) і st(l) st(0) і st(i)
FUCOMI st(i)	Порівняння st(0) із st(i) без генерації винятку у випадку NaN і відповідне встановлення прапорців у *EFLAGS (. ZF, PF, CF)* (P6+)
FUCOMIP st(i)	Порівняння st(0) із st(i) без генерації винятку у випадку NaN і відповідне

	встановлення прапорів у *EFLAGS (ZF, PF, CF) із* вилученням (P6+)
FUCOMP ~st(i)	Порівняння без генерації винятку у випадку NaN із вилученням (387+): st(0) із st(1); pop st(0) із st(i); pop
FUCOMPP st(i)	Порівняння без генерації винятку у випадку NaN із подвійним вилученням (387+): st(0) із st(l); pop; pop
FXAM	Аналіз st(0) - встановлення коду умови в C0, C2, C3
Трансцендентні функції	
F2XM1	Обчислення 2^x- 1: st(0) = $2^{st(o)}$ – 1
FCOS	Косинус: st(0) = COS (st(O)) (387+)
FPATAN	Арктангенс частки з вилученням: st(1) = ATAN(st(1)/st(0)); pop
FPTAN	Обчислення тангенса (аргумент у радіанах, діапазон -2^{63}… $+2^{63}$) і завантаження (push) у стек +1.0: st(0) = TAN(st(0)); push +1.0. Якщо аргумент поза діапазоном, регістри залишаються незмінними і встановлюється прапорець C2
FSIN	Обчислення синуса (аргумент у радіанах, діапазон -2^{63}…$+2^{63}$ *):* st(0)=SIN(st(0)). Якщо аргумент поза діапазоном, регістр залишається незмінним і встановлюється прапорець *C2* (387+)
FSINCOS	Обчислення синуса і косинуса st(0) із розміщенням (push) у стек. В результаті в st(l) буде SIN, а в st(0) – COS (387+)
FYL2X	Обчислення Y x $\log_2$ (X): st(l) = st(1) x $\log_2$st (0); pop
FYL2XP1	Обчислення Y x $\log_2$(X+l): st(1) = st(l) xl $\log_2$ (st(0)+1.0); pop
Керування співпроцесором	
FCLEX	Скидання прапорців винятків із попередньою перевіркою немаскованих винятків, що очікують
FDECSTP	Декремент покажчика стека FPU
FFREE st(i)	Звільнення регістра , позначка st(i), як вільного
FINCSTP	Інкремент покажчика стека FPU
FINIT	Ініціалізація FPU із попередньою перевіркою винятків, що очікують
FLDCW m2byte	Завантаження керуючого слова (FPU *CW)* із пам'яті
FLDENV ml4/28byte	Завантаження стану співпроцесора із образа в пам'яті, збереженого інструкціями *FSTENV/ FNSTENV*
FNCLEX	Скидання прапорців винятків без перевірки тих, що очікують
FNINIT	Ініціалізація FPU без перевірки винятків, що очікують
FNOP	Порожня операція FPU: st(0) = st(0)
FNSAVE	Збереження стану співпроцесора і стека регістрів у пам'яті без перевірки винятків, що очікують
FNSTCW	Збереження керуючого слова без перевірки винятків, що очікують
FNSTENV ml4/28byte	Збереження стану співпроцесора *(SR, CR, TAGW, FIP* і *FDP)* у пам'яті без перевірки винятків, що очікують
FNSTSW AX ~ml6	Запис слова стану в *AX* або в слово пам'яті без перевірки винятків, що очікують
FRSTOR m94/108byte	Завантаження стану співпроцесора і регістрів із пам'яті
FSAVE m94/108byte	Збереження стану співпроцесора і стека регістрів у пам'яті з попередньою перевіркою винятків, що очікують
FSTCW ml6	Збереження керуючого слова з попередньою перевіркою винятків, що очікують
FSTENV ml4/28byte	Збереження стану сопроцесора *(SR, CR, TAGW, FIP* і *FDP)* у пам'яті з попередньою перевіркою винятків, що очікують
FSTSW AX ~ml6	Запис слова стану в *AX* або слово пам'яті для наступного перенесення коду завершення в регістр прапорців із попередньою перевіркою винятків, що очікують
WAIT/FWAIT	Синхронізація, зупинка CPU до завершення поточної операції FPU, перевірка винятків FPU, що очікують

Таблиця Б. 15 - Системні інструкції

ARPL r/ml6, rl6	Вирівнювання *RPL:* якщо в першому операнді поле *RPL* (біти 1, 0) менше, ніж у другому, воно встановлюється з другого і ZF=1; інакше ZF=0
CTS/CLTS	Скидання прапорця перемикання задач у регістрі CRO (TF=0) (286+)
HLT	Зупинка процесора (безупинне виконання *NOP* до апаратного переривання)
INVD	Анулювання даних у внутрішньому кеші і ініціалізація анулювання в зовнішньому кеші без зворотного запису (486+)
INVLPG m	Анулювання елемента таблиці трансляції TLB, що містить адрес m. У ряді випадків анулюється вся таблиця (486+)
LAR rl6,r/ml6 ~r32,r/m32	Завантаження в rl6 байта прав доступу з дискриптора, заданого r/ml6 (завантажується дискриптор із *GDT* або *LDT* із маскою FFOOh), ~заданого r/m32 (із маскою 00FxFF00h)
LGDT ml6&32	Завантаження *GDTR* із пам'яті (6 байт). Якщо поточна розрядність операндів 16 біт, старший байт із пам'яті ігнорується і у *GDTR* заповнюється нулями
LIDT ml6&32	Завантаження *IDTR* із пам'яті (6 байт). Якщо поточна розрядність операндів 16 біт, старший байт із пам'яті ігнорується і *IDTR* заповнюється нулями
LLDT r/ml6	Завантаження *LDTR* (16-бітового селектора) із регістра або пам'яті
LMSW r/ml6	Завантаження *MSW* (частина регістра CR0)
LOCK	Префікс блокування (захоплення) шини на час виконання наступної команди
LSL rl6, r/ml6 ~r32,r/m32	Завантаження rl6 (r32) лімітом сегмента, дискриптор якого заданий r/ml6 (r/m32). Ліміт виражається в байтах, при G=1 (сторінкова розподілюваність) проводиться перерахунок (зсув вліво на 12 розрядів). При завантаженні в rl6 старші біти ліміту відсікаються
LTR r/ml6	Завантаження регістра задачі з r/ml6
MOV CRn, r32	Завантаження керуючого регістра n (n=0, 2, 3, 4)
MOV r32,CRn	Читання керуючого регістра n (n=0, 2, 3, 4)
MOV DRn,r32	Завантаження регістра налагодження n
MOV r32,DRn	Читання регістра налагодження n
MOV TRn,r32	Завантаження регістра тестування n (для 386 і 486)
MOV r32,TRn	Читання регістра тестування n (для 386 і 486)
RDMSR	Читання модельно-специфічного регістра *MSRn, що* адресується *ECX,* у *EDX: EAX* (P5+)
RDPMC	Читання лічильника монітора продуктивності, що адресується *ECX,* у *EDX: EAX* (P5+)
RDTSC	Читання лічильника тактів у *EDX:EAX* (P5+)
RSM	Повернення з режиму SMM
SGDT m48	Зберігання *GDTR* у пам'яті (6 байт)
SIDT m48	Зберігання *IDTR* у пам'яті (6 байт)
SLDT r/ml6 ~r/m32	Зберігання *LDTR* (16-бітового селектора) у регістрі або пам'яті ~ у молодших 16 бітах r/m32
SMSW r/ml6 ~r32	Зберігання *MSW* в регістрі або пам'яті (16 біт) ~ у молодших 16 бітах r/m32
STR r/ml6	Зберігання селектора із TR у r/ml6
VERR r/ml6	Перевірка можливості читання із сегмента, на який посилається r/ml6: установка ZF=1, якщо задачі дозволено читання із сегмента
VERW *r/ml6*	Перевірка можливості запису в сегмент, на який посилається r/ml6:: установка *ZF=1,* якщо задачі дозволений запис у сегмент
WBINVD	Зворотний запис модифікованих рядків, анулювання внутрішньої кеш-пам'яті, ініціалізація анулювання в зовнішньому кеші (486+)
WRMSR	Запис у модельно-специфічний регістр *MSRn,* заданий в *ECX,* із *EDX: EAX* (P5+)