4.2 МАШИННО-ЗАЛЕЖНІ КОМПОНЕНТИ ОС

Одна і та ж операційна система не може без яких-небудь змін встановлюватися на комп'ютерах, що відрізняються типом процесора або/і способом організації усієї апаратури. У модулях ядра ОС не можуть не відбитися такі особливості апаратної платформи, як кількість типів переривань і формат таблиці посилань на процедури обробки переривань, склад регістрів загального призначення і системних регістрів, стан яких треба зберігати в контексті процесу, особливості підключення зовнішніх пристроїв і багато інших.

Проте досвід розробки операційних систем показує: ядро можна спроектувати таким чином, що тільки частина модулів будуть машинно- залежними, а інші не залежатимуть від особливостей апаратної платформи. У добре структурованому ядрі машинно-залежні модулі локалізовані і утворюють програмний шар, що примикає до шару апаратури. Така локалізація машинно- залежних модулів істотно спрощує перенесення операційної системи на іншу апаратну платформу.

Об'єм машинно-залежних компонентів ОС залежить від того, наскільки великі відмінності в апаратних платформах, для яких розробляється ОС. Наприклад, ОС, побудована на 32-бітових адресах, для перенесення на машину з 16-бітовими адресами має бути практично переписана наново. Проте одна з найочевидніших відмінностей – розбіжність системи команд процесорів – долається досить просто. Операційна система програмується мовою високого рівня, а потім відповідним компілятором виробляється код для конкретного типу процесора.

Проте в багатьох випадках відмінності в організації апаратури комп'ютера лежать набагато глибше і здолати їх таким чином не вдається. Наприклад, однопроцесорний і двопроцесорний комп'ютери вимагають застосування в ОС абсолютно різних алгоритмів розподілу процесорного часу. Аналогічна відсутність апаратної підтримки віртуальної пам'яті призводить до принципової відмінності в реалізації підсистеми управління пам'яттю. У таких випадках не обійтися без внесення в код операційної системи специфіки апаратної платформи, для якої ця ОС належить.

Для зменшення кількості машинно-залежних модулів виробники операційних систем обмежують універсальність машинно-незалежних модулів. Це означає, що їх

незалежність носить умовний характер і поширюється тільки на декілька типів процесорів і створених на основі цих процесорів апаратних платформ. По цьому шляху пішли, наприклад, розробники ОС Windows NT, обмеживши кількість типів процесорів для своєї системи чотирма і поставляючи різні варіанти кодів ядра для однопроцесорних і багатопроцесорних комп'ютерів. Особливе місце серед модулів ядра займають низькорівневі драйвери зовнішніх пристроїв. З одного боку ці драйвери, як і високорівневі драйвери, входять до складу менеджера введення-виведення, тобто належать шару ядра, що займає досить високе місце в ієрархії шарів. З іншого боку, низькорівневі драйвери відбивають усі особливості керованих зовнішніх пристроїв, тому їх можна віднести і до шару машинно-залежних модулів. Така двоїстість низькорівневих драйверів ще раз підтверджує схемну моделі ядра із строгою ієрархією шарів.

Для комп'ютерів на основі процесорів Intel x86/Pentium розробка екрануючого машинно-залежного шару ОС дещо спрощується за рахунок вбудованої в постійну пам'ять комп'ютера базової системи введення-виведення – BIOS. BIOS містить драйвери для усіх пристроїв, що входять в базову конфігурацію комп'ютера: жорстких і гнучких дисків, клавіатури, дисплея тощо. Ці драйвери виконують дуже примітивні операції з керованими пристроями, наприклад читання групи секторів даних з певної доріжки диска, але за рахунок цих операцій екрануються відмінності апаратних платформ персональних комп'ютерів і серверів на процесорах Intel різних виробників.

Розробники операційної системи можуть користуватися шаром драйверів BIOS як частиною машинно-залежного шару ОС.

В ідеалі шар машинно-залежних компонентів ядра повністю екранує іншу частину ОС від конкретних деталей апаратної платформи (кеші, контролери переривань введення-виведення тощо), принаймні для того набору платформ, який підтримує ця ОС. У результаті відбувається підміна реальної апаратури деякою уніфікованою віртуальною машиною, однаковою для усіх варіантів апаратної платформи. Усі шари операційної системи, які лежать вище шару машинно-залежних компонентів, можуть бути написані для управління саме цією віртуальною апаратурою. Таким чином, у розробників з'являється можливість створювати один

варіант машинно-незалежної частини ОС (включаючи компоненти ядра, утиліти, системні оброблювальні програми) для всього набору підтримуваних платформ.

Основні елементи комп'ютера

На макрорівні комп'ютер складається з процесора, пам'яті і пристроїв введення-виведення. При цьому кожен компонент представлений одним або декількома модулями. Щоб комп'ютер міг виконувати своє основне призначення, що полягає у виконанні програм, різні компоненти повинні мати можливість взаємодіяти між собою. Можна виділити чотири структурні компоненти комп'ютера:

1. Процесор. Здійснює контроль за діями комп'ютера, а також виконує функцію обробки даних.

2. Основна пам'ять. Тут зберігаються дані і програми.

3. Пристрої введення-виведення. Служать для передачі даних між комп'ютером і зовнішнім оточенням, що складається з різних периферійних пристроїв.

4. Системна шина. Певні структури і механізми, що забезпечують взаємодію між процесором, основною пам'яттю і пристроями введення- виведення.

Процесори

"Керівним центром" комп'ютера є центральний процесор (CPU – Central Processing Unit). Звичайний цикл роботи центрального процесора виглядає так: він читає першу команду з пам'яті, декодує її для визначення типу і операндів команди, виконує команду, потім прочитує, декодує і виконує подальші команди. Таким чином здійснюється виконання програм.

Регістри процесора

Для кожного процесора існує свій набір команд, які він в змозі виконати. Оскільки доступ до пам'яті для отримання команд або набору даних займає набагато більше часу, ніж виконання цих команд, усі процесори містять внутрішні регістри для зберігання ключових змінних і тимчасових результатів. Тому набір інструкцій для будь-якого процесора містить команди для завантаження слова з пам'яті в регістр і збереження слова з регістра в пам'ять. Інші команди об'єднують два операнди з регістрів, пам'яті або того і іншого і отримують результат. Регістри процесора виконують дві функції:

1. Регістри управління і регістри стану. Використовуються в процесорі для контролю над виконуваними операціями. За їх допомогою привілейовані програми ОС можуть контролювати хід виконання інших програм.

2. Регістри, доступні користувачеві. Ці регістри дозволяють програмістові скоротити число звернень до основної пам'яті, оптимізуючи використання регістрів за допомогою мови асемблера або мови високого рівня.

Регістри управління і регістри стану. Однією з функцій процесора є обмін даними з пам'яттю. Для цього використовуються два внутрішні регістри процесора: регістр адреси пам'яті (memory address register – MAR), куди заноситься адреса елементу пам'яті, в якій робитиметься операція читання- запису, і регістр буфера пам'яті (memory buffer register – MBR), куди заносяться дані, призначені для запису в пам'ять. Аналогічно номер облаштування введення-виведення задається в регістрі адреси введення- виведення (I/O address register – I/O AR). Регістр буфера введення-виведення (I/O buffer register – I/O BR).

Окрім цих регістрів є ще декілька **спеціальних регістрів.** Один з них називається **лічильником команд (program counter – PC)**, у ньому міститься адреса наступної команди, що стоїть у черзі на виконання. Після того, як команда вибрана з пам'яті, регістр команд коригується і покажчик переходить до наступної команди. Регістр команд (instruction register – IR) містить останню вибрану в пам'яті команду.

Наступний регістр (чи набір регістрів) називається **PSW (Processor Status Word – слово стану процесора**). Цей регістр містить коди умов і біти коду станів, які задаються командами порівняння, пріоритетом процесора, режимом (режим користувача або режим ядра), та іншу службову інформацію.

Коди умов (відомі також як прапори) – це послідовність бітів, що встановлюються або скидаються процесором залежно від результату виконання операцій. Наприклад, у результаті виконання арифметичної дії може вийти від'ємне число, нуль, або переповнювання. У результаті арифметичних операцій встановлюються також відповідні коди умов. Потім вони можуть бути перевірені умовною операцією галуження. Призначені для користувача програми за допомогою спеціальних команд можуть читати увесь регістр PSW цілком, але писати можуть

тільки в деякі з його полів. Регістр PSW відіграє важливу роль в системних викликах і операціях уведення-виведення.

Регістри, доступні користувачеві. До цих регістрів користувач може звертатися за допомогою команд машинної мови. Серед доступних регістрів є регістри даних, адресні регістри і регістри коду умови.

Регістри даних можна застосовувати в різних цілях. Проте при цьому накладаються певні обмеження. Наприклад, деякі регістри призначені для операцій над числами з плаваючою точкою, тоді як інші – для зберігання цілих чисел.

Адресні регістри призначені для занесення адрес команд і даних в основній пам'яті. У цих регістрах може бути записана тільки частина адреси, що використовується при обчисленні повної адреси.

Ще один регістр процесора називається **покажчиком стека (SP, stack pointer).** Він містить адресу вершини стека в пам'яті. Стек містить по одному фрейму (області даних) для кожної процедури, яка вже почала виконуватися, але ще не закінчилася. У стековому фреймі процедури зберігаються її вхідні параметри, а також локальні і тимчасові змінні, що не зберігаються в регістрах. У деяких машинах виклик процедури або підпрограми призводить до автоматичного збереження вмісту усіх доступних користувачеві регістрів, щоб після повернення їх можна було відновити.

Регістри процесора Pentium. Оскільки в основному в ПК використовуються процесори типу Pentium, то розглянемо їх регістри детальніше. У процесорах Pentium ці регістри діляться на декілька груп:

- регістри загального призначення;
- регістри сегментів;
- покажчик інструкцій;
- регістр прапорів;
- управляючі регістри;
- регістри системних адрес;
- регістри відладки і тестування;
- регістри математичного співпроцесора, що виконує операції з плаваючою точкою.

У процесорі Pentium є вісім 32-розрядних регістрів загального призначення. Чотири з них, які можна умовно назвати A, B, C і D, використовуються для тимчасового зберігання операндів арифметичних, логічних і інших команд. Програміст може звертатися до цих регістрів як до єдиного цілого, використовуючи позначення EAX, EBX, ECX, EDX, а також до деяких їх частин, як це показано на рис. 4.1. Тут позначення AL(L – Low) стосується першого, наймолодшого байта регістра EAX, AH (H – High) – до наступного за старшинством байта, а AX означає обидва молодші байти регістра. Приставка E в позначенні цих регістрів утворена від слова extended (розширений), що вказує на те, що в колишніх моделях процесорів Intel ці регістри були 16-розрядними, а потім їх розрядність була збільшена до 32 біт.

Інші чотири регістри загального призначення – ESI, EDI, EBP і ESP – призначені для задання зміщення адреси відносно початку деякого сегменту даних. Ці регістри використовуються спільно з регістрами сегментів в системі адресації процесора Pentium для задання віртуальної адреси, яка потім за допомогою таблиць сторінок відображається на фізичну адресу.

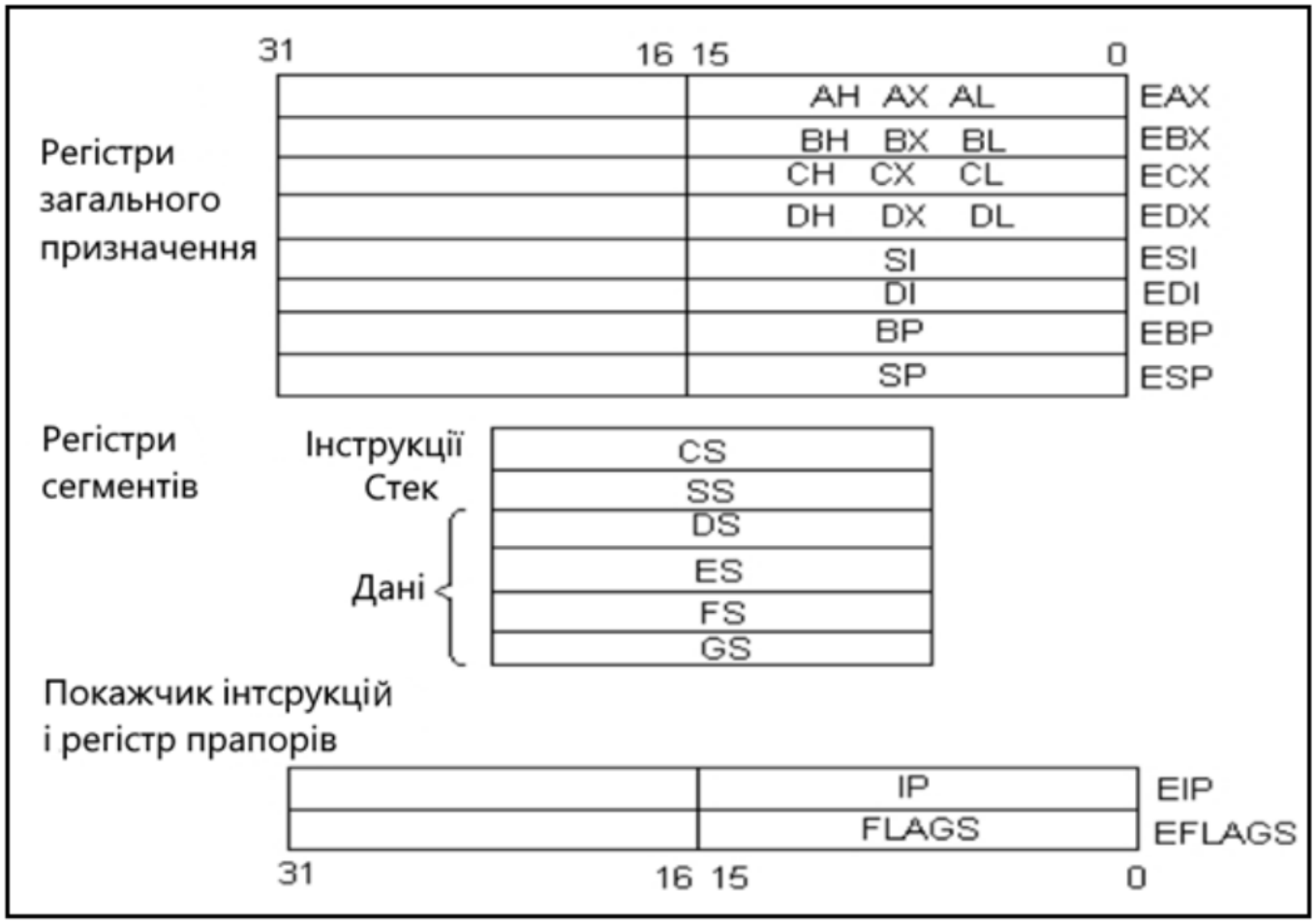

Рисунок 4.1 – Основні регістри процесора Pentium

Регістри сегментів CS, SS, DS, ES, FS і GS в захищеному режимі посилаються на дескриптори сегментів пам'яті – описувачі, в яких містяться такі параметри сегментів, як базова адреса, розмір сегменту, атрибути захисту і деякі інші. Регістри сегментів зберігають 16-розрядне число, що називається селектором, в якому 12 старших розрядів є індексом в таблиці дескрипторів сегментів, 1 розряд вказує, в якій з двох таблиць, GDT або LDT, знаходиться дескриптор, а три розряди поля RPL зберігають значення рівня привілеїв запиту до цього сегменту.

Регістр CS (Code Segment) призначений для зберігання індексу дескриптора кодового сегменту, регістр SS (Stack Segment) – дескриптора сегменту стека, а інші регістри використовуються для вказівки на дескриптори сегментів даних. Усі регістри сегментів, окрім CS, програмно доступні, тобто в них можна завантажити нове значення селектора відповідною командою (наприклад, LDS). Значення регістра CS змінюється при виконанні команд міжсегментних викликів CALL і переходів JMP, а також при перемиканні задач. У цьому розділі термін «задача» часто вживатиметься замість рівнозначного (і поширенішого) терміну «процес» у зв'язку з тим, що саме цей термін вибрали свого часу розробники процесорів Intel x86, і він фігурує в назвах регістрів і структур даних.

Покажчик інструкцій EIP містить зміщення адреси поточної інструкції, яке використовується спільно з регістром CS для отримання відповідної віртуальної адреси.

Регістр прапорів EFLAGS містить ознаки, що характеризують результат виконання операції, наприклад, прапор знаку, прапор нуля, прапор переповнювання, прапор паритету, прапор перенесення і деякі інші. Крім того, тут зберігаються деякі ознаки, що встановлюються і аналізуютсься механізмом переривань, зокрема прапор дозволу апаратних переривань IF.

У процесорі Pentium є п'ять керуючих регістрів – CRO, CR1, CR2, CR3 і CR4, які зберігають ознаки і дані, що характеризують загальні стани процесора (рис. 4.2).

Регістр CR0 містить усі основні ознаки, що істотно впливають на роботу процесора, такі як реальний/захищений режим роботи, включення/виключення сторінкового механізму системи віртуальної пам'яті, а також ознаки, що впливають

на роботу кеша і виконання команд з плаваючою точкою. Молодші два байти регістра CRO мають назву Mashine State Word, MSW – «слово стану машини». Ця назва використовувалася в процесорі 80286 для позначення керуючого регістра, який мав аналогічне призначення.

Рисунок 4.2 – Керуючі і системні регістри процесора Pentium

Регістр CR1 нині не використовується (зарезервований).

Регістри CR2 і CR3 призначені для підтримки роботи системи віртуальної пам'яті. Регістр CR2 містить лінійну віртуальну адресу, яка викликала так звану сторінкову відмову (відсутність сторінки в оперативній пам'яті або відмова із-за порушення прав доступу). Регістр CR3 містить фізичну адресу таблиці розділів, яка використовується сторінковим механізмом процесора.

У регістрі CR4 зберігаються ознаки, що дозволяють роботу архітектурних розширень, наприклад, можливості використання сторінок розміром 4 Мб тощо. Регістри системних адрес містять адреси важливих системних таблиць і структур, використовуваних при управлінні процесами і пам'яттю. Регістр GDTR (Global

Descriptor Table Register) містить фізичну 32-розрядну адресу глобальної таблиці дескрипторів GDT сегментів пам'яті, що утворюють загальну частину віртуального адресного простору усіх процесів.

Регістр IDTR (Interrupt Descriptor Table Register) зберігає фізичну 32- розрядну адресу таблиці дескрипторів переривань IDT, використовувану для виклику процедур обробки переривань у захищеному режимі роботи процесора. Окрім цих адрес у регістрах GDTR і IDTR зберігаються 16-бітові ліміти, що задають обмеження на розмір відповідних таблиць.

Два 16-бітові регістри зберігають не фізичні адреси системних структур, а значення індексів дескрипторів цих структур в таблиці GDT, що дозволяє побічно отримати відповідні фізичні адреси. Регістр TR (Task Register) містить індекс дескриптора сегменту стану задачі TSS. Регістр LDTR (Local Descriptor Table Register) містить індекс дескриптора сегменту локальної таблиці дескрипторів LDT сегментів пам'яті, що утворюють індивідуальну частину віртуального адресного простору процесу. Регістри відладки зберігають значення точок останову, а регістри тестування дозволяють перевірити коректність роботи внутрішніх блоків процесора.

Виконання команд

З метою покращення характеристик процесорів їх розробники давно відмовилися від простої моделі, в якій за один такт може бути зчитана, декодована і виконана тільки одна команда. Багато сучасних процесорів мають можливості виконання декількох команд одночасно. Наприклад, у процесора можуть бути роздільні модулі, що займаються вибіркою, декодуванням і виконанням команд. Під час виконання команди з номером n він може декодувати команду з номером n+1 і прочитувати команду з номером n+2. Подібна організація процесу називається конвеєром (рис. 4.3, а) [9].

Часто зустрічаються і довші конвеєри. У більшості конвеєрних конструкцій зчитана команда має бути виконана, навіть якщо в попередній команді був прийнятий умовний перехід. Ця властивість конвеєрів є проблемою для розробників компіляторів і операційних систем.

Прогресивнішим у порівнянні з конвеєрною конструкцією є скалярний процесор (см. рис. 4.3, б). У цій структурі є присутньою певна кількість виконавчих

вузлів: один для цілочисельних арифметичних операцій, другий – для операцій з плаваючою точкою і ще один – для логічних операцій. За один такт прочитується дві або більше команди, які декодуються і скидаються в буфер зберігання, де вони чекають своєї черги на виконання. Коли виконуючий пристрій звільняється, він «заглядає» в буфер зберігання, і якщо там є команда, яку воно може обробити, то забирає її і виконує. У результаті команди часто виконується не в порядку їх слідування. Проте при цьому підході дуже неприємні ускладнення торкнулися і ОС.

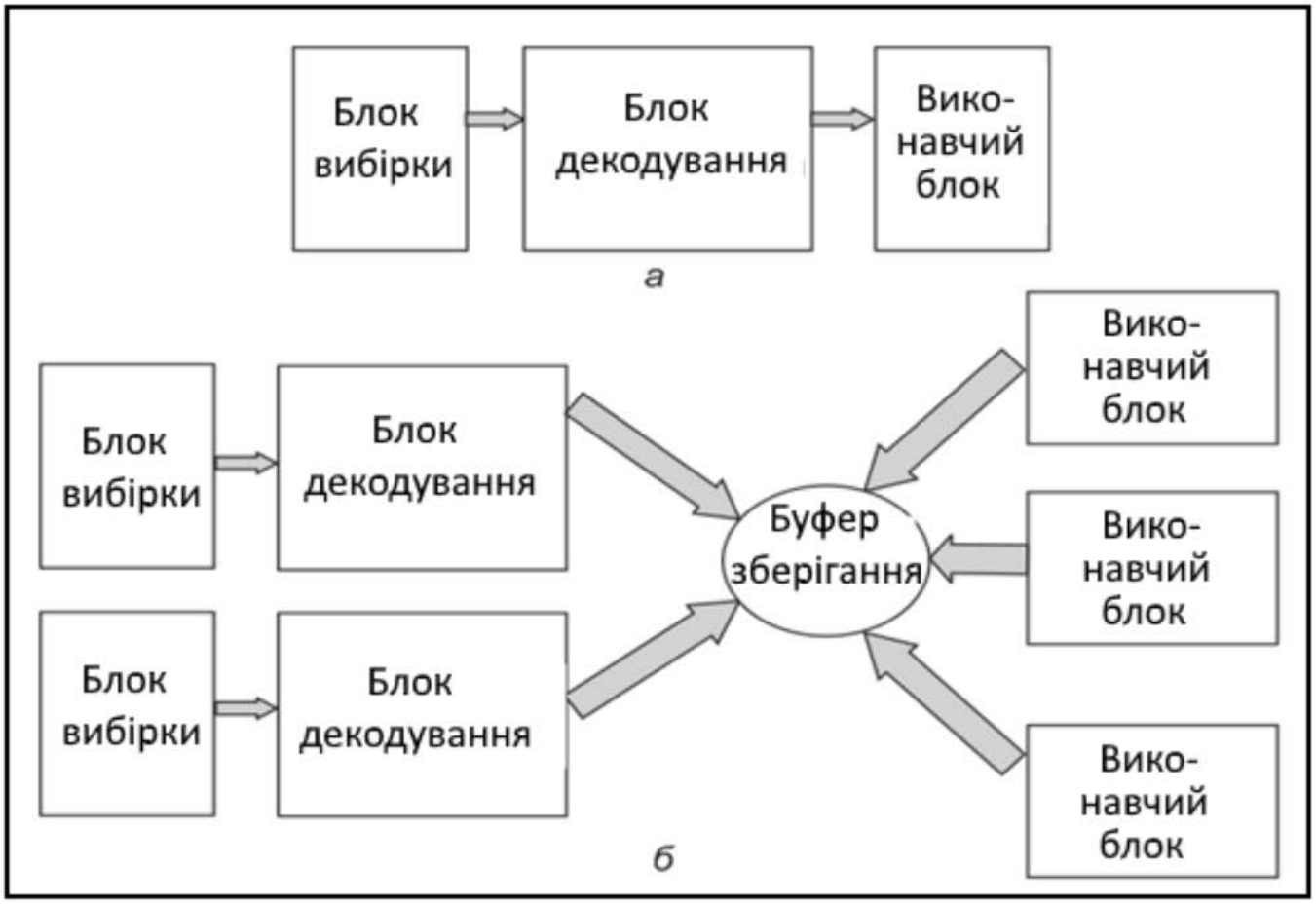

Рисунок 4.3 – Процесор: а – з конвеєром; б – суперскалярний

Якщо зупинитися на найскладнішому елементі комп'ютера, що визначає технічний рівень усього виробу, – центральному процесорі – то можна помітити, що прогрес тут йде двома паралельними шляхами: розвиток елементної бази і вдосконалення архітектури.

Розвиток елементної бази. Технологія виготовлення мікропроцесорів розвивається в напрямі подальшої мініатюризації електронних схем і, як наслідок, підвищення ступеня їх інтеграції. Зменшення розмірів дозволяє «упакувати» на

одному чипі більше число елементів і ускладнити мікросхему. Ще в 1968 році, на зорі мікроелектроніки, один із засновників фірми Intel Гордон Мур сформулював емпіричний «закон Мура», за яким число елементів на одному кристалі повинне подвоюватися кожні півтора роки. Дивно, але факт – пройшло більше 48 років, невпізнанно змінилися технології, проте закон продовжує діяти і зараз.

У повній відповідності із законом Мура, сучасні мікропроцесори є неймовірно складними пристроями. Наприклад, кристал P5 фірми Intel, випущений у 1993 році, що отримав торгову марку «Pentium», містить близько 3 млн транзисторів, P6 – «Pentium Pro» (1996 р.) – 7,5 млн, а процесор P7 (під час розробки він називався «Merced», а в продажу пішов під ім'ям «Itanium»), випуск якого почався в 2000 році, мав близько 20 млн транзисторів.

Зменшення розмірів деталей і довжин провідників, що з'єднують їх, дозволяє удосконалити ще одну характеристику мікропроцесора, що пропорційно впливає на його продуктивність – тактову частоту. Якщо у чіпа i4004 вона дорівнювала 108 кГц, то сучасні схеми допускають збільшення тактової частоти до 2000 – 3000 Мгц.

Удосконалення архітектури. На жаль, нескінченно зменшувати розміри елементарних схем перемикачів неможливо, оскільки вони обмежені знизу розмірами кристалічних решіток. Так само не можна безмежно підвищувати тактову частоту, оскільки швидкість поширення електричного струму кінцева.

Мабуть, найближчим часом елементні можливості мікросхем підійдуть до теоретичної межі, подальше підвищення продуктивності комп'ютерів досягатиметься тільки за рахунок удосконалення архітектури, яке розвивається в чотирьох основних напрямах.

1. Збільшення розрядності. Тенденція до підвищення розрядності виразно простежується в історії мікропроцесорів. Сучасні кристали в основному 32-розрядні, проте просунуті мікросхеми, наприклад, PowerPC, а також перспективні масові моделі, наприклад, «Merced» – 64-розрядні. Мабуть, у майбутньому можна очікувати і появи 128-бітових чіпів.

2. Рух у напрямку RISC. Абревіатура RISC розшифровується як Reduced Instruction Set Computer – комп'ютер із скороченим набором команд.

Для того щоб зрозуміти сенс цього явища, треба повернутися до ранньої історії ЕОМ. В ті часи алгоритмічні мови і компілятори ще не були відомі, і усе програмування велося вручну, в командах процесора. Тому розробники комп'ютерів намагалися зробити систему команд зручною для ручного програмування, наповнивши її складними і місткими командами.

Наприклад, однією машинною командою можна було обчислити функцію log або sin, або перетворити число в іншу систему числення. Репертуар машинних команд виходив досить складним. Наприклад, в IBM-360 були реалізовані 144 команди центрального процесора. Така організація системи команд дістала назву CISC – Complex Instruction Set Computing, тобто обчислення зі складним набором команд.

Основоположником CISC-архітектуру можна вважати компанію IBM з її базовою архітектурою /360, ядро якої використовується з 1964 року і дійшло до наших днів. Лідером в розробці мікропроцесорів з повним набором команд CISC вважається компанія Intel зі своєю серією x86 і Pentium (виключаючи сучасні Intel Pentium 4, Pentium D, Core, AMD Athlon, Phenom, які є гібридними), а також процесори Motorola MC680x0. Ця архітектура є практичним стандартом для ринку мікрокомп'ютерів.

Для CISC-процесорів характерно:

- порівняно невелике число регістрів загального призначення;
- велика кількість машинних команд, деякі з яких навантажені семантично аналогічно операторам високорівневих мов програмування і виконуються за багато тактів;
- велика кількість методів адресації;
- велика кількість форматів команд різної розрядності;
- переважання двоадресного формату команд;
- наявність команд обробки типу регістр-пам'ять.

Стандартний набір команд чіпа i8086 і усіх подальших поколінь процесорів Intel містить близько ста інструкцій найрізноманітнішого призначення і формату. Оскільки формат команди змінний, то вона може бути коректно вибрана з пам'яті тільки після розшифровки коду операції. У результаті кожна інструкція вимагає для свого виконання декілька тактів процесора. Програма, що реалізовує деякий

алгоритм, може бути відносно короткою, проте час виконання цієї програми в комп'ютері виявляється значним.

Процесори з RISC-архітектурою працюють по-іншому. У них набір команд сильно обмежений, усі інструкції максимально спрощені, вони мають однаковий формат і, в ідеалі, можуть виконуватися за один машинний такт. Програма, що виконує той же алгоритм примітивними командами, виходить довшою, проте за рахунок високої швидкодії процесора спостерігається значний виграш в продуктивності.

Зрозуміло, що програмувати вручну для такої машини було б незручно, проте цього ніхто і не робить, оскільки техніка компіляції досягла великих висот. Швидкодіючі оптимізуючі компілятори дозволяють створити такий код, який використовує усі особливості набору команд і дозволяє добитися найвищої обчислювальної потужності.

Зачатки цієї RISC-архітектури йдуть своїми коренями до комп'ютерів CDC6600, розробники яких (Торнтон, Крей та ін.) усвідомили важливість спрощення набору команд для побудови швидких обчислювальних машин. Цю традицію спрощення архітектури С. Крей з успіхом застосував при створенні широко відомої серії суперкомп'ютерів компанії Cray Research. Проте остаточне поняття RISC-процесорів в сучасному його розумінні сформувалося на початку 1980-х років на базі трьох дослідницьких проектів університету Берклі, Стенфордському і Каліфорнійському університетах США. Процесори 801 компанії IBM, процесори RISC університету Берклі і процесори MIPS Стенфордського університету виконували невеликий (50-100) набір команд, тоді як звичайні CISC-процесори виконували 100-200 команд.

Прибічники RISC-архітектури на ділі довели силу своїх аргументів – найпродуктивніші сервери і робочі станції сьогодні використовують RISC-процесори. Проте і прихильники CISC-технології не здаються, на їх боці велетенський об'єм накопиченого програмного забезпечення в кодах ix86. В останніх моделях мікропроцесорів Intel спеціально для мультимедійних додатків введені ще складніші «векторні» команди додаткового набору MMX (MultiMedia eXtention – мультимедійне розширення), що виконують в наддовгих (128 розрядів) регістрах паралельно декілька операцій складання або множення.

Характерні особливості RISC-процесорів.

1. Фіксована довжина машинних інструкцій (наприклад, 32 біти) і простий формат команди.

2. Спеціалізовані команди для операцій з пам'яттю – читання або запису. Операції виду «прочитати-змінити-записати» відсутні. Будь-які операції «змінити» виконуються тільки над вмістом регістрів.

3. Велика кількість регістрів загального призначення (32 і більше в порівнянні з 8-16 регістрами в CISC архітектурі), що дозволяє великому об'єму даних зберігатися в регістрах на процесорному кристалі більший час і спрощує роботу компілятора з розподілу регістрів під змінні.

4. Відсутність підтримки операцій виду «змінити» над укороченими типами даних – байт, 16-бітове слово. Так, наприклад, система команд DEC Alpha містила тільки операції над 64-бітовими словами, і вимагала розробки і подальшого виклику процедур для виконання операцій над байтами, 16- і 32-бітовими словами.

5. Відсутність мікропрограм усередині самого процесора. Те, що в CISC процесорі виконується мікропрограмами, в RISC процесорі виконується як звичайний (хоча і поміщений в спеціальне сховище) машинний код, що не відрізняється принципово від коду ядра ОС і додатків.

Для того щоб об'єднати переваги обох підходів, в останніх розробках компанії Intel (маються на увазі Pentium і Pentium Pro), а також її послідовників – конкурентів (AMD R5, Cyrix M1, NexGen Nx586 та ін.) використовуються ідеї, реалізовані в RISC-мікропроцесорах, на ходу перетворюючи CISC команди в набір RISC команд і виконуючи їх на своєму RISC ядрі (гібридна архітектура). На зовнішньому рівні мікропроцесор виконує стандартний CISC-набір команд, а на внутрішньому – деякий спрощений RISC. Вбудований мікропрограмний емулятор перетворює кожну зовнішню команду в ланцюжок внутрішніх, і виконує її з усією можливою продуктивністю RISC-обчислювача.

3. Ускладнення архітектури процесора. Ще один резерв підвищення продуктивності криється в розпаралелюванні обчислень усередині одного кристала, при цьому розробники мікросхем намагаються реалізувати в конструкції принципи, типові для організації промислового виробництва.

Як відомо, кожна машинна операція складається з декількох фаз: вибірка команди, розшифровка її, читання операндів, безпосереднє виконання операції, запис результату. У старих моделях процесора ці фази виконувалися для кожної операції строго послідовно подібно до того, як в кустарних майстернях йшло колись складання автомобілів – спочатку збирали одну машину, потім другу, при цьому частина робітників постійно простоювала.

Сучасний мікропроцесор влаштований значно складніше. Він схожий на підприємство, в якому величезна кількість робітників збирає на конвеєрі потік автомобілів. Конвеєрний процесор поєднує за часом виконання декількох команд: для однієї відбувається читання операції, для другої – декодування і вибірка регістрів, для третьої – виконання команди обчислювальним блоком тощо. В результаті при тій же тактовій частоті істотно підвищується загальна продуктивність. Більше того, в найдосконаліших конструкціях в чіп мікропроцесора вбудовується декілька самостійних (до 6-8) обчислювальних блоків з фіксованою і плаваючою арифметикою, надшвидка внутрішня пам'ять (кеш) і управляючий пристрій. Неминуча плата за таку організацію – значне підвищення складності і вартості схеми. Проте прогрес мікроелектроніки дозволив реалізувати таку архітектуру в усіх сучасних моделях мікропроцесорів.

4. Багатопроцесорні конфігурації. Коли можливості одного кристала вичерпані, продуктивність комп'ютера в цілому може бути збільшена за рахунок багатопроцесорної організації. Аналіз реальних додатків показує, що довгі ланцюжки машинних команд, які повинні виконуватися строго послідовно, зустрічаються відносно нечасто, в основному в наукових розрахунках.

У принципі число процесорів в комп'ютері нічим не обмежене. Відомі конструкції з сотнями і навіть тисячами процесорів. Проте сумарна продуктивність багатопроцесорної системи росте далеко не лінійно з числом процесорів, оскільки в кожній програмі є деяка межа розпаралелювання, до того ж у багатопроцесорних системах різко зростають накладні витрати на диспетчеризацію обчислювального процесу.

Практика показала, що на стандартних комерційних задачах продуктивність системи росте як sqrt(n), тобто чотирипроцесорна конфігурація всього в два рази

продуктивніше однопроцесорної. Проте на спеціальних задачах, що допускають багатократне розпаралелювання, багатопроцесорні комп'ютери можуть показувати рекорди продуктивності.