Тема 11. Суперскалярні процесори

Оскільки можливості по вдосконаленню елементної бази вже практично вичерпані, подальше підвищення продуктивності ОМ лежить у площині архітектурних рішень. Як уже наголошувалося, один з найбільш ефективних підходів у цьому плані – введення в обчислювальний процес різних рівнів паралелізму. Раніше розглянутий конвеєр команд – типовий приклад такого підходу Тим же цілям служать і арифметичні конвеєри, де конвеєризації піддається процес виконання арифметичних операцій. Додатковий рівень паралелізму реалізується у векторних і матричних процесорах, але тільки під час обробки багатокомпонентних операндів типу векторів і масивів. Тут висока швидкодія досягається за рахунок одночасної обробки всіх компонентів вектора або масиву, проте подібні операнди характерні лише для достатньо вузького круга вирішуваних завдань. Основний об'єм обчислювального навантаження зазвичай приходиться на скалярні обчислення, тобто на обробку поодиноких операндів, таких, наприклад, як цілі числа. Для подібних обчислень додатковий паралелізм реалізується значно складніше, проте він можливий, і прикладом можуть служити суперскалярні процесори.

Суперскалярним (цей термін вперше був використаний у 1987 році) називається центральний процесор (ЦП), який одночасно виконує більш ніж одну скалярну команду. Це досягається за рахунок включення до складу ЦП декількох самостійних функціональних (виконавчих) блоків, кожен з яких відповідає за свій клас операцій і може бути присутнім у процесорі в декількох екземплярах. Структура типового суперскалярного процесора показана на рис. 11.1. Процесор включає шість блоків: вибірки команд, декодування команд, диспетчеризація команд, розподілу команд по функціональних блоках, блок виконання і блок оновлення стану.

Блок вибірки команд витягує команди з основної пам'яті через кеш-пам'ять команд. Цей блок зберігає декілька значень лічильника команд і обробляє команди умовного переходу.

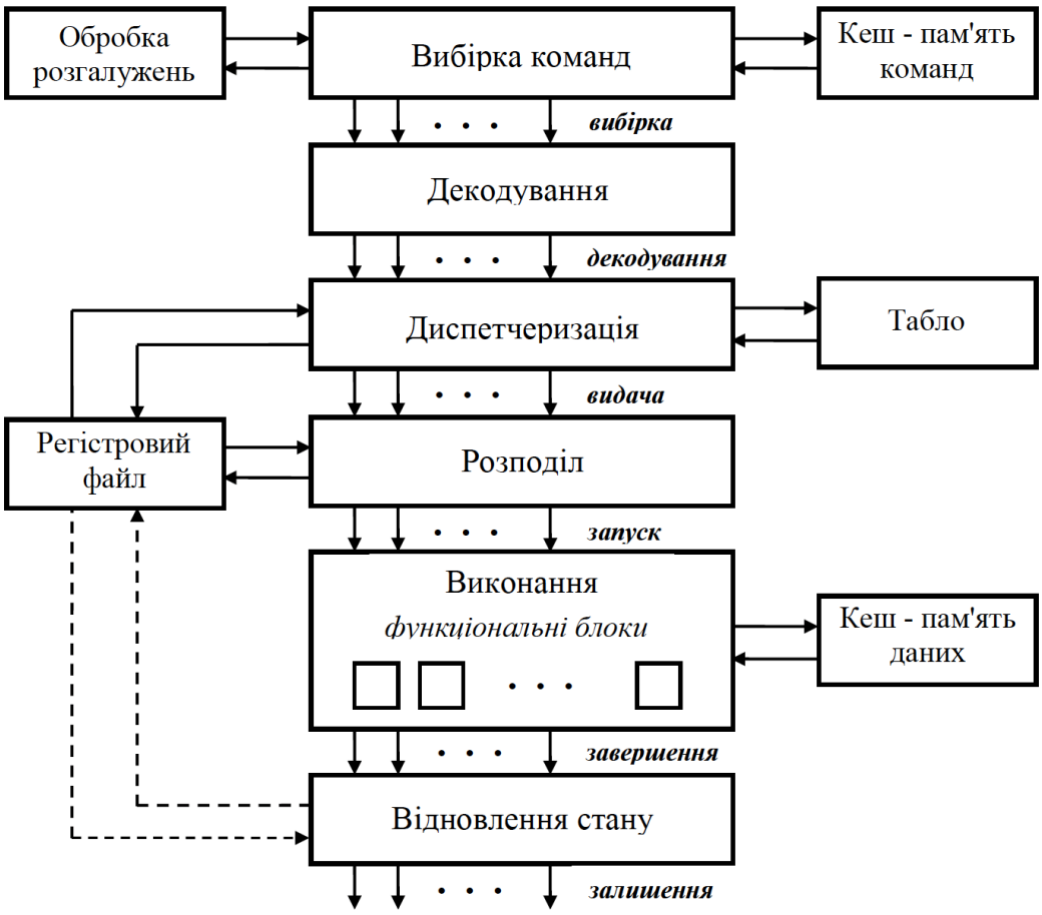

Рисунок 11.1 – Архітектура суперскалярного процесора

Блок декодування розшифровує код операції, що міститься у витягнутих з кеш-пам'яті командах. У деяких суперскалярних процесорах, наприклад у мікропроцесорах фірми Intel, блоки вибірки і декодування суміщені.

Блоки диспетчеризації і розподілу взаємодіють між собою і в сукупності відіграють у суперскалярному процесорі роль контролера трафіка. Обидва блоки зберігають черги декодованих команд. Черга блока розподілу часто розосереджується по декількох самостійних буферах – накопичувачах команд або схемах резервування (reservation station), – призначених для зберігання команд, які вже декодовані, але ще не виконані. Кожен накопичувач команд пов'язаний зі своїм функціональним блоком (ФБ), тому число накопичувачів зазвичай дорівнює числу ФБ, але якщо в процесорі використовується декілька однотипних ФБ, то їм додається загальний накопичувач. По відношенню до блока диспетчеризації накопичувачі команд виступають у ролі віртуальних функціональних пристроїв.

На додаток до черги, блок диспетчеризації зберігає також список вільних функціональних блоків, що називаються табло (Score-board). Табло використовується для відстеження стану черги розподілу. Один раз за цикл блок диспетчеризації витягує команди зі своєї черги, зчитує з пам'яті або регістрів операнди цих команд, після чого, залежно від стану табло, поміщає команди і значення операндів у чергу розподілу. Ця операція називається видачею команд. Блок розподілу в кожному циклі перевіряє кожну команду в своїх чергах на наявність усіх необхідних для її виконання операндів і у разі позитивної відповіді починає виконання таких команд у відповідному функціональному блоці.

Блок виконання складається з набору функціональних блоків. Прикладами ФБ можуть служити цілочисельні операційні блоки, блоки множення і додавання з плаваючою комою, блок доступу до пам'яті. Коли виконання команди завершується, її результат записується і аналізується блоком оновлення стану, який забезпечує облік отриманого результату тими командами в чергах розподілу, де цей результат виступає як один з операндів.

Як було відмічено раніше, суперскалярність передбачає паралельну роботу максимального числа виконавчих блоків, що можливо лише у разі одночасного виконання декількох скалярних команд. Остання умова добре поєднується з конвеєрною обробкою, при цьому бажано, щоб у суперскалярному процесорі було декілька конвеєрів, наприклад два або три.

Подібний підхід реалізований у мікропроцесорі Intel Pentium, де є два конвеєри, кожен зі своїм АЛП (рис. 11.2).

Відзначимо, що тут, на відміну від стандартного конвеєра, в кожному циклі необхідно проводити вибірку більш ніж однієї команди. Відповідно, пам'ять ОМ повинна допускати одночасне зчитування декількох команд і операндів, що найчастіше забезпечується за рахунок її модульної побудови.

Більш інтегрований підхід до побудови суперскалярного конвеєра показаний на рис 11.3.

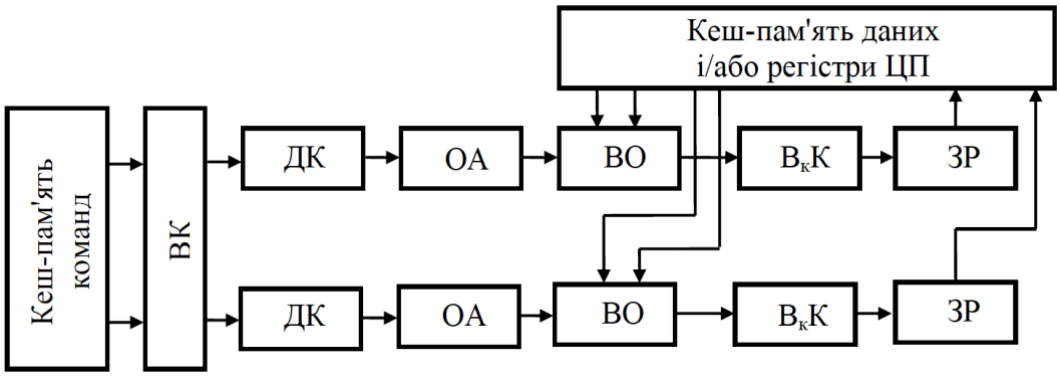

Рисунок 11.2 – Суперскалярний процесор з двома конвеєрами

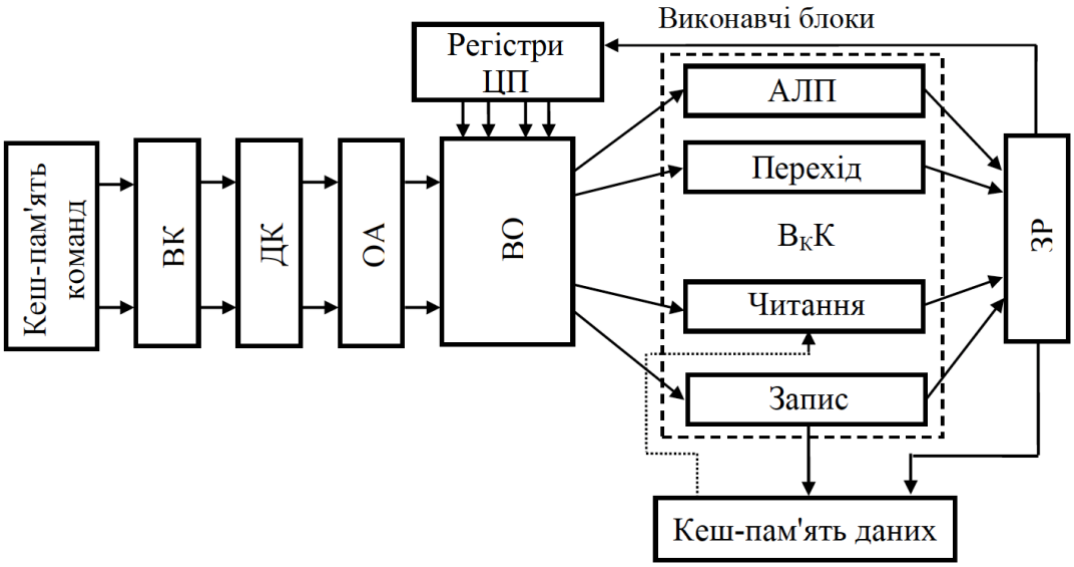

Рисунок 11.3 – Суперскалярний конвеєр із спеціалізованими виконавчими блоками

Тут блок вибірки (ВК) витягує з пам'яті більше однієї команди і передає їх через ступені декодування команди і обчислення адрес операндів у блок вибірки операндів (ВО). Коли операнди стають доступними, команди розподіляються по відповідних виконавчих блоках. Звернемо увагу, що операції «Читання», «Запис» і «Перехід» реалізуються самостійними виконавчими блоками.

Подібна форма суперскалярного процесора використовується в мікропроцесорах Pentium II і Pentium III фірми Intel, а форма з трьома конвеєрами – в мікропроцесорі Athlon фірми AMD. За різними оцінками, застосування суперскалярного підходу приводить до підвищення продуктивності ОМ у межах від 1,8 до 8 разів.

У процесорах деяких ОМ реалізовані як суперскалярність, так і суперконвеєризація. Таке поєднання має місце в мікропроцесорах Athlon і Duron фірми AMD, причому охоплює воно не тільки конвеєр команд, але і блок обробки чисел у формі з плаваючою комою.

Таким чином, суперскалярні мікропроцесори є лідируючим продуктом мікроелектроніки, і їх продуктивність постійно зростає.

11.1 Логіка роботи суперскалярного мікропроцесора

Попередня вибірка команд і передбачення переходів. Оскільки під час суперскалярної обробки необхідно витягати з пам'яті кілька команд за один такт для завантаження паралельно працюючих функціональних модулів, підвищені вимоги ставляться до пропускної здатності інтерфейсу мікропроцесор – пам'ять. В сучасних мікропроцесорах застосовуються багаторівневі роздільні кеш-пам'яті даних і команд.

Для зменшення втрат процесорних циклів, зв'язаних із промахами у разі звертання до кеш-пам'яті у випадку виконання команд розгалуження, до складу системи кешування введені засоби передбачення переходів, основне призначення яких – підвищити імовірність наявності в кеш-пам'яті необхідної команди.

Виконання умовних розгалужень складається з таких етапів:

- розпізнавання команди умовного розгалуження;
- перевірка виконання умови переходу;
- обчислення адреси переходу;
- передача керування у випадку переходу.

На кожному етапі використовуються спеціальні прийоми підвищення продуктивності:

1. Для швидкого декодування використовуються або додаткові біти в полі команди, або переддекодування команд під час вибору з кеш-пам'яті команд.

2. Часто, коли команда вже обрана в кеш, умова переходу ще не вирахувана. Щоб не затримувати потік команд, у даному випадку

використовується передбачення переходу по одній із декількох можливих схем. Деякі предбачувальники використовують статичну інформацію із двійкового коду програми або таку, що спеціально вироблена компілятором. Наприклад, певні коди операцій частіше виробляють розгалуження, ніж інші коди, або розгалуження більш імовірне (під час організації циклів), або компілятор може встановлювати прапор, який вказує напрямки переходу. Може також використовуватися статистична інформація, що отримана під час трасування програми.

Інші передбачувальники використовують динамічно формовану інформацію в процесі виконання програми. Звичайно це інформація, що стосується історії виконання даного розгалуження, що зберігається в таблиці розгалужень або в таблиці передбачень розгалужень. Таблиця передбачення розгалужень організується за асоціативним принципом, подібно кеш-пам'яті, її елементи доступні за адресою команди, розгалуження якої передбачається. В деяких реалізаціях елемент таблиці передбачення розгалуження є лічильником, значення якого збільшується у разі правильного передбачення і зменшується у разі неправильного. При цьому значення лічильника визначає переважний напрямок розгалужень.

У момент визначення дійсного значення умови розгалуження вноситься зміна в історію розгалуження. Якщо передбачення було невірним, то повинна ініціюватися вибірка правильних команд. Результати команд, що були умовно виконані, повинні бути анульовані.

3. Для визначення адреси розгалуження звичайно потрібно виконати цілочислове додавання, що додає до поточного значення лічильника команд зсув, заданий у полі команди розгалуження. І хоча це не вимагає додаткових циклів для звертання до регістрів, прискорення обчислення адреси може бути досягнуте завдяки використанню буфера, який містить раніше використані адреси переходів.

Декодування команд і перейменування регістрів. У суперскалярних процесорах множина функціональних блоків поєднується з множиною конвеєрів

команд. Таким процесорам властиві всі види залежностей, які характерні для поодиноких конвеєрів, причому положення справ посилюється тим, що конвеєрів декілька. В суперскалярних процесорах одночасна робота декількох конвеєрів стає джерелом додаткових неув'язок, зокрема проблеми послідовності надходження команд на виконання і проблеми послідовності завершення команд.

Перша проблема виникає, коли послідовність видачі декодованих команд на функціональні блоки відрізняється від послідовності, яка передбачена програмою. Подібна ситуація відома як неупорядкована видача команд. Термін упорядкована видача команд використовують, коли команди залишають рівні, які стоять до рівня виконання в упорядкованій програмою послідовності. В обох випадках завершення команд звичайно неупорядковане (неупорядковане завершення команд), і це є друга проблема. Упорядковане завершення відбувається рідше.

Крім того, для суперскалярних процесорів є характерним ще один фактор – конфлікт по функціональному блоку, коли на нього претендують декілька команд, які надходять з різних конвеєрів.

Щоб забезпечити неупорядковану видачу команд, у конвеєрі необхідно максимально розв'язати рівні декодування і виконання. Це досягається за допомогою буферної пам'яті, яка має назву вікно команд. Кожна декодована команда спочатку поміщається у вікно команд. Процесор може продовжувати вибірку і декодування нових команд до повного заповнення буфера. Видача команд з буфера на виконання визначається не послідовністю їх надходження, а мірою готовності. Іншими словами, кожна команда, для якої вже відомі значення усіх операндів і вільний потрібний для її використання функціональний блок, негайно видається з буфера на виконання.

Неупорядковані видача і завершення команд – це додатковий потенціал підвищення продуктивності суперскалярного процесора, але для цього необхідно розв'язати дві проблеми:

– усунути залежність команд по даних (мова йде про залежності типу RAW – читання після запису, та WAW – запис після запису), тобто виключити використання як операнда «застарілого» значення регістра і не дозволяти, щоб чергова команда програми через порушення послідовності виконання команд занесла свій результат у регістр ще до того, як це зробила попередня команда;
– зберегти такий порядок виконання команд, щоб загальний результат обчислень залишався ідентичним результату, отриманому у разі строгого дотримання програмної послідовності.

Для усунення залежності команд по даних використовується прийом, відомий як перейменування регістрів. Спосіб розв'язання другої проблеми називають переупорядковуванням команд.

Перейменування регістрів. Коли команди видаються та завершуються упорядковано, кожний регістр у будь-якій точці програми містить саме те значення, яке диктується програмою. Вживання стратегій з неупорядкованою видачею та завершенням команд приводить до того, що запис у регістри може відбуватись також неупорядковано і окремі команди, коли вони звертаються до деякого регістра, замість потрібного одержують «застаріле» або «випереджаюче» значення.

Основна ідея перейменування регістрів у тому, що кожний новий результат записується в один з вільних у даний момент додаткових регістрів. При цьому посилання на регістр, що замінюється в усіх наступних командах, відповідним чином корегуються. Програміст складає програму, використовуючи імена логічних регістрів. Число фізичних регістрів апаратного регістрового файла (АРФ) звичайно більше числа логічних. «Зайві» регістри АРФ використовуються в процедурі перейменування для тимчасового зберігання результатів до моменту вирішення конфліктів по даних, після чого значення з регістра тимчасового зберігання переписується на своє «штатне» місце.

Для подолання зайвих RAW і WAW залежностей, що виникають у результаті обмеженості логічних ресурсів (комірок пам'яті, регістрів),

використовується механізм динамічного відображення обумовлених текстом програми логічних ресурсів на фізичні регістри мікропроцесора. У разі такого підходу з одним логічним ресурсом може бути зв'язано кілька значень у різних фізичних регістрах, кожне з яких відповідає значенню логічної величини в один з моментів часу послідовного виконання програми.

Коли команда створює нове значення для логічного регістра, фізичний регістр, у якому міститься це значення, одержує ім'я. Наступні команди, що використовують це значення, забезпечуються ім'ям фізичного регістра. Дана процедура називається перейменуванням регістрів.

Номери логічних регістрів динамічно відображаються на номери фізичних регістрів за допомогою таблиць підстановки, які оновлюються після декодування кожної команди. Черговий результат записується в новий фізичний регістр, але значення кожного логічного регістра запам'ятовується, завдяки чому легко відновлюється у випадку, якщо виконання команди повинне бути перерване через виникнення виняткової ситуації або неправильного прогнозу напряму умовного переходу.

Перейменування регістрів може бути реалізоване іншим методом – за допомогою буфера перейменування. Це буфер з одним входженням для кожної ініційованої на виконання команди і являє собою асоціативний запам'ятовуючий пристрій або набір регістрів з асоціативним доступом. Цей буфер називається таким, що переупорядковує, тому що він використовується також для встановлення порядку команд у разі переривань. Даний буфер можна розглядати як FIFO чергу, що виконана у вигляді кільцевого буфера з покажчиками “початок” і “кінець”.

Команди поміщаються в кінець буфера. Після завершення команди її результат заноситься в заздалегідь запропонований їй елемент черги, незалежно від місця в черзі, займаного цим елементом. До моменту досягнення командою початку буфера, якщо вона була виконана, її результат поміщається в регістровий файл, а сама команда віддаляється. Команда, що знаходиться в буфері і не виконана через те, що відсутні значення операнда, залишається в

ньому аж до одержання цього значення. Одночасно може вибиратися з черги або поміщатися в неї кілька команд, однак завжди дотримується дисципліна FIFO.

Незалежно від способу перейменування в суперскалярному процесорі усуваються зайві залежності за даними.

Переупорядкування і виконання команд. Після декодування команд і перейменування регістрів команди передаються на виконання. Як уже відмічалося раніше, видача команд у функціональні блоки може виконуватись неупорядковано, по мірі готовності. Після формування для кожної команди упорядкованих трійок, що складаються з коду операції, фізичних операндів – джерела і результату, і розміщення їх у буферах, настає фаза динамічної перевірки готовності значень операндів для виконання команди.

В ідеалі команда готова до виконання, як тільки готові її вхідні операнди. Однак є ряд обмежень, зв'язаних із доступністю фізичних ресурсів, таких як виконавчі пристрої, комутатори і порти регістрових файлів (або буфера, що переупорядковує). Оскільки порядок виконання команд може відрізнятись від порядку, який передбачений програмою, необхідно забезпечити коректність їх операндів (це частково вирішується шляхом перейменування регістрів) і правильну послідовність занесення результатів у регістри регістрового файла. Одним з найбільш розповсюджених прийомів розв'язання цієї проблеми є переупорядкування команд. В його основі лежить використання вікна команд – буферної пам'яті, куди поміщаються усі команди, які пройшли декодування та перейменування регістрів. Вікно команди забезпечує відстрочення передачі команд на виконання до моменту готовності операндів, а також необхідну черговість завершення команд та загрузку їх результатів у регістри регістрового файла.

Відмітимо, що технологія, яка передбачає схему розподілу готових команд по потрібних для їх виконання функціональних блоках з одночасною перевіркою їх доступності, має назву диспетчеризація.

Ця техніка відома також під назвою шелвінг (shelving). Нижче розглядаються два варіанти вікна команд – централізоване і розподілене.

Централізоване вікно команд. Дане вікно реалізується у вигляді так званого табло (Scoreboard). Табло – це буферний запам'ятовуючий пристрій, в якому зберігається деяка кількість останніх витягнутих з пам'яті і декодованих команд, а також поточна інформація про доступність ресурсів, що привертаються для їх виконання. Функціями табло є оперативне виявлення команд, для виконання яких уже доступні всі необхідні операнди і ресурси, і видача таких команд на виконання у відповідні функціональні блоки. Табло можна розглядати як систему попередньої диспетчеризації команд, проте воно здійснює контроль виконання команд і після їх видачі.

Всі витягнуті з пам'яті команди відразу ж після їх декодування і, якщо це необхідно, перейменування регістрів заносяться в табло, причому з дотриманням порядку їх проходження в програмі. Фізично табло реалізується на основі асоціативної пам'яті. Кожній команді виділяється одна комірка, яка складається з декількох полів:

- поля операції, де зберігається дешифрований код операції;
- двох полів операндів, що розміщують значення операндів, якщо вони відомі, або інформацію про те, звідки ці операнди повинні бути отримані;
- поля результату, вказуючого регістру, куди повинен бути поміщений результат виконання даної команди;
- поля бітів достовірності.

В табло також зберігається поточна інформація про доступність пристроїв обробки (функціональних блоків).

Функціонування табло тісно пов'язане з роботою буфера перейменування і може бути описане таким чином. Кожна команда після декодування і перейменування регістрів заноситься в чергову вільну комірку табло. Декодований код операції поміщається в поле операції. Якщо команда припускає завантаження результату в регістр, то на цей регістр є посилання в буфері перейменування (БП) і в поле результату заноситься номер входу БП, в якому зберігається останнє посилання на даний регістр. Далі робиться спроба

заповнити поля операндів значеннями операндів. Спочатку проводиться пошук потрібного значення в апаратному регістровому файлі (АРФ). Якщо операнд не знайдений, виконується асоціативний пошук посилання на регістр у буфері перейменування. Коли результат вдалий, необхідне значення операнда береться з буфера перейменування. У будь-якому варіанті у разі виявлення достовірного значення операнда поле операнда комірки табло заповнюється знайденим значенням, а відповідний цьому полю біт достовірності встановлюється в одиницю. Якщо ж значення операнда ще не обчислене, то в поле операнда комірки табло заноситься ідентифікатор входу буфера перейменування, де знаходиться останнє посилання на шуканий регістр, при цьому біт достовірності такого поля скидається в 0.

Оновлення інформації про готовність операндів і доступність функціональ-них пристроїв виконується в кожному циклі процесора.

Команда може бути лічена з табло і видана на виконання лише після того, як будуть занесені значення всіх операндів, і лише за умови, що потрібний для виконання цієї команди функціональний блок (ФБ) вільний. Після завершення команди у ФБ проводиться запис отриманого результату в ту комірку буфера перейменування, на яку вказує поле результату. Одночасно проводиться асоціативний доступ до всіх команд, що зберігаються в табло, і в тих з них, де в полях операндів вказаний ідентифікатор оновленого входу БП, цей ідентифікатор замінюється занесеним у регістр новим значенням, з відповідною корекцією бітів достовірності. Далі завершена команда покидає табло. Видалення команди з табло є підставою для перезапису значення результату даної команди в регістр АРФ і видалення відповідного запису з буфера перейменування.

Розподілене вікно команд. У варіанті розподіленого вікна команд на вході кожного функціонального блока розміщується буфер декодованих команд, що називається накопичувачем команд або схемою резервування (reservation station). Після вибірки і декодування команди розподіляються по схемах резервування тих ФБ, де команда виконуватиметься. В буфері команда

запам'ятовується і по готовності видається в зв'язаний з даним пунктом функціональний блок. Логіка роботи кожного накопичувача аналогічна централізованому вікну команд. Видача відбувається тільки після того, як команда отримає всі необхідні операнди, і за умови, що ФБ вільний. Під час оновлення вмісту буфера перейменування файла проводиться доступ до всіх накопичувачів команд, і в них ідентифікатори оновлених входів замінюються значеннями операндів, що зберігаються в цих входах.

Відзначимо одну особливість даної схеми: не потрібно, щоб операнд був обов'язково занесений у відведений для нього регістр – він може бути прискорено переданий прямо в накопичувач команд для негайного використання або буферизований там для подальшого використання.

Число незалежних команд, які можуть виконуватися одночасно, варіюється від програми до програми, а також у межах кожної програми. В середньому число таких команд рівне 1-3, часом зростаючи до 5–6. Механізм резервування орієнтований на одночасну видачу декількох команд, що, як правило, легко реалізувати з розподіленим, а не централізованим вікном команд, оскільки темп завантаження розподілених буферів зазвичай менше, ніж потенційний темп видачі команд. Пропускна здатність лінії зв'язку між централізованим вікном команд і функціональними блоками повинна бути вище, ніж у разі розподіленого вікна. Проте для централізованого вікна характерне ефективніше задіювання ємності буфера.

Ємність накопичувача команд у кожному функціональному блоці залежить від очікуваного числа команд для цього блока. Типовий накопичувач розрахований на 1–3 команди.

Для організації вікна виконання використовуються різні методи: однієї черги, багатьох черг або метод станції, що резервує.

Якщо є одна черга, то перейменування регістрів не потрібне, тому що доступність значень операндів може відзначатися бітом резервування, що відповідний кожному регістру. Регістр резервується, коли команда, що його модифікує, призначається на виконання. І регістр звільняється, коли

закінчується виконання команди. Якщо для команди ресурси не були зарезервовані, то вона припиняє своє виконання.

У методі багатьох черг кожна черга організується для команд одного типу. Наприклад, черга команд із плаваючою крапкою або черга команд роботи з пам'яттю.

Третій метод допускає використання станції, що резервує, яка складається із сукупності елементів, кожний з яких містить позиції для розташування коду операції, найменування першого операнда, самого першого операнда, ознаки доступності першого операнда, найменування другого операнда, самого другого операнда, ознаки доступності другого операнда і найменування регістра результату. Коли команда завершує виконання і виробляє результат, то найменування результату порівнюється з найменуваннями операндів у станції, що резервує. Якщо в резервуючої станції виявляється команда, яка чекає цього результату, то дані записуються у відповідну позицію і встановлюється ознака їх доступності. Коли в команді доступні всі операнди, ініціюється її виконання. Станція, що резервує, стежить за доступністю операндів. Коли команда під час диспетчеризації потрапляє у резервуючу станцію, всі готові операнди із регістрового файла переписуються в поля цієї команди. Коли всі операнди готові, команда виконується. Інколи резервуюча станція містить не самі операнди, а покажчики на них в регістровому файлі або буфері, що переупорядковує.

Завершення виконання команд. Завершальною фазою виконання команди є фаза зміни стану процесора відповідно до виконаної команди. Призначення цієї фази – збереження послідовної моделі виконання програми, у разі реального рівнобіжного виконання окремих команд і умовного виконання команд розгалуження. Для зміни стану процесора застосовуються два основних способи, причому обидва основані на використанні двох станів: стану, зміненого в результаті операції, і стану, необхідного для відновлення.

При першому способі зберігається стан процесора в наборі контрольних точок або у буфері історії обчислень, які, за необхідності, використовуються для відновлення стану.

Другий спосіб передбачає розгляд логічного (архітектурного) і фізичного стану процесора. Фізичний стан змінюється негайно після завершення чергової команди. Архітектурний стан змінюється тоді, коли відомий результат умовно виконаних команд. Для реалізації цього способу використовується буфер, що переупорядковує: результати з буфера відправляються у файл архітектурних регістрів і пам'ять.

У буфері, що переупорядковує, для кожної команди є відповідне їй значення лічильника команд і значення інших регістрів, які необхідні для коректного обслуговування переривань.

Ефективність використання суперскалярних архітектур обмежують принаймні дві обставини.

По-перше, є обмеження на ступінь паралелізму на рівні команд, навіть якщо застосовується найдосконаліша техніка суперскалярних обчислень. Перше обмеження виникає з умовних переходів. Інше виходить з того, що розмір вікна виконання (число активних команд, що можуть виконуватися паралельно) обмежує можливий властивий програмі паралелізм, тому що не розглядається рівнобіжне виконання команд, що знаходяться на відстані, яка перевищує розмір вікна.

По-друге, складність суперскалярного процесора зростає зі зростанням кількості паралельних команд, що виконуються, і навіть швидше.

Найімовірніше, що межею розпаралелювання у разі суперскалярної обробки є запуск на одночасне виконання в кожному такті 7–8 команд.

Таким чином, суперскалярні мікропроцесори є лідируючим продуктом мікроелектроніки, і їхня продуктивність постійно зростає, але у ході використання цих процесорів необхідно ретельно досліджувати архітектурні прийоми одержання високої продуктивності і перевіряти адекватність цих прийомів проблемній області, для розв'язання задач якої створюється

обчислювальна система. Подальше підвищення продуктивності мікропроцесорів зв'язується в даний час зі статичним і динамічним аналізом коду з метою виявлення резервів паралелізму рівня окремих команд і програмних сегментів з використанням інформації, наданої компілятором мови високого рівня. Дослідження в даному напрямку привели до розробки мультискалярної архітектури процесорів, що є подальшим розвитком суперскалярної архітектури.