УДК 004.7

ЛУЦЬКИЙ Г.М.,
ДОЛГОЛЕНКО О.М.,
АКСЬОНЕНКО С.В.,
СТОРОЖУК В.О.

МОДЕЛЮВАННЯ ОБМЕЖЕНОЇ РЕАЛІЗАЦІЇ АРХІТЕКТУРИ ПОТОКУ ДАНИХ В СТРУКТУРІ СУПЕРСКАЛЯРНОГО ПРОЦЕСОРА

У статті представлені результати дослідження особливостей обмеженої реалізації архітектури потоку даних (RDF) у структурі ядра суперскалярного процесора, промодельовані схемотехнічні рішення, пов'язані з реалізацією RDF, уточнено кількість рядків у кожній з трьох станцій резервування досліджуваного ядра.

The results of the study features a restricted implementation of a data flow architecture (RDF), in the structure of the superscalar processor core modeled circuit solutions related to the implementation RDF, the quantity of rows in each of three stations of reservation of the studied core is specified.

Вступ

Архітектура Data Flow [1], в якій відсутній лічильник команд, а момент готовності команд до виконання визначається моментом готовності їхніх даних поки не знайшла свого впровадження у вигляді, запропонованому J. Dennis, через складність її реалізації. Складність пов'язана з тим, що визначення моменту готовності до виконання кожної окремо взятої команди здійснюється на основі використання активної комірки пам'яті для її подання. Для реалізації великої кількості активних комірок пам'яті не тільки потрібні значні апаратурні витрати, але й ускладнюється організація розпаралелювання виконання команд при настанні одночасної готовності великої їх кількості до виконання.

Однак, досить швидко низкою дослідників [2,3] була запропонована обмежена реалізації архітектури потоку даних (Restricted Data Flow (RDF)) в структурі суперскалярного процесора, яка функціонує на основі розширеного алгоритму Tomasulo і дозволяє використовувати дрібнозернистий паралелізм при виконанні RISC операцій (Reduced Instruction Set Computing - обчисленнях зі скороченим набором команд) [4]. Так, наприклад, починаючи з мікроархітектури Intel P6 (1995 р.), в суперскалярних мікропроцесорах фірми Intel реалізовується технологія OoOE (Out-of-Order Execution), заснована на використанні RDF.

Метою роботи є дослідження особливостей реалізації RDF в структурі суперскалярного процесора, розробка схемотехнічних рішень, пов'язаних з реалізацією RDF.

Архітектура CISC-RISC

В даний час домінуючою архітектурою, для побудови мікропроцесорів [4], стала архітектура CISC-RISC («CISC-outside RISC-inside», CISC - Complex Instruction Set Computing). Наприклад, останні процесори Intel: Nehalem, Sandy Bridge, Ivy Bridge, Haswell, а також AMD: K10, Bulldozer, Piledriver, Jaguar побудовані по архітектурі CISC-RISC.

У цих процесорах паралелізм на рівні команд (ILP-Instruction-Level Parallelism) досягається шляхом використання методик реалізації неявного динамічного паралелізму, присутнього на етапі виконання CISC команд, декодованих на безліч RISC-операцій (їх часто називають μops, або подібними термінами). Однією з таких методик є використання RDF.

Для реалізації паралелізму на рівні потоків CISC команд (TLP - Thread-Level Parallelism) або мультитредової архітектури - ядра цих процесорів одночасно обробляють два потоки команд, кожен з яких реалізується відповідно до принципів ILP - архітектури.

Архітектура CISC-RISC отримала назву суперскалярной архітектури [4].

У модельованій мікроархітектурі RISC - ядра є новітні аспекти вирішення проблеми оптимізації процесів виконання команд в конвеєрі та підвищення його пропускної спроможності. Стало майже універсальною нормою споряджати процесори центральним контролюючим і управляючим пристроєм, який на усіх стадіях перевіряє правильність обробки команд в конвеєрі. Цей контролер визначає і координує зміни станів команд, що проходять через конвеєр.

У модельованій мікроархітектурі ядра застосовано децентралізовану схему пристрою контролю і управління процесами виконання команд в конвеєрі, що розподілена по всіх сегментах конвеєра. Ядро одночасно обробляє 4 потоки CISC команд.

Для досягнення ILP, модельована мікроархітектура декодує CISC команди x86-64 в прості стилізовані елементарні обчислювальні операційні одиниці, що названі RISC–операціями. При цьому мікроархитектура може змінювати свої машинні стани в єдиний спосіб – шляхом виконання RISC – операцій.

Мікроархітектура RISC-ядра є глибоко конвеєрною, відносно відомих реалізацій. Глибокий конвеєр реалізовано як декілька коротких сегментних потоків, сполучених чергами. Такий підхід дає можливість застосувати вищу тактову частоту, а негативні наслідки глибокої конвеєризації знижуються за рахунок застосування вдосконаленого пристрою передбачення галужень, надшвидкісного доступу до L2–кеш (другого рівня) і гранично високої тактової частоти.

Модельована мікроархітектура також є архітектурою з динамічним прогнозуванням галужень (speculations). Результати динамічно прогнозованого виконання (виконання, що працює з припущеннями відносно ще не обчисленої умови галуження) не повинні змінювати архітектурні стани процесора доти, доки не буде обрахований результат, припущення відносно якого і викликало це динамічне прогнозування. Після перевірки припущень динамічно прогнозовані обчислення або стають не динамічно прогнозованими, або – недійсними.

Декодер команд

Схема декодера команд (ID) показана на *Рис.1*. На першій стадії його роботи здійснюється запам'ятовування байтів командного вікна в черзі поточного треду буферу розмічених команд. Байти командного вікна надходять з буфера-переддекодера (IFU) через вирівнювач.

Завантаження нового вікна команд оброблюваного потоку проводиться при будь-якій з наступних умов:

- Через неправильне передбачення галужень, внаслідок чого проводиться скидання ID;
- Правильно передбачене за допомогою цільового буфера галужень (BTB) галуження, яке повинне прийматися;
- Усі повні команди, що знаходяться в даний момент у буфері успішно декодовані.

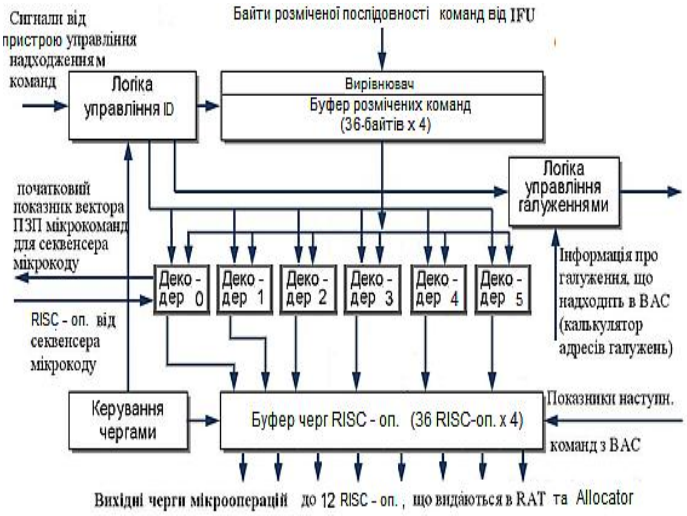

Рис. 1. Декодер команд

З черг буферу розмічені команди спрямовуються на два комплексні декодери та чотири прості. Комплексні декодери: декодер 1 використовуються для декодування довгих команд та макрокоманд, що декодуються не більше ніж в 4 RISC-операції; декодер 0 використовується для декодування складних команд, котрі декодуються в 4 і більше RISC-операції і потребують використання секвенсора мікрокоду (MS). Декодери 2, 3, 4 та 5 призначені для декодування тільки простих команд і макрокоманд, які завжди складають більшість у потоках команд.

Через змінну довжину x86-64 команд ускладняється подача в одному такті шістьох правильно вирівняних команд на шість декодерів. Навіть саме визначення довжини однієї команди не просте, оскільки спочатку повинні декодуватися перші байти, щоб інтерпретувати подальші байти. Оскільки процес подачі шістьох команд змінної довжини по суті послідовний, то необхідно заздалегідь знати розташування меж кожної команди. Пристрій виявлення довжини команд, що знаходиться в буфері-переддекодері IFU виконує функцію попереднього декодування; тобто визначення меж знаходження команд за допомогою сканування потоку байтів команд, маркування першого байта коду операції і останнього байта команди. Крім того, IFU відмічає байти, щоб вказати BTB на передбачення галужень і на контрольні точки можливих запинок коду.

Кожна з чергових шести команд командного вікна подається на один з шести декодерів, що можуть її декодувати. Якщо буфер команд не

містить шести повних команд, то в декодери буде подано стільки розпізнаних і вирівняних команд, скільки їх залишилося в черзі. Логіка управління подачею команд використовує маркери першого байта коду операції для вирівнювання і одночасної подачі команд декодерам.

Оскільки в черги командного буферу може одночасно поступати до 32 команд оброблюваного потоку, то для декодування усіх їх може знадобитися декілька тактів. Розташування початкового байту чергових шести команд, що подаються в деякому такті на декодери може знаходитися де завгодно у черзі буферу. Але спеціальна апаратура вирівнює шість команд і спрямовує їх на шість відповідних декодерів.

Хоча шість команд одночасно подаються на декодери за один цикл, деякі з команд можуть не декодуватися до кінця. Коли команда успішно не декодована, то певний декодер скидається і усі RISC-операції, отримані від цього декодера стають недійсними і анулюються. Це може зайняти множину циклів, що витратяться на успішне декодування усіх команд, котрі знаходяться у черзі буфера. До анулювання RISC-операцій приводять наступні ситуації, внаслідок чого проводиться повторна подача відповідних команд на декодери в подальших циклах:

Якщо на декодері 0 повинна декодуватися макрокоманда чи складна команда, яка вимагає перемикання її на MS з ПЗП мікрокоду (UROM), а він ще не завершив декодування попередньої команди.

Якщо зустрічається галуження, то всі RISC-операції, що вироблені наступними декодерами, стають недійсними. В одному циклі може декодуватися тільки одне галуження.

Число команд і їх складність, які можуть одночасно декодуватися, безпосередньо не мають відношення до числа виданих ID RISC-операцій, тому що буфер черг RISC-операцій декодера запам’ятовує RISC-операції та видає їх пізніше.

Складні команди

Складні команди – це ті команди, які для свого декодування потребують використання MS. В модельованому ядрі такі команди може оброблювати тільки декодер 0. Виклик роботи MS відбувається за двома умовами:

Якщо команда декодується в більше ніж 4 RISC-операції, то 4 перші з них виробляються декодером 0, а наступні – формує MS. При цьому MS отримує початкову адресу UROM від декодера 0 і починає виробляти послідовність RISC-операції, поки не досягне кінця потоку мікрокоду.

Коли зустрічаються повільні команди з великим числом циклів, декодер 0 не видає ніяких RISC-операцій і передає управління MS, який формує послідовність RISC-операцій за допомогою UROM.

Після декодування RISC - операції передаються далі по конвеєру разом з усією інформацією, необхідною для їх планування, диспетчеризації, виконання та вилучення.

Таблиця альтернативних імен регістрів

Використання таблиці альтернативних імен регістрів (RAT) забезпечує перейменування регістрів – цілочислових та з плаваючою точкою, а також регістрів ознак (прапорів), в процесі виконання команд, трансформованих в RISC-операції. Це дає можливість розширити невеликий набір архітектурний регістрів, початково закладений в x86-64. Але головною функцією перейменування регістрів є усунення залежностей між командами і забезпечення більш високого рівня ILP. В RAT логічні адреси регістрів операндів і регістру результату RISC-операції відображаються на відповідні фізичні адреси комірок буферу переупорядковування (ROB). При цьому відповідний масив в RAT, що відповідає поточному потоку команд оновлюється новою фізичною адресую приймача результату даних для кожної нової RISC-операції. Ці нові адреси надаються пристроєм призначення ресурсів (allocator).

Принцип перейменувань регістрів для одного потоку за допомогою RAT показано на *Рис. 2*. У кожному тактовому циклі RAT по своїм масивам повинна швидко відшукувати фізичні адреси ROB, які відповідають логічним посиланням кожної з RISC-операцій. Такі фізичні посилання стають частиною стану RISC-операцій і з цього моменту переміщуються разом з ними, стаючи їх атрибутами. Будь-який машинний стан, який буде модифіковано RISC-операцією також перейменовується, через інформацію, надану пристроєм виділення. Це посилання на фізичний регістр одержувача результату RISC-операції записується в RAT для використання подальшими RISC-операціями, джерела операндів яких посилаються до того ж логічного регістру. Оскільки значення фізичного приймача результату буде унікальне (в результаті перейменування воно вже не співпадає з логічним значенням) для кожної RISC-операції, воно використовується як ідентифіка-

тор для RISC-операції скрізь, де проводиться її позачергове виконання. Усі перевірки і посилання на RISC-операцію виконуються з використанням цього фізичного приймача (physical destination – PDst), як його нове ім'я.

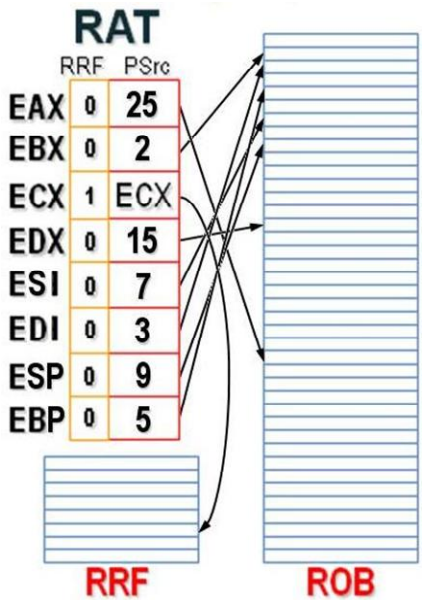

Рис. 2. Принцип перейменування регістрів за допомогою RAT для одного з оброблюваних потоків команд та знаходженням даних в ROB і RRF для цього потоку

Логічні регістри, з якими оперують RISC-операції, повинні перейменовуватися при динамічному прогнозованому виконанні. Якщо виникає пряма залежність за даними, наприклад така:

RISC0: ADD EAX, EBX; src 1 = EBX, src 2 = EAX, dst = EAX;

RISC1: ADD EAX, ECX; src 1 = ECX, src 2 = EAX, dst = EAX;

RISC2: ADD EAX, EDX. src 1 = EDX, src 2 = EAX, dst = EAX;,

то RAT повинна надати в ході обробки, причому "на льоту" – не зупиняючись, перейменовані адреси джерел операндів за допомогою спеціальної логіки обхідних передач. Ця логіка безпосередньо забезпечить для RISC1 джерело регістрового операнда src 2, EAX, щоб уникнути необхідності очікування запису результату в приймач RISC0 того ж EAX, який записуватиметься в RAT і потім читатиметься як джерело операнда RISC1 src.

Коли відбувається неправильне прогнозування галуження, RAT повинна скинути непотрібний стан, який було накопичено і знову повернутися до відображень логічних адрес регістрів на фізичні, які проводитимуться вже з наступним набором RISC-операцій. Схема відновлення неправильного прогнозування галуження гарантує, що RAT не виконуватиме ніяких нових перейменувань, поки ядро позачергового виконання не скине весь непотрібний неправильно прогнозований стан. Це означає, що регістрові посилання тепер знаходитимуться в регістровому файлі вилучення результатів (retirement register file – RRF) та відповідатимуть логічним значенням доти, поки не почнуть з'являтися нові динамічно прогнозовані RISC-операції. Всі незавершені RISC-операції, що ще очікують свого виконання в станціях резервування (RS) і мають посилання на неправильно прогнозовані фізичні регістри, стають недійсними та звільняють RS.

Реалізація RAT

Архітектура процесорів x86-64 дозволяє проводити читання і записи з різними форматами регістрових даних: байти, півслова, слова, подвійні слова з цілочисловими регістрами загального призначення (наприклад: AL, AH, AX, EAX, RAX), що представляє проблему при перейменуванні регістрів. Ця проблема виникає, коли запис часткової ширини стався перед читанням регістрових даних з більшою довжиною формату. В цьому випадку дані, потрібні при читанні формату більшої довжини, можуть бути перекриті множиною попередніх записів до різних частин регістра.

Вирішення цієї проблеми вимагає, щоб RAT запам'ятовувала ширину кожного вхідного запису цілочислового формату, що туди надходить. Це виконується за допомогою додаткового 2-бітового поля для кожного запису, що надходить, в окремі банки менших і більших цілочислових форматів. Дворозрядне кодування допомагає розрізнити чотири розміри записів в регістри шириною: 64, 32, 16 та 8 біт. RAT використовує інформацію про розмір регістрів, щоб визначити чи необхідне більше регістрове значення, ніж воно було раніше записане. В деяких випадках RAT повинна генерувати призупинення при виконанні часткових записів.

Іншим випадком, поширеним при 16-бітовому кодуванні, є незалежне використання 8-бітових регістрових даних. Якби тільки одним псевдонімом підтримувався доступ до цілочислових регістрів для усіх розмірів, то незалежне використання 8-бітових піднаборів регістрів викликало б величезне число помилкових

залежностей. Наприклад, розглянемо послідовність RISC-операцій:

RISC0: MOV AL,#DATA1
RISC1: MOV AH,#DATA2
RISC2: ADD AL,#DATA3
RISC3: ADD AH,#DATA4.

RISC-операції 0 і 1 пересилають незалежні дані в AL і AH. RISC-операції 2 і 3 використовують операнди в AL і AH для виконання підсумовувань. Якщо для регістра "A" було б призначено тільки один псевдонім, то дані в AH операцією RISC1 перезаписувалися б поверх даних, що записуються операцією RISC0 в AL. Потім, коли RISC2 спробує прочитати AL, RAT вже не буде містити правильний показник і виконання RISC2 і RISC3 повинно бути зупинене, поки повністю не завершиться RISC1. Операнд-джерело для RISC2 буде втрачено внаслідок запису в AL. Для відновлення цієї інформації знадобилося б відновлення даних і послідовна обробка команд, що привело б до різкого падіння продуктивності.

Для запобігання цьому, в RAT підтримуються два цілочислові регістрові банки даних (меншого і більшого форматів). Для 32-бітових і 16-бітових доступів в RAT, дані читаються тільки з меншого банку, але запис проводиться в обидва банки одночасно. Для 8-бітових доступів, тільки відповідний більший або менший форматний банк проводить запис або читання, залежно від доступу до старшого або молодшого байту півслова (наприклад, до AH або AL). Тому, старший або молодший байти регістрів використовують різні записи надходжень, але обидва вони можуть бути перейменовані незалежно. Банк великих форматів містить тільки 15 масивів вхідних надходжень, тому що вісім 64-бітових цілочислових регістрів, а саме: R8 - R15, шість 16-бітових сегментних регістрів, а саме: CS, DS, ES, FS, GS, SS та 18-бітовий регістр станів EFlags, відповідно до специфікації x86-64, не розділяються на старші і молодші частини і не мають будь-яких часткових доступів. Масиви вхідних надходжень для шістнадцяти регістрів 128-бітових векторних операндів та операндів з плаваючою комою, а саме: XMM0 – XMM15 не мають часткових доступів, але послідовно можуть попарно об'єднуватися, утворюючи вісім регістрів для 256-бітових векторних операндів команд AVX.

Показники фізичних джерел операнда (PSrc) RAT вказує на комірки в регістровому масиві ROB, де знаходяться необхідні для обчислень дані. Фактично дані не з'являються в ROB, поки RISC-операція не сформує результат виконання і не запише його назад в ROB по шині зворотного запису. Поки не виконається зворотний запис на місце PSrc, ROB містить недійсні значення ("сміття").

Кожен запис в RAT має біт RRF, котрий вказує на місцезнаходження даних архітектурного регістру, що відповідає цьому запису: в RRF, чи в ROB (див. рис. 2). Якщо біт RRF встановлено, то дані знаходяться в RRF і фізична адреса в RAT вказує на відповідний регістр в RRF. Якщо біт RRF скинуто, то дані знаходяться в ROB і фізична адреса вказує на коректну позицію в ROB. По 10-бітовому полю фізичної адреси можна отримати доступ до будь-якого із записів в ROB. Шини влаштовані так, щоб RRF міг передавати початкові дані так само, як і ROB.

RAT здійснює перейменування логічних джерел-операндів (LSrc's) в RISC-операціях, що видаються з буферу черг RISC-операцій. Ці 24 джерела використовуються як індекси для цілочислового масиву RAT. RAT має 24 порти читання, щоб дозволити одночасне читання всіх LSrc's для чергового надходження RISC-операцій.

Після завершення фази читання, цілочисловий масив RAT має бути оновлений новими фізичними приймачами результатів (PDst's) від allocator, що є місцями запису результатів поточних оброблюваних RISC-операцій в ROB, чи RRF. Оскільки можливі залежності усередині циклів виділення приймачів, застосована пріоритетна схема для записів, щоб гарантувати, що для кожного приймача записується коректний PDst. Так, для прикладу з попереднього розділу: вищий: - фізичні приймачі результату в ROB для поточних RISC-операцій RISC3, RISC2, RISC1 та RISC0; нижчий – логічний приймач результату в RRF для будь-якої RISC-операції, що завершується та вилучається з ROB.

Розглянемо дії RAT при завершенні RISC-операції та вилученні з ROB її результату (retirement) і фіксації її кінцевих архітектурних станів (committing) у місці їх постійного зберігання у RRF. ROB повідомляє RAT, що місце розміщення результату приймача RISC-операції (PSrc), котра завершується, звільняється, а її результат повинен бути прийнятим в RRF. По цьому сигналу співпадаючий запис, або мно-

жина записів, в RAT із отриманим значенням поля PSrc від ROB анулюється і надалі звернення до відповідного логічного регістру вказуються на RRF (див. рис. 2).

Механізм вилучення RISC-операції вимагає, щоб в RAT виконалися асоціативні зіставлення отриманого значення поля PSrc від ROB з полем PSrc для кожного з 8 цілочислових масивів RAT. Для усіх знайдених збігів скидаються відповідні записи у масивах, що надалі вказують на RRF. У вилучення є найвищий пріоритет в пріоритетному механізмі зворотного запису – вилучення повинно відбутися раніше, ніж будь-яка нова RISC-операція проведе зворотний запис в ROB. Тому, записи результатів виконання RISC-операцій у ROB здійснюються після скидання RISC-операцій, що вилучаються.

Скидання в перейменуваннях регістрів з плаваючою точкою (FP) ускладнене через застосування стекової організації FP-регістрів в Intel Architecture x86-64. Для використання вершини стека (top-of-stack – TOS) в посиланнях на FP-регістри передбачені спеціальні апаратні ресурси. В RAT підтримується таблиця FP-регістрів, що знаходяться в RRF (retirement FP RAT table – RfRAT), яка містить інформацію про стан регістрового стека для операцій з плаваючою точкою. Кожен запис в RfRAT має 5-бітову ширину: 1 біт виділено як біт недійсності регістру при його вилученні із стека (retired stack valid) і 4 біта для показника адреси регістра в RRF. Застосування RfRAT забезпечує можливість відновлення з неправильно передбачених галужень шляхом запам’ятовування в спеціальному буфері виконаних з динамічним прогнозуванням макрооперацій FXCH.

Макрооперація FXCH з операндом переставляє (виконує свопінг) вершини стека регістрів з плаваючою точкою з «операндом», ставлячи в вершину стека номер регістра, що є операндом в макрооперації FXCH, а номер регістра, що до цього розміщувався у вершині – розміщує у стеку на місці «операнда». Макрооперація FXCH без операнда виконує циклічний зсув стеку на одну позицію вгору.

Макрооперація FXCH могла б бути реалізована як три RISC-операції MOV із застосуванням регістрів для тимчасового зберігання проміжних результатів. Але починаючи з процесора Pentium код обчислень з плаваючою точкою оптимізовано під використання макрооперації FXCH для впорядкування даних та їх подальшої обробки виконавчими блоками. Використання трьох RISC-операцій для реалізації свопінгу FXCH збільшило б продуктивність процесора, але погіршило б оптимізацію коду при обчисленнях з плаваючою точкою, тому в модельованій мікроархітектурі макрооперація FXCH реалізована за допомогою однієї RISC-операції.

Обробка RISC-операції FXCH в RAT здійснюється як перестановка елементів масиву RfRAT. У зв’язку з тим, що ця RISC-операція не потребує ресурсів виконавчих пристроїв, FXCH маркірується як “завершена” в ROB, як тільки ROB отримує її від RAT. Таким чином, FXCH не забирає ресурси RS і виконується за цикл.

В RfRAT може з динамічним прогнозуванням переставлятися будь-яке число її записів, перш ніж станеться неправильне передбачення галуження. При цьому всі раніше видані на виконання RISC-операції повинні бути анульованими. Після неправильно передбаченого галуження ROB видає сигнал, вказуючи, що всі наступні RISC-операції, починаючи з RISC-операції неправильно передбаченого галуження, мають бути вилучені. Це означає, що стан мікроархітектури має бути відновлений до машинного стану, що існував під час неправильного передбачення галуження. Таке відновлення в RAT здійснюється шляхом запам’ятовування всіх динамічно прогнозованих FXCH та інших RISC-операцій з плаваючою крапкою в спеціальному буфері. Вміст цього буферу, або скидається при підтвердженні передбачення, або використовується для відновлення масиву RfRAT при неправильному передбаченні галуження з наступним скиданням вмісту цього буферу. При неправильному передбаченні галуження регістри-приймачі всіх RISC-операцій буферу, крім FXCH, вилучаються із стеку (біт retired stack valid відповідних регістрів в стекові установлюється в “0”).

Перевизначення джерел операндів в RISC-операціях

Коли PDst RISC-операції, що вилучається, все ще викликається RISC-операціями, які тільки проводять читання з таблиць в RAT, в якості PSrc, то ці виклики переадресовуються до відповідного регістру в RRF. Це означає, що при вилученні RISC-операція, що завершується, повинна мати пріоритет над RISC-операціями, які тільки проводять читання з таблиць. Для

цього RISC-операція, що вилучається, виконується як обхідна після апаратного табличного читання. Таким чином, прочитані з таблиць дані будуть перевизначені (overridden) самою останньою інформацією в поточних RISC-операціях.

Механізм перевизначення вимагає виконання асоціативного зіставлення значення обох PSrc всіх RISC-операціями, які проводять читання з таблиць зі всіма показниками PDst, що вилучаються з таблиць в RAT.

Для здійснення одночасного перейменування LSrc's і LDst's у всіх одночасно оброблюваних RISC-операціях та забезпечення широких можливостей швидкого перевизначення PSrc's, RAT реалізовано відповідно до схеми, представленої на *Рис. 3.*

Розроблювана схема RAT, крім вищерозглянутих перевизначень PSrc's, здійснює також перевизначення PSrc's спрямовані на подолання залежностей RAW серед одночасно оброблюваних RISC-операцій. Розглянемо роботу RAT, коли серед одночасно оброблюваних RISC-операцій є наступні RISC-операції:

RISC0: r1 + r3 -> r3

RISC1: r3 + r2 -> r3

RISC2: r3 + r4 -> r5

В цьому прикладі джерело операнда r3 в RISC1 залежить від приймача результату RISC-операції RISC0 за тим же посиланням. Це означає, що між цими двома RISC-операціями існує залежність RAW. Для її подолання дані з r3 необхідні для RISC1 будуть записані в комірку RS, де буде набувати стану готовності RISC1, із зворотної шини запису результату від функціональних пристроїв після виконання RISC0. Це буде здійснено як перевизначення прочитаного з таблиць RAT значення PSr для r3 на очікування операнда із зворотної шини запису результату від функціональних пристроїв після виконання RISC0. Аналогічно, для RISC2 буде здійснено перевизначення прочитаного з таблиць RAT значення PSr для r3 на очікування операнда із зворотної шини запису від виконання RISC1, що визначається за допомогою allocator.

RAT може призупиняти заміну джерел операндів, за зовнішніми та внутрішніми призупинками.

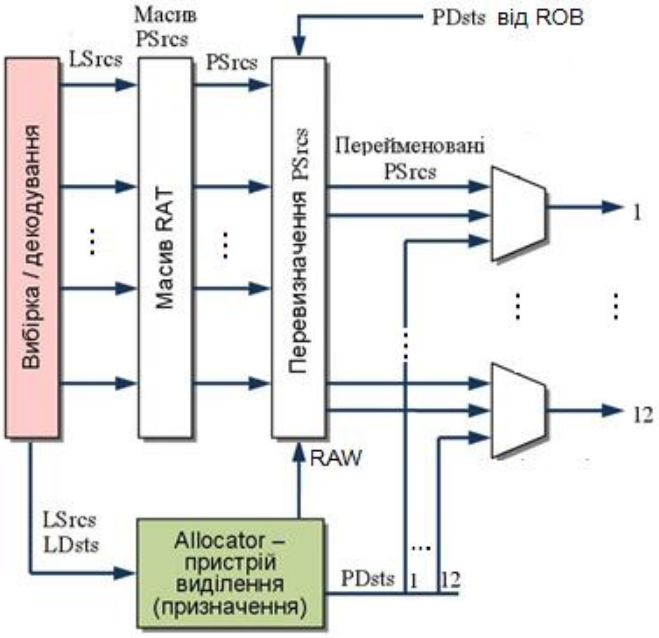

Рис. 3. Одночасні перейменування LSrc's і LDst's та перевизначення PSrc's в RAT

Внутрішні призупинки відбуваються, якщо RAT не може повністю обробити поточний набір RISC-операцій внаслідок наявності часткового регістрового запису в одній із RISC-операцій та наступного читання більшої частини цього ж регістру іншою RISC-операцією набору. В цьому випадку RAT повинен виконати призупинку, поки останній запис часткової довжини необхідного регістра не буде видалено з ROB та здійснено його перенесення в RRF. RAT виконує цю функцію, підтримуючи інформацію про розмір: 8, 16, 32 та 64 біта для кожного регістрового псевдоніма регістрів: A, B, C і D; 16, 32 та 64 біта для кожного регістрового псевдоніма регістрів: Source Index, Destination Index, Base Pointer, Stack Pointer і Instruction Pointer. Згідно специфікації Intel Architecture x86-64 інші регістри не мають часткових записів.

Зовнішнє призупинення RAT відбувається, коли allocator не може обробити чергову порцію RISC-операцій внаслідок відсутності вільних рядків в RS, або комірок в ROB.

Пристрій призначення ресурсів

У кожному тактовому циклі allocator здійснює виділення ресурсів для надходжень, що записуються в: ROB, RS, буфер завантажень (LB – Load Buffer), а також в буфер запам'ятовувань (SB – Store Buffer). Для цього він декодує RISC-операції, що надходять від ID, для визначення потрібних для них ресурсів. У тому випадку, коли allocator не може здійснити виділення потрібних ресурсів для чергової пор-

ції RISC-операцій, подальша видача RISC-операцій призупиняється, поки не стають доступними достатньо ресурсів для подальшого виконання шляхом вилучення з обробки попередніх RISC-операцій.

Призначення для ROB

Для ROB allocator визначає PDst's – номери фізичних регістрів, куди будуть записуватися результати виконаних RISC-операцій. Ці призначення використовуються RAT для створення відповідних записів в таблицях альтернативних імен архітектурних регістрів. RAT використовує PDst's, щоб безпосередньо адресувати ROB. Це означає, що якщо ROB заповнений, allocator повинен встановити сигнал призупинення роботи RAT для запобігання перезапису дійсних даних у ROB.

Для роботи з ROB allocator підтримує список вільних комірок в ROB. Адреси ROB для RISC-операцій allocator призначає послідовно від 0 і до найвищої адреси, а потім пошук вільної адреси для призначення знову починається з 0.

Призначення для станцій резервування

Для RS allocator призначає номери вільних рядків в RS, формує для них біти дозволу запису нових надходжень і передає цю інформацію до RAT. Якщо RS не мають вільних рядків, то allocator виставляє сигнал "призупинення" для запобігання перезапису дійсних даних в RS. Фактично, якщо RS заповнені, то біти дозволу будуть скинуті і, таким чином, ніякий запис, що надходить, не буде дозволено.

Призначення рядків RS здійснюється не так, як призначення для буферу ROB. Оскільки RS диспетчеризують RISC-операції для виконання в позачерговому порядку – у міру готовності їх даних, записи RISC-операцій, що надходять до рядків RS, зазвичай розподілені відповідно до порядку звільнення цих комірок від раніше записаних в них RISC-операцій. Таким чином, модель циклічної буферизації тут не працює. Замість цього використовується схема бітової матриці (побітового відображення), де кожен запис, що поступає, в RS відображається на біт в накопичувачі розміщень в RS. Таким чином, записи в RS можуть заміщатися у будь-якому порядку. Allocator шукає вільні рядки в RS, скануючи бітову матрицю від місця завершення виділення для попередньої порції RISC-операцій і поки не знаходить потрібну кількість перших вільних рядків в RS.

Станції резервування

Станції резервування (RS) у своїй основі є місцем, де RISC-операції, очікують надходжень коректних операндів (тобто поки не вирішаться залежності між RISC-операціями та стануть готовими операнди для їх виконання) і доки не вивільняться потрібні виконавчі пристрої (EU). У кожному тактовому циклі RS здійснюють визначення: достовірності операндів в готових до виконання RISC-операціях, наявності готовності потрібних EU та виконують планування позачергової обробки RISC-операцій. Це планування, або диспетчеризація RISC-операцій, полягає в організації їх відправки на виконання в EU та управлінні ланцюгами обходу (data bypassing) даних, минувши RS і EU, по паралельних обхідних ланцюгах за допомогою мультиплексорів. Надходження в RS можуть містити послідовності RISC-операцій від всіх одночасно оброблюваних потоків, але кожна RS може містити тільки ті RISC-операції, котрі можуть бути виконані на підключених до її портів EU. Всього ядро містить три RS: RS операцій з пам`яттю, RS операцій з фіксованою точкою, RS операцій з плаваючою точкою та з XMM (SSE) регістрами.

Кожна з RS складається із рядків (див. *Рис. 4*). В кожному з рядків зберігається одна RISC-операція, яка набуває в ньому стану готовності, очікуючи своїх операндів. Отримавши операнди, рядок RS, подібно до "активних комірок пам'яті" в архітектурі Деніса [1] "вистрілює на виконання" в EU. Для цього кожен рядок RS розділений на поля.

RISC-операції надходять в рядки RS від RAT. Номер рядка для надходження RISC-операції виділяє allocator на підставі отриманих повідомлень від RS про звільнення рядків (з бітом Busy = 0). При надходженні RISC-операції в рядок RS, значення його біта зайнятості Busy змінюється на протилежне.

Крім того кожен рядок RS містить: поле операції, що має виконатися; поля для запису значень операндів; біти наявності в комірці дійсного значення першого операнда Valid1 та другого – Valid2; біт готовності Ready, який вказує, що всі операнди для цієї RISC-операції надійшли і вона стає готовою до виконання. Рядок містить також поле тегу PDst, що видається разом з результатом виконання RISC-операції на шину зворотного запису і використовується для запису результату в комірку ROB та в інші рядки RS в якості операнда. Для команд завантаження/ запам'ятовування в цьому полі знаходиться адреса.

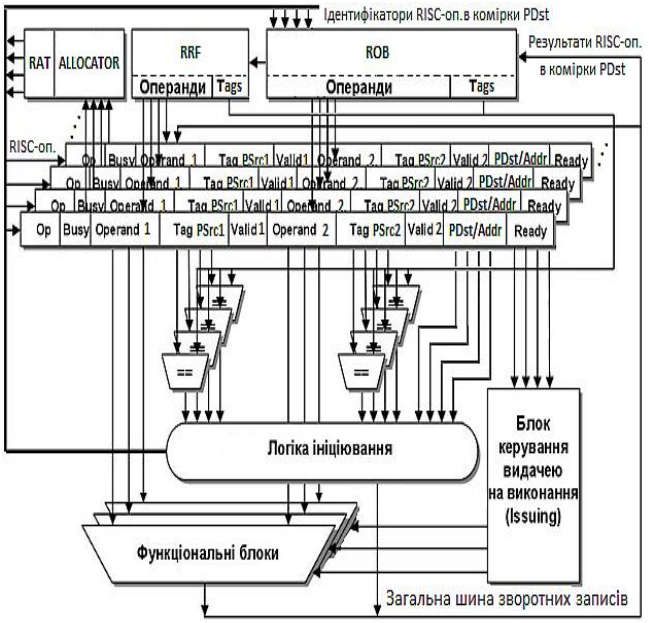

Рис. 4. Надходження RISC-операції до рядків RS, набуття готовності та видача їх на виконання

RISC-операція залишається в RS, поки не стануть дійсними всі джерела її операндів. Чергова порція операндів для RISC-операцій, що зберігаються в RS, може надходити в кожному такті по одинадцяти портам: чотирьом від ROB, чотирьом від RRF і трьом від шини зворотних записів масиву EU. Пристрої адресації пам'яті по вмісту тегів операндів PSrc (content addressable memories – CAMs) в RS управляють "захопленням" операндів з цих портів та записом їх в відповідні поля RISC-операцій з відповідним значенням Src. Запис операнда в поле RISC-операції відбувається при співпадінні бітів тегу фізичного джерела PSrc RISC-операції з бітами тегу фізичного приймача PDst на шині відповідного порту. При цьому RS виконують також наступні операції:

- відшуковують вільні рядки зі значенням біту зайнятості Busy=0 та передають їх номери до allocator;
- встановлюють біт зайнятості рядку Busy=1 при запису в нього нової RISC-операції;
- встановлюють біт наявності в рядку першого операнда Valid1=1 чи біт наявності в рядку другого операнда Valid2=1 при запису в поле операндів рядку значення відповідного операнда;
- встановлюють біти Valid1=Valid2=1 при запису в поле операндів значення операнда для одноопераднної RISC-операції;
- формують біт готовності рядку Ready = Valid1&Valid2;
- здійснюють моніторинг бітів Busy всіх пристроїв з масиву EU;
- видають RISC-операцію для виконання в пристрій з масиву EU з бітом Busy=0 з рядку, що має біт Ready = 1 та потребує для свого виконання цього блоку, встановлюючи значення бітів рядку Busy= Valid1=Valid2 = 0.

RS визначають наявність вільних EU та готових до виконання, RISC-операцій, що потребують для обробки цих EU. Для цього вони використовують пріоритетний показник для визначення рядка початку пошуку готових до виконання RISC-операцій. Пріоритетний показник змінюється згідно алгоритму псевдо-FIFO. Це застосовується для збільшення продуктивності RS.

За один тактовий цикл RS можуть диспетчеризувати до дванадцяти RISC-операцій: три операції з плаваючою точкою, три операції з фіксованою точкою, чотири операції формування адрес (AGU) і дві операції запам'ятовування даних (STD).

Деякі RISC-операції можуть диспетчеризуватися для виконання більш ніж до одного порту в RS. Дія по фіксації такої RISC-операції до конкретного порту називається прив'язкою. Прив'язка RISC-операції в RS до порту EU, пов'язаного з відповідним інтерфейсом, виконується за допомогою алгоритму вирівнювання балансу завантаження, при якому відомо скільки RISC-операцій в RS чекають своєї черги на виконання через цей інтерфейс. Цей алгоритм використовується тільки для RISC-операції, які можуть бути виконаними одним з декількох EU. Така диспетчеризація називається "статичної прив'язкою з вирівнюванням балансу завантаження" готових до виконання RISC-операцій.

Буфер переупорядковування (ROB)

Цей циклічний буфер застосовується в мікроархітектурі RISC-ядра суперскалярного процесора для забезпечення двох найважливіших функцій обробки команд: перейменування регістрів і позачергового виконання команд. В деякій мірі ROB аналогічний регістровому файлу в процесорній частині чергового виконання команд, але з додатковими функціями підтримки вилучення команд з приведенням у відпові-

дність всіх архітектурних станів після перейменувань регістрів та динамічно прогнозованого виконання RISC-операцій.

ROB підтримує позачергове виконання, буферизуючи результати виконання RISC-операцій з масиву EU, перш ніж остаточно привести видимі архітектурні стани у відповідність до цих результатів.

ROB містить усі виконувані RISC-операції від усіх виконуваних в даний момент потоків, у тому числі й службового потоку, що знаходяться в виконавчому конвеєрі. Тобто команди, які були диспетчеризовані, але ще архітектурно не завершилися. Вони включають усі RISC-операції, що знаходяться в RS, усі RISC-операції, що знаходяться на стадії виконання в масиві EU, а також ті RISC-операції, що вже завершили своє виконання та їх результати знаходяться у ROB, очікуючи свого завершення та вилучення з одночасним приведенням у відповідність архітектурних станів виконуваного потоку, відповідно до його початкового програмного порядку. Визначення черговості завершення RISC-операцій відбувається на підставі місця її зберігання у ROB, відносно покажчика початку (голови) ROB (див. *Рис. 5 а*) та значення ідентифікаторів RISC-операцій, що містять інформацію про порядок їх декодування (див. *Рис. 5 в*).

Стан кожної команди у ROB може відстежуватися за допомогою декількох бітів в кожному записі до ROB, що надійшов. Кожна команда тут може знаходитися в одному з декількох станів: очікуванні виконання, виконанні операцій і завершенні виконання. У міру того як команда переходить від одного стану до іншого, ці біти станів оновлюються. Одним з бітів є біт динамічно прогнозованого стану для позначення стану прогнозованого напрямку галуження. ROB здійснює перевірку умови галуження. Коли напрям галуження був визначений правильно, раніше динамічно прогнозовані команди відзначаються вже не динамічно прогнозованими.

В іншому випадку, коли напрям галуження був некоректним, результати виконання RISC-операцій становляться недійсними і відкидаються. Завершуватися з приведенням у відповідність усіх архітектурних станів можуть тільки не динамічно прогнозовані RISC-операції.

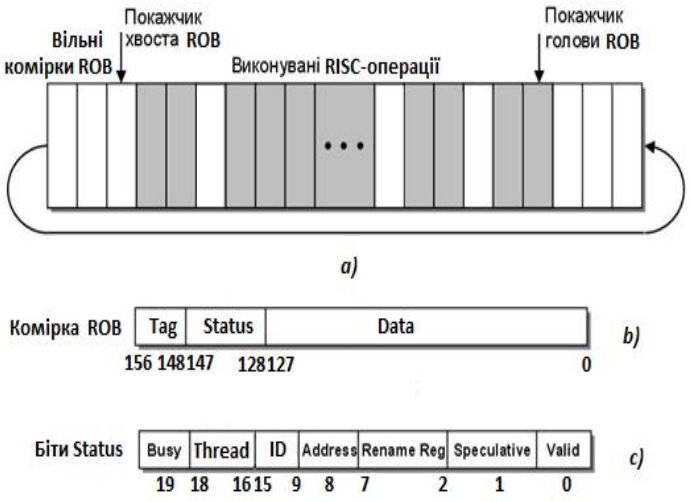

Рис. 5. Структура ROB
a) - завершення і вилучення RISC-операцій з ROB, починаючи з його «голови», та надходження нових RISC-операцій в ROB, починаючи з його «хвоста»;
b) - структура комірки ROB;
c) - структура поля Status в комірці ROB

При неправильно передбаченому галуженні, або виключенні під час виконання CISC-команди, пристрій управління ROB не змінює архітектурні стани у відповідності до отриманих результатів з масиву EU, а просто відкидає результати динамічно прогнозованих RISC-операцій. Для цього він помічає їх тегом Valid = 0 і відправляє allocator повідомлення про звільнення відповідних комірок ROB.

Ключовою функцією ROB є керування процесом вилучення або завершення виконання RISC-операцій. Для підтримки перейменувань регістрів використовується буферний накопичувач результатів з масиву EU. Виконавчі блоки записують дані результатів обчислень в регістри перейменувань. Логіка вилучення з ROB оновлює архітектурні регістри на підставі вмісту кожної перейменованої копії архітектурного регістру, суворо дотримуючись програмного порядку обчислень. RISC-операції, що набувають стану готовності в RS, джерела операндів яких містяться в архітектурних регістрах, можуть отримувати вміст або фактичного архітектурного регістра з RRF, або вміст перейменованого регістра з ROB. Оскільки мікроархітектура RISC-ядра суперскалярна, різні RISC-операції, що використовують в одному циклі один і той же архітектурний регістр, фактично можуть отримати доступ до різних фізичних регістрів.

ROB підтримує out-of-order – позачергове виконання RISC-операції в масиві EU, дозво-

ляючи виконавчим блокам завершувати свої RISC-операції і зворотно записувати результати обчислень незалежно від одночасного виконання інших RISC-операцій. Логіка вилучення RISC-операцій з ROB оновлює архітектурні стани під час вилучення у початковому програмному порядку дотримання, добуваючи потрібну інформацію з ідентифікаторів RISC-операцій, згенерованих декодерами команд і занесених в виділені allocator комірки ROB, минаючи RS і EU.

ROB активний в трьох окремих частинах процесорного конвеєра: на стадіях перейменувань і читання регістрів регістрового файлу; на стадіях виконання і зворотного запису та на стадіях вилучення з приведенням у відповідність архітектурних станів.

ROB сполучений і розташований у безпосередній близькості з allocator і RAT. Allocator виділяє фізичні регістри в ROB для підтримки виконання динамічно прогнозованих операцій і перейменуванням регістрів у RAT. Фактичним перейменуванням архітектурних регістрів в ROB керує RAT. Пристрої allocator і RAT функціонують в схемній частині чергової обробки команд процесорного конвеєра. Таким чином, стадії перейменування і читання регістрів з ROB виконують функції послідовності програмного потоку.

Інтерфейс ROB сполучений з RS, масивом EU та RRF. Дані, прочитувані з ROB в RRF впродовж конвеєрної стадії читання регістрів, можуть бути джерелами операндів RISC-операції в RS, що набувають стану готовності. Ці операнди запам'ятовуються в комірках RS, поки RISC-операції ще не диспетчеризувалися для EU. Виконавчі блоки проводять зворотні записи результатів виконання RISC-операцій в ROB через десять портів зворотного запису (шість портів виділено повністю для зворотних записів, чотири для часткових зворотних записів при виконанні STD і STA). Зворотні записи результатів проводяться в позачерговому порядку відносно порядку декодування RISC-операцій.

Результати обчислень від EU разом з обчисленими бітами регістра EFLAGS (Status register) записуються в комірку ROB, на яку вказує PDst виконуваної RISC-операції. Якщо виявляється, що виконана RISC-операція була помилково динамічно прогнозованою, тоді біти регістра EFLAGS анулюються разом з цією RISC-операцією і іншими виявленими помилково динамічно прогнозованими RISC-операціями. Інакше, ROB аналізує біти регістра EFLAGS RISC-операції що вилучається разом з повідомленням про можливе виключення, закодоване в ідентифікаторі RISC-операції. Якщо виключення має місце, ROB викликає відповідну дію обробки виключення, перш, ніж прийняти рішення про передачу результату виконаної RISC-операції до RRF для приведення у відповідність архітектурних станів.

Обробник вилучення має інтерфейс, що поєднує його з MS. За допомогою цього інтерфейсу ROB повідомляє MS про виникнення виключення, змушуючи його перейти до певної підпрограми мікрокоду обробника виключень.

В конвеєрі процесорного ядра ROB активний на різних стадіях його роботи. Він задіяний на одній з стадій роботи конвеєрного блоку попередньої обробки. На цій стадії ROB отримує від RAT ідентифікатори RISC-операцій, що згенеровані декодерами команд, і заносить їх в виділені allocator свої комірки. Ці ідентифікатори ROB використовуватиме при завершенні і вилученні RISC-операцій, щоб це завершення і вилучення відбувалось відносно до первинного програмного порядку. Для зменшення ширини записів в RS, а також скорочення обсягу інформації, яка має бути виданою в EUs, ці ідентифікатори RISC-операцій заносяться прямо в комірки ROB, куди в наступному будуть занесені результати виконання цих RISC-операцій.

Відразу після розміщення в RS чергової послідовності RISC-операцій, з ROB читаються до RRF результати попередніх RISC-операцій, завершених відповідно до програмного порядку їх виконання. Ці результати можуть бути одночасно операндами для RISC-операцій, що набувають стану готовності в RS. Взагалі операнди для RISC-операцій можуть надходити від одного з трьох місць: від RRF, від ROB, або з шини зворотного запису від масиву EUs.

На конвеєрній стадії читання результатів з EUs результати виконаних RISC-операцій, разом з обчисленими бітами регістра EFLAGS, записуються в комірки ROB (перейменовані фізичні регістри) відповідно до значень PDst виконаних RISC-операцій. Ця стадія зворотного запису відокремлена від стадій перейменування і читання регістрів, тому що RISC-операції видаються з RS в позачерговому порядку. Арбітраж шини зворотного запису здійснюється арбітром блоку EUs.

На конвеєрній стадії завершення виконання RISC-операцій логіка управління вилученням з

ROB фіксує архітектурні стани в RRF відповідного потоку та звільняє комірки ROB, надсилаючи повідомлення про адреси звільнених комірок до allocator. Конвеєрна стадія вилучення відокремлені від стадії зворотного запису, тому що зворотні записи проводяться в позачерговому порядку обробки RISC-операцій відносно початково заданого програмного порядку. Вилучення фактично зворотно переупорядковує RISC-операції та приводить їх у відповідність до чергового порядку виконання машинних команд. Операції вилучення проводиться за два тактові цикли та є повністю конвеєризованими.

Логіка управління вилученням з ROB працює з ним як з FIFO, починаючи з RISC-операцій, які раніше були розміщені згідно з конвеєром в послідовному порядку черги FIFO. При вилученнях з ROB логіка управління вилученням також аналізує ідентифікатори RISC-операцій, що згенеровані декодерами команд. Це гарантує, що вилучення відповідатиме початковому програмному порядку виконання машинних команд і не змінить дійсні машинні стани.

ROB реалізовано як багатопортовий регістровий файл з окремими портами для розміщення записів полів RISC-операцій, необхідних для подальшого вилучення, зворотних записів від EU, читань з ROB операндів для RS і читань для логіки управління вилученнями з ROB.

Кожен з одночасно оброблюваних потоків команд мають свій окремий RRF, де містяться архітектурні регістри, що відображують архітектурні стани оброблюваного потоку. Не всі дійсні стани процесорного RISC-ядра при обробці відповідного потоку зосереджені в RRF, але будь-який перейменований стан там є.

Блок обчислення адреси наступного командного вікна видає архітектурний показник команд для оброблюваного потоку команд.

Коли ROB визначив, що ядро почало виконувати операції по помилковому шляху галуження, то будь-яким RISC-операціям по цьому шляху не дозволено вилучатися зі зміною архітектурних станів оброблюваного потоку. Замість цього встановлюється сигнал “очищення” та відбувається видалення відповідних RISC-операцій.

Висновки

Досліджена реалізація RDF в ядрі суперскалярного процесора, який здійснює одночасне декодування на RISC-операції командного вікна розміром 32 байта одного з 4 оброблюваних потоків CISC команд з набору x86-64. Декодування командних вікон здійснюється почергово, відносно оброблюваних потоків. Декодування командних вікон, відносно команд окремо оброблюваного потоку, здійснюється з прогнозом їх галуження і позачерговим запуском на виконання. В модельованому ядрі застосована децентралізована схема пристрою контролю та управління процесами виконання команд в конвеєрі, яка розподілена по всіх сегментах конвеєра.

У наслідок буферизації операндів в RS можна уникнути зупинок конвеєра, пов'язаних з погрозами збоїв WAR, WAW і RAW.

Уточнено кількість рядків у кожній з трьох станцій резервування досліджуваного ядра та кількість комірок в ROB.

Список посилань

1. J. Dennis: Data Flow Supercomputers; IEEE Computer, pp. 48-56, Nov. 1980.
2. M. Simone, A. Essen, A. Ike, A. Krishnamoorthy, T. Maruyama, N. Patkar, M. Ramaswami, M. Shebanow, V. Thirumalaiswamy, D. Tovey (1995). Implementation trade-offs in using a restricted data flow architecture in a high performance RISC microprocessor. New York. pp. 151-162.
3. Y. Patt, W. Hwu, et al, Experiments with HPS, a Restricted Data Flow Micro architecture for High Performance Computers, Digest of Papers, COMPCON 86, (March 1986), pp. 254-258.
4. Hennessy John L., Patterson David A. Computer Architecture. A Quantitative Approach: Fifth Edition.-USA: Morgan Kaufman, 2012.-708.