Тема 10. Конвеєризація обчислень

Вдосконалення елементної бази вже не приводить до кардинального зростання продуктивності ОМ. Комп'ютер у цілому являє собою складну логічну схему, і, як правило, в певні періоди тактових сигналів одні вузли цієї схеми працюють, а інші ні. Тому для підвищення швидкості обробки інформації схему потрібно побудувати так, щоб у ній працювала одночасно максимальна кількість вузлів. Перспективнішими в цьому плані являються архітектурні прийоми, серед яких один з найбільш значущих – конвеєризація (рис. 10.1).

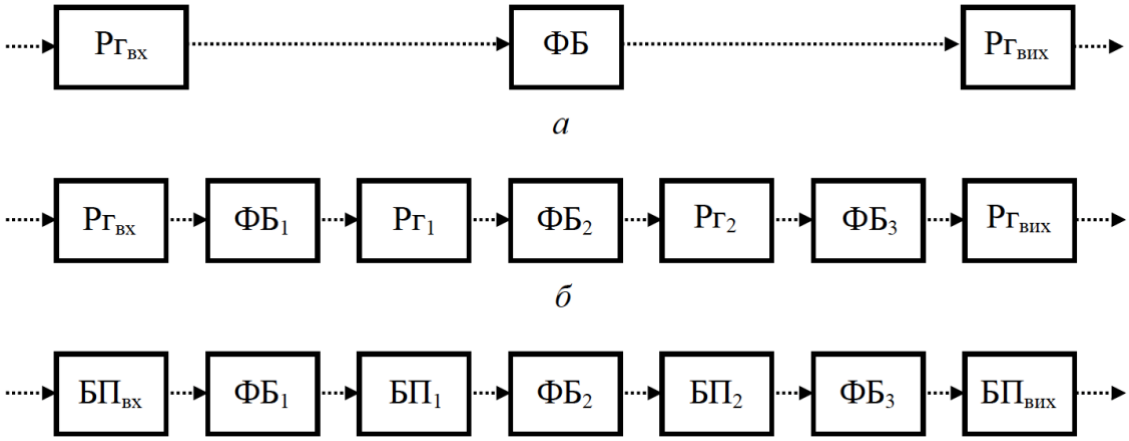


Рисунок 10.1 – Обробка інформації: а – в поодинокому блоці; б –в конвеєрі з регістрами; в – в конвеєрі з буферною пам'яттю

Для пояснення ідеї конвеєра спочатку звернемося до рис. 10.1, а, де показаний окремий функціональний блок (ФБ). Початкові дані поміщаються у вхідний регістр $Рг_{вх}$, обробляються у функціональному блоці, а результат обробки фіксується у вихідному регістрі $Рг_{вих}$. Якщо максимальний час обробки у ФБ дорівнює T_{max}, то нові дані можуть бути занесені у вхідний регістр $Рг_{вх}$ не раніше, ніж опісля T_{max}.

Тепер розподілимо функції, що виконуються у функціональному блоці *ФБ* (рис. 4.32, а), між трьома послідовними незалежними блоками: $ФБ_1$, $ФБ_2$ і $ФБ_3$, причому так, щоб максимальний час обробки в кожному $ФБ_i$ був однаковим і дорівнював $T_{max}/3$. Між блоками розмістимо буферні регістри $Рг_i$, (рис. 4.32, б),

призначені для зберігання результату обробки у $ФБ_i$, на випадок, якщо наступний за ним функціональний блок ще не готовий використовувати цей результат. В розглянутій схемі дані на вхід конвеєра можуть подаватися з інтервалом $T_{max}/3$ (втричі частіше), і хоча затримка від моменту надходження першої одиниці даних в $Рг_{вх}$ до моменту появи результату її обробки на виході $Рг_{вих}$ як і раніше складає T_{max}, подальші результати з'являються на виході $Рг_{вих}$ вже з інтервалом $T_{max}/3$.

На практиці рідко вдається добитися того, щоб затримки в кожному $ФБ_i$, були однаковими. Як наслідок, продуктивність конвеєра знижується, оскільки період надходження вхідних даних визначається максимальним часом їх обробки в кожному функціональному блоці. Для усунення цього недоліку або принаймні часткової його компенсації кожен буферний регістр $Рг_i$, слід замінити буферною пам'яттю $БП_i$, здатною зберігати багато даних і організованою за принципом FIFO, – «першим увійшов – першим вийшов» (рис. 4.32, в). Обробивши елемент даних, $ФБ_i$ заносить результат в $БП_i$, витягає з $БП_{i-1}$ новий елемент даних і приступає до чергового циклу обробки, причому ця послідовність здійснюється кожним функціональним блоком незалежно від інших блоків. Обробка в кожному блоці може продовжуватися до тих пір, поки не ліквідовується попередня черга або поки не переповниться наступна черга. Якщо ємкість буферної пам'яті достатньо велика, відмінності в часі обробки не позначаються на продуктивності, проте бажано, щоб середня тривалість обробки у всіх $ФБ_i$, була однаковою.

В архітектурі обчислювальних машин можна знайти множину об'єктів, де конвеєризація забезпечує відчутний приріст продуктивності ОМ, але найбільш відчутний ефект досягається під час конвеєризації етапів машинного циклу.

За способом синхронізації роботи ступенів конвеєри можуть бути синхронними і асинхронними. Для традиційних ОМ характерні синхронні конвеєри. Зв'язано це, перш за все, з синхронним характером роботи процесорів. Асинхронні конвеєри виявляються корисними, якщо зв'язок між ступенями не великий, а довжина сигнальних трактів між різними ступенями сильно різниться.

Прикладом використання асинхронних конвеєрів можуть служити систоличні комп'ютери.

10.1 Синхронні лінійні конвеєри

Ефективність синхронного конвеєра багато в чому залежить від правильного вибору довжини тактового періоду $T_к$.

Мінімально допустиму $T_к$ можна визначити як суму найбільшого з часів обробки на окремому рівні конвеєра $T_{РМАХ}$ і часу запису результатів обробки в буферні регістри між рівнями конвеєра $T_{БР}$:

$$T_{\text{к}} = T_{\text{РМАХ}} + T_{\text{БР}}$$

Через вірогідний «перекос» у надходженні тактуючого сигналу в різні рівні конвеєра попередній вираз слід доповнити ще одним елементом – максимальною величиною «перекосу» $T_{\text{ПК}}$:

$$T_{\text{к}} = T_{\text{РМАХ}} + T_{\text{БР}} + T_{\text{ПК}}$$

Кожний рівень конвеєра може містити багато логічних трактів обробки. $T_к$ визначається найбільш довгими трактами в усіх рівнях конвеєра. Під час розробки конвеєра необхідно враховувати, що для всіх послідовних елементів, які обробляються одним і тим же рівнем, обробка першого елемента може проходити по тракту максимальної довжини, а другого елемента – по мінімальному тракту. Таким чином результат обробки другого елемента може з'явитись на виході рівня до того, як у вихідному регістрі рівня буде запам'ятований попередній результат. Щоб уникнути подібної ситуації, сума $T_{БР}$ + $T_{ПК}$ повинна бути менша мінімального часу обробки в рівні $T_{РМІН}$, звідки

$$T_{\text{БР}} = T_{\text{РМІН}} - T_{\text{ПК}}$$

Вибір довжини тактового періоду для конвеєра повинен відбуватися з урахуванням співвідношення

$$T_{\text{РМАХ}} + T_{\text{БР}} + T_{\text{ПК}} = T_{\text{к}} = T_{\text{РМАХ}} + T_{\text{РМІН}} - T_{\text{ПК}}$$

Незважаючи на очевидні переваги вибору $T_к$, рівним нижній межі, звичайно орієнтуються на середнє значення між нижньою і верхньою межами.

Щоб охарактеризувати ефект, який досягається за рахунок конвеєризації обчислень, звичайно використовують три показники: прискорення, ефективність і продуктивність.

Під прискоренням розуміється відношення часу обробки без конвеєра і за його наявності. Теоретично найкращий час обробки вхідного потоку з N значень T_{NK} на конвеєрі з K рівнями і тактовим періодом $T_к$ можна визначити виразом

$$T_{\text{NK}} = (K + (N-1))T_{\text{к}}$$

Вираз відображає той факт, що до появи на виході конвеєра результату обробки першого елемента повинно пройти K тактів, а наступні результати будуть проходити в кожному такті.

У процесорі без конвеєра загальний час виконання складає NKT_K. Таким чином, прискорення обчислень S за рахунок конвеєризації обчислень може описуватись виразом

$$S = \frac{NKT_K}{(K+(N-1))T_K} = \frac{NK}{K+(N-1)}$$

При $N \to \infty$ прискорення прагне до величини, яка дорівнює кількості рівнів у конвеєрі.

Другою характеристикою конвеєрного процесора є ефективність E – доля прискорення, яка приходиться на один рівень конвеєра:

$$E = \frac{S}{K} = \frac{N}{K+(N-1)}$$

Третя характеристика конвеєрного процесора – пропускна здатність або продуктивність P – ефективність, яка є діленою на довжину тактового періоду:

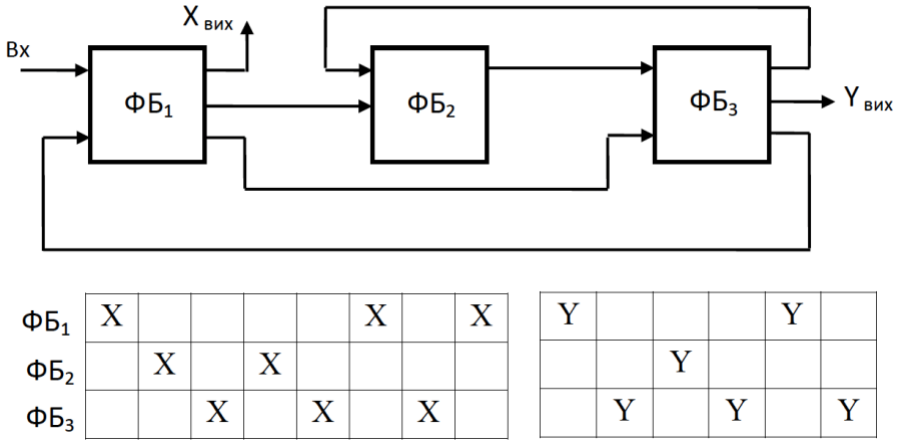
$$P = \frac{N}{T_K(K+(N-1))}$$

При $N \to \infty$ ефективність прагне до одиниці, а продуктивність – до частоти тактування конвеєра:

$$P = \frac{1}{T_K} = F_K$$

10.2 Нелінійні конвеєри

Конвеєр не завжди є лінійним ланцюжком етапів. У ряді ситуацій виявляється вигідним, коли функціональні блоки з'єднані між собою не послідовно, а відповідно до логіки обробки, при цьому одні блоки в ланцюжку можуть пропускатися, а інші − створювати циклічні структури. Це дозволяє за допомогою одного і того ж конвеєра одночасно обчислювати більше однієї функції, проте якщо ці функції конфліктують між собою, то такий конвеєр важко завантажити повністю. Структура нелінійного конвеєра, що одночасно обчислює дві функції X і Y, приведена на рис. 10.2. Там же показана послідовність, в якій функції X і Y застосовують ті або інші функціональні блоки.



ФБ1	X					X		X	Y				Y	
ФБ2		X		X							Y			
ФБ3			X		X		X			Y		Y		Y

Рисунок 10.2 – Нелінійний конвеєр

Щоб визначити, коли пора приступати до повторного обчислення тієї або іншої функції, необхідно побудувати діаграму одноразової реалізації цієї функції і відстежити по ній моменти, коли такий запуск не приведе до конфлікту, пов'язаного з одночасним зверненням до одного і того ж функціонального блока.

Так, у ході реалізації функції X запуск чергового її обчислення можливий після 1, 3 і 6 тактів. Запуск паралельного обчислення функції Y можливий після 2 і 4 тактів. У разі запуску функції Y черговий її запуск дозволений після тактів 1, 3 і 5, а паралельний запуск функції X дозволений після 2 і 4 тактів.

10.3 Конвеєр команд. Конфлікти в конвеєрі команд

Ідея конвеєра команд була запропонована в 1956 році академіком С. А. Лебедєвим. Як відомо, циклом команди є послідовність етапів. Поклавши реалізацію кожного з них на самостійний пристрій і послідовно з'єднавши такі пристрої, ми отримаємо класичну схему конвеєра команд. У циклі команди можна виділити шість етапів:

1. Вибірка команди (ВК). Читання чергової команди з пам'яті і занесення її в регістр команди.

2. Декодування команди (ДК). Визначення коду операції та способів адресації операндів.

3. Обчислення адрес операндів (ОА). Обчислення виконавських адрес кожного з операндів відповідно до вказаного в команді способу їх адресації.

4. Вибірка операндів (ВО). Витягання операндів з пам'яті. Ця операція не потрібна для операндів, що знаходяться в регістрах.

5. Виконання команди (ВкК). Виконання вказаної операції.

6. Запис результату (ЗР). Занесення результату в пам'ять.

Використання конвеєра дозволяє значно скоротити час обробки команд. Проте на практиці через конфліктні ситуації, що виникають у конвеєрі, досягти потенційної продуктивності не вдається.

Конфліктні ситуації в конвеєрі прийнято позначати терміном ризик, а обумовлені вони можуть бути трьома причинами:

- спробою декількох команд одночасно звернутись до одного й того ж ресурсу комп'ютера (структурний ризик);
- взаємозв'язком команд по даних (ризик по даних);
- неоднозначністю під час вибірки наступної команди у випадку команд переходу (ризик по управлінню).

Структурний ризик має місце, коли декілька команд, які знаходяться на різних рівнях конвеєра, спробують одночасно використати один і той же ресурс, частіше за все – пам'ять. Подібних конфліктів частково удається уникнути за рахунок модульної побудови пам'яті і кеш-пам'яті. В цьому випадку є

вірогідність того, що команди будуть звертатись або до різних модулів оперативної пам'яті (ОП), або одна з них стане звертатись до ОП, а інша – до кеш-пам'яті.

Ризик по даних – типова та регулярно виникаюча ситуація. Для пояснення сутності взаємозв'язку команд по даних призначимо, що дві команди в конвеєрі (i та j) передбачають звертання до однієї й тієї ж змінної x, причому команда i попереджає команду j (рис. 10.3).

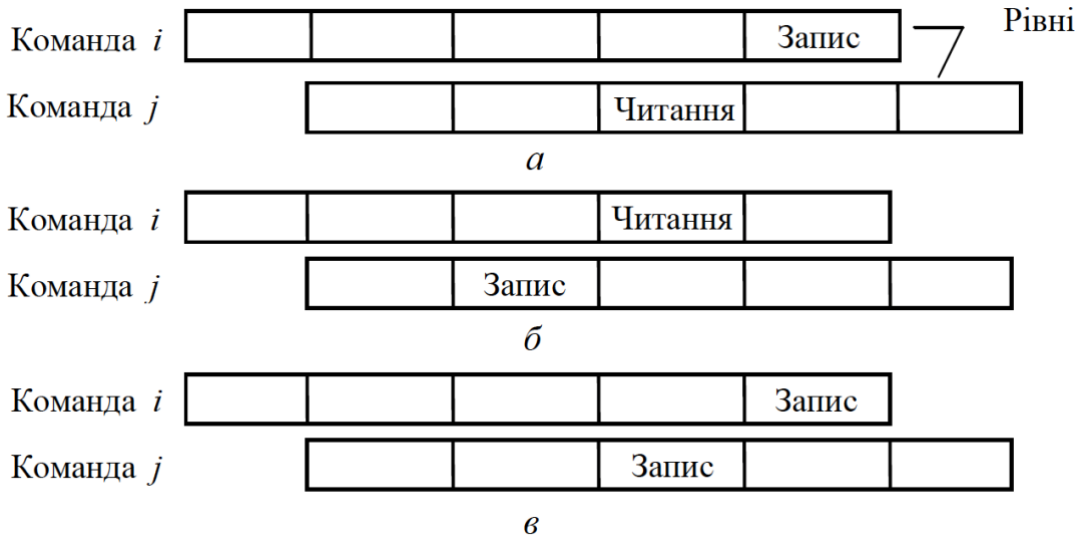


Рисунок 10.3 – Конфлікти по даних: а – «читання після запису»; б – «запис після читання»; в – «запис після запису»

В загальному випадку між i та j можуть бути три типи конфліктів по даних:

а) «Читання після запису» (RAW): команда j читає x до того, як команда i встигла записати нове значення x , тобто команда j помилково отримає старе значення x замість нового.

б) «Запис після читання» (WAR): команда j записує нове значення x до того, як команда i встигла прочитати x, тобто команда i помилково отримає нове значення x замість старого.

в) «Запис після запису» (WAW): команда j записує нове значення x до того, як команда i встигла записати як x власне значення, тобто x помилково містить i-те значення x замість j-го.

Для боротьби з конфліктами по даних використовують як програмні, так і апаратні методи.

Програмні методи орієнтовані на усунення самої можливості конфліктів ще на стадії компіляції програми. Оптимізуючий компілятор намагається створити такий об'єктний код, щоб між командами, які схильні до конфліктів, знаходилась достатня кількість нейтральних у цьому плані команд. Якщо таке не вдається, то між командами, які конфліктують, компілятор вставляє необхідну кількість команд типу «Немає операції».

Фактичне розв'язання конфліктів покладається на апаратні методи. Найбільш ефективним рішенням є зупинка команди j на декілька тактів з тим, щоб команда і встигла завершитися або принаймні минути рівень конвеєра, який викликав конфлікт. Таку ситуацію називають «бульбашкою» в конвеєрі.

Зрозуміло, що зупинки конвеєра знижують його ефективність і розробники ОМ всіляко прагнуть скоротити загальне число зупинок або хоч би їх тривалість. Оскільки найбільш часті конфлікти по даних RAW, основні зусилля витрачаються на протидію саме цьому типу конфліктів. Серед відомих методів боротьби з RAW найбільшого поширення набув прийом прискореного просування інформації (forwarding). Зазвичай між двома сусідніми рівнями конвеєра розташовується буферний регістр, через який попередній рівень передає результат своєї роботи на подальший рівень, тобто передача інформації можлива лише між сусідніми рівнями конвеєра. У разі прискореного просування, коли для виконання команди потрібний операнд, вже обчислений попередньою командою, цей операнд може бути отриманий безпосередньо з відповідного буферного регістра, минувши всі проміжні рівні конвеєра. З даною метою в конвеєрі передбачаються додаткові тракти пересилки інформації (тракти випередження, тракти обходу), які забезпечені засобами мультиплексування.

Найбільші проблеми під час створення ефективного конвеєра обумовлені командами, що змінюють природний порядок обчислень. Простий конвеєр орієнтований на лінійні програми. В ньому ступінь вибірки витягує команди з послідовних елементів пам'яті, використовуючи для цього лічильник команд

(ЛчК). Адреса чергової команди в лінійній програмі формується автоматично, за рахунок додавання до вмісту ЛчК числа, рівного довжині поточної команди в байтах. Реальні програми практично ніколи не бувають лінійними. В них обов'язково присутні команди управління, які змінюють послідовність обчислень: безумовний та умовний перехід, виклик процедури, повернення з процедури і тому подібне Частка подібних команд у програмі оцінюється як 10-20% (за деякими джерелами вона істотно більша). Виконання команд, які змінюють послідовність обчислень (надалі їх називатимемо командами переходу), може приводити до припинення конвеєра на декілька тактів, через що продуктивність процесора знижується. Припинення конвеєра під час виконання команд переходу обумовлено двома чинниками.

Перший чинник характерний для будь-якої команди переходу і пов'язаний з вибіркою команди з точки переходу (за адресою, вказаною в команді переходу). Те, що поточна команда відноситься до команд переходу, стає ясним тільки після декодування (після проходження рівня декодування), тобто після двох тактів з моменту надходження команди на конвеєр. За цей час на перші рівні конвеєра вже поступлять нові команди, які вже витягнуті в припущенні, що природний порядок обчислень не буде порушений. У разі переходу ці рівні потрібно очистити і завантажити в конвеєр команду, розташовану за адресою переходу, для чого потрібна виконавська адреса останньої. Оскільки в командах переходу зазвичай вказані лише спосіб адресації і адресний код, виконавська адреса заздалегідь повинна бути обчисленою, що і робиться на третьому рівні конвеєра. Таким чином, реалізація переходу в конвеєрі вимагає певних додаткових операцій, виконання яких рівносильно зупинці конвеєра як мінімум на два такти.

Друга причина порушення ритмічності роботи конвеєра має відношення тільки до команд умовного переходу. В разі появи команди умовного переходу, конвеєр повинен бути очищений від непотрібних команд, що виконувалися до даного моменту і ще протягом декількох тактів (ДК, ОА, ВО, ВкК) не буде завершеною жодна інша команда. Це і є витрати через неможливість перед-бачення результату команди умовного переходу. Як видно, вони або істотно

більші, ніж для інших команд переходу (якщо перехід відбувається), або відсутні зовсім (якщо перехід не відбувається).

Для скорочень затримок, обумовлених вибіркою команди з точки переходу, застосовуються декілька підходів:

- обчислення виконавчої адреси переходу на рівні декодування команди;
- використання буфера адрес переходу;
- використання кеш-пам'яті для зберігання команд, розташованих у точці переходу;
- використання буфера циклу.

У результаті декодування команди з'ясовується не тільки її приналежність до команд переходу, але також спосіб адресації і адресний код точки переходу. Це дозволяє відразу ж приступити до обчислення виконавської адреси переходу, не чекаючи передачі команди на третій рівень конвеєра, і тим самим скоротити час зупинки конвеєра з двох тактів до одного. Для реалізації цієї ідеї до складу рівня декодування вводяться додаткові суматори, за допомогою яких і обчислюється виконавська адреса точки переходу.

Буфером адрес переходу (BTB, Branch Target Buffer) є кеш-пам'ять невеликої ємності, в якій зберігаються виконавські адреси точок переходу декількох останніх команд, для яких перехід мав місце. Перед вибіркою чергової команди її адреса (вміст лічильника команд) порівнюється з адресами команд, поданих у BTB. Для команди, знайденої в буфері адрес переходу, виконавська адреса точки переходу не обчислюється, а береться з BTB, завдяки чому вибірка команди з точки переходу може бути почата на один такт раніше. Команда, посилання на яку у BTB відсутнє, обробляється стандартним чином. Якщо це команда переходу, то отримана під час її виконання виконавська адреса точки переходу заноситься у BTB, за умови, що команда завершилася переходом.

Застосування BTB дає найбільший ефект, коли окремі команди переходу в програмі виконуються багато разів, що типово для циклів.

Кеш-пам'ять команд, розташованих у точці переходу (BTIC, Branch Target Instruction Cache), – це вдосконалений варіант BTB, де в кеш-пам'ять крім

виконавської адреси команди в точці переходу записується також і код цієї команди. За рахунок збільшення ємності кеш-пам'яті BTIC дозволяє у разі повторного виконання команди переходу виключити не тільки етап обчислення виконавської адреси точки переходу, але і етап вибірки розташованої там команди. Переваги даного підходу найбільшою мірою виявляються у разі багатократного виконання одних і тих же команд переходу, головним чином під час реалізації програмних циклів.

Буфер циклу є маленькою швидкодіючою пам'яттю, що входить до складу першого рівня конвеєра, де проводиться вибірка команд. У буфері зберігаються коди n останніх команд у тій послідовності, в якій вони вибиралися. Коли має місце перехід, апаратура спочатку перевіряє, чи немає потрібної команди в буфері, і якщо це так, то команда витягується з буфера. Стратегія найбільш ефективна під час реалізації циклів та ітерацій, чим і пояснюється назва буфера Якщо буфер достатньо великий, щоб охопити все тіло циклу, вибірку команд з пам'яті досить виконати тільки один раз у першій ітерації, оскільки необхідні для подальших ітерацій команди вже знаходяться в буфері.

За принципом використання буфер циклу схожий на BTIC, з тією різницею, що в ньому зберігається послідовність виконання команд, а сам буфер менший по ємності і дешевший.

Серед ОМ, де реалізований буфер циклу, можна згадати деякі обчислювальні машини фірми CDC (Star 100, 6600, 7600) і супер-ЕОМ CRAY-1.

10.4 Методи вирішення проблеми умовного переходу

Проблеми умовних переходів приводять до найбільших витрат у роботі конвеєра. Для усунення або часткового скорочення цих витрат існують різні способи, які умовно можна розділити на чотири групи: буфери передвибірки; множинні потоки; затриманий перехід; передбачення переходу.

Рівномірність надходження команд на вхід конвеєра часто порушується, наприклад через зайнятість пам'яті або під час вибірки команд, які складаються з декількох слів. Щоб добитися ритмічної подачі команд на вхід конвеєра, між

рівнем вибірки команди та іншою частиною конвеєра часто розміщують буферну пам'ять, яка організована за принципом черги (FIFO) і має назву буфер передвибірки.

Буфер передвибірки можна розглядати як декілька додаткових рівнів конвеєра. Подібне подовження конвеєра не позначається на його продуктивності (при заданій тактовій частоті); воно лише приводить до збільшення часу проходження команди через конвеєр. Типові буфери передвибірки у відомих процесорах розраховані на 2-8 команд.

Щоб одночасно з забезпеченням ритмічної роботи конвеєра розв'язати і проблему умовного переходу, в конвеєр включають два таких буфери (рис. 10.4).

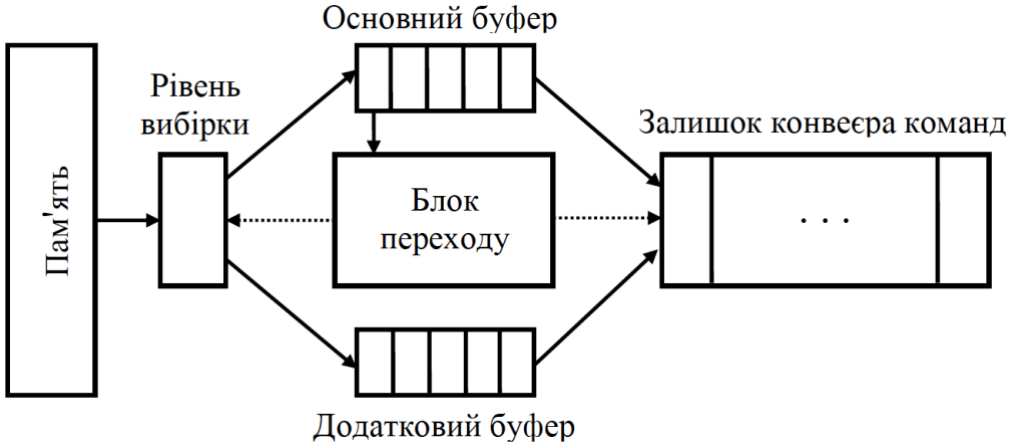


Рисунок 10.4 – Конвеєр з буферами передвибірки команд

Крім необхідності дублювання частини обладнання, у методу є ще один недолік. Так, якщо в потоці команд декілька команд УП проходять одна за одною або знаходяться достатньо близько, кількість можливих галужень збільшується і, відповідно, повинно бути збільшено і число буферів передвибірки.

Іншим рішенням проблеми переходів служить дублювання початкових рівнів конвеєра і створення тим самим двох паралельних потоків команд (рис. 10.5).

В одній гілці такого конвеєра послідовність вибірки і виконання команд відповідає випадку, коли умова переходу не виконалась, у другій гілці – випадку виконання цієї умови. Обидва потоки сходяться в точці, де результат перевірки

умови переходу стає очевидним. Подальше просування по конвеєру продовжує тільки «правильний» потік. Основний недолік методу полягає в тому, що на конвеєр або в потік може поступити нова команда УП до того, як буде ухвалено остаточне рішення по поточній команді переходу. Кожна така команда вимагає додаткового потоку. Незважаючи на це, стратегія дозволяє поліпшити продуктивність конвеєра.

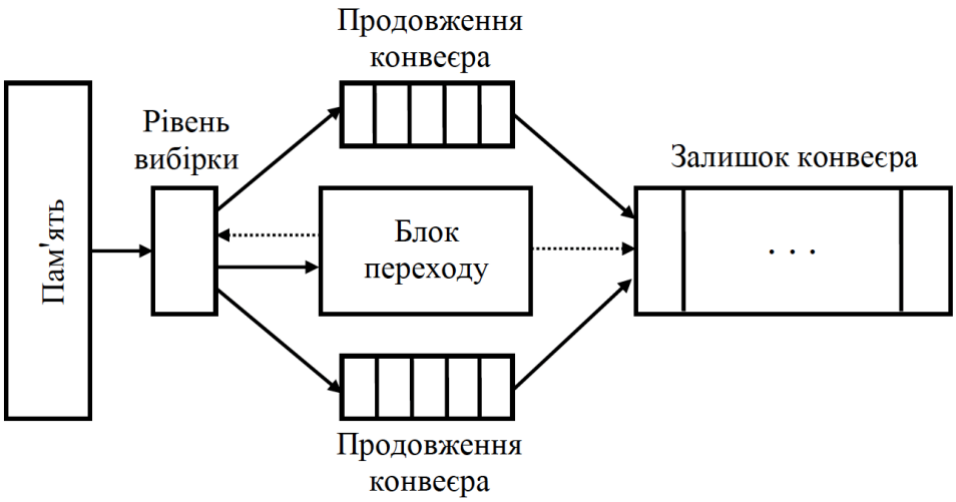


Рисунок 10.5 – Конвеєр з множинними потоками

Стратегія затриманого переходу передбачає продовження виконання команд, які йдуть за командою УП, незалежно від її результату. Природно, що це має сенс, коли наступні команди є «корисними», тобто такими, які все одно повинні бути колись виконані, незалежно від того, відбувається перехід або ні, і якщо команда переходу ніяк не впливає на результат їх виконання.

Для реалізації цієї ідеї на етапі компіляції програми після кожної команди переходу вставляється команда «Немає операції». Потім на стадії оптимізації програми проводяться спроби «перемішати» команди так, щоб по можливості більшу кількість команд «Немає операції» замінити «корисними» командами програми. Зрозуміло, заміщати команду «Немає операції» можна лише на таку, яка не впливає на умову виконуваного переходу, інакше отримана послі-довність команд не буде функціонально еквівалентною початковій. В оптимі-зованій програмі вдається замінити більше 20% команд «Немає операції» [25].

Передбачення переходів на сьогоднішній день розглядається як один з найбільш ефективних способів боротьби з конфліктами по управлінню. Ідея полягає в тому, що ще до моменту виконання команди умовного переходу або відразу ж після її надходження на конвеєр робиться припущення про найбільш вірогідний результат такої команди (перехід станеться або ні). Наступні команди надходять на конвеєр відповідно до передбачення. У разі помилкового передбачення конвеєр необхідно повернути до стану, з якого почалася вибірка «непотрібних» команд (очистити початкові рівні конвеєра), і перейти до завантаження, починаючи з «правильної» точки, що по ефекту еквівалентно призупиненню конвеєра. Ціна помилки може стати достатньо високою, але за правильних передбачень виграш буде більшим.

Серед механізмів передбачення переходів розрізняють статичне та динамічне передбачення переходів.

Статичне передбачення переходів здійснюється на основі апріорної інформації про програму, яка буде виконуватись. Передбачення робиться на етапі компіляції програми і в процесі обчислень вже не змінюється. Головна різниця між відомими механізмами статичного прогнозування полягає у виді інформації, яка використовується для передбачення, та її трактовці. Вихідна інформація може бути отримана двома шляхами: на основі аналізу коду програми або в результаті її «профілювання». Під «профілюванням» розуміють виконання програми при деякому еталонному наборі вихідних даних, яке супроводжується збором інформації про результати кожної команди умовного переходу. Вибір команд, які завершились переходом, фіксується в спеціальному біті коду операції. У ході виконання програми поведінка конвеєра команд визначається після вибірки команди по стану згаданого біта в коді операції.

У динамічних стратегіях рішення про найбільш вірогідний результат команди умовного переходу приймається в ході обчислень, на основі інформації про попередні переходи (історії переходів), яка збирається в процесі виконання програми. В цілому, динамічні стратегії у порівнянні зі статичними забезпечують більш високу точність передбачення.

Ідея динамічного передбачення переходів передбачає накопичення інформації про результат виконання попередніх команд УП. Історія переходів фіксується у формі таблиці, кожний елемент якої складається з m бітів. Потрібний елемент таблиці вказується за допомогою k- розрядної двійкової комбінації – шаблона. Цим пояснюється загальноприйнята назва таблиці передісторії переходів – таблиця історії для шаблонів (PHT, Pattern History Table). Як шаблони для доступу до PHT можуть бути взяті:

- адреса команди умовного переходу;
- регістр глобальної історії;
- регістр локальної історії;
- комбінація попередніх варіантів.

Схема, де для доступу до PHT вибрана адреса команди УП (вміст лічильника команд), дозволяє враховувати поведінку кожної конкретної команди УП. Кожній команді умовного переходу в PHT відповідає свій лічильник. Коли команда закінчується переходом, вміст лічильника збільшується на одиницю, а інакше – зменшується на одиницю. Як шаблон для пошуку в PHT служать молодші k розрядів вмісту лічильника команд.

Регістр глобальної історії (GHR, Global History Register) являє собою l - розрядний регістр зсуву. Після виконання чергової команди УП вміст регістру зсовується на один розряд, а у звільнену позицію заноситься одиниця (якщо був перехід) або нуль (якщо переходу не було). Отже, кодова комбінація в GHR відображає історію виконання останніх l команд умовного переходу. Як шаблон для пошуку в PHT служать молодші k розрядів GHR (частіше $l = k$).

Регістр локальної історії (LHR, Local History Register) по логіці роботи аналогічний регістру глобальної історії, але призначений для фіксації результатів однієї й тієї ж команди УП. У схемах передбачення з LHR присутня так звана таблиця локальної історії, яка являє собою масив регістрів локальної історії. Як і в схемі з адресою команди переходу, кожний лічильник в PHT фіксує історію результату тільки однієї команди УП, але базується на більш детальних знаннях, які відображають ще і послідовність результатів.

Розмір РНТ впливає на точність передбачень – зі збільшенням розміру РНТ точність передбачень зростає. Типова кількість елементів РНТ (елементарних лічильників) у різних процесорах варіюється від 256 до 4096.