Лекция8. Структурные конфликты и конфликты по данным. Методы их минимизации

В конвейерных системах все команды разделяются на несколько ступеней, каждая из которых выполняется отдельным блоком аппаратуры, причем эти блоки функционируют параллельно. Если на выполнение каждой ступени расходуется одинаковое время (один такт), то на выходе конвейера на каждом такте появляется результат очередной команды.

В реальных системах из-за возникновения конфликтов при работе конвейера среднее количество тактов $N_{\text{конв}}$ для выполнения команды, измеряемое в CPI (Cycles Per Instruction — циклов на инструкцию), определяется как:

$$N_{\text{конв}} = N_{\text{ид. конв}} + N_{\text{стр}} + N_{\text{дан}} + N_{\text{упр}},$$

где $N_{\text{конв}}$ — CPI конвейера; $N_{\text{ид. конв}}$ — CPI идеального конвейера; $N_{\text{стр}}$, $N_{\text{дан}}$, $N_{\text{упр}}$ — количество циклов на разрешение структурных конфликтов, конфликтов по данным и конфликтов по управлению соответственно.

CPI идеального конвейера есть не что иное как максимальная пропускная способность конвейера при идеальной загрузке. Для увеличения пропускной способности, а следовательно, уменьшения общего CPI конвейера необходимо минимизировать влияние конфликтов на работу конвейера.

Классификация методов минимизации структурных конфликтов и конфликтов по данным представлена на рис. 2.3.1.

Структурные конфликты

Совмещенный режим выполнения команд в общем случае требует конвейеризации функциональных устройств и дублирования ресурсов для разрешения всех возможных комбинаций команд в конвейере. Если какая-нибудь комбинация команд не может быть принята из-за конфликта по ресурсам, то говорят, что в процессоре имеется *структурный конфликт*. Типичным примером вычислительных

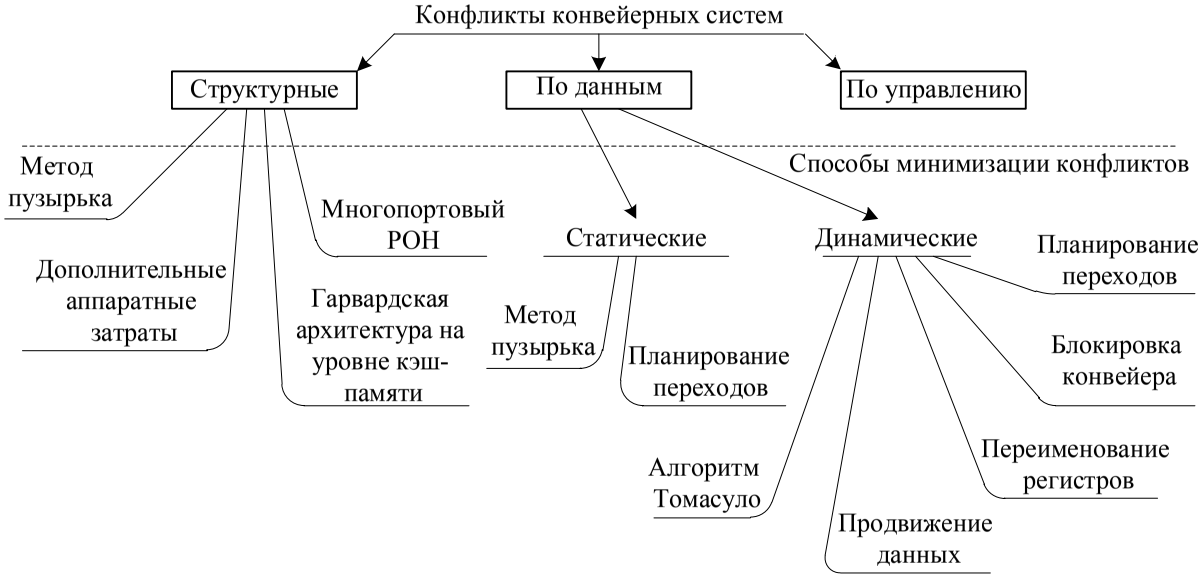

Рис. 2.3.1. Классификация основных методов минимизации структурных конфликтов и конфликтов по данным

систем, в которых возможно появление структурных конфликтов, являются машины с частично конвейеризированными функциональными устройствами. Время работы такого устройства может составлять несколько тактов синхронизации конвейера. В этом случае команды, которые используют данное функциональное устройство, не могут поступать в него в каждом такте.

Возникновение структурных конфликтов может быть связано с недостаточным дублированием некоторых ресурсов, что препятствует выполнению произвольной последовательности команд в конвейере без его приостановки. Например, процессор имеет только один порт записи в регистровый файл, а конвейеру при определенных обстоятельствах требуется выполнить две записи в регистровый файл в одном такте. Когда последовательность команд наталкивается на такой конфликт, конвейер приостанавливает выполнение одной из команд до тех пор, пока не станет доступным необходимое устройство.

Конвейеризация всех функциональных устройств может оказаться слишком дорогой. Процессоры, допускающие два обращения к памяти в одном такте, должны иметь удвоенную пропускную способность памяти, например, путем организации кэш-памяти раздельно для команд и данных. Аналогично, полностью конвейеризированное устройство деления с плавающей точкой требует большого количества вентилей. Поэтому разработчики следуют принципу: если структурные конфликты не будут возникать слишком часто, то не стоит платить за то, чтобы их обойти. В целом влияние структурного конфликта на производительность конвейера относительно невелико.

Конфликты по данным

Одним из факторов, влияющих на производительность конвейерных систем, являются межкомандные логические зависимости. Такие зависимости существенно ограничивают потенциальный параллелизм смежных операций, обеспечиваемый соответствующими аппаратными средствами обработки. Степень влияния этих зависимостей определяется как архитектурой процессора (в основном структурой устройства управления, конвейером команд и параметрами функциональных устройств), так и характеристиками программ.

Конфликты по данным возникают в том случае, когда применение конвейерной обработки может изменить порядок обращений

к операндам так, что он будет отличаться от порядка, который наблюдается при последовательном выполнении команд на неконвейерной машине.

Конфликты по данным возникают, когда имеет место зависимость между командами, и они расположены по отношению друг к другу достаточно близко, так что совмещение операций, происходящее при конвейеризации, может привести к изменению порядка обращения к операндам.

Известны три возможных типа конфликтов по данным в зависимости от порядка операций чтения и записи:

1) чтение-после-записи;
2) запись-после-чтения;
3) запись-после-записи.

Рассмотрим две команды K_1 и K_2, при этом команде K_1 предшествует K_2.

Чтение-после-записи (Read-After-Write — RAW) — команда K_2 пытается прочитать операнд-источник данных прежде чем его запишет команда K_1, так что команда K_2 может некорректно получить старое значение. Это наиболее распространенный тип конфликтов; способ их преодоления с помощью механизма «обходов» будет рассмотрен позже.

Запись-после-чтения (Write-After-Read — WAR) — команда K_2 пытается записать результат в приемник прежде чем он считывается командой K_1, так что команда K_1 может некорректно получить новое значение. Этот тип конфликтов, как правило, не возникает в системах с централизованным управлением потоком команд, обеспечивающим выполнение команд в порядке их поступления, так что последующая запись всегда выполняется позже, чем предшествующее считывание. Особенно часто конфликты такого типа возникают в системах, допускающих выполнение команд не в порядке их расположения в программном коде.

Запись-после-записи (Write-After-Write — WAW) — команда K_2 пытается записать операнд прежде чем будет записан результат команды K_1, т. е. записи заканчиваются в неверном порядке, оставляя в приемнике значение, записанное командой K_1, а не K_2. Этот тип конфликтов присутствует только в конвейерах, которые выполняют запись с многих ступеней (или позволяют команде выполняться даже в случае, когда предыдущая команда приостановлена).

В борьбе с конфликтами по данным выделяют два основных аспекта: своевременное обнаружение потенциального конфликта и его устранение.

Признаком возникновения конфликта по данным между двумя командами K_1 и K_2 служит невыполнение хотя бы одного из трех условий Бернстейна:

1) пересечение $W(K_1)$ с $W(K_2)$ пусто;
2) пересечение $W(K_1)$ с $R(K_2)$ пусто;
3) пересечение $R(K_1)$ с $W(K_2)$ пусто,

где $W(K_1)$ — набор выходных данных команды K_1; $R(K_1)$ — набор входных данных команды K_1. Если все условия выполняются, то операторы K_1 и K_2 могут быть выполнены одновременно разными исполнителями в параллельной вычислительной системе.

Для борьбы с конфликтами по данным применяются как статические, так и динамические методы.

Статические методы ориентированы на устранение самой возможности конфликтов еще на стадии составления или компиляции программы. Компилятор пытается создать такой код, чтобы между командами, склонными к конфликтам, находилось достаточное количество нейтральных в их отношении команд. Если это невозможно, то между конфликтующими командами компилятор вставляет необходимое количество команд типа «Нет операции» (NOP). Такой подход называют *методом пузырька*.

Многие типы приостановок конвейера могут происходить достаточно часто. Например, для оператора $A = B + C$ компилятор, скорее всего, сгенерирует последовательность команд, представленную на рис. 2.3.2.

MOV R1, B	ВК	ДК	ФА	ПО	ПР	РР				
MOV R2, C		ВК	ДК	ФА	ПО	ПР	РР			
ADD R2, R1			ВК	ДК	ПР	ПР	ОЖ	ВО	РР	
MOV A, R2				ВК	ДК	ФА	ПР	ПР	ОЖ	РР

Рис. 2.3.2. Конвейерное выполнение оператора $A = B + C$

Очевидно, что выполнение команды ADD должно быть приостановлено до тех пор, пока не станет доступным поступающий из памя-

ти операнд C. Дополнительной задержки выполнения команды MOV не произойдет в случае применения цепей обхода для пересылки результата операции АЛУ непосредственно в регистр данных памяти для последующей записи.

В данном простом примере компилятор никак не может улучшить ситуацию, однако в ряде более общих случаев он может реорганизовать последовательность команд так, чтобы избежать приостановок конвейера. Этот метод, называемый *планированием загрузки конвейера* (pipeline scheduling), или *планированием потока команд* (instruction scheduling), использовался начиная с 1960-х годов, но особую актуальность приобрел в 1980-х годах, когда конвейерные системы получили широкое распространение.

Рассмотрим фрагмент программы, составленной на языке высокого уровня и выполняющей арифметические операции над переменными. Положим для примера следующую последовательность вычислений:

$$A = B + C;$$
$$D = E - F.$$

(Напомним, что *переменная* — это поименованная область памяти, адрес которой можно использовать для осуществления доступа к данным.) Чтобы сгенерировать код, не вызывающий остановок конвейера, допустим, что задержка загрузки из памяти составляет один такт.

На рис. 2.3.3 приведен пример устранения конфликтов компилятором, символом $(*)$ отмечены команды, вызывающие блокировку.

В результате оптимизации устранены обе блокировки (командой загрузки MOV R_C, AdrC команды ADD R_B, R_C и командой MOV R_F, AdrF команды SUB R_E, R_F). Заметим, что использование разных регистров для первого и второго операторов было достаточно важным для реализации такого правильного планирования. В частности, если бы переменная E была загружена в тот же регистр, что B или C, то планирование не было бы корректным. В общем случае планирование конвейера может требовать увеличенного количества регистров. Такое увеличение может оказаться особенно существенным для машин, которые способны выдавать на выполнение несколько команд в одном такте.

Многие современные компиляторы используют метод планирования потока команд для улучшения производительности конвейе-

а	MOV R_B, AdrB	***б***	MOV R_B, AdrB
	MOV R_C, AdrC (*)		MOV R_C, AdrC
	ADD R_B, R_C		MOV R_E, AdrE
	MOV AdrA, R_B		ADD R_B, R_C
	MOV R_E, AdrE		MOV R_F, AdrF
	MOV R_F, AdrF (*)		MOV AdrA, R_B
	SUB R_E, R_F		SUB R_E, R_F
	MOV AdrD, R_E		MOV AdrD, R_E

Рис. 2.3.3. Пример устранения конфликтов компилятором: *а* — неоптимизированная последовательность команд; *б* — оптимизированная последовательность команд

ра. В простейшем алгоритме компилятор планирует распределение команд в одном и том же базовом блоке. Базовый блок представляет собой линейный участок последовательности программного кода, в котором отсутствуют команды перехода, за исключением начала и конца участка (переходы внутрь этого участка тоже должны отсутствовать). Планирование такой последовательности команд осуществляется достаточно просто, поскольку компилятор знает, что каждая команда в блоке будет выполняться, если выполняется первая из них, и можно просто построить граф зависимостей этих команд и упорядочить их так, чтобы минимизировать приостановки конвейера. Для простых конвейеров стратегия планирования на основе базовых блоков вполне удовлетворительна. Однако когда конвейеризация становится более интенсивной и действительные задержки конвейера растут, требуются более сложные алгоритмы планирования.

Существуют аппаратные методы, позволяющие изменить порядок выполнения команд программы так, чтобы минимизировать приостановки конвейера. Эти методы получили общее название *методов динамической оптимизации* (в англоязычной литературе в последнее время часто применяются также термины «out-of-order execution» — *«неупорядоченное выполнение»* и «out-of-order issue» — *«неупорядоченная выдача»*).

Основными средствами методов динамической оптимизации являются:

1) размещение схемы обнаружения конфликтов в возможно более низкой точке конвейера команд так, чтобы позволить команде продвигаться по конвейеру до тех пор, пока ей реально не потребуется операнд, являющийся также результатом логически более ранней, но еще не завершившейся команды. Альтернативный подход — это централизованное обнаружение конфликтов на одной из ранних ступеней конвейера;

2) буферизация команд, ожидающих разрешения конфликта, и выдача последующих логически не связанных команд в «обход» буфера; в этом случае команды могут выдаваться на выполнение не в том порядке, в котором они расположены в программе, однако аппаратура обнаружения и устранения конфликтов между логически связанными командами обеспечивает получение результатов согласно заданной программе;

3) соответствующая организация коммутирующих магистралей, реализующая засылку результата операции непосредственно в буфер, хранящий логически зависимую команду, задержанную из-за конфликта, или непосредственно на вход функционального устройства до того, как этот результат будет записан в регистровый файл или в память (short-circuiting, data forwarding или data bypassing — метод, который будет рассмотрен далее).

В примере, представленном на рис. 2.3.4, все команды, следующие за командой ADD, используют результат ее выполнения. Команда ADD записывает результат в регистр R1, а команда SUB читает это значение. Если не предпринять никаких мер для того, чтобы предотвратить этот конфликт (например, введением цикла ожидания, как и показано на рисунке), команда SUB прочитает неправильное значение и попытается его использовать. На самом деле значение, используемое командой SUB, является даже неопределенным (хотя логично предположить, что SUB всегда будет использовать значение R1, которое было присвоено какой-либо командой, предшествовавшей ADD, это не всегда так). Если произойдет прерывание между командами ADD и SUB, то команда ADD завершится, и значение R1 в этой точке будет соответствовать результату ADD. Такое непрогнозируемое поведение очевидно неприемлемо.

Проблема, поставленная в этом примере, может быть решена с помощью достаточно простой организации коммутирующих магистра-

ADD R1, R2	ВК	ДК	ПР	ПР	ВО	РР					
SUB R3, R1		ВК	ДК	ПР	ПР	ОЖ	ВО	РР			
AND R4, R1			ВК	ДК	ПР	ПР	ОЖ	ВО	РР		
OR R5, R1				ВК	ДК	ПР	ПР	ОЖ	ВО	РР	
XOR R6, R1					ВК	ДК	ПР	ПР	ОЖ	ВО	РР

Рис. 2.3.4. Пример последовательности команд в конвейере, имеющих зависимость по данным

лей, которая называется пересылкой или продвижением данных (data forwarding), «обходом» (data bypassing), иногда «закороткой» (short-circuiting). Эта организация состоит в следующем (рис. 2.3.5). Результат операции АЛУ с его выходного регистра всегда снова подается на входы АЛУ. Если аппаратура обнаруживает, что предыдущая операция АЛУ записывает результат в регистр, соответствующий источнику операнда для следующей операции АЛУ, то логические схемы управления выбирают в качестве входа для АЛУ результат, поступающий по цепи «обхода», а не значение, прочитанное из регистрового файла.

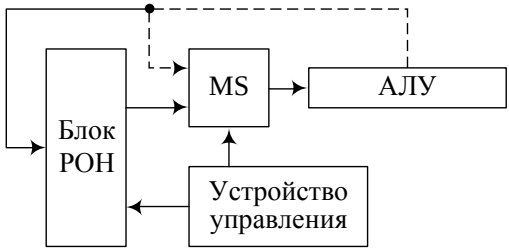

Рис. 2.3.5. Схема продвижения данных

Эта организация «обхода» может быть обобщена для того, чтобы включить передачу результата прямо в то функциональное устройство, которое в нем нуждается: результат с выхода одного устройства «пересылается» на вход другого, а не с выхода некоторого устройства только на его вход.

Не все потенциальные конфликты по данным могут обрабатываться с помощью организации «обхода». Рассмотрим последовательность команд, представленную на рис. 2.3.6.

MOV R1, (R6)	ВК	ДК	ФА	ПО	ПР	РР				
ADD R7, R1		ВК	ДК	ПР	ПР	ОЖ	ВО	РР		
SUB R5, R1			ВК	ДК	ПР	ПР	ОЖ	ОЖ	ВО	РР
AND R6, R1				ВК	ДК	ПР	ПР	ОЖ	ОЖ	ОЖ

Рис. 2.3.6. Пример последовательности команд с приостановкой конвейера

Эта последовательность отличается от последовательности подряд идущих команд АЛУ. Команда загрузки (MOV) регистра R1 из памяти по адресу, содержащемуся в регистре R6 (косвенная регистровая адресация), имеет задержку, которая не может быть устранена обычным обходным путем. В этом случае, чтобы обеспечить корректное выполнение программы, применяют *метод блокировки конвейера* (pipeline interlock). Аппаратура обнаруживает конфликт и приостанавливает конвейер до тех пор, пока он существует. Приостановка начинается с команды, которая хочет использовать данные в то время, когда предыдущая команда, результат которой является операндом для нашей, вырабатывает этот результат. Аппаратура вызывает приостановку конвейера или появление «пузырька» точно так же, как и в случае структурных конфликтов.

Еще одним динамическим методом минимизации конфликтов по данным является *метод переименования регистров* (register renaming), который служит для устранения конфликтов по данным типа запись-после-чтения (WAR) и запись-после-записи (WAW). Поясним суть метода на примере возникновения конфликта по данным типа запись-после-чтения (рис. 2.3.7).

$$
\begin{aligned}
K_1 &: \text{ADD R1, R2} \\
K_2 &: \text{MUL R2, R5} \\
&\cdots \\
K_2' &: \text{MUL R33, R5}
\end{aligned}
$$

Рис. 2.3.7. Пример переименования регистров

Команда K_1 использует регистр R2 в качестве операнда, следующая за ней команда использует этот же регистр в качестве приемника результата. Для разрешения конфликтной ситуации при одновременном обращении к регистру R2 результат выполнения команды K_2 записывается в R33 вместо R2, т. е. происходит переименование регистра. Отметим, что при выполнении последующих команд все обращения к переименованным регистрам должны перенаправляться до тех пор, пока это переименование остается актуальным.

При аппаратной реализации метода переименования регистров выделяются логические регистры, обращение к которым выполняется с помощью соответствующих полей команды (специальных тегов), и физические регистры, которые размещаются в регистровом файле процессора. Номера логических регистров динамически отображаются на номера физических регистров посредством таблиц отображения, которые обновляются после декодирования каждой команды. Новый результат записывается в новый физический регистр. Однако предыдущее значение каждого логического регистра сохраняется и может быть восстановлено в случае, если выполнение команды будет прервано из-за возникновения исключительной ситуации или неправильного предсказания направления условного перехода.

В процессе выполнения программы генерируется множество временных регистровых результатов. Эти временные значения записываются в регистровые файлы вместе с постоянными значениями. Временное значение становится новым постоянным значением, когда завершается выполнение команды (фиксируется ее результат). В свою очередь, завершение выполнения команды происходит, когда все предыдущие команды успешно завершились в заданном программой порядке.

Программист имеет дело только с логическими регистрами. Реализация физических регистров от него скрыта.

Метод переименования регистров упрощает контроль зависимостей по данным. В машине, которая способна выполнять команды не в порядке их расположения в программе, номера логических регистров могут стать двусмысленными, поскольку один и тот же регистр может быть назначен последовательно для хранения различных значений. Однако поскольку номера физических регистров уникально идентифицируют каждый результат, все неоднозначности устраняются.

Контрольные вопросы

1. Какие существуют типы конфликтов в конвейерных системах?
2. Расскажите об основных методах борьбы со структурными конфликтами.
3. Приведите классификацию конфликтов по данным.
4. Расскажите об основных методах борьбы с конфликтами по данным.
5. В чем суть метода планирования загрузки конвейера? Приведите пример.
6. В чем суть метода переименования регистров? Приведите пример.

Литература

1. Микропроцессоры. В 3-х кн. Кн. 1. Архитектура и проектирование микроЭВМ: учебник для втузов / ***Под ред. Л.Н. Преснухина***. — М.: Высшая школа, 1986. — 495 с.

2. ***Карпов В.Е.*** Введение в распараллеливание алгоритмов и программ // Компьютерные исследования и моделирование. — 2010. — Т. 1, № 3. — С. 231 – 272.

3. ***Цилькер Б.Я., Орлов С.А.*** Организация ЭВМ и систем: учебник для вузов. — СПб.: Питер, 2004. — 654 с.