7.1 Что такое конвейерная обработка

Разработчики архитектуры компьютеров издавна прибегали к методам проектирования, известным под общим названием "совмещение операций", при котором аппаратура компьютера в любой момент времени выполняет одновременно более одной базовой операции. Этот общий метод включает два понятия: параллелизм и конвейеризацию. Хотя у них много общего и их зачастую трудно различать на практике, эти термины отражают два совершенно различных подхода. При параллелизме совмещение операций достигается путем воспроизведения в нескольких копиях аппаратной структуры. Высокая производительность достигается за счет одновременной работы всех элементов структур, осуществляющих решение различных частей задачи.

Конвейеризация (или конвейерная обработка) в общем случае основана на разделении подлежащей исполнению функции на более мелкие части, называемые ступенями, и выделении для каждой из них отдельного блока аппаратуры. Так обработку любой машинной команды можно разделить на несколько этапов (несколько ступеней), организовав передачу данных от одного этапа к следующему. При этом конвейерную обработку можно использовать для совмещения этапов выполнения разных команд. Производительность при этом возрастает благодаря тому, что одновременно на различных ступенях конвейера выполняются несколько команд. Конвейерная обработка такого рода широко применяется во всех современных быстродействующих процессорах.

7.2 Уровни конвейеризации

•**Макроконвейер –** конвейеризация на уровне процессоров

•**Конвейер команд** - конвейеризация команд процессора

•**Конвейер арифметический** - конвейеризация на уровне выполнения команд процессора

7.3 Простейшая организация конвейера и оценка его производительности

Выполнение типичной команды можно разделить на следующие этапы:

•выборка команды - **IF** (по адресу, заданному счетчиком команд, из памяти извлекается команда);

•декодирование команды / выборка операндов из регистров - **ID**;

•выполнение операции / вычисление эффективного адреса памяти - **EX**;

•обращение к памяти - **MEM**;

•запись результата - **WB.**

Чтобы конвейеризовать выполнение команд можно просто разбить выполнение каждой команды на указанные выше этапы, отведя для выполнения каждого этапа один такт синхронизации, и начинать в каждом такте выполнение новой команды. Для хранения промежуточных результатов каждого этапа необходимо использовать регистровую память. Использование промежуточных регистров обеспечивают передачу данных и управляющих сигналов с одной ступени конвейера на следующую. Хотя общее время

выполнения одной команды в таком конвейере будет составлять пять тактов, в каждом такте аппаратура будет выполнять в совмещенном режиме пять различных команд.

Работу конвейера можно представить в виде временной диаграммы (рис. 7.3.1), на которой обычно изображаются выполняемые команды, номера тактов и этапы выполнения команд.

Номер команды	**Номер такта**								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Команда i	IF	ID	EX	MEM	WB				
Команда i+1		IF	ID	EX	MEM	WB			
Команда i+2			IF	ID	EX	MEM	WB		
Команда i+3				IF	ID	EX	MEM	WB	
Команда i+4					IF	ID	EX	MEM	WB

Рис. 7.3.1 Представление о работе конвейера

Конвейеризация увеличивает пропускную способность процессора (количество команд, завершающихся в единицу времени), **но она не сокращает время выполнения отдельной команды.** В действительности, она даже несколько увеличивает время выполнения каждой команды из-за накладных расходов, связанных с управлением регистровыми станциями. Однако увеличение пропускной способности означает, что программа будет выполняться быстрее по сравнению с простой не конвейерной схемой.

Тот факт, что время выполнения каждой команды в конвейере не уменьшается, накладывает некоторые ограничения на практическую длину конвейера. Кроме ограничений, связанных с задержкой конвейера, имеются также ограничения, возникающие в результате несбалансированности задержки на каждой его ступени и из-за накладных расходов на конвейеризацию. Частота синхронизации не может быть выше, а, следовательно, такт синхронизации не может быть меньше, чем время, необходимое для работы наиболее медленной ступени конвейера.

Накладные расходы на организацию конвейера возникают из-за

•задержки сигналов в конвейерных регистрах и

•из-за перекосов сигналов синхронизации.

Конвейерные регистры к длительности такта добавляют время установки и задержку распространения сигналов. В предельном случае длительность такта можно уменьшить до суммы накладных расходов и перекоса сигналов синхронизации, однако при этом в такте не останется времени для выполнения полезной работы по преобразованию информации.

В качестве примера рассмотрим не конвейерную машину с пятью этапами выполнения операций, которые имеют длительность 50, 50, 60, 50 и 50 нс соответственно (рис. 7.3.1).

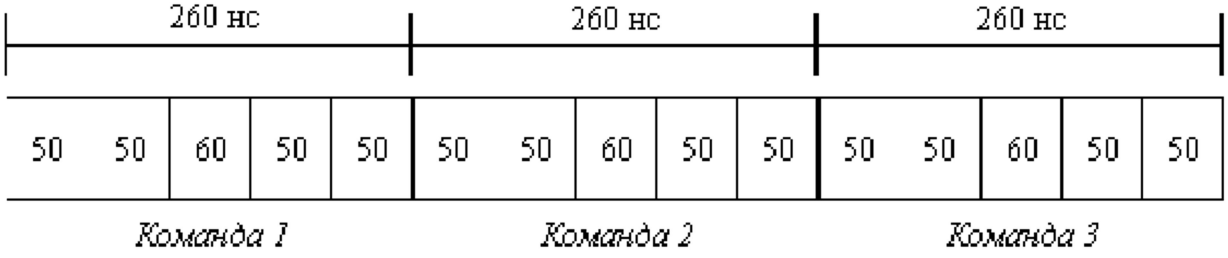

рис. 7.3.2 Неконвейерное выполнение команд

Пусть накладные расходы на организацию конвейерной обработки составляют 5 нс . Тогда среднее время выполнения команды в не конвейерной машине будет равно 260 нс. Если же используется конвейерная организация, длительность такта будет равна длительности самого медленного этапа обработки плюс накладные расходы, т.е. 65 нс. Это время соответствует среднему времени выполнения команды в конвейере. Таким образом, ускорение, полученное в результате конвейеризации, будет равно отношению 260/65=4.

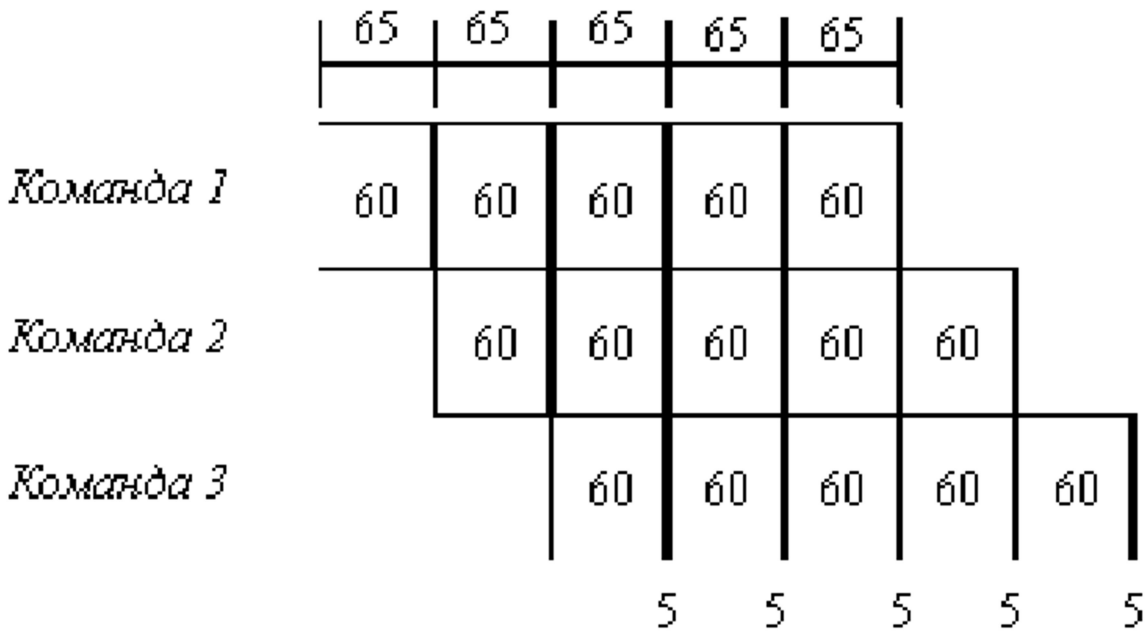

рис. 7.3.3 Конвейерное выполнение команд

Общая формула для расчета ускорения от конвейеризации операции имеет вид:

$$\xi = n*k/(n+k),$$

где k – число ступеней конвейера, а n – длина одной последовательности исходных данных (команд).

Конвейеризация эффективна только тогда, когда загрузка конвейера близка к полной, а скорость подачи новых команд и операндов соответствует максимальной производительности конвейера. Если произойдет задержка, то параллельно будет выполняться меньше операций и суммарная

производительность снизится. Такие задержки могут возникать в результате возникновения конфликтных ситуаций.

При реализации конвейерной обработки возникают ситуации, которые препятствуют выполнению очередной команды из потока команд в предназначенном для нее такте. Такие ситуации называются конфликтами.

Конфликты снижают реальную производительность конвейера, которая могла бы быть достигнута в идеальном случае.

Рассмотрим различные типы конфликтов, возникающие при выполнении команд в конвейере, и способы их разрешения.

Существуют три класса конфликтов:

1.**Структурные конфликты**, которые возникают из-за конфликтов по ресурсам, когда аппаратные средства не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.

2.**Конфликты по данным**, возникающие в случае, когда выполнение одной команды зависит от результата выполнения предыдущей команды.

3.**Конфликты по управлению**, которые возникают при конвейеризации команд переходов и других команд, которые изменяют значение счетчика команд.

Конфликты в конвейере приводят к необходимости приостановки выполнения команд (pipeline stall). Обычно в простейших конвейерах, если приостанавливается какая-либо команда, то все следующие за ней команды также приостанавливаются. Команды, предшествующие приостановленной, могут продолжать выполняться, но во время приостановки не выбирается ни одна новая команда.

7.4 Структурные конфликты и способы их минимизации

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 машине имеется структурный конфликт.**

Примеры архитектур, в которых возможно появление структурных конфликтов:

•Машины с не полностью конвейерными функциональными устройствами.

Время работы такого устройства может составлять несколько тактов синхронизации конвейера. В этом случае последовательные команды, которые используют данное функциональное устройство, не могут поступать в него в каждом такте.

•Машины с недостаточным дублированием некоторых ресурсов, что препятствует выполнению произвольной последовательности команд в конвейере без его приостановки. Например, машина может иметь только один порт записи в регистровый файл, но при определенных обстоятельствах конвейеру может потребоваться выполнить две записи в регистровый файл в одном такте. Это также приведет к структурному конфликту. Когда

последовательность команд наталкивается на такой конфликт, конвейер приостанавливает выполнение одной из команд до тех пор, пока не станет доступным требуемое устройство.

•Машины, в которых имеется единственный конвейер памяти для команд и данных. В этом случае, когда одна команда содержит обращение к памяти за данными, оно будет конфликтовать с выборкой более поздней команды из памяти. Чтобы разрешить эту ситуацию, можно просто приостановить конвейер на один такт, когда происходит обращение к памяти за данными. Подобная приостановка часто называются "конвейерным пузырем" (pipeline bubble) или просто пузырем, поскольку пузырь проходит по конвейеру, занимая место, но не выполняя никакой полезной работы.

Команда	**Номер такта**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Команда загрузки	IF	ID	EX	MEM	WB					
Команда 1		IF	ID	EX	MEM	WB				
Команда 2			IF	ID	EX	MEM	WB			
Команда 3				stall	IF	ID	EX	MEM	WB	
Команда 4						IF	ID	EX	MEM	WB
Команда 5							IF	ID	EX	MEM
Команда 6								IF	ID	EX

Рис. 7.4.1 Диаграмма работы конвейера при структурном конфликте

При всех прочих обстоятельствах, машина без структурных конфликтов будет всегда иметь более низкий **CPI (среднее число тактов на выдачу команды).**

Возникает вопрос: почему разработчики допускают наличие структурных конфликтов? Для этого имеются две причины: снижение стоимости и уменьшение задержки устройства. Конвейеризация всех функциональных устройств может оказаться слишком дорогой. Машины, допускающие два обращения к памяти в одном такте, должны иметь удвоенную пропускную способность памяти, например, путем организации раздельных кэшей для команд и данных. Аналогично, полностью конвейерное устройство деления с плавающей точкой требует огромного количества вентилей. Если структурные конфликты не будут возникать слишком часто, то может быть и не стоит платить за то, чтобы их обойти. Как правило, можно разработать не конвейерное, или не полностью конвейерное устройство, имеющее меньшую общую задержку, чем полностью конвейерное. Например, разработчики устройств с плавающей точкой компьютеров CDC7600 и MIPS R2010 предпочли иметь меньшую задержку выполнения операций вместо полной их конвейеризации.

7.5 Конфликты по данным, остановы конвейера и реализация механизма обходов

Одним из факторов, который оказывает существенное влияние на производительность конвейерных систем, являются меж командные логические зависимости. Такие зависимости в большой степени ограничивают потенциальный параллелизм смежных операций, обеспечиваемый соответствующими аппаратными средствами обработки. Степень влияния этих зависимостей определяется как архитектурой процессора (в основном, структурой 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 за операндами так, что этот порядок будет отличаться от порядка, который наблюдается при последовательном выполнении команд на неконвейерной машине. Рассмотрим конвейерное выполнение последовательности команд на рисунке 7.5.1.

ADD	R1,R2,R3	IF	ID	EX MEM WB
SUB	R4,R1,R5		IF	ID EX MEM WB
AND	R6,R1,R7			IF ID EX MEM WB
OR	R8,R1,R9			IF ID EX MEM WB
OR	R10,R1,R11			IF ID EX MEM WB

Рис. 7.5.1 а. Последовательность команд в конвейере и ускоренная пересылка данных (data forwarding, data bypassing, short circuiting)

ADD	R1,R2,R3		IF		ID		EX		MEM		WB								
				R						W									
SUB	R4,R1,R5			IF			ID		EX		MEM		WB						
							R					W							
AND	R6,R1,R7						IF		ID		EX		MEM		WB				
								R						W						
OR	R8,R1,R9							IF		ID		EX		MEM		WB			
									R						W				
XOR	R10,R1,R11								IF		ID		EX		MEM		WB		
										R						W				

Рис. 7.5.1, б. Совмещение чтения и записи регистров в одном такте

В этом примере 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, 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 разрешена с помощью достаточно простой аппаратной техники, которая называется **пересылкой** или **продвижением данных** (data forwarding),**обходом** (data bypassing), иногда **закороткой** (short-circuiting).

Эта аппаратура работает следующим образом:

•Результат операции АЛУ с его выходного регистра всегда снова подается назад 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 файл.

Эта техника "обходов" 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.

7.6 Классификация конфликтов по данным

Конфликт возникает везде, где 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. В нашем примере был проиллюстрирован конфликт, происходящий с регистровыми операндами, *но для пары команд возможно появление зависимостей при записи или чтении одной и той же ячейки памяти.* Однако, если все обращения к памяти выполняются в строгом порядке, то появление такого типа конфликтов предотвращается.

Известны три возможных конфликта по данным в зависимости от порядка операций чтения и записи. Рассмотрим две команды i и j, при этом i предшествует j.

Возможны следующие конфликты:

•**RAW** (**чтение после записи**) - j пытается прочитать операнд-источник данных прежде, чем i туда запишет. Таким образом, j может некорректно получить старое значение. Это наиболее общий тип конфликтов, способ их преодоления с помощью механизма "обходов" рассмотрен ранее.

•**WAR** (**запись после чтения**) - j пытается записать результат в приемник прежде, чем он считывается оттуда командой i, так что i может некорректно получить новое значение. Особенно часто конфликты такого рода могут возникать в системах, допускающих выполнение команд не в порядке их расположения в программном коде.

Этот тип конфликтов как правило не возникает в системах с централизованным управлением потоком команд, обеспечивающих выполнение команд в порядке их поступления, так как последующая запись всегда выполняется позже, чем

предшествующее считывание.

•**WAW** (**запись после записи**) - j пытается записать операнд прежде, чем будет записан результат команды i, т.е. записи заканчиваются в неверном порядке, оставляя в приемнике значение, записанное командой i, а не j. Этот тип конфликтов присутствует только в конвейерах, которые выполняют запись *со многих ступеней* (или позволяют команде выполняться даже в случае, когда предыдущая приостановлена).

7.7 Конфликты по данным, приводящие к приостановке конвейера

К сожалению не все потенциальные конфликты по данным могут обрабатываться с помощью механизма "обходов", рассмотренного ранее. Пусть имеем следующую последовательность команд (рис. 7.7.1):

Команда	IF	ID	EX	MEM	WB				
LW R1,32(R6)		IF	ID	EX	MEM	WB			
ADD R4,R1,R7			IF	ID	stall	EX	MEM	WB	
SUB R5,R1,R8				IF	stall	ID	EX	MEM	WB
AND R6,R1,R7					stall	IF	ID	EX	MEM	WB

Рис. 7.7.1 Последовательность команд с приостановкой конвейера

Этот случай отличается от последовательности подряд идущих команд АЛУ. Команда загрузки (LW) регистра R1 из памяти имеет задержку, которая не может быть устранена обычной "пересылкой". Вместо этого нам нужна дополнительная аппаратура, называемая **аппаратурой внутренних блокировок конвейера** (pipeline interlook), чтобы обеспечить корректное выполнение примера. Вообще такого рода аппаратура обнаруживает конфликты и приостанавливает конвейер до тех пор, пока существует конфликт. В этом случае эта аппаратура приостанавливает конвейер, начиная с команды, которая хочет использовать данные в то время, когда предыдущая команда, результат которой является операндом для нашей, вырабатывает этот результат. Эта аппаратура вызывает приостановку конвейера или появление "пузыря" точно также как и в случае структурных конфликтов.

7.8. Методика планирования компилятора для устранения конфликтов по данным

Пусть, например, имеется последовательность операторов:

a = b + c;

d = e - f;

Как сгенерировать код, не вызывающий остановок конвейера? Предполагается, что задержка загрузки из памяти составляет один такт. Можно сгенерировать следующую последовательность команд: (рис. 7.8.1):

Неоптимизированная последовательность команд	*Оптимизированная последовательность команд*
LW R_b,b	**LW R_b,b**
LW R_c,c	**LW R_c,c**
ADD R_a,R_b,R_c	**LW R_e,e**
SW a,R_a	**ADD R_a,R_b,R_c**
LW R_e,e	**LW R_f,f**
LW R_f,f	**SW a,R_a**
SUB R_d,R_e,R_f	**SUB R_d,R_e,R_f**
SW d,R_d	**SW d,R_d**

Рис.7.8.1 Пример устранения конфликтов компилятором

В результате устранены обе блокировки: командой **LW Rc,c** команды **ADD Ra,Rb,Rc** и командой **LW Rf,f** команды **SUB Rd,Re,Rf**. Имеется зависимость между операцией АЛУ и операцией записи в память, но структура конвейера допускает пересылку результата с помощью цепей "обхода". Заметим, что использование разных регистров для первого и второго операторов было достаточно важным для реализации такого правильного планирования. В частности, если переменная **e** была бы загружена в тот же самый регистр, что **b** или **c**, такое планирование не было бы корректным. В общем случае планирование конвейера может требовать *увеличенного количества регистров*. Такое увеличение может оказаться особенно существенным для машин, которые могут выдавать на выполнение несколько команд в одном такте.

Стратегия планирования на основе базовых блоков

Многие современные компиляторы используют технику планирования команд для улучшения производительности конвейера. В простейшем алгоритме компилятор просто 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.

К счастью, 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-orderexecution" - неупорядоченное выполнение и"out-of-order issue" - неупорядоченная выдача).

Основными средствами 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. Альтернативным подходом является 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 в соответствии с заданной программой.

3. Организация коммутирующих магистралей, обеспечивающая 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, data bypassing - методы).

4.Метод переименования регистров (register renaming). Получил свое название от широко применяющегося в компиляторах метода переименования - метода размещения данных, способствующего сокращению числа зависимостей и тем самым увеличению производительности при отображении необходимых исходной программе объектов (например, переменных) на аппаратные ресурсы (например, ячейки памяти и регистры).

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. Как уже отмечалось, номера логических регистров ставятся в соответствие номерам физических регистров. Отображение реализуется с помощью таблиц отображения, которые обновляются после декодирования каждой команды. Каждый новый результат записывается в физический регистр. Однако до тех пор, пока не завершится выполнение соответствующей команды, значение в этом физическом регистре рассматривается как временное.

Метод переименования регистров упрощает контроль зависимостей по данным. В машине, которая может 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. Но поскольку номера физических регистров уникально идентифицируют каждый результат, все неоднозначности устраняются.

7.9 Конфликты по управлению

Конфликты по управлению могут вызывать даже большие потери производительности конвейера, чем конфликты по данным. Когда выполняется команда условного перехода, она может либо изменить, либо не изменить значение счетчика команд.

Если команда условного перехода заменяет счетчик команд значением адреса, вычисленного в команде, то **переход называется выполняемым**; в противном случае, он называется **невыполняемым**.

Простейший метод работы с условными переходами заключается в **приостановке конвейера**, как только обнаружена команда условного перехода до тех пор, пока она не достигнет ступени конвейера, которая вычисляет новое значение счетчика команд (см. рис. 7.9.1).

Команды перехода	IF	ID	EX	MEM	WB								
Следующая команда		IF	stall	stall	IF	ID	EX	MEM	WB				
Следующая команда +1			Stall	stall	stall	IF	ID	EX	MEM	WB			
Следующая команда +2				stall	stall	stall	IF	ID	EX	MEM			
Следующая команда +3					stall	stall	stall	IF	ID	EX			
Следующая команда +4						stall	stall	stall	IF	ID			
Следующая команда +5							Stall	stall	stall	IF			

Рис. 7.9.1 Приостановка конвейера при выполнении команды условного перехода

Такие приостановки конвейера из-за конфликтов по управлению должны реализовываться иначе, чем приостановки из-за конфликтов по данным, поскольку выборка команды, следующей за командой условного перехода, должна быть выполнена как можно быстрее, как только мы узнаем окончательное направление команды условного перехода.

Например, если конвейер будет приостановлен на три такта на каждой команде условного перехода, то это может существенно отразиться на производительности машины. При частоте команд условного перехода в программах, равной 30% и идеальном CPI, равным 1, машина с приостановками условных переходов достигает примерно только половины ускорения, получаемого за счет конвейерной организации. Таким образом, **снижение потерь от условных переходов становится критическим вопросом.**

Число тактов, теряемых при приостановках из-за условных переходов, может быть уменьшено двумя способами:

1.**Обнаружением** является ли условный переход **выполняемым** или **невыполняемым на более ранних ступенях конвейера.**

2.Более ранним вычислением значения счетчика команд для выполняемого перехода (т.е. вычислением целевого адреса перехода).

Реализация этих условий требует модернизации исходной схемы конвейера, приведенного на рис. 7.9.1 Представление модернизированного конвейера приведено на рис. 7.10.1.

В некоторых машинах конфликты из-за условных переходов являются даже еще более дорогостоящими по количеству тактов, чем в нашем примере, поскольку время на оценку условия перехода и вычисление адреса перехода может быть даже большим. Например, машина с раздельными

ступенями декодирования и выборки команд, возможно, будет иметь задержку условного перехода (длительность конфликта по управлению), которая по крайней мере на один такт длиннее. Многие компьютеры VAX имеют задержки условных переходов в **четыре и более тактов**, а большие машины с глубокими конвейерами имеют потери по условным переходам, равные **шести или семи** тактам. В общем случае, **чем глубина конвейера больше, тем больше потери на командах условного перехода, исчисляемые в тактах.** Конечно эффект снижения относительной производительности при этом зависит от общего CPI машины. Машины с высоким CPI могут иметь условные переходы большей длительности, поскольку процент производительности машины, которая будет потеряна из-за условных переходов, меньше.

7.10 Снижение потерь на выполнение команд условного перехода

Имеется несколько методов сокращения приостановок конвейера, возникающих из-за задержек выполнения условных переходов.

Обсудим четыре простые схемы, используемые во время компиляции. В этих схемах прогнозирование направления перехода выполняется статически, т.е. прогнозируемое направление перехода фиксируется для каждой команды условного перехода на все время выполнения программы.

После обсуждения этих схем исследуем вопрос о правильности предсказания направления перехода компиляторами, поскольку все эти схемы основаны на такой технологии.

Далее, в следующей лекции рассмотрим более мощные схемы, используемые компиляторами (такие, например, как разворачивание циклов), которые уменьшают частоту команд условных переходов при реализации циклов, а также динамические, аппаратно реализованные схемы прогнозирования.

Метод выжидания

Простейшая схема обработки команд условного перехода заключается в замораживании или подавлении операций в конвейере, путем блокировки выполнения любой команды, следующей за командой условного перехода, до тех пор, пока не станет известным направление перехода. Рис. 7.9.1 отражал именно такой подход. Привлекательность такого решения заключается в его простоте.

Метод возврата

Более хорошая и не на много более сложная схема состоит в том, чтобы прогнозировать условный переход как невыполняемый. При этом аппаратура должна просто продолжать выполнение программы, как если бы условный переход вовсе не выполнялся. В этом случае необходимо позаботиться о том, чтобы не изменить состояние машины до тех пор, пока направление перехода не станет окончательно известным. В некоторых машинах эта схема с невыполняемыми по прогнозу условными переходами реализована путем продолжения выборки команд, как если бы условный переход был обычной командой. Поведение конвейера выглядит так, как будто ничего необычного

не происходит. Однако если условный переход на самом деле выполняется, то необходимо просто очистить конвейер от команд, выбранных вслед за командой условного перехода и заново повторить выборку команд (рис. 7.10.1).

Невыполняемый условный переход	IF	ID	EX MEM WB
Команда i+1		IF	ID EX MEM WB
Команда i+2			IF ID EX MEM WB
Команда i+3			IF ID EX MEM WB
Команда i+4			IF ID EX MEM WB
Выполняемый условный переход	IF	ID	EX MEM WB
Команда i+1		IF	ID EX MEM WB //удалена из конв.
Команда i+2			stall IF ID EX MEM WB
Команда i+3			Stall IF ID EX MEM WB
Команда i+4			stall IF ID EX MEM

Рис. 7.10.1. Диаграмма работы модернизированного конвейера

Альтернативная схема прогнозирует переход как выполняемый.

Как только команда условного перехода декодирована и вычислен целевой адрес перехода, предполагаем, что переход выполняемый, и осуществляем выборку команд и их выполнение, начиная с целевого адреса. Если мы не знаем целевой адрес перехода раньше, чем узнаем окончательное направление перехода, у этого подхода нет никаких преимуществ. Если бы условие перехода зависело от непосредственно предшествующей команды, то произошла бы приостановка конвейера из-за конфликта по данным для регистра, который является условием перехода, и мы бы узнали сначала целевой адрес. В таких случаях прогнозировать переход как выполняемый было бы выгодно.

Конвейер инструкций можно назвать «полностью конвейерным», если он может принимать новую инструкцию каждый машинный цикл. Иначе в конвейер должны быть вынужденно вставлены задержки, которые выравнивают конвейер, при этом ухудшая его производительность.

Преимущества:

1. Время цикла процессора уменьшается, таким образом увеличивая скорость обработки инструкций в большинстве случаев.
2. Некоторые комбинационные логические элементы, такие, как сумматоры или умножители, могут быть ускорены путём увеличения количества логических элементов. Использование конвейера может предотвратить ненужное наращивание количества элементов.

Недостатки:

1. Бесконвейерный процессор исполняет только одну инструкцию за раз. Это предотвращает задержки веток инструкций (фактически каждая ветка задерживается), и проблемы, связанные с последовательными инструкциями, которые исполняются параллельно. Следовательно, схема такого процессора проще, и он дешевле для изготовления.
2. Задержка инструкций в бесконвейерном процессоре слегка ниже, чем в конвейерном эквиваленте. Это происходит из-за того, что в конвейерный процессор должны быть добавлены дополнительные триггеры.
3. У бесконвейерного процессора скорость обработки инструкций стабильна. Производительность конвейерного процессора предсказать намного сложнее, и она может значительно различаться в разных программах.

Множество схем включают в себя конвейеры в 7, 10 или даже 20 уровней (как, например, в процессоре Pentium 4). Поздние ядра Pentium 4 с кодовыми именами Prescott и Cedar Mill (и их Pentium D-производные) имеют 31-уровневый конвейер.

Процессор Xelerator X10q имеет конвейер длиной более чем в тысячу шагов. Обратной стороной медали в данном случае является необходимость сбрасывать весь конвейер в случае, если ход программы изменился (например, по условному оператору). Эту проблему пытаются решать предсказатели переходов. Предсказание переходов само по себе может только усугубить ситуацию, если предсказание производится плохо. В некоторых областях применения, таких, как вычисления на суперкомпьютерах, программы специально пишутся так, чтобы как можно реже использовать условные операторы, поэтому очень длинные конвейеры весьма позитивно скажутся на общей скорости вычислений, так как длинные конвейеры проектируются так, чтобы уменьшить CPI (количество тактов на инструкцию).

Если ветвление происходит постоянно, перестановка машинных инструкций таким образом, чтобы те инструкции, которые, скорее всего, понадобятся, были размещены в конвейере, может значительно уменьшить

потери скорости по сравнению с необходимостью каждый раз полностью сбрасывать конвейер.

Высокая пропускная способность конвейеров приводит к уменьшению производительности в случае, если в исполняемом коде содержится много условных переходов: процессор не знает, откуда читать следующую инструкцию, и поэтому вынужден ждать, когда закончится инструкция условного перехода, оставляя за ней пустой конвейер. После того, как ветка будет пройдена и станет известно, куда процессору необходимо переходить в дальнейшем, следующая инструкция должна будет пройти весь путь через конвейер перед тем, как результат становится доступным и процессор снова «работает». В крайнем случае, производительность конвейерного процессора может теоретически упасть до производительности бесконвейерного, или даже быть хуже за счёт того, что будет занят только один уровень конвейера и между уровнями присутствует небольшая задержка.

Если процессор оснащён конвейером, код, читаемый из памяти, не выполняется сразу, а помещается в очередь (очередь предвыборки, prefetch input queue). Если код, содержащийся в памяти, будет изменён, код, содержащийся очереди конвейера, останется прежним. Также не изменятся инструкции, находящиеся в кэше инструкций. Стоит учитывать, что данная проблема характерна только для самомодифицирующихся программ и упаковщиков исполняемых файлов.