

## Лекции 9-10

### Тема: Конвейерная организация

- 1. Что такое конвейерная обработка
- 2. Уровни конвейеризации
- 3. Простейшая организация конвейера и оценка его производительности
- 4. Структурные конфликты и способы их минимизации
- 5. Конфликты по данным, останова конвейера и реализация механизма обходов
  - 5.1. Классификация конфликтов по данным
  - 5.2. Конфликты по данным, приводящие к приостановке конвейера
  - 5.3. Методика планирования компилятора для устранения конфликтов по данным
- 6. Конфликты по управлению. Сокращение потерь на выполнение команд перехода и минимизация конфликтов по управлению
  - 6.1. Снижение потерь на выполнение команд условного перехода
- 7. Проблемы реализации точного прерывания в конвейере
- 8. Обработка многотактных операций и механизмы обходов в длинных конвейерах
- 9. Конфликты и ускоренные пересылки в длинных конвейерах

#### **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.**

## 2. Уровни конвейеризации

- **Макроконвейер** – конвейеризация на уровне процессоров
- **Конвейер команд** - конвейеризация команд процессора
- **Конвейер арифметический** - конвейеризация на уровне выполнения команд процессора

## 3. Простейшая организация конвейера и оценка его производительности

Для иллюстрации основных принципов построения процессоров мы будем использовать простейшую архитектуру, содержащую 32 целочисленных регистра общего назначения ( $R_0, \dots, R_{31}$ ), 32 регистра плавающей точки ( $F_0, \dots, F_{31}$ ) и счетчик команд PC. Будем считать, что набор команд нашего процессора включает типичные арифметические и логические операции, операции с плавающей точкой, операции пересылки данных, операции управления потоком команд и системные операции. В арифметических командах используется трехадресный формат, типичный для RISC-процессоров, а для обращения к памяти используются операции загрузки и записи содержимого регистров в память.

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**;
- запоминание 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 станцию (память). Промежуточные регистровые станции 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 условно представить в виде сдвинутых во времени схем процессора (рис. 1). Этот рисунок хорошо отражает совмещение во времени выполнения различных этапов команд. Однако чаще для представления работы

конвейера используются временные диаграммы (рис. 2), на которых обычно изображаются выполняемые команды, номера тактов и этапы выполнения команд.

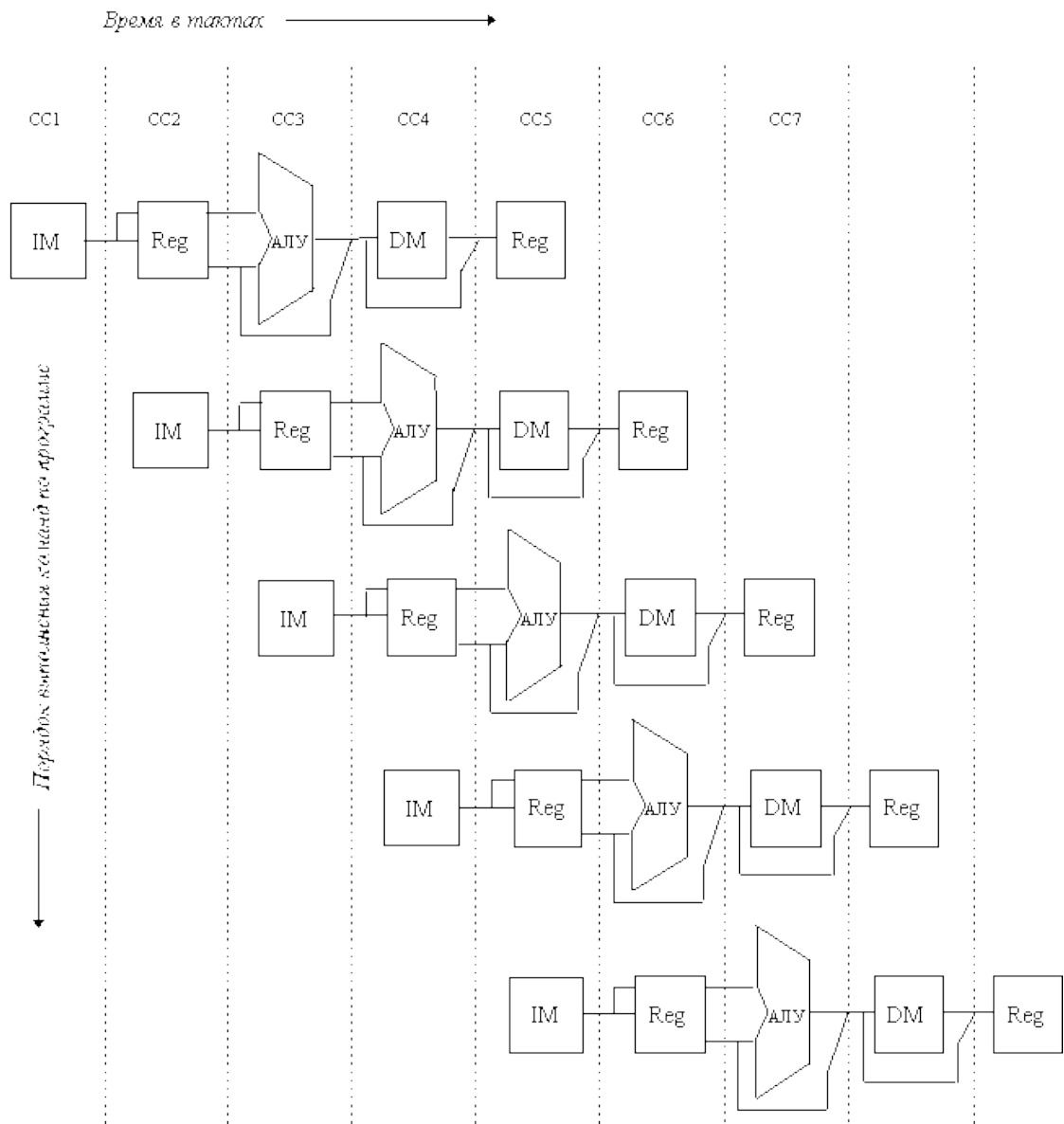


Рис. 1. Представление о работе конвейера

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. 2. 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 (защелках) 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. 3). 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  $260/65=4$ :

**Среднее время выполнения команды в неконвейерном режиме = 260**

**Среднее время выполнения команды в конвейерном режиме = 65**

**Ускорение от конвейеризации команды = Ср. время выполнения команды в неконвейерном режиме / Ср. время выполнения команды в конвейерном режиме**

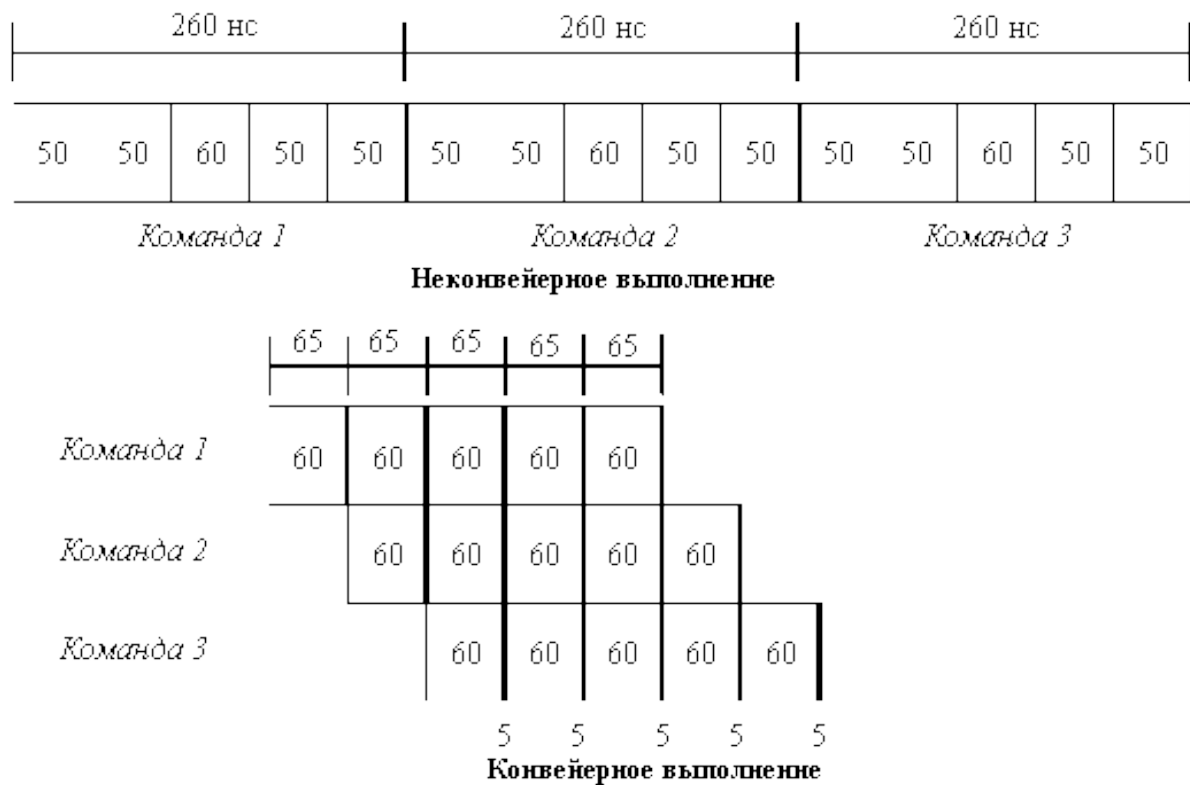
Ускорение от конвейеризации =  $260/65=4$

Общая формула для арифметического конвейера

**Ускорения от конвейеризации операции имеет вид:**

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

где  $k$  – число ступеней конвейера, а  $n$  – длина одной последовательности исходных данных (команд) (вывод см. на примере сложении двух векторов длины  $n$  с плавающей запятой)



**Рис. 3. Эффект конвейеризации при выполнении 3-х команд - четырехкратное ускорение**

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.

#### 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* (рис. 5). 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 называется "конвейерным пузырем" (pipeline bubble) или просто пузырем, поскольку пузырь проходит по конвейеру, занимая место, но не выполняя никакой полезной работы.

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 отдельных кэшей для команд и данных. Аналогично, полностью конвейерное устройство деления с плавающей точкой требует огромного количества вентилях. 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.

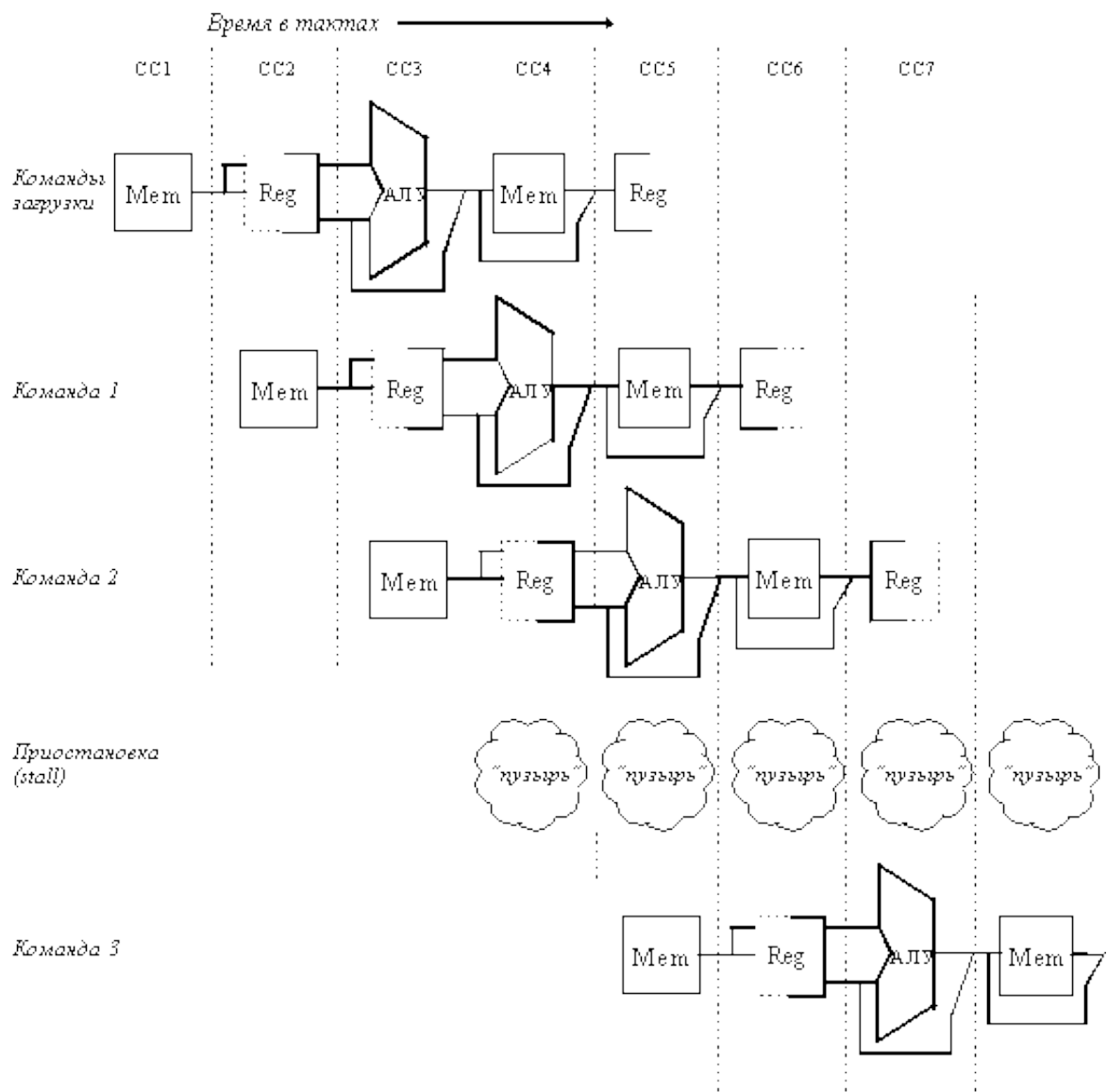


Рис. 5, а. Пример структурного конфликта при реализации памяти с одним портом

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. 5, б. Диаграмма работы конвейера при структурном конфликте

## 5. Конфликты по данным, остановки конвейера и реализация механизма обходов

Одним из факторов, который оказывает существенное влияние на производительность конвейерных систем, являются межкомандные 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 6.

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. 6, 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. 6, 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 файла (рис. 7).

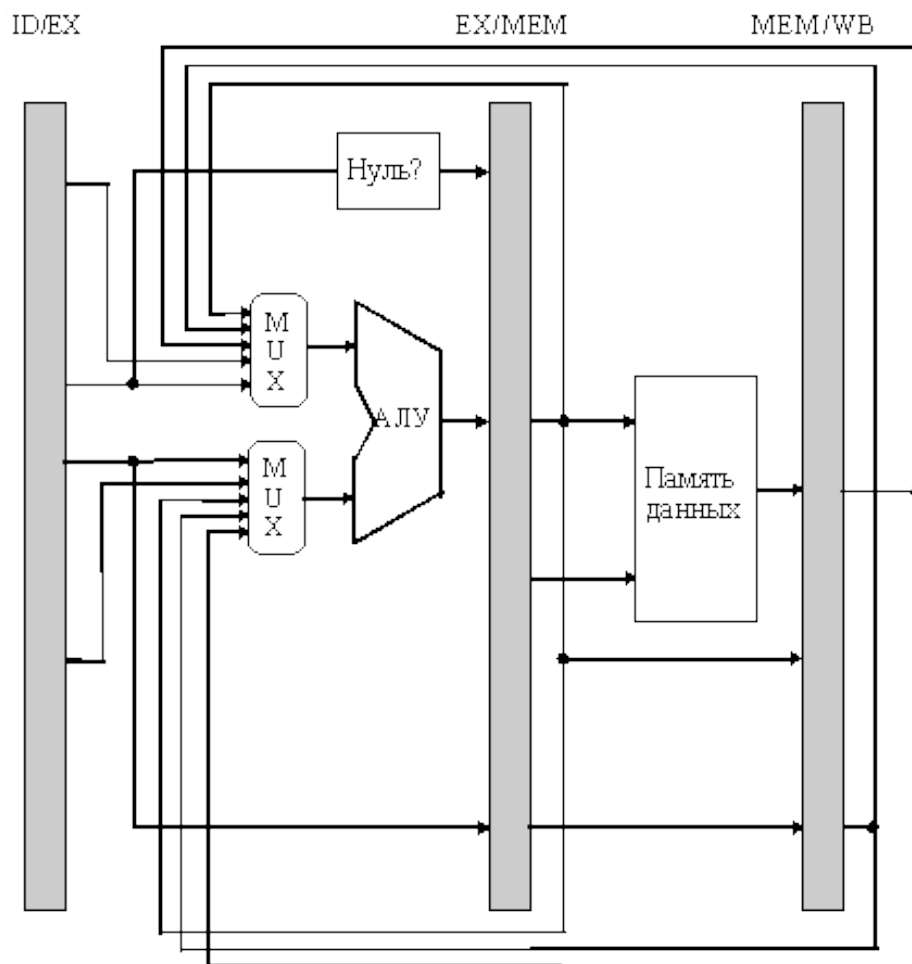


Рис. 7. АЛУ с цепями обхода и ускоренной пересылки

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.

## 5.1. 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. 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. Особенно часто конфликты такого рода могут возникать в системах, допускающих выполнение команд не в порядке их расположения в программном коде.
- **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).

## 5.2. 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. 8):

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. 8. 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 interlock), 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.

### 5.3. Методика планирования компилятора для устранения конфликтов по данным

Многие типы приостановок конвейера могут происходить достаточно часто. Например, для оператора  $A = B + C$  компилятор скорее всего сгенерирует следующую последовательность команд (рис.9):

LW R1,B	IF	ID	EX	MEM WB				
LW R2,C		IF	ID	EX	MEM	WB		
ADD R3,R1,R2			IF	ID	stall	EX	MEM	WB
SW A,R3				IF	stall	ID	EX	MEM WB

Рис. 9. Конвейерное выполнение оператора  $A = B + C$

Очевидно, выполнение команды ADD должно быть приостановлено до тех пор, пока не станет доступным поступающий из памяти операнд C. Дополнительной задержки выполнения команды SW не произойдет в случае применения цепей обхода для пересылки результата операции АЛУ непосредственно в регистр данных памяти для последующей записи.

Для данного простого примера компилятор никак не может улучшить ситуацию, однако в ряде более общих случаев он может реорганизовать последовательность команд так, чтобы избежать приостановок конвейера. Эта техника, называемая **планированием загрузки конвейера** (pipeline scheduling) или **планированием потока команд** (instruction scheduling), использовалась начиная с 60-х годов и стала особой областью интереса в 80-х годах, когда конвейерные машины получили наибольшее распространение.

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. 10):

Неоптимизированная последовательность команд	Оптимизированная последовательность команд
LW R <sub>b</sub> ,b	LW R <sub>b</sub> ,b
LW R <sub>c</sub> ,c	LW R <sub>c</sub> ,c
ADD R <sub>a</sub> ,R <sub>b</sub> ,R <sub>c</sub>	LW R <sub>e</sub> ,e
SW a,R <sub>a</sub>	ADD R <sub>a</sub> ,R <sub>b</sub> ,R <sub>c</sub>
LW R <sub>e</sub> ,e	LW R <sub>f</sub> ,f
LW R <sub>f</sub> ,f	SW a,R <sub>a</sub>
SUB R <sub>d</sub> ,R <sub>e</sub> ,R <sub>f</sub>	SUB R <sub>d</sub> ,R <sub>e</sub> ,R <sub>f</sub>

SW d,R <sub>d</sub>	SW d,R <sub>d</sub>
---------------------	---------------------

*Рис.10. Пример устранения конфликтов компилятором*

В результате устранены обе блокировки: командой **LW R<sub>e</sub>,c** команды **ADD R<sub>a</sub>,R<sub>b</sub>,R<sub>c</sub>** и командой **LW R<sub>f</sub>,f** команды **SUB R<sub>d</sub>,R<sub>e</sub>,R<sub>f</sub>**. 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.

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-order execution" - 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).

При аппаратной реализации метода переименования регистров выделяются логические регистры, обращение к которым выполняется с помощью соответствующих полей команды, и физические регистры, которые размещаются в аппаратном регистровом файле процессора. Номера логических регистров динамически отображаются на номера физических регистров посредством таблиц отображения, которые обновляются после декодирования каждой команды. Каждый новый результат записывается в новый физический регистр. Однако предыдущее значение каждого логического регистра сохраняется и может быть восстановлено в случае, если выполнение команды должно быть 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. 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.

## 6. Конфликты по управлению. Сокращение потерь на выполнение команд перехода и минимизация конфликтов 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. 11).

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.

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. 11. Приостановка конвейера при выполнении 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. 11. Представление модернизированного конвейера приведено на рис. 12.

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 отдельными 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.

### 6.1. 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. 11 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. 12).

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
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. 12. Диаграмма работы модернизированного конвейера

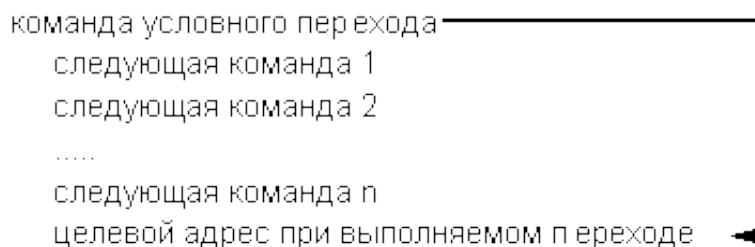
### Альтернативная схема прогнозирует переход как выполняемый.

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.

Дополнительно в некоторых машинах (особенно в машинах с устанавливаемыми по умолчанию кодами условий или более мощным, а потому и более медленным набором условий перехода) целевой адрес перехода известен раньше окончательного направления перехода, и схема прогноза перехода как выполняемого имеет смысл.

### Задержанные переходы

Четвертая схема, которая используется в некоторых машинах, называется "задержанным переходом". В задержанном переходе такт выполнения с задержкой перехода длиной  $n$  есть:



Команды с 1 по  $n$  находятся в слотах (временных интервалах) задержанного перехода. Задача программного обеспечения заключается в том, чтобы сделать команды, следующие за командой перехода, действительными и полезными. Аппаратура гарантирует реальное выполнение этих команд перед выполнением собственно перехода. Здесь используются несколько *приемов оптимизации*.

На рис. 13,а показаны три случая, при которых может планироваться задержанный переход. В верхней части рисунка для каждого случая показана исходная последовательность команд, а в нижней части - последовательность команд, полученная в результате планирования. В случае (а) слот задержки заполняется независимой командой, находящейся перед командой условного перехода. Это наилучший выбор. Стратегии (b) и (c) используются, если применение стратегии (a) невозможно.

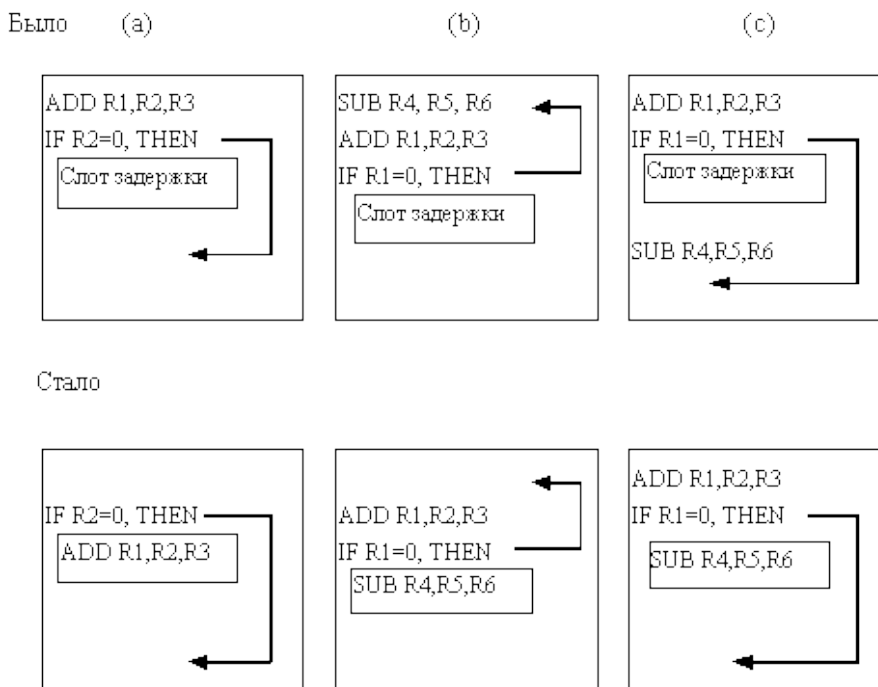


Рис. 13,а. Требования к переставляемым командам при планировании задержанного перехода

В последовательностях команд для случаев (b) и (c) использование содержимого регистра R1 в качестве условия перехода препятствует перемещению

команды ADD (которая записывает результат в регистр R1) за команду перехода. В случае (b) слот задержки заполняется командой, находящейся по целевому адресу команды перехода. Обычно такую команду приходится копировать, поскольку к ней возможны обращения и из других частей программы. Стратегии (b) отдается предпочтение, когда с высокой вероятностью переход является выполняемым, например, если это переход на начало цикла.

Наконец, слот задержки может заполняться командой, находящейся между командой невыполняемого перехода и командой, находящейся по целевому адресу, как в случае (c). Чтобы подобная оптимизация была законной, необходимо, чтобы можно было все-таки выполнить команду SUB, если переход пойдет не по прогнозируемому направлению. При этом предполагаем, что команда SUB выполнит ненужную работу, но вся программа при этом будет выполняться корректно. Это, например, может быть в случае, если регистр R4 используется только для временного хранения промежуточных результатов вычислений, когда переход выполняется не по прогнозируемому направлению.

В таблице на рис. 13,б указываются различные ограничения для всех приведенных выше схем планирования условных переходов, а также ситуации, в которых они дают выигрыш. Компилятор должен соблюдать требования при подборе подходящей команды для заполнения слота задержки. Если такой команды не находится, слот задержки должен заполняться пустой операцией.

Рассматриваемый случай	Требования	Когда увеличивается производительность
(a)	Команда условного перехода не должна зависеть от переставляемой команды	Всегда
(b)	Выполнение переставляемой команды должно быть корректным, даже если переход не выполняется Может потребоваться копирование команды	Когда <b>переход выполняется</b> . Может увеличивать размер программы в случае копирования команды
(c)	Выполнение переставляемой команды должно быть корректным, даже если переход выполняется	Когда <b>переход не выполняется</b>

Рис.13, б

Планирование задержанных переходов осложняется:

1. наличием ограничений на команды, размещение которых планируется в слотах задержки и
2. необходимостью предсказывать во время компиляции, будет ли условный переход выполняемым или нет.

Имеются небольшие дополнительные затраты аппаратуры на реализацию задержанных переходов. Из-за задержанного эффекта условных переходов, для

корректного восстановления состояния в случае появления прерывания нужны несколько счетчиков команд (один плюс длина задержки).

## 7. Проблемы реализации точного прерывания в конвейере

*Обработка прерываний в конвейерной машине оказывается более сложной из-за того, что совмещенное выполнение команд затрудняет определение возможности безопасного изменения состояния машины произвольной командой.* В конвейерной машине команда выполняется по этапам, и ее завершение осуществляется через несколько тактов после выдачи для выполнения. Еще в процессе выполнения отдельных этапов команда может изменить состояние машины. Тем временем возникшее прерывание может вынудить машину прервать выполнение еще не завершенных команд.

Как и в не конвейерных машинах **двумя основными проблемами** при реализации прерываний **являются**:

- (1) прерывания, которые возникают в процессе выполнения некоторой команды;
- (2) механизм возврата из прерывания для продолжения выполнения программы.

Например, для нашего простейшего конвейера прерывание по отсутствию страницы виртуальной памяти при выборке данных не может произойти до этапа выборки из памяти (MEM). В момент возникновения этого прерывания в процессе обработки уже будут находиться несколько команд. Поскольку подобное прерывание должно обеспечить возврат для продолжения программы и требует переключения на другой процесс (операционную систему), необходимо надежно **очистить конвейер и сохранить состояние машины** таким, чтобы повторное выполнение команды после возврата из прерывания осуществлялось при корректном состоянии машины. Обычно это реализуется путем сохранения адреса команды (PC), вызвавшей прерывание. Если выбранная после возврата из прерывания команда не является командой перехода, то сохраняется обычная последовательность выборки и обработки команд в конвейере. Если же это команда перехода, то необходимо оценить условие перехода и в зависимости от выбранного направления начать выборку либо по целевому адресу команды перехода, либо следующей за переходом команды.

Когда происходит прерывание, для корректного сохранения состояния машины необходимо выполнить следующие шаги:

1. В последовательность команд, поступающих на обработку в конвейер, *принудительно вставить команду перехода на прерывание.*
2. Пока выполняется команда перехода на прерывание, *погасить все требования записи*, выставленные командой, вызвавшей прерывание, а также всеми следующими за ней в конвейере командами. Эти действия позволяют предотвратить все изменения состояния машины командами, которые не завершились к моменту начала обработки прерывания.
3. После передачи управления подпрограмме обработки прерываний операционной системы, она немедленно должна сохранить значение адреса команды (PC), вызвавшей прерывание. Это значение будет использоваться позже для организации возврата из прерывания.

Если используются механизмы задержанных переходов, состояние машины уже невозможно восстановить с помощью одного счетчика команд, поскольку в процессе восстановления команды в конвейере могут оказаться вовсе не последовательными. В частности, если команда, вызвавшая прерывание, находилась в слоте задержки перехода, и переход был выполненным, то необходимо заново повторить выполнение команд из слота задержки плюс команду, находящуюся по целевому адресу команды перехода. Сама команда перехода уже выполнялась и ее повторения не требуется. При этом адреса команд из слота задержки перехода и целевой адрес команды перехода естественно не являются последовательными. Поэтому **необходимо сохранять и восстанавливать несколько счетчиков команд**, число которых на единицу превышает длину слота задержки. Это выполняется на третьем шаге обработки прерывания.

После обработки прерывания специальные команды осуществляют возврат из прерывания путем перезагрузки счетчиков команд и инициализации потока команд. *Если конвейер может быть остановлен так, что команды, непосредственно предшествовавшие вызвавшей прерывание команде, завершаются, а следовавшие за ней могут быть заново запущены для выполнения, то говорят, что конвейер обеспечивает точное прерывание.* В идеале команда, вызывающая прерывание, не должна менять состояние машины, и для корректной обработки некоторых типов прерываний требуется, чтобы команда, вызывающая прерывание, не имела никаких побочных эффектов. Для других типов прерываний, например, для прерываний по исключительным ситуациям плавающей точки, вызывающая прерывание команда на некоторых машинах записывает свои результаты еще до того момента, когда прерывание может быть обработано. В этих случаях аппаратура должна быть готовой для восстановления операндов-источников, даже если местоположение результата команды совпадает с местоположением одного из операндов-источников.

Поддержка точных прерываний во многих системах является обязательным требованием, а в некоторых системах была бы весьма желательной, поскольку она упрощает интерфейс операционной системы. Как минимум в машинах со страничной организацией памяти или с реализацией арифметической обработки в соответствии со стандартом IEEE средства обработки прерываний должны обеспечивать точное прерывание либо целиком с помощью аппаратуры, либо с помощью некоторой поддержки со стороны программных средств.

Необходимость реализации в машине точных прерываний иногда оспаривается из-за некоторых проблем, которые осложняют повторный запуск команд. Повторный запуск сложен из-за того, что команды могут изменить состояние машины еще до того, как они гарантировано завершают свое выполнение (иногда гарантированное завершение команды называется фиксацией команды или фиксацией результатов выполнения команды). Поскольку команды в конвейере могут быть взаимозависимыми, блокировка изменения состояния машины может оказаться непрактичной, если конвейер продолжает работать. Таким образом, по мере увеличения степени конвейеризации машины возникает необходимость отката любого изменения состояния, выполненного до фиксации

команды. К счастью, в простых конвейерах, подобных рассмотренному, эти проблемы не возникают. На рис. 14 показаны ступени рассмотренного конвейера и причины прерываний, которые могут возникнуть на соответствующих ступенях при выполнении команд.

Ступень конвейера	Причина прерывания
IF	Ошибка при обращении к странице памяти при выборке команды; невыровненное обращение к памяти; нарушение защиты памяти
ID	Неопределенный или запрещенный код операции
EX	Арифметическое прерывание
MEM	Ошибка при обращении к странице памяти при выборке данных; невыровненное обращение к памяти; нарушение защиты памяти
WB	Отсутствует

*Рис. 14. Причины прерываний в простейшем конвейере*

## 8. Обработка многотактных операций и механизмы обходов в длинных конвейерах

В рассмотренном нами конвейере стадия выполнения команды (EX) составляла всего один такт, что вполне приемлемо для целочисленных операций. Однако для большинства операций плавающей точки было бы непрактично требовать, чтобы все они выполнялись за один или даже за два такта. Это привело бы к существенному увеличению такта синхронизации конвейера, либо к сверхмерному увеличению количества оборудования (объема логических схем) для реализации устройств плавающей точки. Проще всего представить, что команды плавающей точки используют тот же самый конвейер, что и целочисленные команды, но с двумя важными изменениями:

**Во-первых**, такт EX может повторяться многократно столько раз, сколько необходимо для выполнения операции.

**Во-вторых**, в процессоре может быть несколько функциональных устройств, реализующих операции плавающей точки. При этом могут возникать приостановки конвейера, если выданная для выполнения команда, либо вызывает структурный конфликт по функциональному устройству, которое она использует, либо существует конфликт по данным.

Конвейер с такими дополнениями называется **длинным конвейером**.

Допустим, что в нашей реализации процессора имеются четыре отдельных функциональных устройства (см.рис.15):

1. Основное целочисленное устройство.
2. Устройство умножения целочисленных операндов и операндов с плавающей точкой.
3. Устройство сложения с плавающей точкой.
4. Устройство деления целочисленных операндов и операндов с плавающей точкой.

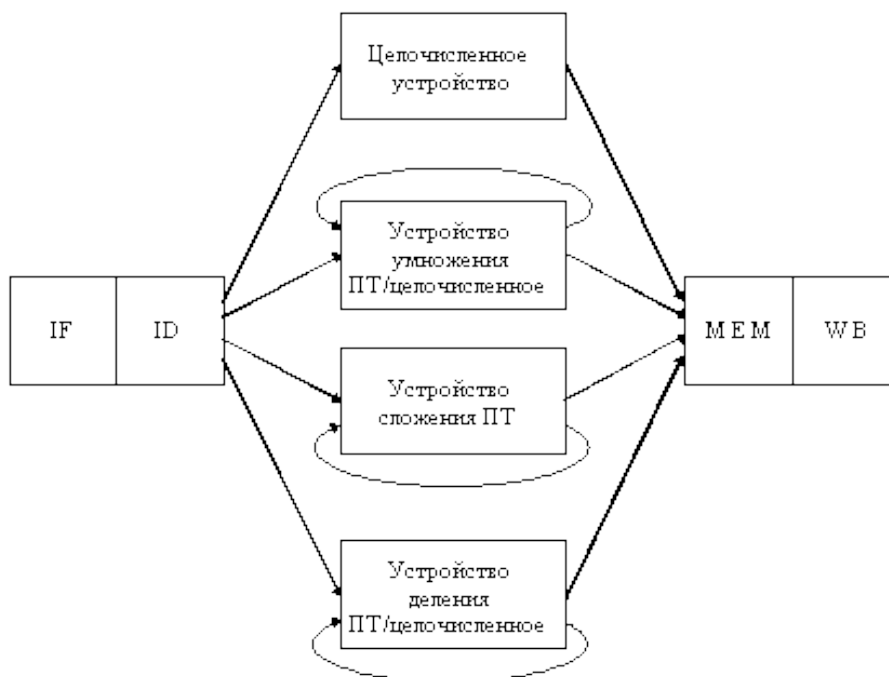


Рис. 15. Конвейер с дополнительными функциональными устройствами

Целочисленное устройство обрабатывает **все команды загрузки и записи в память** при работе с двумя наборами регистров (целочисленных и с плавающей точкой), **все целочисленные операции** (за исключением команд умножения и деления) и **все команды переходов**.

Если предположить, что стадии выполнения других функциональных устройств неконвейерные, то рис. 15 показывает структуру такого конвейера. Поскольку стадия EX является неконвейерной, никакая команда, использующая функциональное устройство, не может быть выдана для выполнения до тех пор, пока предыдущая команда не покинет ступень EX. Более того, если команда не может поступить на ступень EX, весь конвейер за этой командой будет приостановлен.

В действительности промежуточные результаты, возможно, не используются циклически ступенью EX, как это показано на рис. 15, и ступень EX имеет задержки длительностью более одного такта. Можно обобщить структуру конвейера плавающей точки, допустив конвейеризацию некоторых ступеней и параллельное выполнение нескольких операций. Чтобы описать работу такого конвейера, необходимо определить задержки функциональных устройств, а также скорость инициаций или скорость повторения операций. Это скорость, с которой новые операции данного типа могут поступать в функциональное устройство.

Например, предположим, что имеют место следующие задержки функциональных устройств и скорости повторения операций:

Функциональное устройство	Задержка Скорость повторения
Целочисленное АЛУ	1 1
Сложение с ПТ	4 2



Умножение с ПТ (и целочисленное)	6 3
Деление с ПТ (и целочисленное)	15 15

На рис. 16 представлена структура подобного конвейера. Ее реализация требует введения конвейерной регистровой станции EX1/EX2 и модификации связей между регистрами ID/EX и EX/MEM.

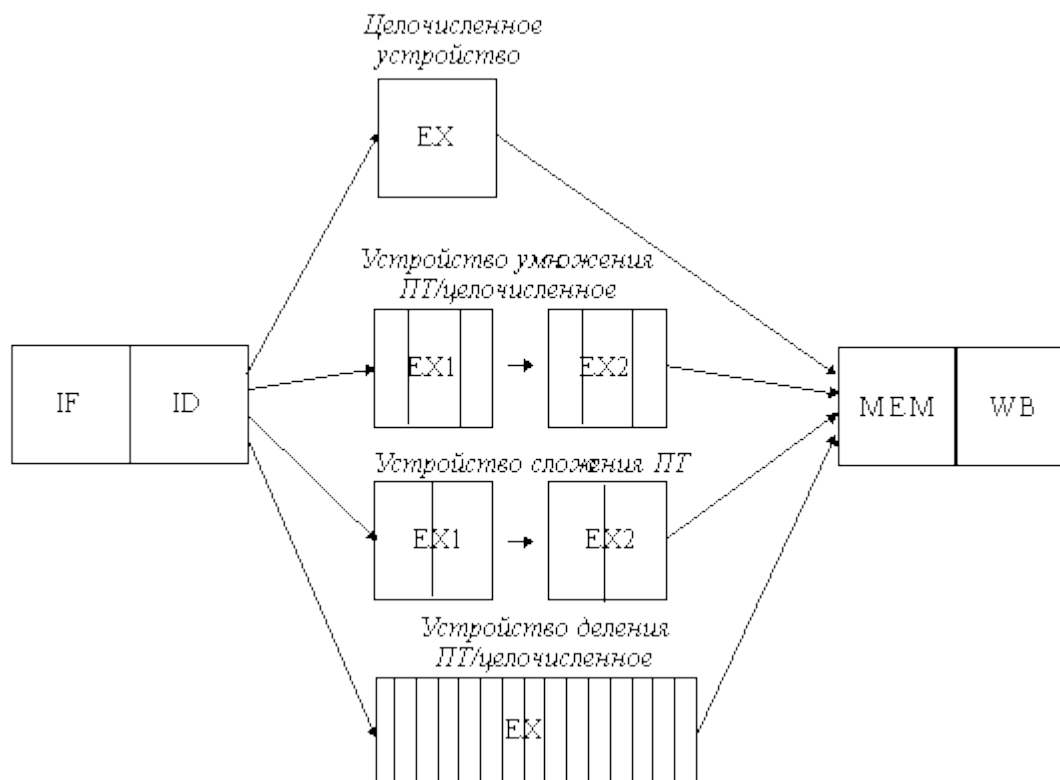


Рис.16. Конвейер с многоступенчатыми функциональными устройствами

## 9. Конфликты и ускоренные пересылки в длинных конвейерах

Имеется несколько различных аспектов обнаружения конфликтов и организации ускоренной пересылки данных в длинных конвейерах, подобных представленному на рис. 16:

1. Поскольку устройства не являются полностью конвейерными, в данной схеме возможны структурные конфликты. Эти ситуации необходимо обнаруживать и приостанавливать выдачу команд.
2. Поскольку устройства имеют разные времена выполнения, количество записей в регистровый файл в каждом такте может быть больше 1.
3. Возможны конфликты типа **WAW**, поскольку команды больше не поступают на ступень **WB** в порядке их выдачи для выполнения. Заметим, что конфликты типа **WAR** невозможны, поскольку чтение регистров всегда осуществляется на ступени **ID**.
4. Команды могут завершаться не в том порядке, в котором они были выданы для выполнения, что вызывает проблемы с реализацией прерываний.

Прежде чем представить общее решение для реализации схем обнаружения конфликтов, рассмотрим вторую и третью проблемы.

Если предположить, что файл регистров с плавающей точкой (ПТ) имеет только один порт записи, то последовательность операций с ПТ, а также операция загрузки ПТ совместно с операциями ПТ может вызвать конфликты по порту записи в регистровый файл.

Рассмотрим последовательность команд, представленную на рис. 17, которая является примером конфликта по записи в регистровый файл.

Команда	Номер такта									
	1	2	3	4	5	6	7	8	9	10
MULTD F0,F4,F6	IF	ID	EX1 <sub>1</sub>	EX1 <sub>2</sub>	EX1 <sub>3</sub>	EX2 <sub>1</sub>	EX2 <sub>2</sub>	EX2 <sub>3</sub>	MEM	WB
...		IF	ID	EX	MEM	WB				
ADDD F2,F4,F6			IF	ID	EX1 <sub>1</sub>	EX1 <sub>2</sub>	EX2 <sub>1</sub>	EX2 <sub>2</sub>	MEM	WB
...				IF	ID	EX	MEM	WB		
...					IF	ID	EX	MEM	WB	
LD F8,0(R2)						IF	ID	EX	MEM	WB

Рис. 17. Пример конфликта по записи в регистровый файл

В такте 10 все три команды достигнут ступени **WB** и должны произвести запись в регистровый файл. При наличии только одного порта записи в регистровый файл машина должна обеспечить последовательное завершение команд. Этот единственный регистровый порт является источником структурных конфликтов. Чтобы решить эту проблему, **можно увеличить количество портов** в регистровом файле, но такое решение может оказаться неприемлемым, поскольку эти дополнительные порты записи скорее всего будут редко использоваться. Однако в установившемся состоянии максимальное количество необходимых портов записи равно 1. Поэтому в реальных машинах разработчики предпочитают отслеживать обращения к порту записи в регистры и **рассматривать одновременное к нему обращение как структурный конфликт**.

Имеется два способа для обхода этого конфликта. Первый заключается в **отслеживании использования порта записи на ступени ID конвейера и приостановке выдачи команды как при структурном конфликте**. Схема обнаружения такого конфликта обычно реализуется с помощью сдвигового регистра. Альтернативная схема предполагает приостановку конфликтующей команды, когда она пытается попасть на ступень **MEM** конвейера. Преимуществом такой схемы является то, что она не требует обнаружения конфликта до входа на ступень **MEM**, где это легче сделать. Однако подобная реализация усложняет управление конвейером, поскольку приостановки в этом случае могут возникать в двух разных местах конвейера.

**Другой проблемой является возможность конфликтов типа WAW.** Можно рассмотреть тот же пример, что и на рис. 17. Если бы команда **LD** была

выдана на один такт раньше и имела в качестве месторасположения результата регистр **F2**, то возник бы конфликт типа **WAW**, поскольку эта команда выполняла бы запись в регистр **F2** на один такт раньше команды **ADDD**.

Имеются два способа обработки этого конфликта типа **WAW**:

**Первый подход** заключается в задержке выдачи команды загрузки до момента передачи команды **ADDD** на ступень **MEM**.

**Второй подход** заключается в подавлении результата операции сложения при обнаружении конфликта и изменении управления таким образом, чтобы команда сложения не записывала свой результат. Тогда команда **LD** может выдаваться для выполнения сразу же. Поскольку такой конфликт является редким, обе схемы будут работать достаточно хорошо. В любом случае конфликт может быть обнаружен на ранней стадии **ID**, когда команда **LD** выдается для выполнения. Тогда приостановка команды **LD** или установка блокировки записи результата командой **ADDD** реализуются достаточно просто.

Таким образом, для *обнаружения возможных конфликтов* необходимо рассматривать конфликты между командами ПТ, а также конфликты между командами ПТ и целочисленными командами. За исключением команд загрузки/записи с ПТ и команд пересылки данных между регистрами ПТ и целочисленными регистрами, команды ПТ и целочисленные команды достаточно хорошо разделены, и все целочисленные команды работают с целочисленными регистрами, а команды ПТ - с регистрами ПТ.

Таким образом, для *обнаружения конфликтов между целочисленными командами и командами ПТ* необходимо рассматривать только команды загрузки/записи с ПТ и команды пересылки регистров ПТ. Это упрощение управления конвейером является дополнительным преимуществом поддержания отдельных регистровых файлов для хранения целочисленных данных и данных с ПТ. (Главное преимущество заключается в удвоении общего количества регистров и увеличении пропускной способности без увеличения числа портов в каждом наборе). Если предположить, что конвейер выполняет обнаружение всех конфликтов на стадии **ID**, перед выдачей команды для выполнения в функциональные устройства должны быть выполнены три проверки:

1. **Проверка наличия структурных конфликтов.** Ожидание освобождения функционального устройства и порта записи в регистры, если он потребуется.
2. **Проверка наличия конфликтов по данным типа RAW.** Ожидание до тех пор, пока регистры-источники операндов указаны в качестве регистров результата на конвейерных станциях **ID/EX** (которая соответствует команде, выданной в предыдущем такте), **EX1/EX2** или **EX/MEM**.
3. **Проверка наличия конфликтов типа WAW.** Проверка того, что команды, находящиеся на конвейерных станциях **EX1** и **EX2**, не имеют в качестве месторасположения результата регистр результата выдаваемой для выполнения команды. В противном случае выдача команды, находящейся на ступени **ID**, приостанавливается.

Хотя логика обнаружения конфликтов для многотактных операций ПТ несколько более сложная, концептуально она не отличается от такой же логики для целочисленного конвейера. То же самое касается логики для ускоренной пересылки данных. Логика ускоренной пересылки данных может быть реализована с помощью проверки того, что указанный на конвейерных станциях **EX/MEM** и **MEM/WB** регистр результата является регистром операнда команды ПТ. Если происходит такое совпадение, для пересылки данных разрешается прием по соответствующему входу мультиплексора. Многотактные операции ПТ создают также новые проблемы для механизма прерывания.

## Поддержка точных прерываний

Другая проблема, связанная с реализацией команд с большим временем выполнения, может быть проиллюстрирована с помощью следующей последовательности команд:

DIVF F0,F2,F4

ADDF F10,F10,F8

SUBF F12,F12,F14

Эта последовательность команд выглядит очень просто. В ней отсутствуют какие-либо зависимости. Однако она приводит к появлению новых проблем из-за того, что выданная раньше команда может завершиться после команды, выданной для выполнения позже. В данном примере можно ожидать, что команды ADDF и SUBF завершаться раньше, чем завершится команда DIVF. Этот эффект является типичным для конвейеров команд с большим временем выполнения и называется *внеочередным завершением команд (out-of-order completion)*. Тогда, например, если команда DIVF вызовет арифметическое прерывание после завершения команды ADDF, мы не сможем реализовать точное прерывание на уровне аппаратуры. В действительности, поскольку команда ADDF меняет значение одного из своих операндов, невозможно даже с помощью программных средств восстановить состояние, которое было перед выполнением команды DIVF.

Имеются четыре возможных подхода для работы в условиях внеочередного завершения команд.

**1. Игнорирование проблемы и использование механизма неточного прерывания.** Этот подход использовался в 60-х и 70-х годах и все еще применяется в некоторых суперкомпьютерах, в которых некоторые классы прерываний запрещены или обрабатываются аппаратурой без остановки конвейера. Такой подход трудно использовать в современных машинах при наличии концепции виртуальной памяти и стандарта на операции с плавающей точкой IEEE, которые требуют реализации точного прерывания путем комбинации аппаратных и программных средств. В некоторых машинах эта проблема решается путем введения двух режимов выполнения команд: быстрого, но с возможно не точными прерываниями, и медленного, гарантирующего реализацию точных прерываний.

**2.Буферизация результатов операции до момента завершения выполнения всех команд, предшествовавших данной.** В некоторых машинах используется этот подход, но он становится все более дорогостоящим, если отличия во времени выполнения разных команд велики, поскольку становится большим количество результатов, которые необходимо буферизовать. Более того, результаты из этой буферизованной очереди необходимо пересылать для обеспечения продолжения выдачи новых команд. Это требует большого количества схем сравнения и многовходовых мультиплексоров. Имеются две вариации этого основного подхода. Первая называется буфером истории (history file), использовавшемся в машине CYBER 180/990. Буфер истории отслеживает первоначальные значения регистров. Если возникает прерывание и состояние машины необходимо откатить назад до точки, предшествовавшей некоторым завершившимся вне очереди командам, то первоначальное значение регистров может быть восстановлено из этого буфера истории. Подобная методика использовалась также при реализации автоинкрементной и автодекрементной адресации в машинах типа VAX. Другой подход называется буфером будущего (future file). Этот буфер хранит новые значения регистров. Когда все предшествующие команды завершены, основной регистровый файл обновляется значениями из этого буфера. При прерывании основной регистровый файл хранит точные значения регистров, что упрощает организацию прерывания. В следующей главе будут рассмотрены некоторые расширения этой идеи.

**3.Метод** заключается в том, чтобы **разрешить в ряде случаев неточные прерывания, но при этом сохранить достаточно информации, чтобы подпрограмма обработки прерывания могла выполнить точную последовательность прерывания.** Это предполагает наличие информации о находившихся в конвейере командах и их адресов. Тогда после обработки прерывания, программное обеспечение завершает выполнение всех команд, предшествовавших последней завершившейся команде, а затем последовательность может быть запущена заново.

Рассмотрим следующий наихудший случай:

*Команда 1* - длинная команда, которая в конце концов вызывает прерывание

*Команда 2, ... , Команда n-1* - последовательность команд, выполнение которых не завершилось

*Команда n* - команда, выполнение которой завершилось

Имея значения адресов всех команд в конвейере и адрес возврата из прерывания, программное обеспечение может определить состояние команды 1 и команды n. Поскольку команда n завершила выполнение, хотелось бы продолжить выполнение с команды n+1. После обработки прерывания программное обеспечение должно смоделировать выполнение команд с 1 по n-1. Тогда можно осуществить возврат из прерывания на команду n+1. Наибольшая неприятность такого подхода связана с усложнением подпрограммы обработки прерывания. Но для простых конвейеров, подобных рассмотренному нами, имеются и упрощения. Если команды с 2 по n все являются целочисленными, то мы просто знаем, что в случае завершения

выполнения команды  $n$ , все команды с 2 по  $n-1$  также завершили выполнение. Таким образом, необходимо обрабатывать только операцию с плавающей точкой. Чтобы сделать эту схему работающей, количество операций ПТ, выполняющихся с совмещением, может быть ограничено. Например, если допускается совмещение только двух операций, то только прерванная команда должна завершаться программными средствами. Это ограничение может снизить потенциальную пропускную способность, если конвейеры плавающей точки являются достаточно длинными или если имеется значительное количество функциональных устройств. Такой подход использовался в архитектуре SPARC, позволяющей совмещать выполнение целочисленных операций с операциями плавающей точки.

**4. Метод представляет собой гибридную схему, которая позволяет продолжать выдачу команд только если известно, что все команды, предшествовавшие выдаваемой, будут завершены без прерывания.** Это гарантирует, что в случае возникновения прерывания ни одна следующая за ней команда не будет завершена, а все предшествующие будут завершены. Иногда это означает необходимость приостановки машины для поддержки точных прерываний. Чтобы эта схема работала, необходимо, чтобы функциональные устройства плавающей точки определяли возможность появления прерывания на самой ранней стадии выполнения команд так, чтобы предотвратить завершение выполнения следующих команд. Такая схема используется, например, в микропроцессорах R2000/R3000 и R4000 компании MIPS.