Оценка производительности идеального конвейера

Выполнение каждой команды складывается из ряда последовательных этапов (шагов, стадий), суть которых не меняется от команды к команде. С целью увеличения быстродействия процессора и максимального использования всех его возможностей в современных микропроцессорах используется **конвейерный принцип обработки** информации. Этот принцип подразумевает, что в каждый момент времени процессор работает над различными стадиями выполнения нескольких команд, причем на выполнение каждой стадии выделяются отдельные аппаратные ресурсы. По очередному тактовому импульсу каждая команда в *конвейере* продвигается на следующую стадию обработки, выполненная команда покидает *конвейер*, а новая поступает в него.

В различных процессорах количество и суть этапов различаются. Рассмотрим принципы конвейерной обработки информации на примере пятиступенчатого *конвейера*, в котором выполнение команды складывается из следующих этапов:

1. `IF` (**Instruction Fetch**) - считывание команды в процессор;
2. `ID` (**Instruction Decoding**) - декодирование команды;
3. `OR` (**Operand Reading**) - считывание операндов;
4. `EX` (**Executing**) - выполнение команды;
5. `WB` (**Write Back**) - запись результата.

Выполнение команд в таком *конвейере* представлено в таблица 11.1.

Таблица 11.1.

Команда	Такт								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
i	IF	ID	OR	EX	WB				
i+1		IF	ID	OR	EX	WB			
i+2			IF	ID	OR	EX	WB		
i+3				IF	ID	OR	EX	WB	
i+4					IF	ID	OR	EX	WB

Пусть для выполнения отдельных стадий обработки требуются следующие затраты времени (в некоторых условных единицах):

$T_{IF} = 20,\ T_{ID} = 15,\ T_{OR} = 20,\ T_{EX} = 25,\ T_{WB} = 20.$

Тогда, предполагая, что дополнительные расходы времени составляют dt = 5 единиц, получим время *такта*:

$T = \max \{T_{IF},\ T_{ID},\ T_{OR},\ T_{EX},\ T_{WB}\} + \Delta t = 30.$

Оценим время выполнения одной команды и некоторой группы команд при последовательной и конвейерной обработке.

При последовательной обработке время выполнения N команд составит:

$$T_{посл} = N*(T_{IF} + T_{ID} + T_{OR} + T_{EX} + T_{WB}) = 100N.$$

Анализ таблица 11.1 показывает, что при конвейерной обработке после того, как получен результат выполнения первой команды, результат очередной команды появляется в следующем *такте* работы процессора. Следовательно,

$$T_{конв} = 5T + (N-1) * T.$$

Примеры длительности выполнения некоторого количества команд при последовательной и конвейерной обработке приведены в таблица 11.2.

Таблица 11.2.

Количество команд	Время	
	при последовательном выполнении	при конвейерном выполнении
1	100	150
2	200	240
10	1000	420
100	10000	3120

Очевидно, что при достаточно длительной работе *конвейера* его быстродействие будет существенно превышать быстродействие, достигаемое при последовательной обработке команд. Это увеличение будет тем больше, чем меньше длительность *такта конвейера* и чем больше количество выполненных команд. Сокращение длительности *такта* достигается, в частности, разбиением выполнения команды на большое число этапов, каждый из которых включает в себя относительно простые операции и поэтому может выполняться за короткий промежуток времени. Так, если в микропроцессоре **Pentium** длина *конвейера* составляла 5 ступеней (при максимальной тактовой частоте 200 МГц), то в **Pentium-4** - уже 20 ступеней (при максимальной тактовой частоте на сегодняшний день 3,4 ГГц).

Конфликты в конвейере и способы минимизации их влияния на производительность процессора

Значительное преимущество конвейерной обработки перед последовательной имеет место в *идеальном конвейере*, в котором отсутствуют конфликты и все команды выполняются друг за другом без перезагрузки *конвейера*. Наличие конфликтов снижает реальную производительность *конвейера* по сравнению с идеальным случаем.

Конфликты - это такие ситуации в конвейерной обработке, которые препятствуют выполнению очередной команды в предназначенном для нее *такте*.

Конфликты делятся на три группы:

- *структурные*,
- *по управлению*,
- по данным.

Структурные конфликты возникают в том случае, когда аппаратные средства процессора не могут поддерживать все возможные комбинации команд в режиме одновременного выполнения с совмещением.

Причины *структурных конфликтов*.

1. Не полностью конвейерная структура процессора, при которой некоторые ступени отдельных команд выполняются более одного *такта*.

   Пусть этап выполнения команды `i+1` занимает 3 *такта*. Тогда диаграмма работы *конвейера* будет иметь вид, представленный в таблица 11.3.

Таблица 11.3.

Команда	Такт								
	1	2	3	4	5	6	7	8	9
i	IF	ID	OR	EX	WB				
i+1		IF	ID	OR	EX	EX	EX	WB	
i+2			IF	ID	OR	O	O	EX	WB
i+3				IF	ID	OR	O	O	EX
i+4					IF	ID	OR	O	O

   При этом в работе *конвейера* возникают так называемые "пузыри" (обработка команд `i+2` и следующих за ней, начиная с *такта* 6), которые снижают производительность процессора.

   Эту ситуацию можно было бы ликвидировать двумя способами. Первый предполагает увеличение времени *такта* до такой величины, которая позволила бы все этапы любой команды выполнять за один *такт*. Однако при этом существенно снижается эффект конвейерной обработки, так как все этапы всех команд будут выполняться значительно дольше, в то время как обычно нескольких *тактов* требует выполнение лишь отдельных этапов очень небольшого количества команд. Второй способ предполагает использование таких аппаратных решений, которые позволили бы значительно снизить затраты времени на выполнение данного этапа (например, использовать матричные схемы умножения). Но это приведет к усложнению схемы процессора и невозможности реализации на этой БИС других, функционально более важных, узлов. Так как представленная в таблица 11.3 ситуация возникает при реализации команд, относительно редко встречающихся в программе, то обычно разработчики процессоров ищут компромисс между увеличением длительности *такта* и усложнением того или иного устройства процессора.

2. Недостаточное дублирование некоторых ресурсов.

   Одним из типичных примеров служит конфликт из-за доступа к запоминающим устройствам. Из таблица 11.1 видно, что в случае, когда операнды и команды находятся в одном запоминающем устройстве, начиная с *такта* 3, работу *конвейера* придется постоянно приостанавливать, поскольку различные команды в одном и том же *такте* обращаются к памяти на считывание команды, выборку операнда, запись результата.

   Борьба с конфликтами такого рода проводится путем увеличения количества однотипных функциональных устройств, которые могут одновременно выполнять одни и те же или схожие функции. Например, в современных микропроцессорах обычно разделяют кэш-память для хранения команд и кэш-память данных, а также используют многопортовую схему доступа к регистровой памяти, при которой к регистрам можно одновременно обращаться по одному каналу для записи, а по другому - для считывания информации. Конфликты из-за исполнительных устройств обычно сглаживаются введением в

состав микропроцессора дополнительных блоков. Так, в микропроцессоре Pentium-4 предусмотрено 4 АЛУ для обработки целочисленных данных. Процессоры, имеющие в своем составе более одного *конвейера*, называются **суперскалярными**.

Недостатком суперскалярных микропроцессоров является необходимость синхронного продвижения команд в каждом из *конвейеров*. В таблица 11.4 представлена последовательность выполнения команд в микропроцессоре, имеющем два *конвейера*, при условии, что команде К1 требуется 3 *такта* на этапе `EX`.

Таблица 11.4.

Этап	Такт													
	1		**2**		**3**		**4**		**5**		**6**		**7**	
`IF`	K1	K2	K3	K4	K5	K6	K7	K8	K7	K9	K7	K10	K11	K12
`ID`			K1	K2	K3	K4	K5	K6	K5	K8	K5	K9	K7	K10
`OR`					K1	K2	K3	K4	K3	K6	K3	K8	K5	K9
`EX`							K1	K2	K1	K4	K1	K6	K3	K8
`WB`										K2		K4	K1	K6

При этом команды будут завершаться в последовательности

```
K2-K4-K1-K6-...
```

Следовательно, для обеспечения правильной работы суперскалярного микропроцессора при возникновении затора в одном из *конвейеров* должны приостанавливать свою работу и другие. В противном случае может нарушиться исходный порядок завершения команд программы. Но такие приостановки существенно снижают быстродействие процессора. Разрешение этой ситуации состоит в том, чтобы дать возможность выполняться командам в одном *конвейере* вне зависимости от ситуации в других *конвейерах*. Это приводит к **неупорядоченному выполнению команд**. При этом команды, стоящие в программе позже, могут завершиться ранее команд, стоящих впереди. Аппаратные средства микропроцессора должны гарантировать, что результаты выполненных команд будут записаны в приемник в том порядке, в котором команды записаны в программе. Для этого в микропроцессоре результаты этапа выполнения команды обычно сохраняются в специальном буфере восстановления последовательности команд. Запись результата очередной команды из этого буфера в приемник результата проводится лишь после того, как выполнены все предшествующие команды и записаны их результаты.

Конфликты по управлению возникают при конвейеризации команд переходов и других команд, изменяющих значение счетчика команд.

Суть конфликтов этой группы наиболее удобно проиллюстрировать на примере команд условного перехода. Пусть в программе, представленной в таблица 11.1, команда `i+1` является командой условного перехода, формирующей адрес следующей команды в зависимости от результата выполнения команды `i`. Команда `i` завершит свое выполнение в *такте* 5. В то же время команда условного перехода уже в *такте* 3 должна прочитать необходимые ей признаки, чтобы правильно сформировать адрес следующей команды.
Если *конвейер* имеет большую глубину (например, 20 ступеней), то промежуток времени между формированием признака результата и *тактом*, где он анализируется, может быть еще большим. В инженерных задачах примерно

каждая шестая команда является командой условного перехода, поэтому приостановки конвейера при выполнении команд переходов до определения истинного направления перехода существенно скажутся на производительности процессора.

Наиболее эффективным методом снижения потерь от *конфликтов по управлению* служит **предсказание переходов**. Суть данного метода заключается в том, что при выполнении команды условного перехода специальный блок микропроцессора определяет наиболее вероятное направление перехода, не дожидаясь формирования признаков, на основании анализа которых этот переход реализуется. Процессор начинает выбирать из памяти и выполнять команды по предсказанной ветви программы (так называемое **исполнение по предположению**, или "спекулятивное" исполнение). Однако так как направление перехода может быть предсказано неверно, то получаемые результаты с целью обеспечения возможности их аннулирования не записываются в память или регистры (то есть для них не выполняется этап `WB`), а накапливаются в специальном буфере результатов.

Если после формирования анализируемых признаков оказалось, что направление перехода выбрано верно, все полученные результаты переписываются из буфера по месту назначения, а выполнение программы продолжается в обычном порядке. Если направление перехода предсказано неверно, то буфер результатов очищается. Также очищается и *конвейер*, содержащий команды, находящиеся на разных этапах обработки, следующие за командой условного перехода. При этом аннулируются результаты всех уже выполненных этапов этих команд. *Конвейер* начинает загружаться с первой команды другой ветви программы. Так как конвейерная обработка эффективна при большом числе последовательно выполненных команд, то перезагрузка конвейера приводит к значительным потерям производительности. Поэтому вопросам эффективного предсказания направления ветвления разработчики всех микропроцессоров уделяют большое внимание.

Методы предсказания переходов делятся на статические и динамические. При использовании статических методов до выполнения программы для каждой команды условного перехода указывается направление наиболее вероятного ветвления. Это указание делается или программистом с помощью специальных средств, имеющихся в некоторых языках программирования, по опыту выполнения аналогичных программ либо результатам тестового выполнения программы, или программой-компилятором по заложенным в ней алгоритмам.

Методы динамического прогнозирования учитывают направления переходов, реализовывавшиеся этой командой при выполнении программы. Например, подсчитывается количество переходов, выполненных ранее по тому или иному направлению, и на основании этого определяется направление перехода при следующем выполнении данной команды.

В современных микропроцессорах вероятность правильного предсказания направления переходов достигает 90-95 %.

Конфликты по данным возникают в случаях, когда выполнение одной команды зависит от результата выполнения предыдущей команды.

При обсуждении этих конфликтов будем предполагать, что команда `i` предшествует команде `j`.

Существует несколько типов *конфликтов по данным*.

1. Конфликты типа RAW (**Read After Write**): команда j пытается прочитать операнд прежде, чем команда i запишет на это место свой результат. При этом команда j может получить некорректное старое значение операнда.

   Проиллюстрируем этот тип конфликта на примере выполнения команд, представленных в таблица 11.1. Пусть выполняемые команды имеют следующий вид:

```
i)       ADD R1,R2;    R1 = R1+R2
i+1=j)   SUB R3,R1;    R3 = R3-R1
```

   Команда i изменит состояние регистра R1 в *такте* 5. Но команда i+1 должна прочитать значение операнда R1 в *такте* 4. Если не приняты специальные меры, то из регистра R1 будет прочитано значение, которое было в нем до выполнения команды i.

   Уменьшение влияния конфликта типа RAW обеспечивается методом обхода (продвижения) данных. В этом случае результаты, полученные на выходах исполнительных устройств, помимо входов приемника результата передаются также на входы всех исполнительных устройств микропроцессора. Если устройство управления обнаруживает, что данный результат требуется одной из последующих команд в качестве операнда, то он сразу же, параллельно с записью в приемник результата, передается на вход исполнительного устройства для использования следующей командой.

   Конфликты типа RAW обусловлены именно конвейерной организацией обработки команд.

   Главной причиной двух других типов *конфликтов по данным* является возможность неупорядоченного выполнения команд в современных микропроцессорах, то есть выполнение команд не в том порядке, в котором они записаны в программе.

2. Конфликты типа WAR (**Write After Read**): команда j пытается записать результат в приемник, прежде чем он считается оттуда командой i, При этом команда i может получить некорректное новое значение операнда:

```
3. i)       ADD R1,R2
   i+1 =j)  SUB R2,R3
```

   Этот конфликт возникнет в случае, если команда j вследствие неупорядоченного выполнения завершится раньше, чем команда i прочитает старое содержимое регистра R2.

4. Конфликты типа WAW (**Write After Write**): команда j пытается записать результат в приемник, прежде чем в этот же приемник будет записан результат выполнения команды i, то есть запись заканчивается в неверном порядке, оставляя в приемнике результата значение, записанное командой i:

```
5. i)   ADD R1,R2
6.          . . .
   j)   SUB R1,R3
```

   Устранение *конфликтов по данным* типов WAR и WAW достигается путем отказа от неупорядоченного исполнения команд, но чаще всего путем введения буфера восстановления последовательности команд.

   Как отмечалось выше, наличие конфликтов приводит к значительному снижению производительности микропроцессора. Определенные типы

конфликтов требуют приостановки *конвейера*. При этом останавливается выполнение всех команд, находящихся на различных стадиях обработки (до 20 ти команд в **Pentium-4**). Другие конфликты, например, при неверном предсказанном направлении перехода, ведут к необходимости полной перезагрузки *конвейера*. Потери будут тем больше, чем более длинный *конвейер* используется в микропроцессоре. Такая ситуация явилась одной из причин сокращения числа ступеней в микропроцессорах последних моделей. Так, в микропроцессоре **Itanium** *конвейер* содержит всего 10 ступеней. При этом его тактовая частота составляет около 1 МГц [[2]]. Однако на каждой ступени выполняется больше функциональных действий, чем в **Pentium-4**.