

Суперскалярная и VLIW архитектуры

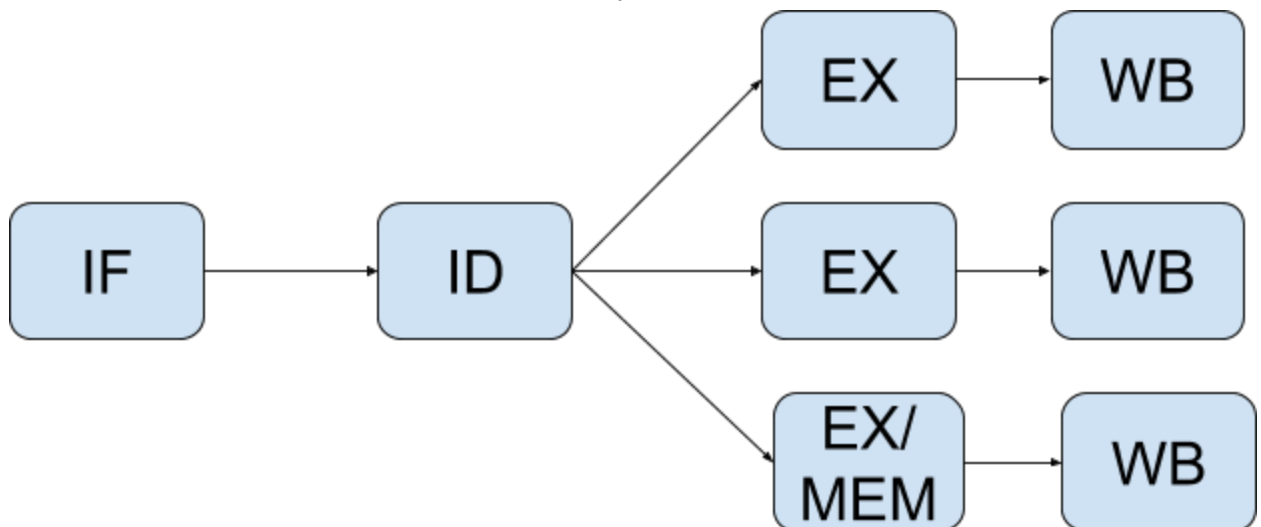
В современных компьютерах принцип последовательного выполнения Фон Неймана строго не выполняется. Железо выполняет команды параллельно но делает это так, чтобы результат был идентичен результату последовательного выполнения программ. То есть железо может отойти от принципа последовательного выполнения ровно настолько, чтобы результат был неизменен, но команды выполнялись как можно быстрее. Есть два способа использовать несколько конвейеров **одновременно**:

1. **VLIW** - Very Long Instruction Word.

Теперь инструкции будут приходить пачками¹ в каждой, в блок **IF** (Instruction Fetch).

Далее, блок **ID** (Instruction Decode) раскидывает команды по конвейерам.

Конвейеры могут быть различны: например, можно вынести операции для работы с памятью в отдельные конвейеры, а в парочку арифметических добавить работу с числами с плавающей точкой. Как видно, планировка происходит на этапе компиляции. Пример простой VLIW архитектуры:



IF - Instruction Fetch. Достаёт команды из памяти. Поддерживает в себе Instruction Pointer, указывающий на текущий адрес команды.

ID - Instruction Decode. Раскидывает команды по конвейерам: *i*-ой команде в пачке соответствует *i*-ый конвейер. Так и рождаются **NOP**ы - иногда мы не можем забить все конвейеры, так как некоторые не поддерживают специфические операции, которые нужны нам в данный момент. Поэтому и ждём до следующей пачки.

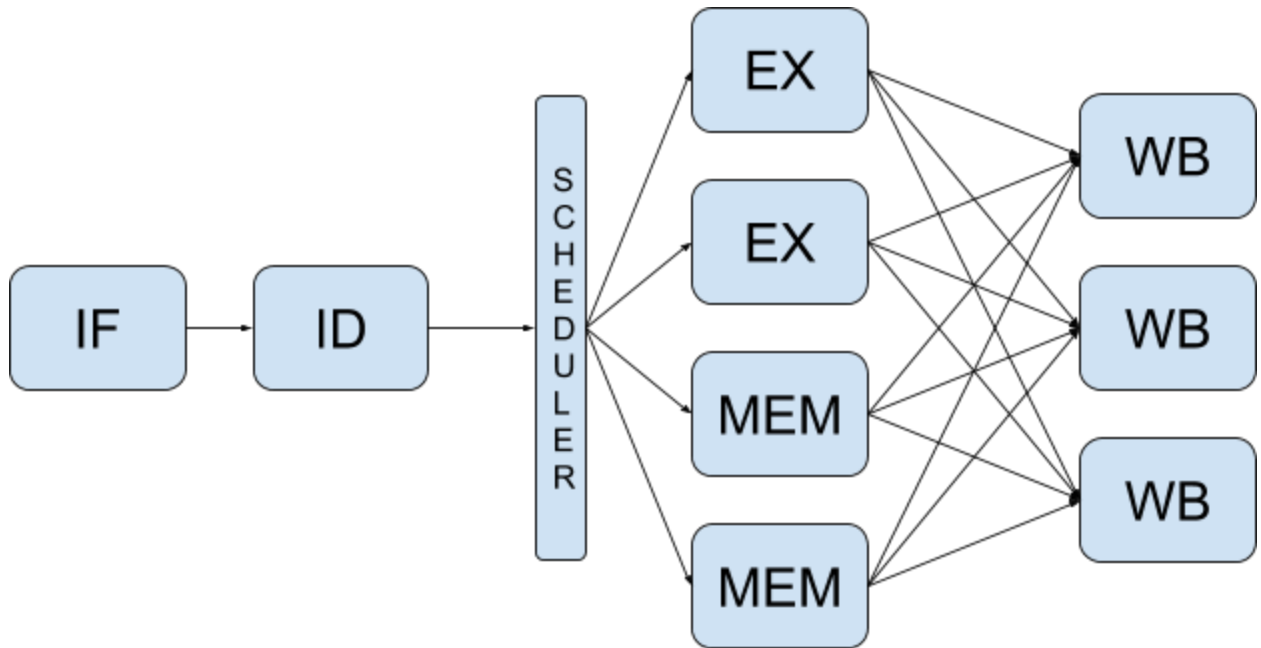
EX - Executor. Выполняющий блок (основная часть конвейера).

EX/MEM - Executor + Memory management - Он же, но с возможностью чтения из памяти.

WB - Writeback. Модуль записи результата в память или регистр.

¹ Размер пачки фиксирован для каждого процессора. Часто он равен 3, но, в общем случае, варьируется от 2 до 6.

2. **Superscalar.** Инструкции поступают пачками, но после **ID** они **разбиваются на микрокоманды, поступающие в буффер планировщика (Scheduler)**. **Размер буфера может достигать до нескольких сотен команд, 170 - примерный средний размер** (его устроит).



- В данном примере показано, что из-за того, что **WB** - достаточно сложный блок, их число может быть меньше числа конвейеров. Поэтому и строят такие сети :) Если вдруг случилось, что все блоки **WB** заняты, то весь **конвейер ждёт**.
- Кроме того, мы вынесли **МЕМы**, работу с памятью. Почему? Обращение к памяти может быть очень долгим, а из-за этого будет простаивать конвейер, что плохо. Поэтому здоровые люди выносят обращение, чтобы не портить саму идею конвейеров.

Плюсы и минусы

VLIW	Superscalar
<ul style="list-style-type: none"> • +/- Из-за планирования при компиляции, производительность зависит от планировщика² в компиляторе. <ul style="list-style-type: none"> ○ + Из-за того, что с нормальным компилятором всё будет здорово и код в общем станет быстрым. ○ - Из-за того, что народ компилирует всякой фигнёй и код 	<ul style="list-style-type: none"> • + При добавлении/изменении/удалении планировщик решает наши проблемы. • + Нет бессмысленных НОРов в коде, соотв. код меньше и кэши не забиты. • + Run-time информация о кэшах у планировщика (дальнейшем коде, переменных в них)- ещё лучше планирование.

² Только в этом случае под планировщиком я понимаю "планировщик в компиляторе", но не в процессоре.

<p>становится медленным. Когда вы так делаете, в мире плачет один котик-VLIWотик</p> <ul style="list-style-type: none"> • + Просто сделать. • + Просто добавить новый конвейер (обычная копия паста). • - Из-за планирования на этапе компиляции, НОРы в коде забивают кэши и их приходится увеличивать. • - При добавлении/изменении/удалении конвейеров старый код приходится перекомпилировать под данный процессор. 	<ul style="list-style-type: none"> • + Для сохранения энергии можно отключать некоторые конвейеры. • - Сложно сделать. • - Чем тупее superscalar - тем чувствительнее он к компилятору. Код компилируется по спецификации какой-нибудь общей ISA (x86, например). При этом неизвестно, что внутри суперскаляра на этапе компиляции. Из-за этого при компиляции на всяких, кхм, сомнительных компиляторах (прости, визуалка), код может отставать паза в 2 по скорости работы, по сравнению с норм компиляторами. • - Занимает место на кристалле
---	---

Hazards

Write-after-write	Write-after-read
<pre>mul R1,R2,R3 or R4,R4,R1 add R1,R2,R3 sub R5,R5,R1</pre>	<pre>mul R1,R2,R3 or R4,R4,R1 ; просто для красоты add R5,R1,R3 sub R5,R5,R1</pre>
Решается аппаратным переименованием регистров	Это отслеживает scheduler

Пояснение к п. 1: из-за того, что **mul** и **add** могут выполняться за разное время, после выполнения **add**, в **R1** запишется результат перемножения.

Пояснение к п. 2: аналогично, сперва в команде **add** мы считаем **R1**, а потом запишем результат перемножения.

NB: Внутри суперскаляра всё ещё конвейеры с сопутствующими для них хазардами. Не забывайте про билет 2.4 :)

Про переименование регистров

Проблема - Write-After-Write hazards.

Решение: давайте каждый регистр будет не на конкретный кусок памяти, а на тот, на который мы ему скажем. Вот как это поможет:

mul R1,R2,R3 ; R1 ссылается на какой-то регистр h0, собираемся записать в h0




or R4,R4,R1 ; перемножение R2,R3.
 add R1,R2,R3 ; Теперь R1 ссылается на другой регистр, например, h5, куда
 ; и будет записана сумма.
 sub R5,R5,R1 ; R1 ссылается на h5. Мы сказали положить туда сумму в команде
 ; add. Результат перемножения лежит в h0 и никому не мешает ;)
 NB: как выбираются такие регистры - даже не спрашивайте.

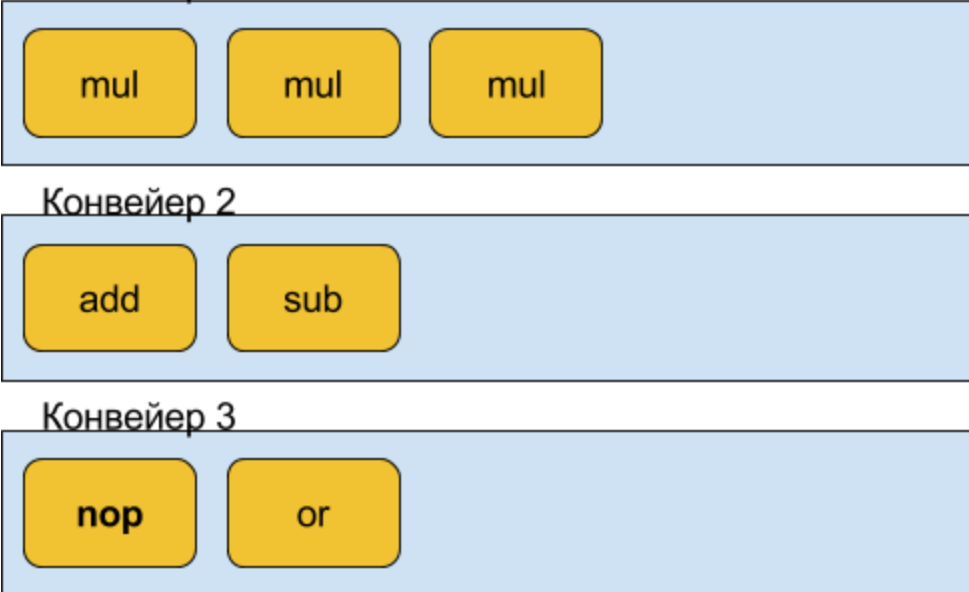
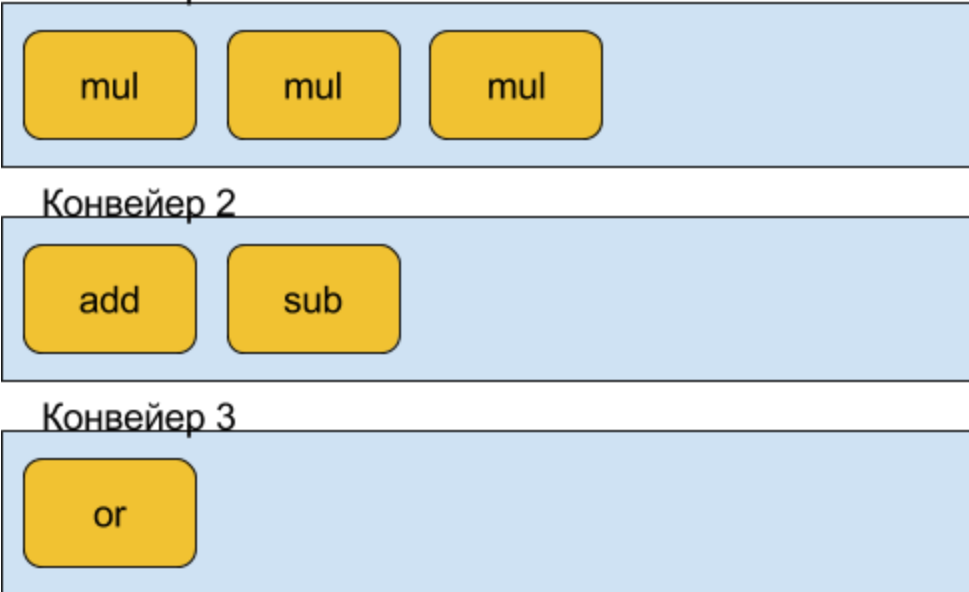
Планировщики

Пример кода:

```

mul R0,R0,R1
add R2,R2,R3
sub R2,R2,R3
or R4,R5,R6
  
```

Тип входа/ тип выхода	Описание
InO/InO	<p>Размер буфера команд \leq количеству конвейеров. Делать его больше нет смысла, ибо выполнения последовательное, т.е. буфер превратится в обычную очередь. Зачем? Расписание выполнения команд примера ниже</p> <p>Конвейер 1</p>  <p>Конвейер 2</p>  <p>Конвейер 3</p> 
InO/OoO	<p>Размер буфера команд \leq количеству конвейеров. Порядок выход команд из конвейеров может не совпадать с порядком входа.</p>

	<p>Конвейер 1</p>  <p>Конвейер 2</p> <p>Конвейер 3</p> <p>Пример: старые Intel Atom, некоторые ARM.</p>
<p>ОоО</p>	<p>Суперскаляр сам планирует порядок входа микрокоманд. Порядок выход команд из конвейеров может не совпадать с порядком входа.</p> <p>Конвейер 1</p>  <p>Конвейер 2</p> <p>Конвейер 3</p>

Типы планировщиков отсортированы по возрастанию сложности и увеличению их эффективности.

Современные Intel суперскаляры имеют около 8 конвейеров: например, 4 арифметических, 2 на запись в память, 2 на чтение.

К ОоО можно отнести все современные процессоры (для хомячков и *продвинутых*): Intel, AMD, ARM (высокопроизводительные)/

Примеры

VLIW	Superscalar
<ul style="list-style-type: none">• Эльбрус• Itanium• Старые видеокарты AMD<ul style="list-style-type: none">○ Мем в том, что железо общается не с программами пользователей, а с драйверами, которые AMD же и писали, из-за чего проблем с программированием не было. VLIW архитектура проста, что дало больше площади на кристалле -> больше вычислителей. Хороший шаг, но потом они от него ушли, ибо в современном мире, помимо работы с графикой, видеокарты используются как обычные программируемые вычислители: бетховены майнятся, хэши ломаются и т.д.	См. Выше :)