1 Конвеер

Заметим, что если выполнять все инструкции на железке последовательно, то во время выполнения каждой инструкции много железа будет просто не использоваться.
Давайте разделим процесс выполнения на несколько независимых стадий. И будем выполнять каждую за один такт. Собственно, когда первая команда перешла на вторую стадию выполнения, на первую уже можно подать следующую команду и так далее.

2 Конвеер MIPS

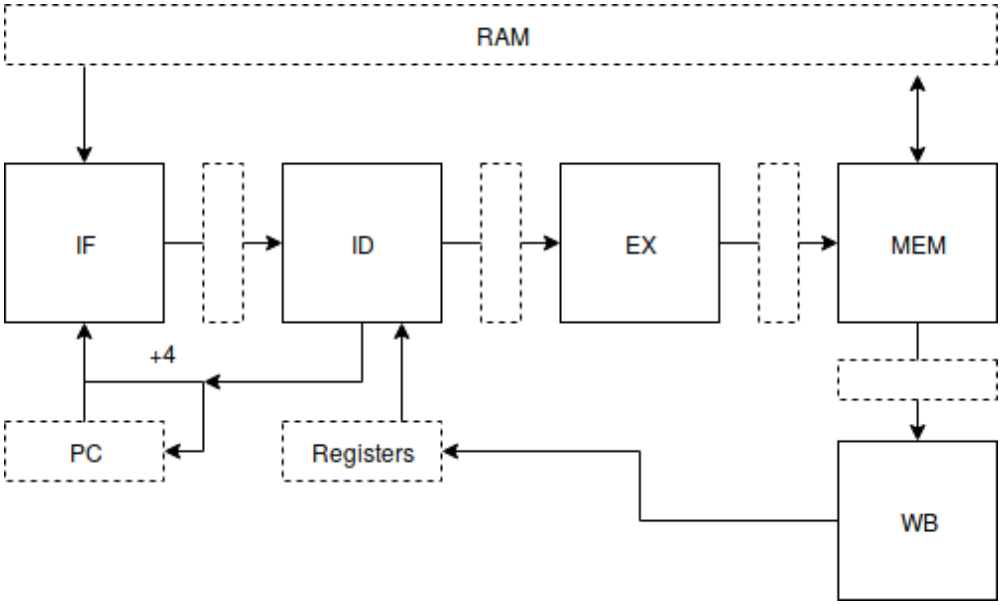


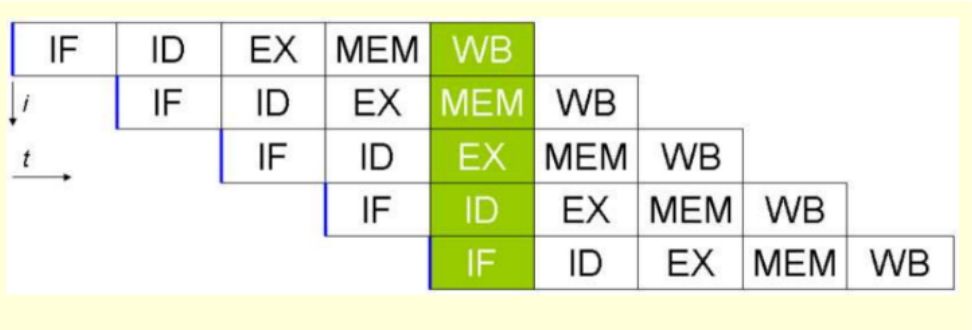
Рис. 1: MIPS конвеер

Стадии:

- Instruction Fetch
- Instruction Decode
- Execute
- MEM
- Write-back

2.1 Instruction Fetch

На этой стадии выполняется чтение команды из памяти.
У CPU есть специальный регистр, **PC** (Program Counter в MIPS, в x86 он называется IP - Instruction Pointer), в нем хранится адрес выполняемой команды.
На этой стадии из памяти считывается четыре байта (размер одной команды в MIPS), затем счетчик PC увеличивается на четыре.



2.2 Instruction Decode

На этой стадии команда декодируется. Т.е. разбираем команду на:

- код операции
- коды регистров-операндов
- код константы (если есть)

Если в команде фигурировали значения регистров, то мы считываем эти значения во второй половине такта (на спаде импульса синхронизации).

2.3 Execute

На этой стадии выполняются операции с операндами (которые декодировали на предыдущей стадии). Для операций LD и ST на этой стадии считается адрес.

2.4 MEM

На этой стадии происходит чтение/запись в память.

2.5 Write-Back

На этой стадии (в первой половине такта, по фронту имульса синхронизации) всё, что было посчитано или считано из памяти пишется в регистры.

3 Почему это быстрее?

Во-первых, так как стадии простые, выполняются они быстрее. Следовательно можно увеличить тактовую частоту.
Во-вторых, не смотря на то, что у нас могла увеличиться lattency (м.б. раньше мы делали одну команду за 2 такта, а теперь делаем её за 5), у нас сильно увеличилсь throughput (новые команды получаем каждый такт). Одна команда выполнится за 5 тактов, а сто команд - за 104 такта (в среднем каждая команда выполнится за 1 такт, ничего так).
Скорость работы $= Clock * IPC$
IPC - Instruction Per Clock - сколько полезного делаем за один такт. На конвеере мы увеличиваем оба параметра, поэтому скорость растёт.

4 Примеры

ADD R0 R1 5

1. IF: Из памяти считана инструкция. PC += 4
2. ID: Инструкция декодирована. Команда: ADD, Операнды: regs[R1], 5. Записано должно быть в R0.
3. EX: Выполнено сложение.
4. MEM: NOP. Эта команда не обращается к памяти, на этой стадии с ней ничего не происходит.
5. WB: Результат сложения пишется в регистр R0.

5 Конфликты конвеера

5.1 Data hazard

Data hazards - конфликты, возникающие из-за того, что инструкции не независимы.
Пример:
ADD R1 R2 R3
SUB R4 R5 R1
XOR ...

IF	ID	EX	MEM	WB	
ADD					
SUB	ADD				
XOR	SUB	ADD			
XOR	SUB	NOP	ADD		
XOR	SUB	NOP	NOP	ADD	
XOR	SUB	NOP	NOP	NOP	<-На MIPS эта строка пропускается из-за хитрости с ID и WB
	XOR	SUB	NOP	NOP	

Рис. 2: Пример RAW хазарда

Есть несколько подходов к решению такого конфликта:

1. Притормозить конвеер, т.е. закинуть на стадии NOP вместо полезных вычислений. В примере можно оставить SUB на стадии ID. Тогда ID отменит изменение PC, а на следующую стадию передаст NOP.
2. Forwarding. Запросить у одной из следующих стадий уже посчитанное значение. ID может попросить читать значение из следующего буфера. Но если перед SUB была команда загрузки из памяти? Тогда нужна еще одна линия Forwarding. Но команда LD выполняется только на стадии MEM -> на стадию EX всё равно придется кинуть NOP.
3. Документация. Можно сказать, что то, что команда записывает результат выполнения только через 4 такта - не баг, а фича.

В MIPS конфликты решаются с помощью Forwarding и документации -> там NOP не появляются сами по себе.

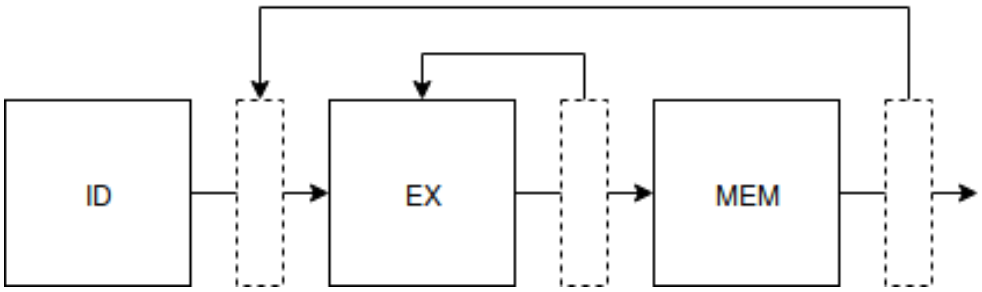


Рис. 3: Forwarding

5.2 Control hazards

Control hazards - конфликты, возникающие из-за инструкций, которые изменяют PC.
Не рассказывать про branch prediction))0))00
Пример:
ADD
JMP L1
SUB
SUB
L1: XOR
Есть два способа починить:

1. Документация
2. Кидать NOP

Кроме того, обычно JMP исполняется на стадии ID.
В MIPS: документация. JMP срабатывает на такт позже -> команда, написанная после команды JMP тоже выполнится. Это место после команды JMP называется "delay slot".

5.3 Structural hazard

Structural hazard - конфликты, связанные с нехваткой ресурсов. Т.е. когда железка не может выпонять некоторые команды одновременно на каких-то стадиях.
Возникает, например, если у нас один канал к памяти. Если на стадии MEM выполняется LD или ST - не можем считать команду на стадии IF.
Как пофиксить:
Добавить железа. Либо жить с этим.
В MIPS такой проблемы нет, т.к. MIPS - гарвардская архитектура.