Архитектура вычислительных систем

Лекция 12.

Яревский Е.А.

Кафедра вычислительной физики

КОНВЕЙЕРНЫЙ ПРОЦЕССОР

Разработаем конвейерный процессор, разделив однотактный процессор на пять стадий. Пять команд смогут выполняться одновременно, по одной в каждой из стадий.

В идеальном случае латентность команд не изменится, а пропускная способность вырастет в пять раз. Конвейеризация требует накладных расходов, так что в реальности пропускная способность будет ниже, чем в идеальном случае.

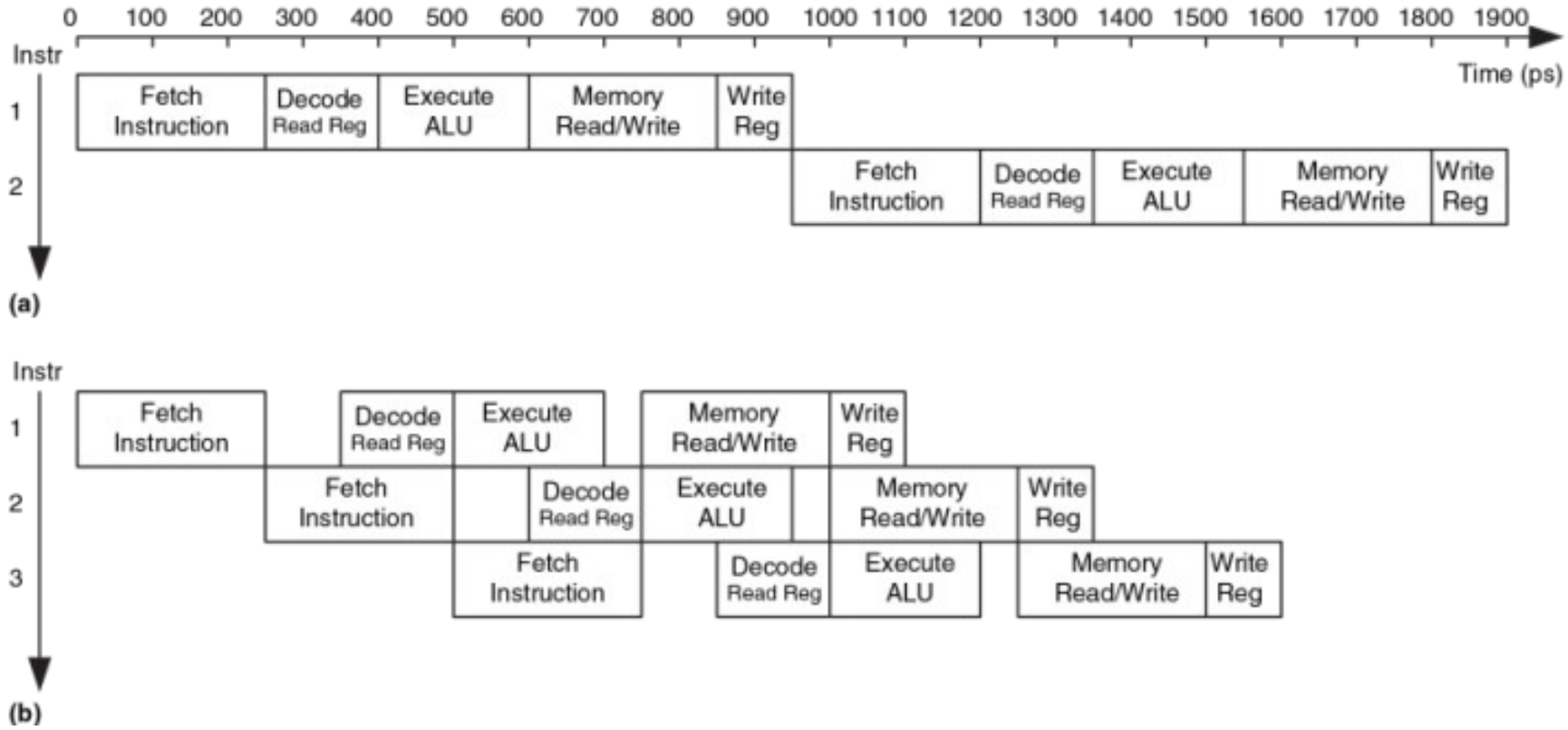
Наибольшие задержки: чтение и запись в память и регистровый файл, использование АЛУ.

Поделим конвейер на стадии так, чтобы каждая стадия включала ровно одну из этих операций.

Стадии:

- Fetch (выборка),
- Decode (дешифрация),
- Execute (выполнение),
- Memory (доступ к памяти),
- Writeback (запись результатов).

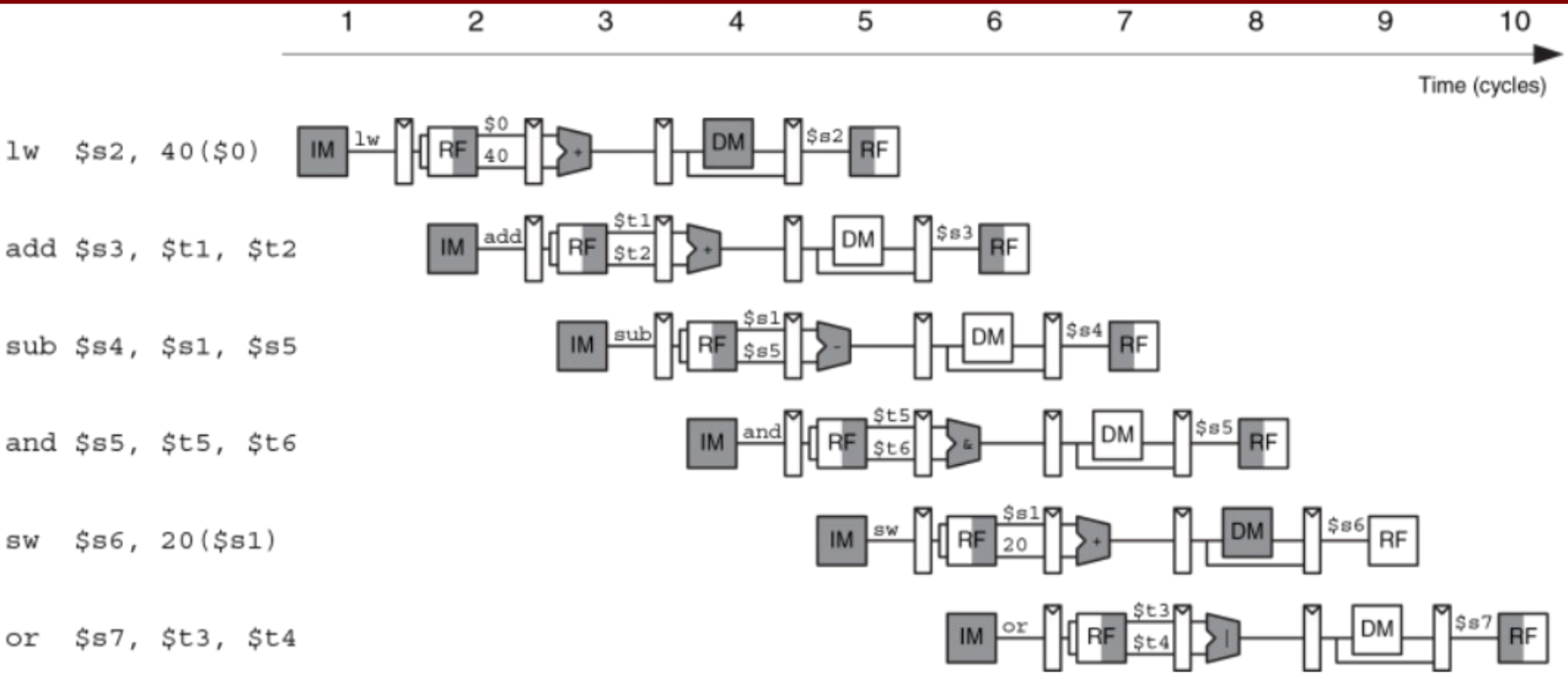
Временная диаграмма процессоров



Однотактный процессор: латентность команд = 950 пс,
пропускная способность – 1 команда за 950 пс.

Конвейерный процессор:, длина стадии = 250 пс, определяется самой медленной стадией – доступ к памяти (Fetch или Memory).
Латентность команд = 1250 пс.
Пропускная способность – 1 команда за 250 пс.
Латентность команд в конвейерном процессоре больше, чем в однотактном, потому что стадии конвейера не идеально сбалансированы.

Абстрактное представление работающего конвейера

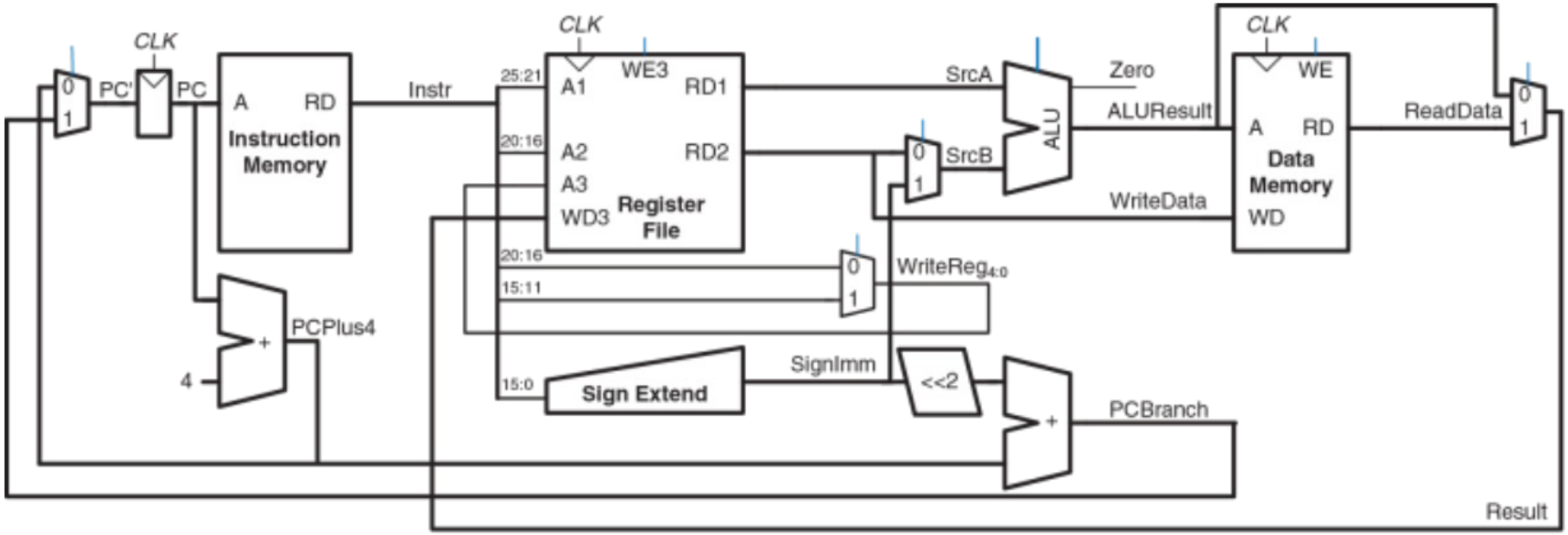


Каждая стадия изображена главным компонентом стадии:
память команд (instruction memory, IM), чтение регистрового файла (register file, RF), АЛУ, память данных (DM) и запись в регистровый файл (writeback).

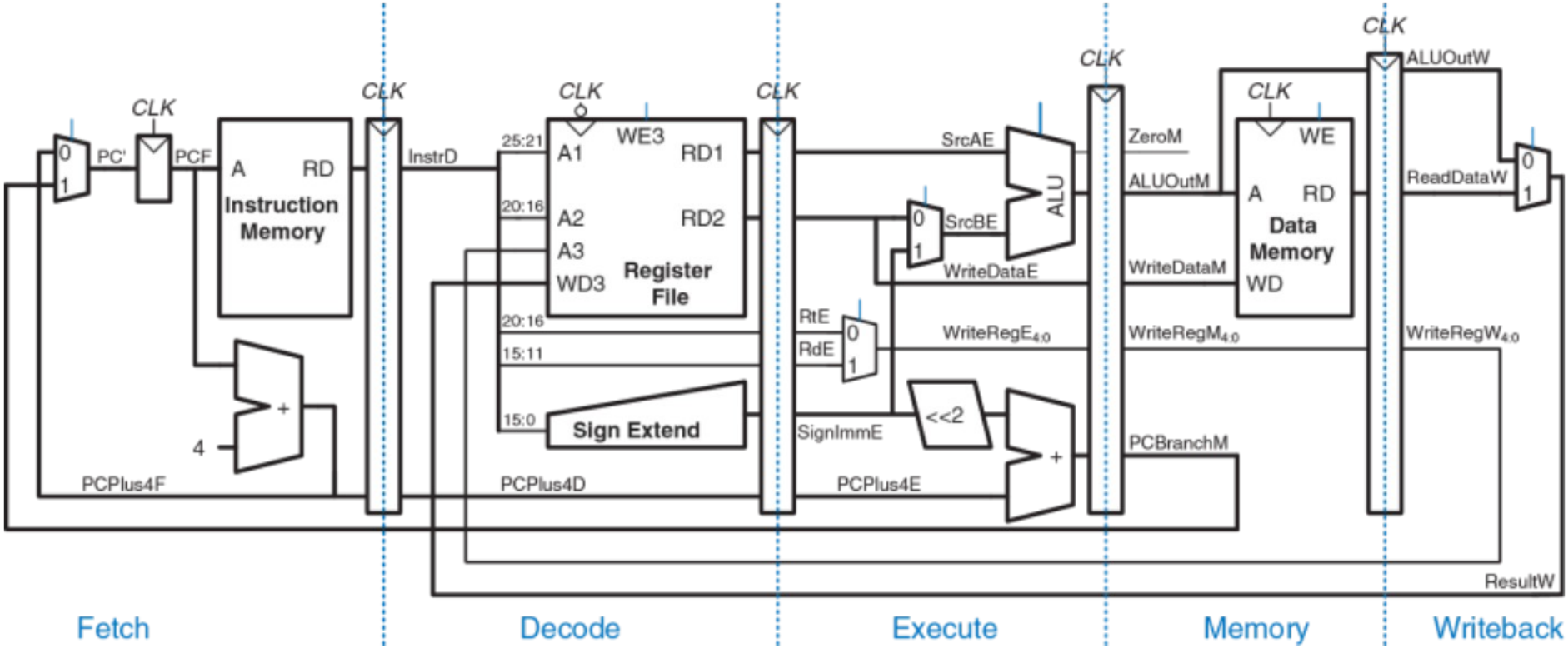
Главная проблема конвейера:
разрешение конфликтов (hazards), возникающих, когда результаты одной из команд требуются для выполнения последующей команды до того, как первая завершится.
Например, если бы команда add использовала регистр $s2 вместо $t2, то возник бы конфликт, потому что регистр $s2 еще не был бы записан командой lw в тот момент, когда команда add должна была его прочитать.

Методы разрешения конфликтов:
– пересылка данных через байпас (bypassing, или forwarding),
– приостановка (stalls) и
– сброс (flushes)
конвейера.

Однотактный тракт данных

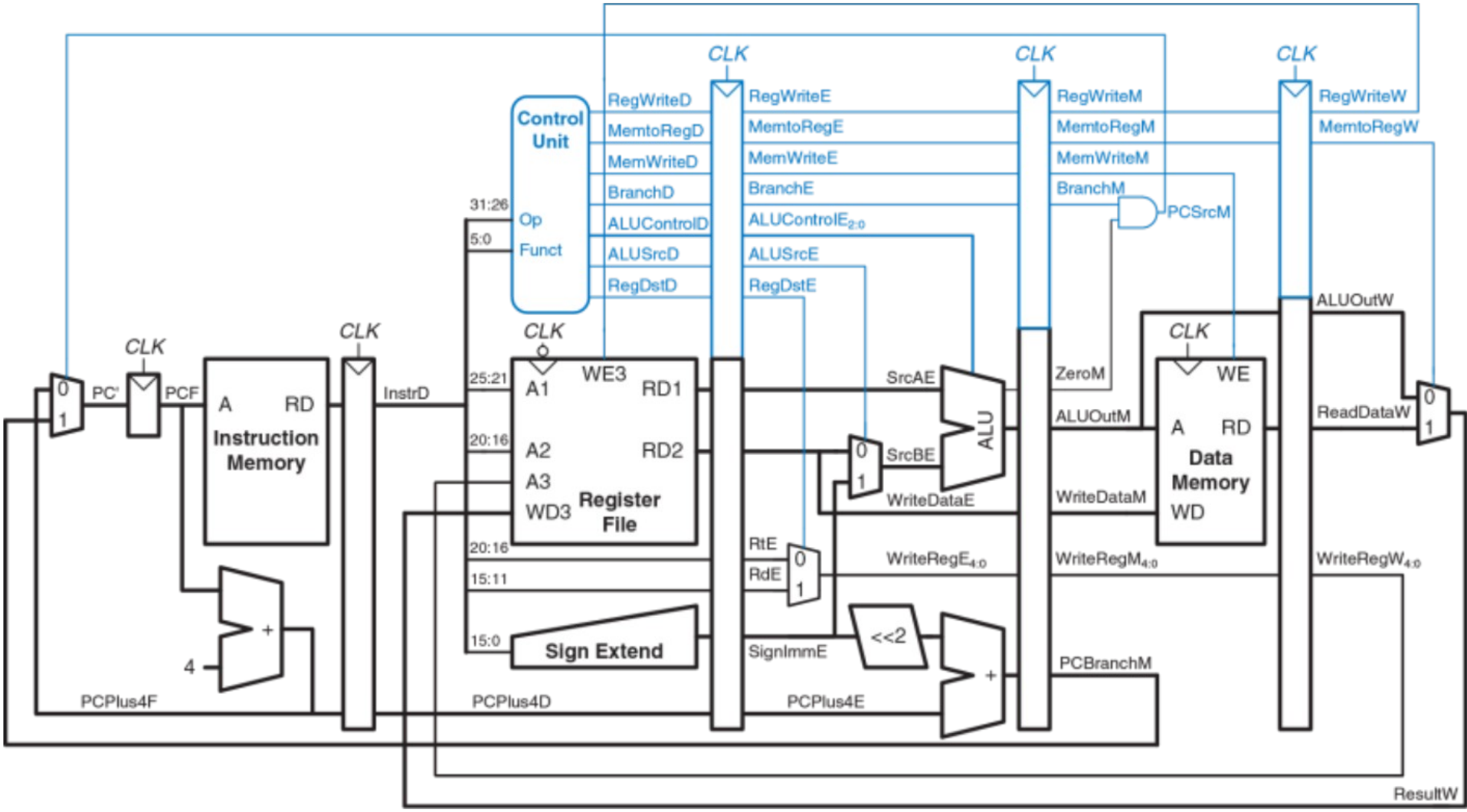


Конвейерный тракт данных



Конвейерный тракт данных можно получить, порезав однотактный тракт данных на пять стадий, разделенных регистрами (pipeline registers).
Одна важная проблема организации конвейерной обработки данных – это то, что все сигналы, относящиеся к конкретной команде, должны обязательно продвигаться по конвейеру одновременно друг с другом.

Конвейерное устройство управления

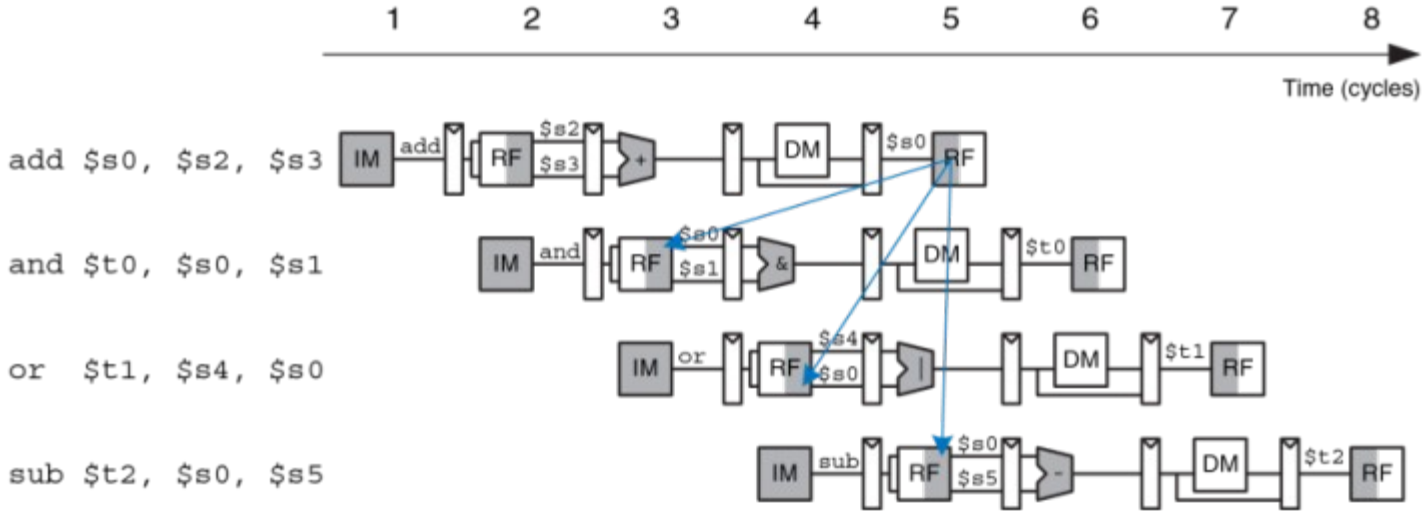


Конвейерный процессор использует те же управляющие сигналы, что и однотактный процессор, поэтому использует такое же устройство управления. Эти сигналы конвейеризуются точно так же, как и тракт данных, чтобы оставаться синхронными с командой, перемещающейся из одной стадии в другую.

Конфликты

В конвейерном процессоре выполняются несколько команд одновременно. Когда одна из них зависит от результатов другой, еще не завершенной команды, то говорят, что произошел конфликт (hazard) в конвейере.

Типы конфликтов: конфликты данных (data hazards) и конфликты управления (control hazards).
Конфликт данных: команда пытается прочитать из регистра значение, которое еще не было записано предыдущей командой.
Конфликт управления: процессор выбирает из памяти следующую команду до того, как стало ясно, какую именно команду надо выбрать.



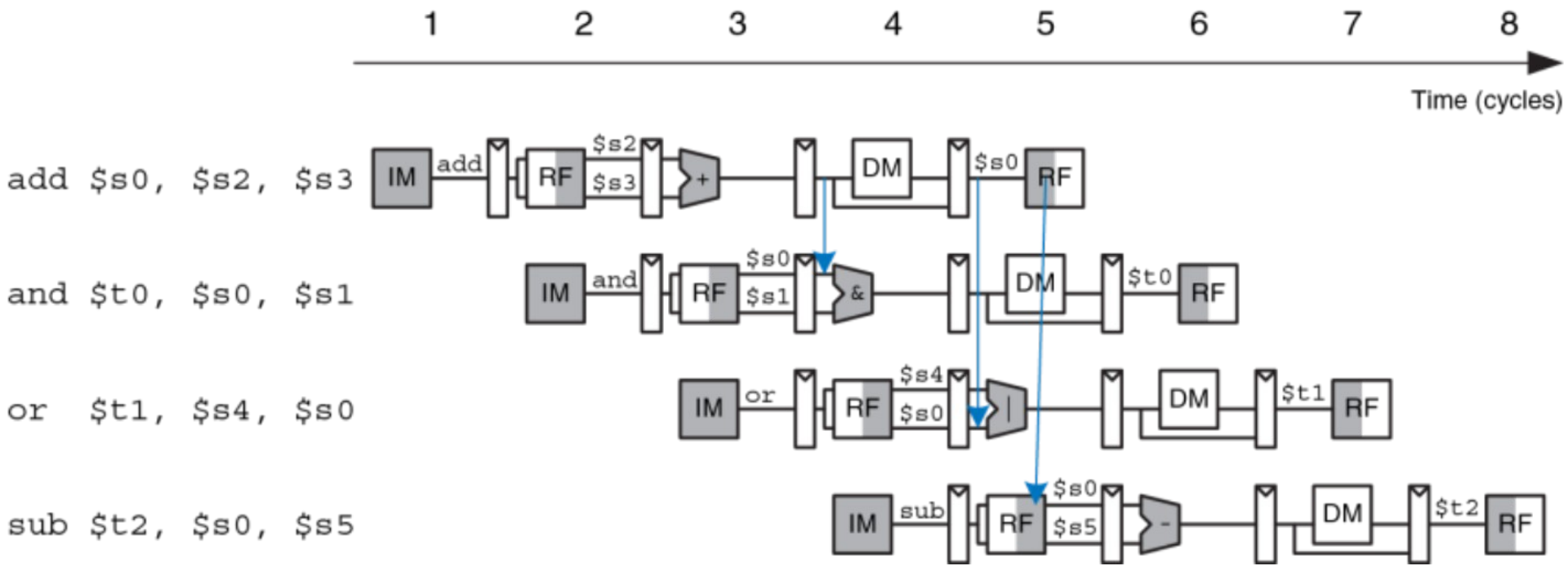
Добавим в процессор блок разрешения конфликтов (hazard unit),

Разрешение конфликтов пересылкой через байпас

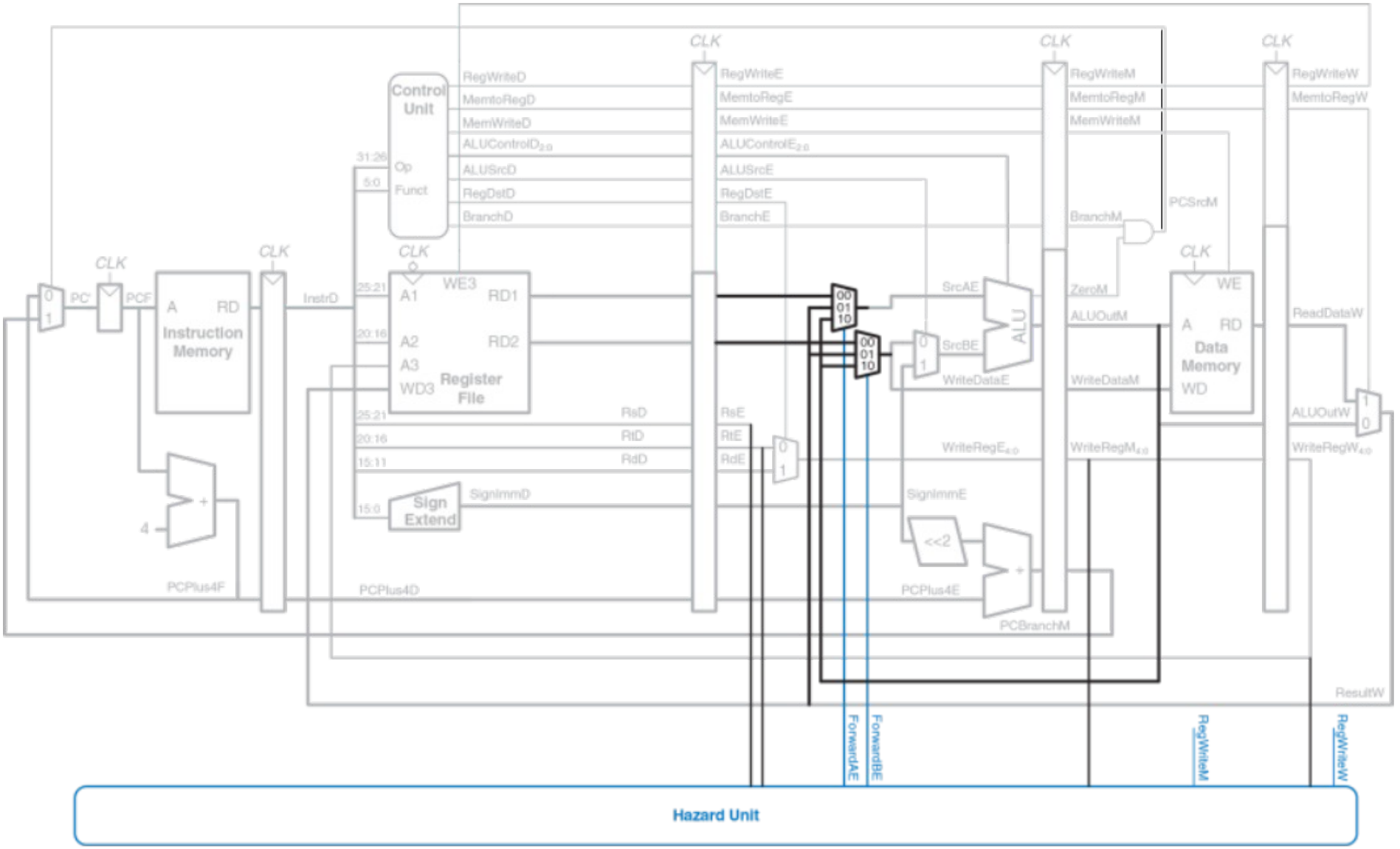
Некоторые конфликты данных можно разрешить пересылкой результата через байпас (bypassing, forwarding) из стадий Memory или Writeback команду, находящуюся в стадии Execute.

Чтобы организовать байпас, понадобится добавить мультиплексоры перед АЛУ.

Пересылка данных через байпас необходима, если номер любого из регистров операндов команды, находящейся в стадии Execute, равен номеру регистра результата команды, находящейся в стадии Memory или Writeback.



Разрешение конфликтов пересылкой через байпас



Разрешение конфликтов пересылкой через байпас

У модуля байпасс есть блок обнаружения конфликтов и два мультиплексора. Блок обнаружения конфликтов управляет мультиплексорами байпаса (forwarding multiplexers), которые определяют, взять ли операнды из регистрового файла или переслать их напрямую из стадии Memory или Writeback.
Если номера регистров результатов в стадиях Memory и Writeback одинаковы, то приоритет отдается стадии Memory (она содержит более новую команду).

Функция, определяющая логику пересылки данных в операнд SrcA:

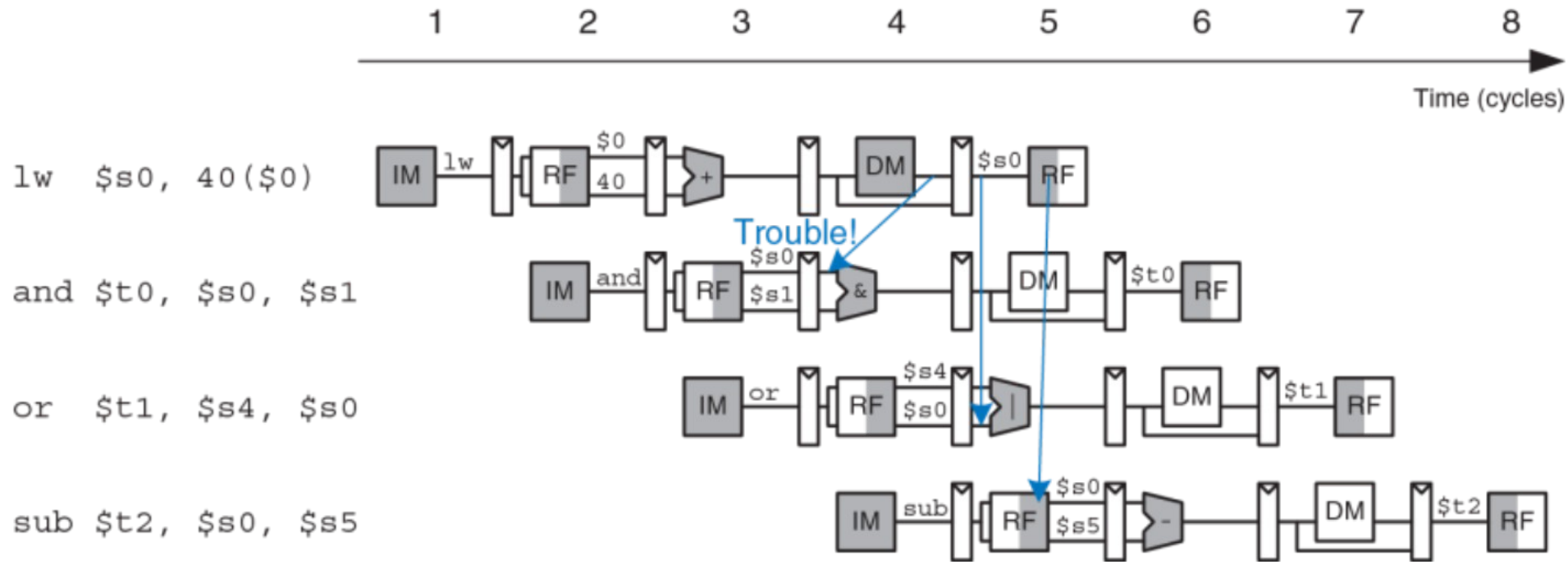
```
if      ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM) then
                                 ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW) then
                                 ForwardAE = 01
else                             ForwardAE = 00
```

Логика для операнда SrcB (ForwardBE) такая же, но проверяется поле rt, а не rs.

Разрешение конфликтов данных приостановками конвейера

Пересылка данных через байпас может разрешить конфликт при чтении после записи, только если результат вычисляется в стадии Execute – только в этом случае его можно сразу переслать в стадию Execute следующей команды. Команда lw не может прочитать данные раньше, чем в конце стадии Memory, поэтому ее результат нельзя переслать в стадию Execute следующей команды. (латентность команды lw равна двум тактам).

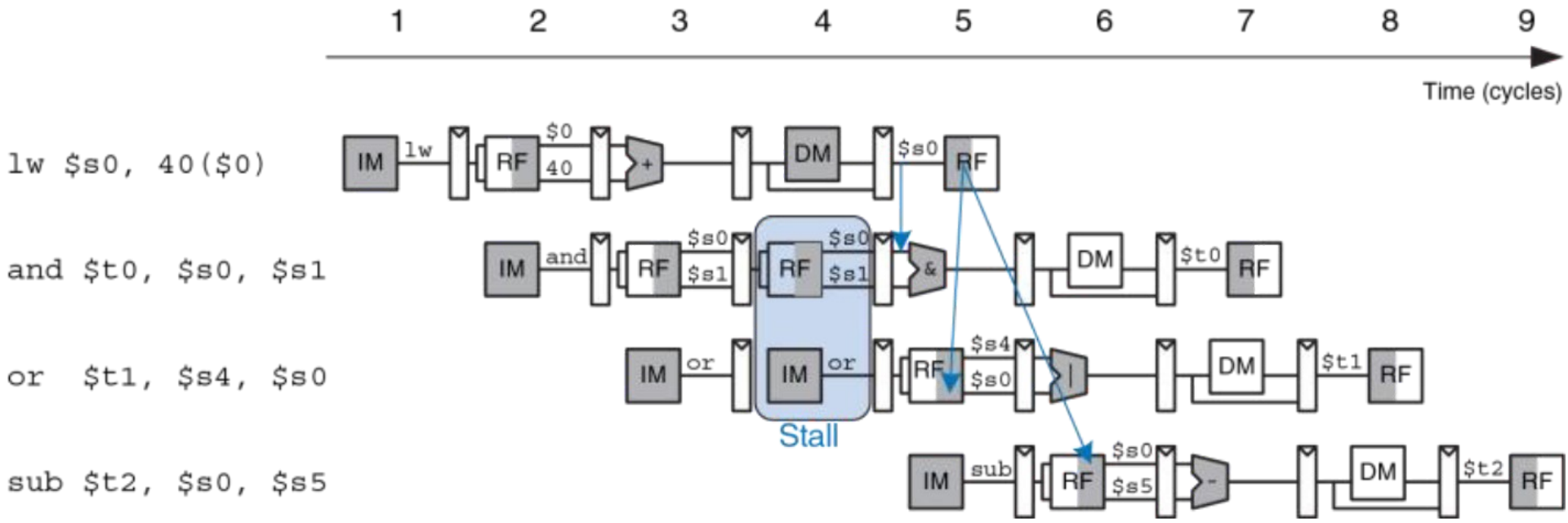
Альтернативное решение – приостановить конвейер, задержав все операции до тех пор, пока данные не станут доступны.



Разрешение конфликтов данных приостановками конвейера

Приостановим зависимую команду (*and*) в стадии Decode.
Команда *and* попадает в эту стадию на 3-ем такте и остается там на 4-ом такте.
На 5-ом такте результат команды *lw* можно через байпас переслать из стадии Writeback в стадию Execute, где будет находиться команда *and*.

Стадия Execute на 4-ом такте не используется. Стадия Memory не используется на 5-ом такте, а Writeback – на 6-ом. Эта неиспользуемая стадия, проходящая по конвейеру, называется пузырьком (bubble).
Пузырек получается путем обнуления всех управляющих сигналов стадии Execute на время приостановки стадии Decode.



Разрешение конфликтов данных приостановками конвейера

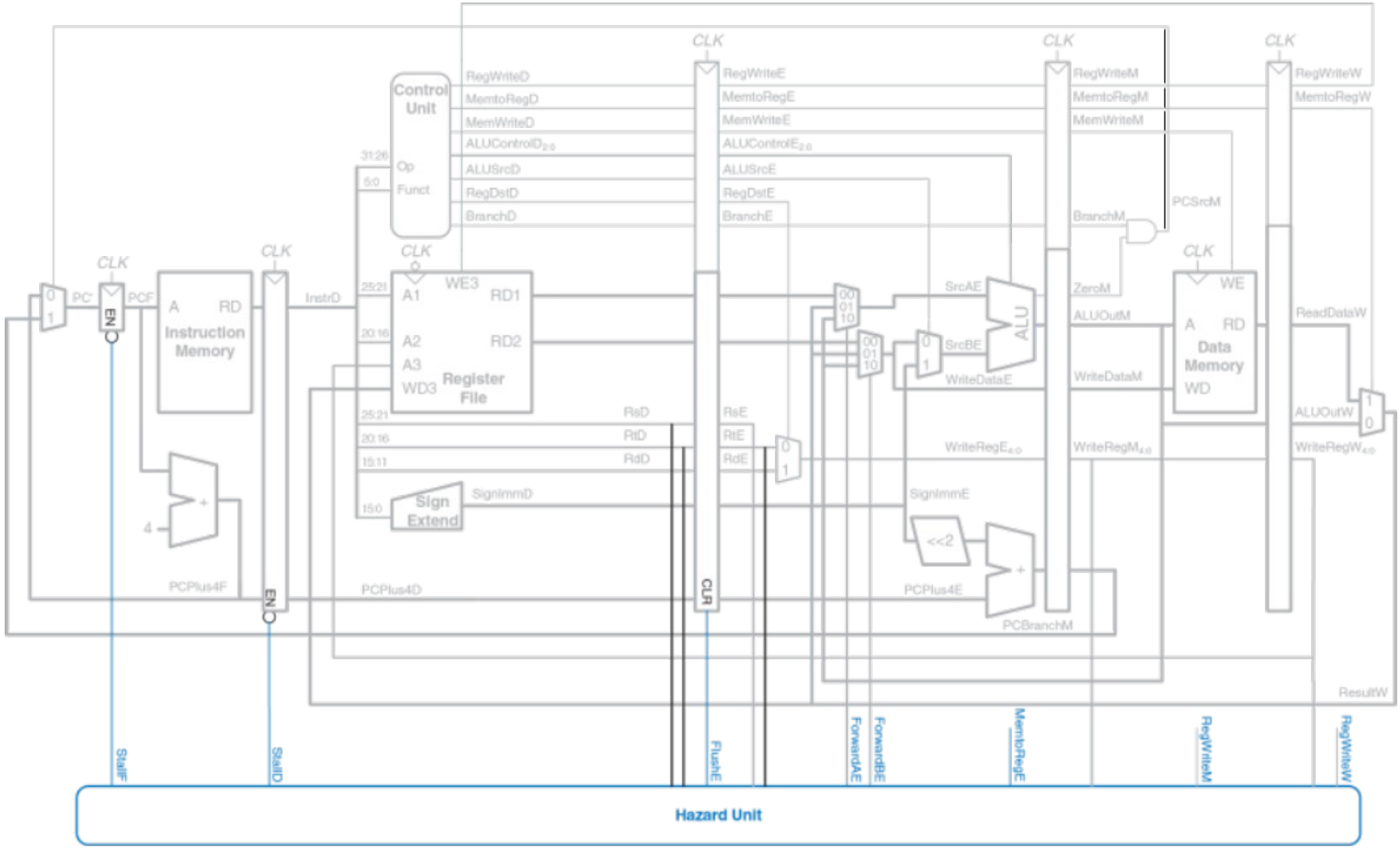
Стадию конвейера можно приостановить, если запретить обновление регистра, находящегося между этой и предыдущей стадиями. Как только какая-либо стадия приостановлена, все предыдущие стадии тоже должны быть приостановлены, чтобы ни одна из команд не пропала. Регистр, находящийся сразу после приостановленной стадии, должен быть очищен, чтобы «мусор» не попал в конвейер.

Блок управления конфликтами смотрит, какая команда находится в стадии Execute. Если это lw, а номер регистра результата (rtE) совпадает с номером любого из регистров операндов команды, находящейся в стадии Decode (rsD или rtD), то стадия Decode должна быть приостановлена до тех пор, пока операнд не будет прочитан из памяти.
Для приостановки стадий Fetch и Decode нужно добавить вход разрешения работы (EN) временным регистрам, расположенным перед этими стадиями, а также вход синхронного сброса (CLR) временному регистру, расположенному перед стадией Execute.

Для команды *lw* всегда устанавливается сигнал MemtoReg.
Логика формирования сигналов приостановки (stall) и очистки (flush) выглядит так:

lwstall = ((rsD = = rtE) OR (rtD = = rtE)) AND MemtoRegE
StallF = StallD = FlushE = lwstall

Разрешение конфликтов данных приостановками конвейера



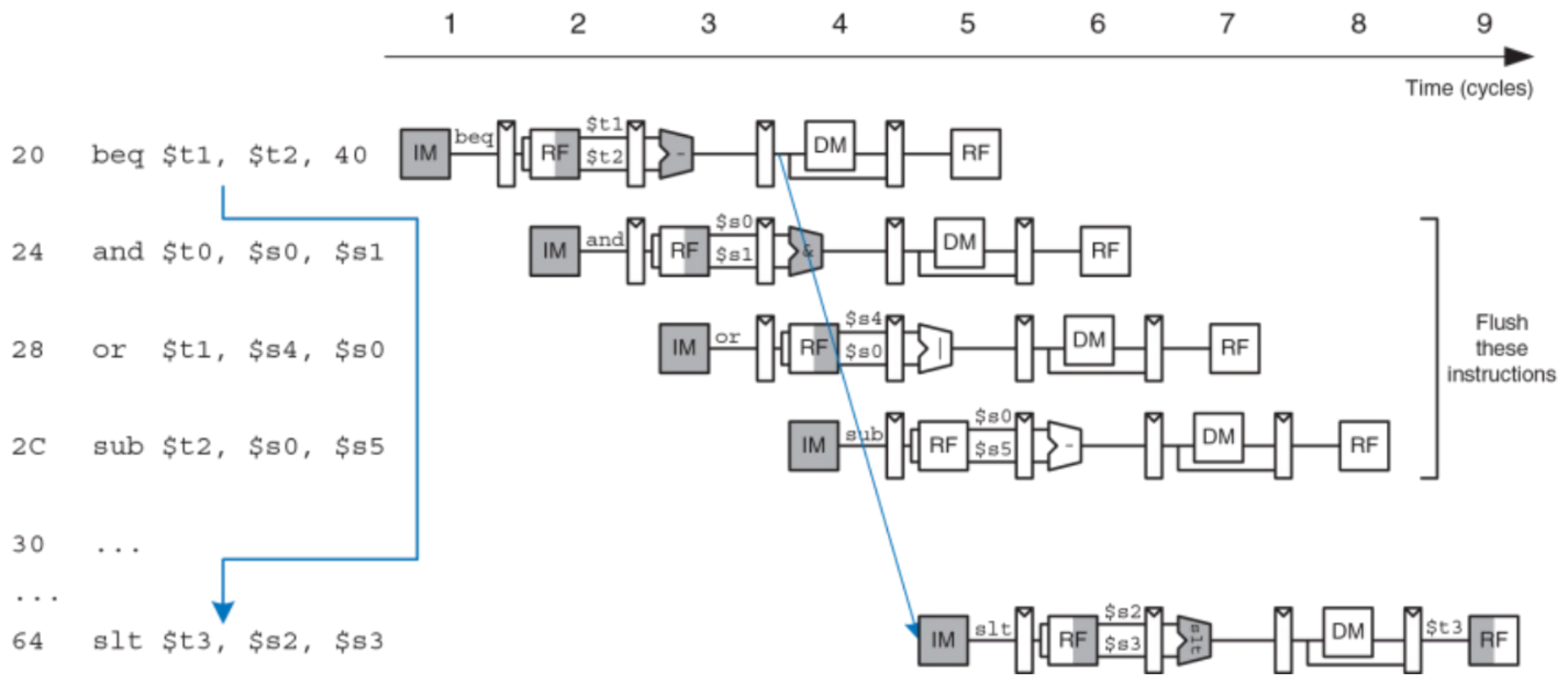
Разрешение конфликтов управления

Выполнение команды beq приводит к конфликту управления: конвейерный процессор не знает, какую команду выбрать следующей, так как в этот момент еще не ясно, нужно ли будет выполнить условный переход или нет.

Один из способов разрешить конфликт – приостановить конвейер, пока не будет принято нужное решение (т.е. до тех пор, пока не будет вычислен сигнал PCSrc). Решение принимается в стадии Memory, так что для каждой команды условного перехода придется приостанавливать конвейер на три такта.

Другой способ – предсказать, будет ли выполнен условный переход или нет, и начать выполнять команды, основываясь на этом. Как только условие перехода будет вычислено, процессор может прервать эти команды, если предсказание было неверным. Пусть предсказано, что условный переход не будет выполнен, и команды выполняются в порядке следования. Если окажется, что переход должен был быть выполнен, то конвейер должен быть очищен (flushed) от трех команд, идущих сразу за командой перехода, путем очистки соответствующих временных регистров. Зря потраченные в этом случае такты называются простоем из-за неправильно предсказанного перехода (branch misprediction penalty).

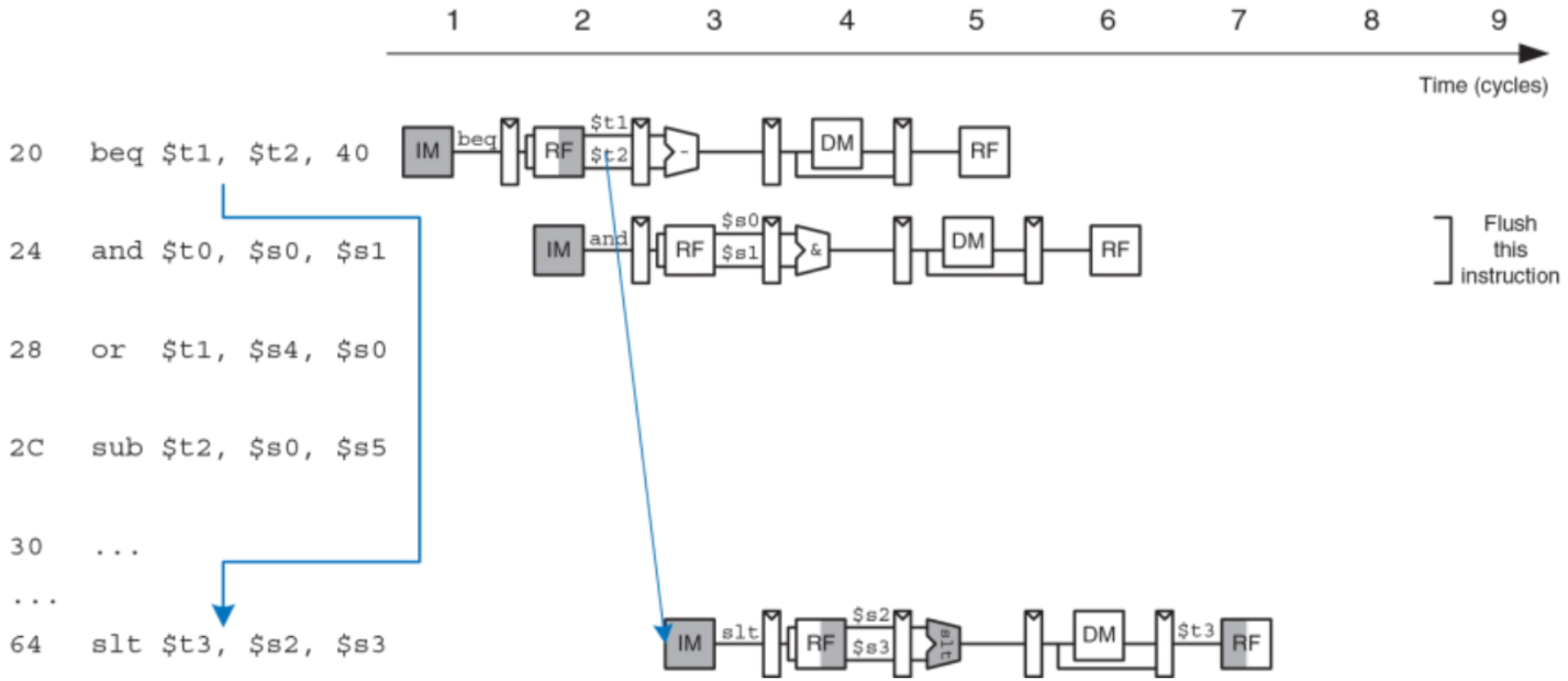
Разрешение конфликтов управления



Очистка трех команд.

Простой из-за неправильно предсказанного перехода можно уменьшить, если вычислить условие перехода пораньше. Так как вычисление условия заключается в определении равенства двух регистров, то можно использовать отдельный компаратор – это гораздо быстрее, чем вычитать два числа и проверять результат на равенство нулю. Если компаратор достаточно быстр, то можно перенести его в стадию Decode, чтобы прочитанные из регистрового файла операнды тут же сравнивались, а результат сравнения использовался для вычисления нового значения счетчика команд к концу стадии Decode.

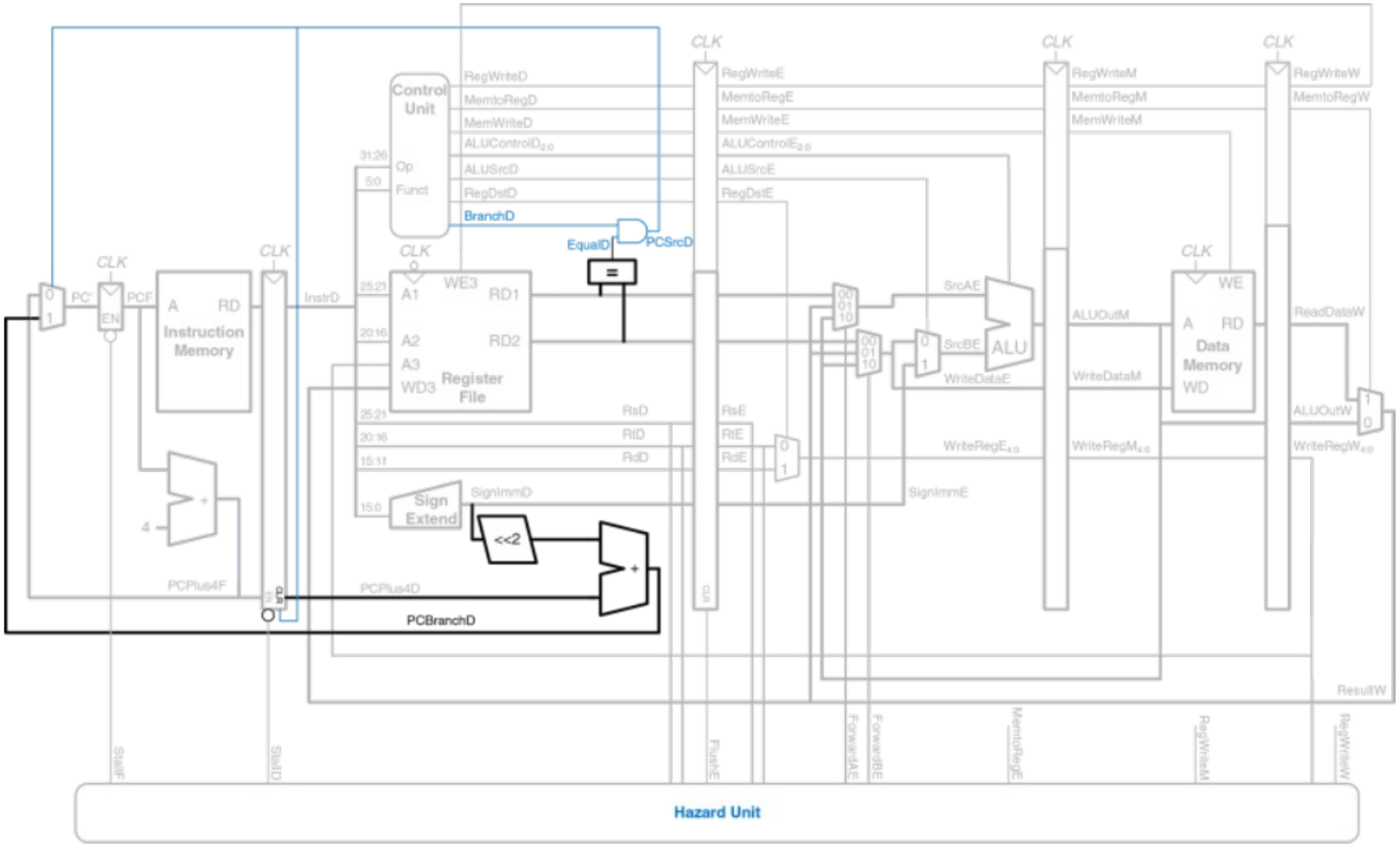
Разрешение конфликтов управления



Функционирование конвейера с ранним вычислением условий перехода, происходящим на втором такте.

На третьем такте процессор очищает конвейер от команды *and* и выбирает из памяти команду *slt*, то есть простой из-за неправильно предсказанного перехода уменьшился с трех тактов до одного.

Разрешение конфликтов управления



Однако, раннее вычисление условия переходов приводит к новому конфликту данных при чтении после записи.