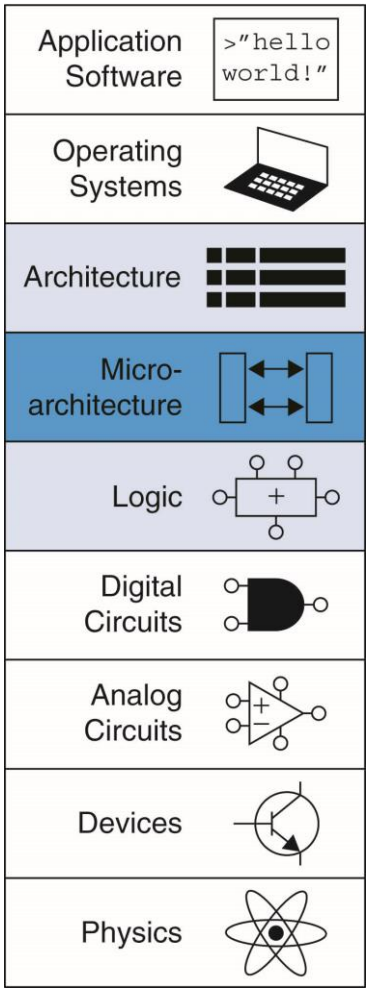
КАРХ: Тема_11: Микроархитектура

Въведение.

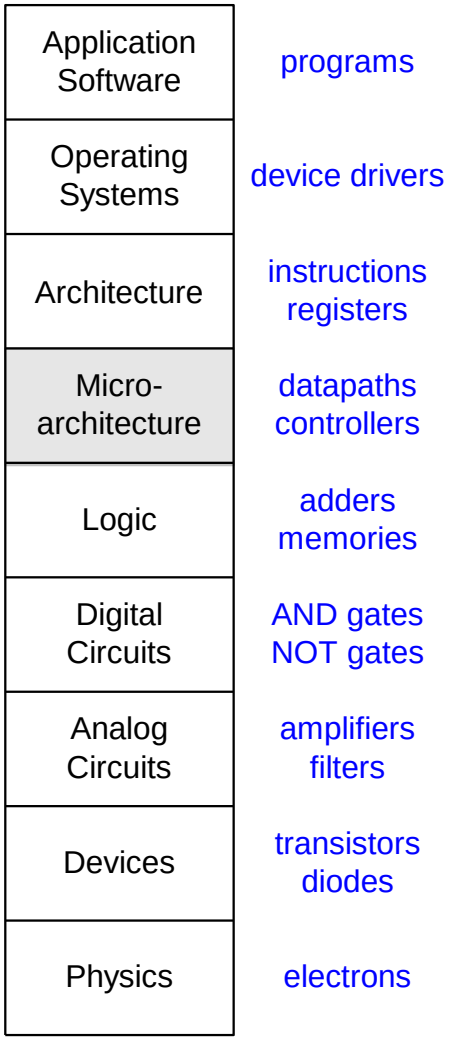
- Микроархитектурата – това е връзката между логиката и архитектурата. Тя е специфичното подреждане на регистри, АЛУ-та, крайни автомати, памети и други изграждащи блокове, необходими за осъществяване на архитектурата.
- Архитектурите имат множество различни микроархитектури, които се различават по производителност, цена и сложност. Въпреки че изпълняват един и същ код, вътрешният им дизайн може широко да варира.
- Компютърната архитектура се дефинира от набор инстукции, който оформя **архитектурното ниво** (architectural state). При MIPS процесорите тя се състои от програмен брояч и 32 регистъра. Някои процесори имат и nano-architectural state.



КАРХ: Тема_11: Микроархитектура

Въведение.

- **Микроархитектурата (Microarchitecture)** на процесора се разделя на две части: **Datapath** и **Control.**
  - **Datapath:** оперира с думи от данни. Има структури като памети, регистри, АЛУ-та, мултиплексори. MIPS е 32-bit архитектура и използва 32-bit datapath.
  - **Control (unit):** получава текущата инструкция от datapath и определя как datapath да я изпълни, като генерира управляващи сигнали (control signals).



КАРХ: Тема_11: Микроархитектура

Въведение.

- **Три Микроархитектури** на MIPS процесор:
  - **С един цикъл (Single-cycle):** Изпълнява цяла инстукция за един цикъл. Той е лесен за обяснение и има прост контролен блок. Не изисква допълнителни елементи (non-architectural state).
  - **С много цикли (Multicycle):** Процесорът изпълнява инструкциите на последователни кратки цикли. Простите инструкции изискват по-малък брой цикли от сложните. Мултицикълната архитектура намалява хардуера като цена използвайки многократно скъпите хардуерни блокове, като суматори и памети, в рамките на една инструкция. За целта е небходимо добавянето на нано-архитектурни регистри за запазването на междинните резултати. За изпълнението на 1 инструкция – няколко тактови цикъла.
  - **Конвеерни (Pipelined):** Конвеерната архитектура се прилага върху процесорът с единичен цикъл, което води до едновременно изпълнение на няколко инструкции. Това изисква допълнителна логика и регистри за отчитане на зависимостите между едновременно изпълняваните инструкции, както и pipeline регистри. Всички високо-производителни процесори днес са конвеерни.

КАРХ: Тема_11: Микроархитектура

Производителност (Ефективност) на процесора (Processor Performance).

- За оценка на производителността на процесорите се използват **benchmark-програми.**
- Времето за изпълнение на една програма:

  Execution Time = (брой инструкции)(цикли/инструкция)(секунди/цикъл)
- Определения:
  - Цикли за инструкция (Cycles per instruction) – CPI
  - T_c – clock period: seconds/cycle
  - IPC: (instructions per cycle) = IPC

Program Execution Time

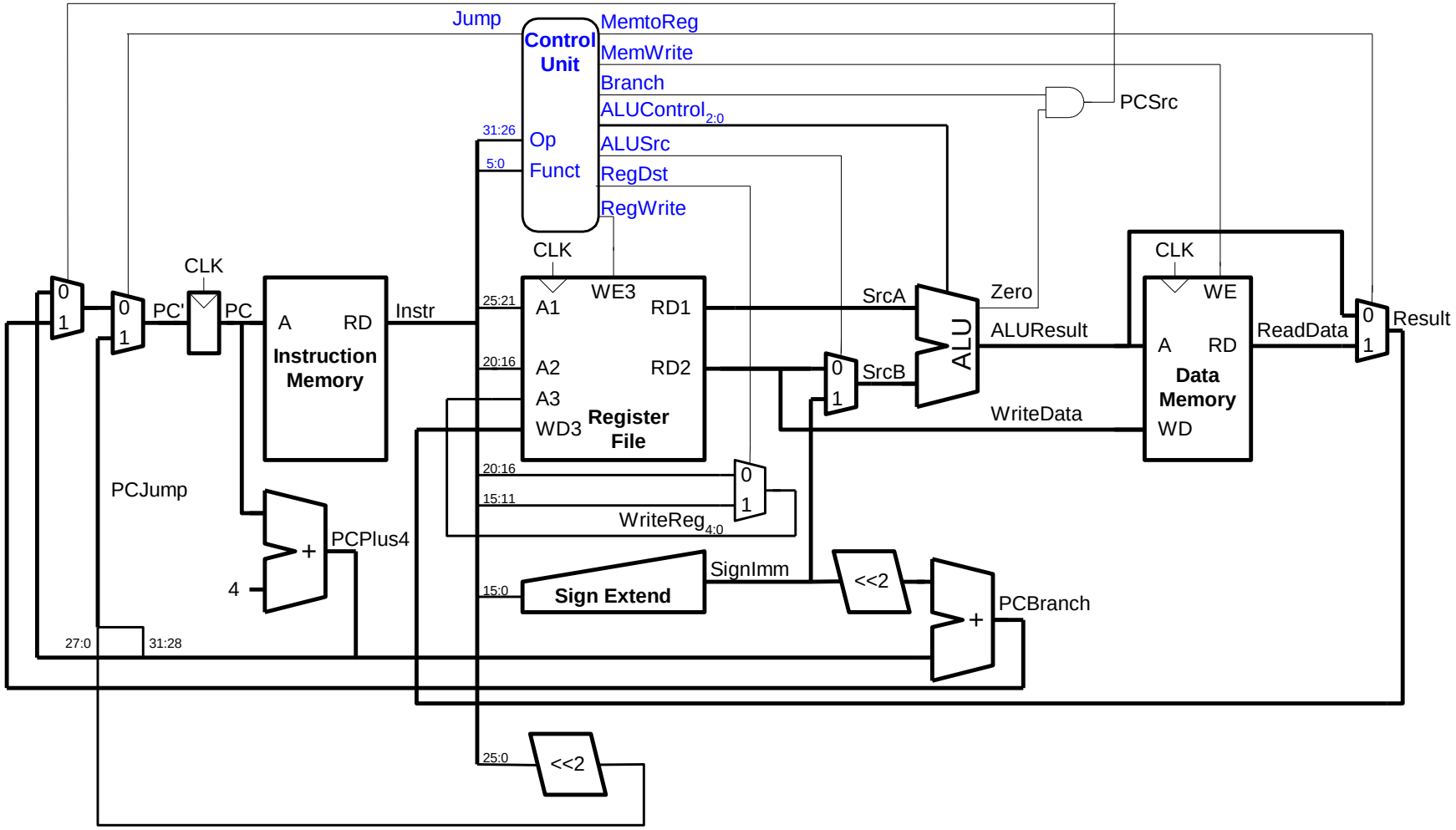
= **(брой инструкции)(цикли/инструкция)(секунди/цикъл)**

= **брой инструкции x CPI x T_C**

- Предизвикателство (Challenge) е удовлетворяване на противоречащите си изисквания за :
  - Цена (Cost)
  - Мощност (Power)
  - Производителност (Performance)

КАРХ: Тема_11: Микроархитектура

Процесор с единичен цикъл (Single-Cycle Processor).



T_C limited by critical path (lw)

КАРХ: Тема_11: Микроархитектура

Процесор с много цикли (Multicycle Processor).

- **Процесор с 1 цикъл (Single-cycle):**
  - \+ прост
  - \- T_c – clock period (cycle time) се лимитира от най-дългата инструкция (lw)
  - \- 2 суматора/ALUs & 2 памети
- **Процесор с много цикли (Multicycle):**
  - \+ къс период T_c
  - \+ простите инструкции се изпълняват по-бързо
  - \+ многократно използване на скъпия хардуер в рамките на една инструкция (благодарение на многото цикли)
  - \- изисква повече време
- **Използва същите стъпки в дизайна: datapath & control**

КАРХ: Тема_11: Микроархитектура

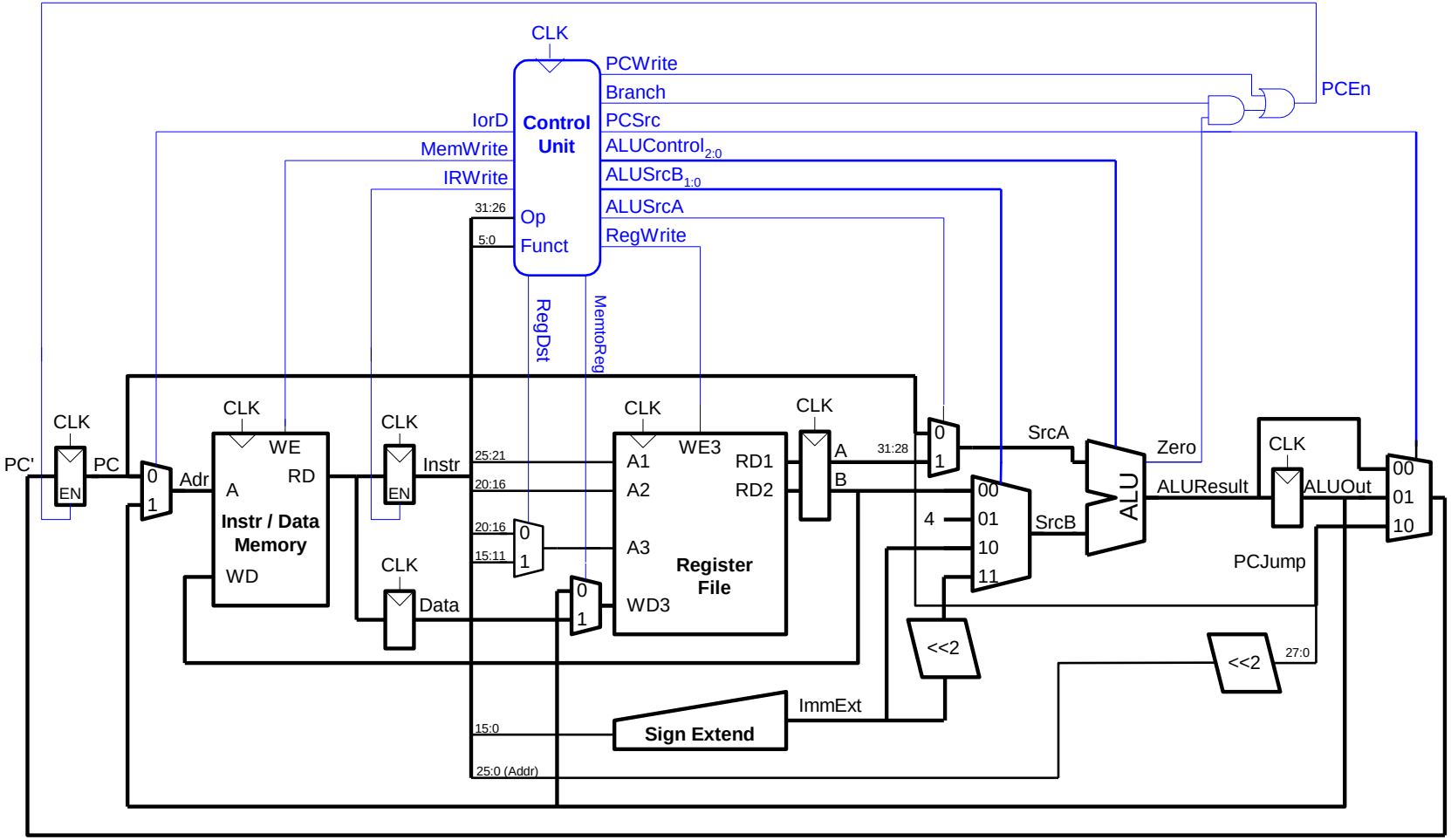
Производителност на процесор с много цикли (Multicycle ProcessorPerformance).

- Инструкциите използват различен брой цикли:
  - 3 cycles: beq, j
  - 4 cycles: R-Type, sw, addi
  - 5 cycles: lw
- CPI е претеглено средно (weighted average)
- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

- **Average CPI =** (0.11 + 0.02)(3) + (0.52 + 0.10)(4) + (0.25)(5) **= 4.12**

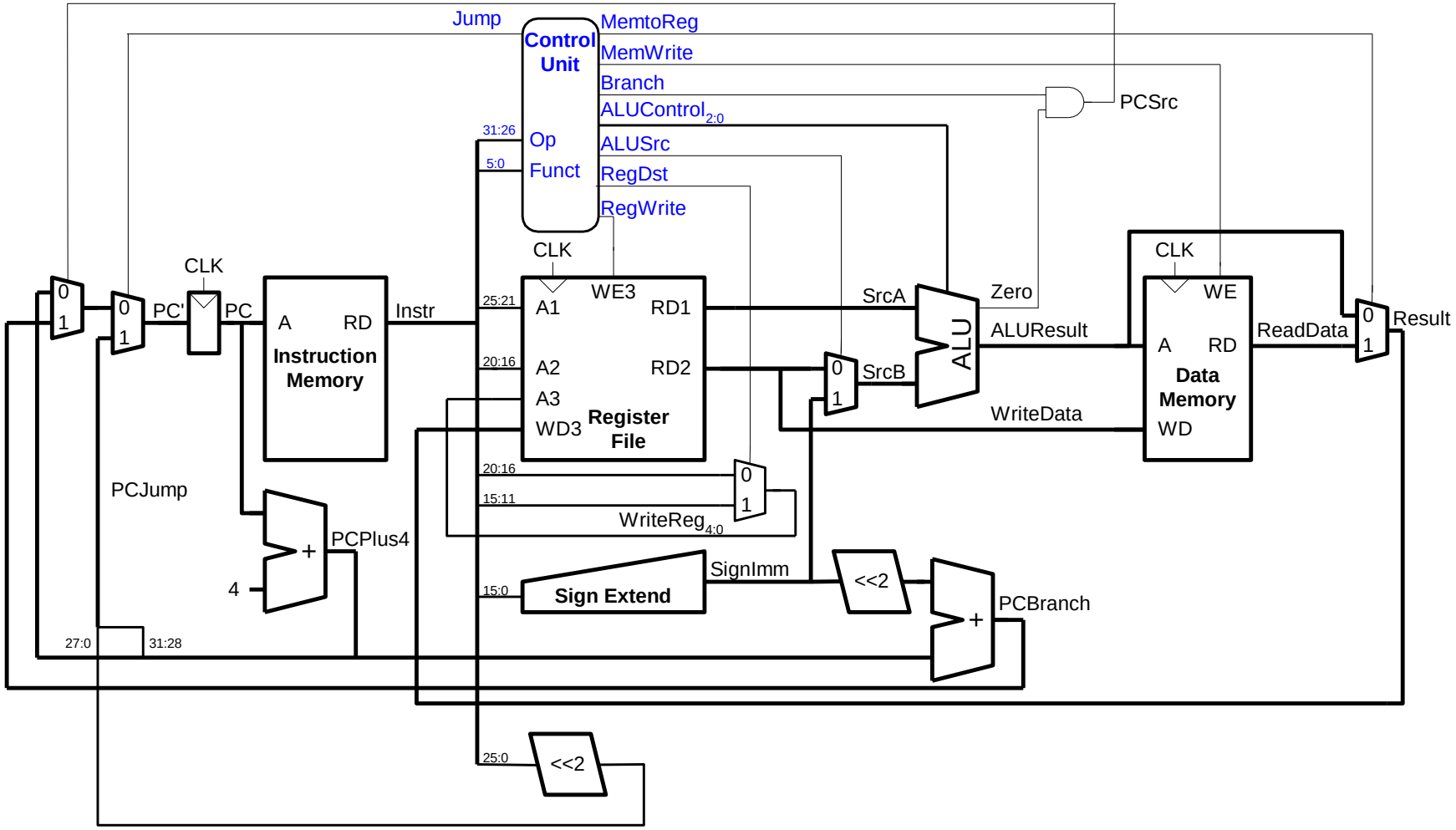
КАРХ: Тема_11: Микроархитектура

Процесор с много цикли (Multicycle Processor).



КАРХ: Тема_11: Микроархитектура

Процесор с единичен цикъл (Single-Cycle Processor).



T_C limited by critical path (lw)

КАРХ: Тема_11: Микроархитектура

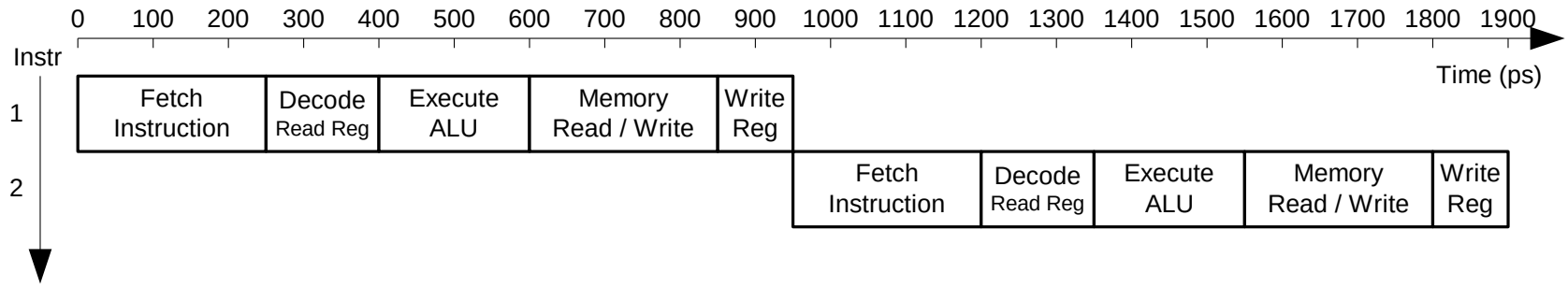
Конвеерен процесор (Pipelined MIPS Processor).

- Използва времеви паралелизъм (Temporal parallelism)
- Разделя single-cycle processor на 5 конвеерни стъпала (stages), което означава 5 инструкции да се изпълняват едновременно – по една във всяко стъпало.Четенето и писането в паметта и регистрите, както и работата на АЛУ-то изискват най-много време на процесора, т.е. това са най-бавните стъпки. Затова те се избират в 5-те конвеерни стъпала:
  - Fetch (Извличане) – процесорът чете инструкция от паметта;
  - Decode (Декодиране) – процесорът чете операндите източници от регистър файла и декодира инструкциите за да генерира контролните сигнали;
  - Execute (Изпълнение) – извършва се изчислението в АЛУ;
  - Memory (Памет) – процесорът чете или записва данни в паметта;
  - Writeback (Върнат запис) – процесорът записва резултата в регистър файла (ако е приложимо).
- Добавят се pipeline регистри между отделните стъпала.

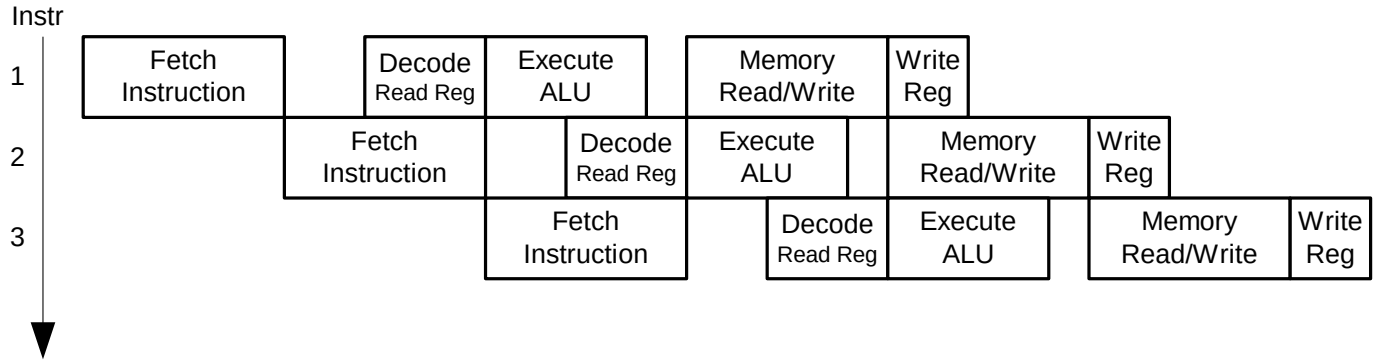
КАРХ: Тема_11: Микроархитектура

Сравнение на процесорите (Single-Cycle vs. Pipelined).

Single-Cycle

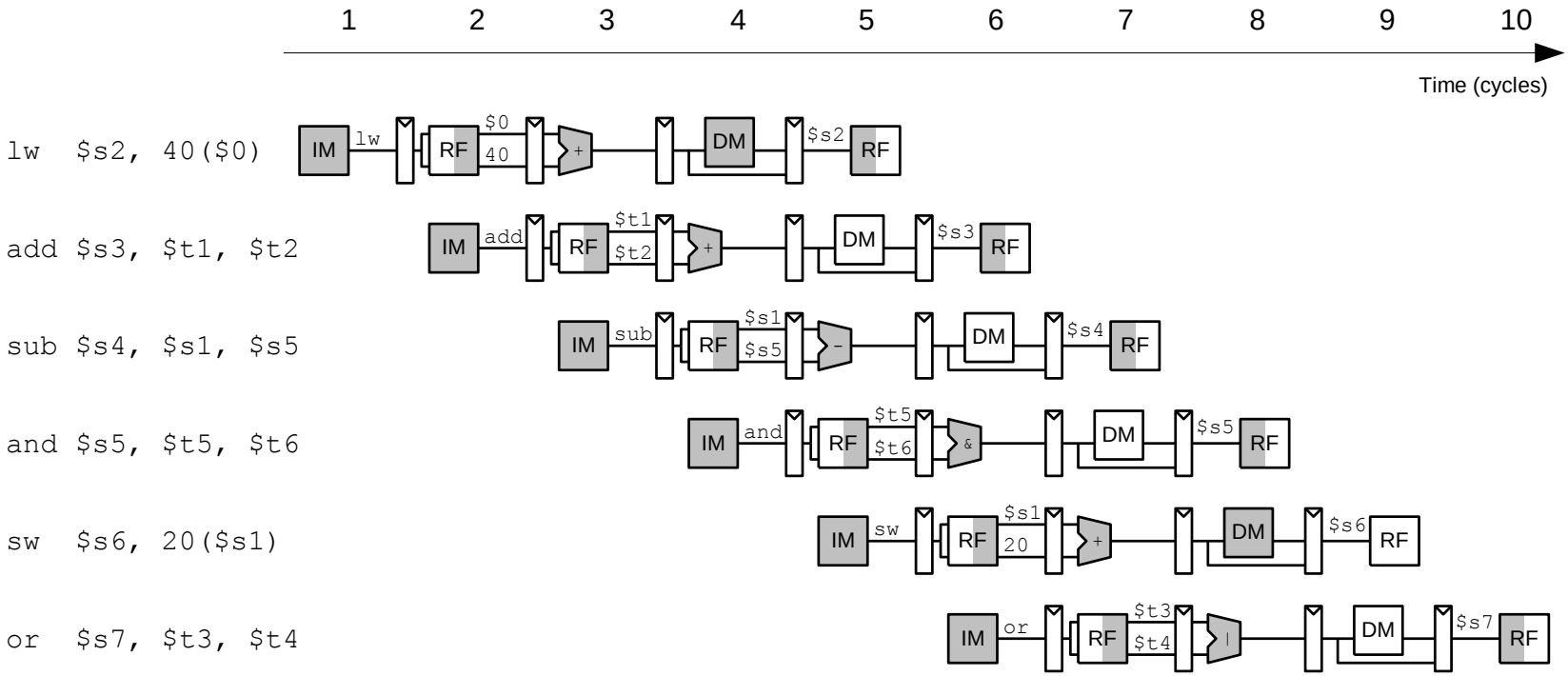


Pipelined



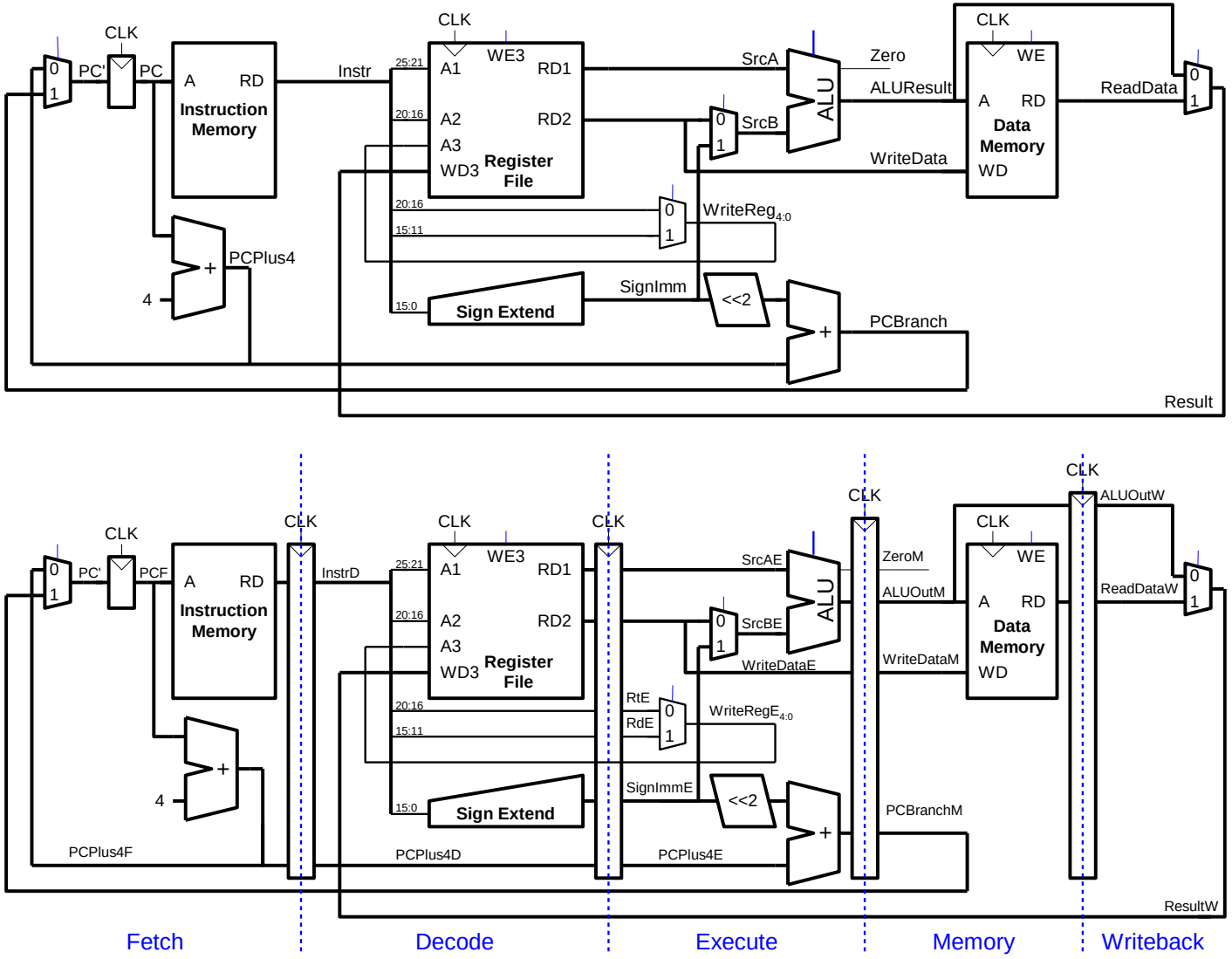
КАРХ: Тема_11: Микроархитектура

Примерна илюстрация (Pipelined Processor Abstraction).



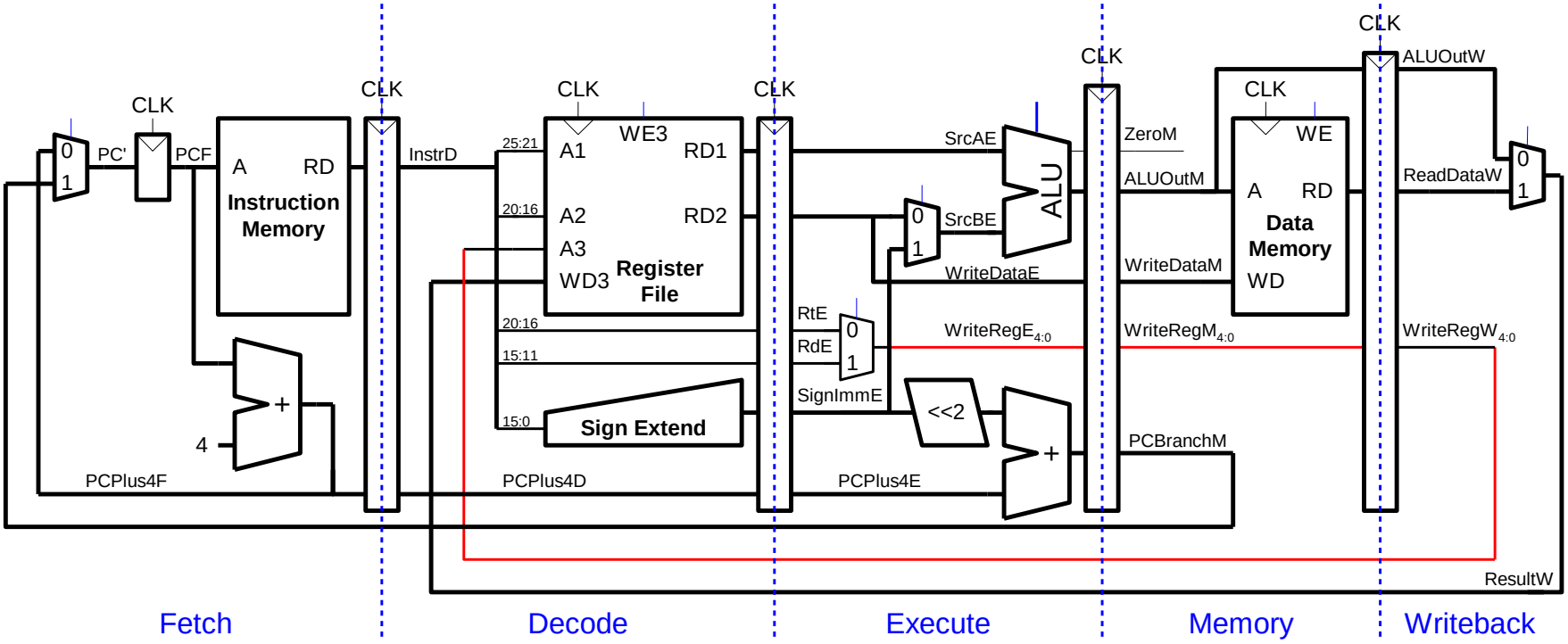
КАРХ: Тема_11: Микроархитектура

Разделяне на конвеерните стъпала (Single-Cycle & Pipelined Datapath).



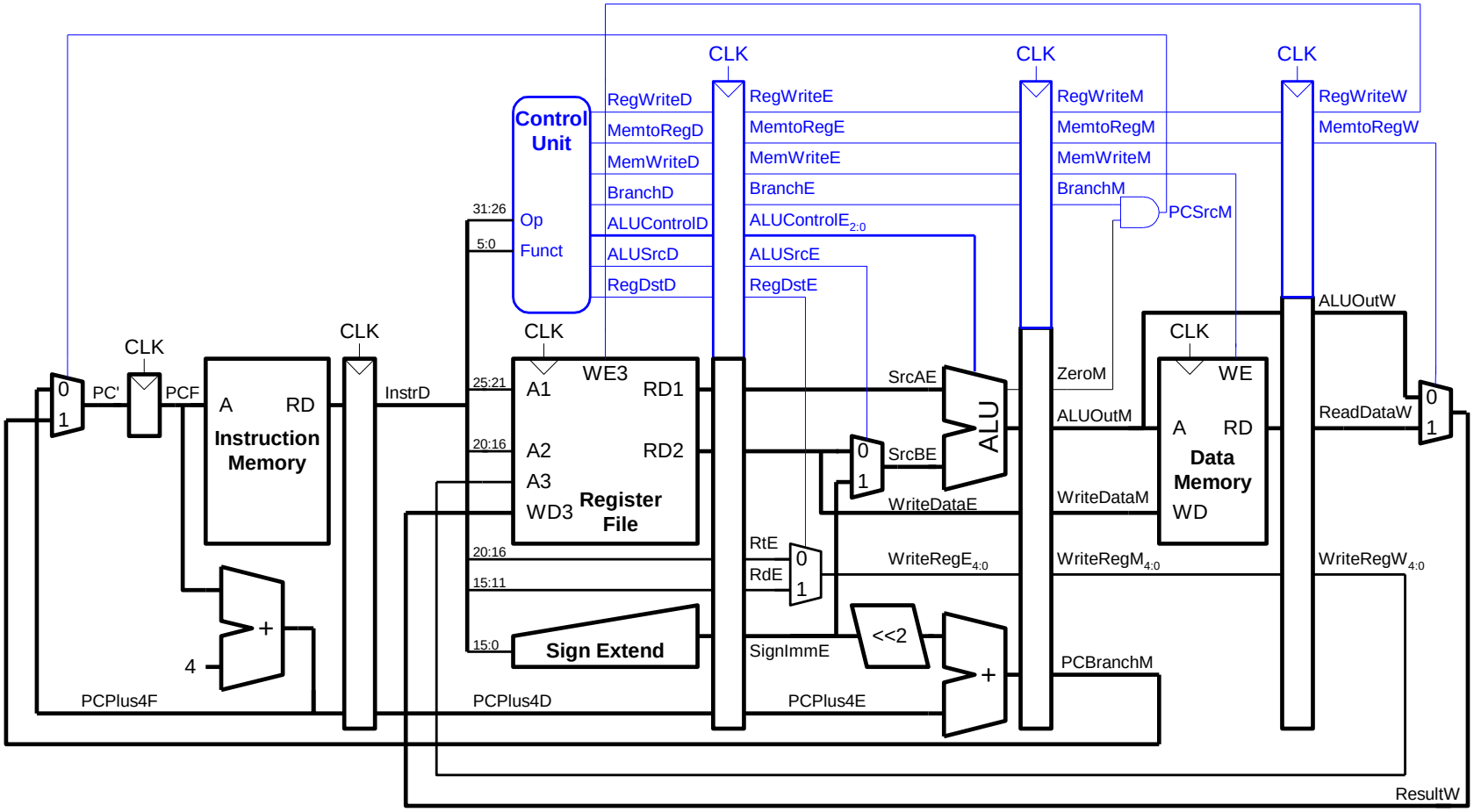
КАРХ: Тема_11: Микроархитектура

Извършване на необходимите корекции (Corrected Pipelined Datapath).



КАРХ: Тема_11: Микроархитектура

Контролен блок (Pipelined Processor Control).



- Същият контролен блок, както при **single-cycle processor.**
- **Контролните сигнали** закъсняват според съответното стъпало (pipeline stage).

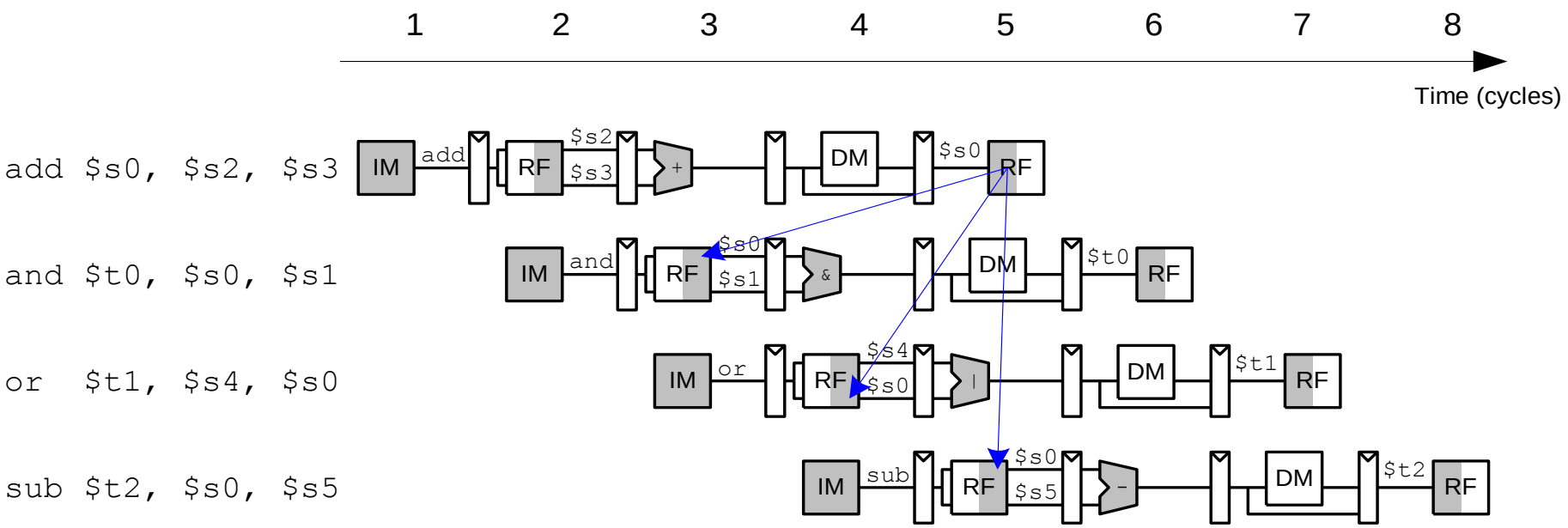
КАРХ: Тема_11: Микроархитектура

Опасности (Рискове) при конвеерната организация (Pipeline Hazards).

- Възникват когато една инструкция е зависима от резултата от друга такава, която още не е изпълнена.
- Регистър файлът може да чете и записва в един и същ цикъл, като записът става през първата част от цикъла, а четенето – през втората. Така това не води до рискове.
- Видове рискове:
  - **Data hazard:** стойността на регистър не е записана (върната) обратно в регистър файла
  - **Control hazard:** следващата инструкция не е определена още (поради възможни преходи)

КАРХ: Тема_11: Микроархитектура

Пример на Data Hazard при конвеерна организация (Data Hazard).



RAW (*Read after Write*) Hazard!

КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Data Hazards).

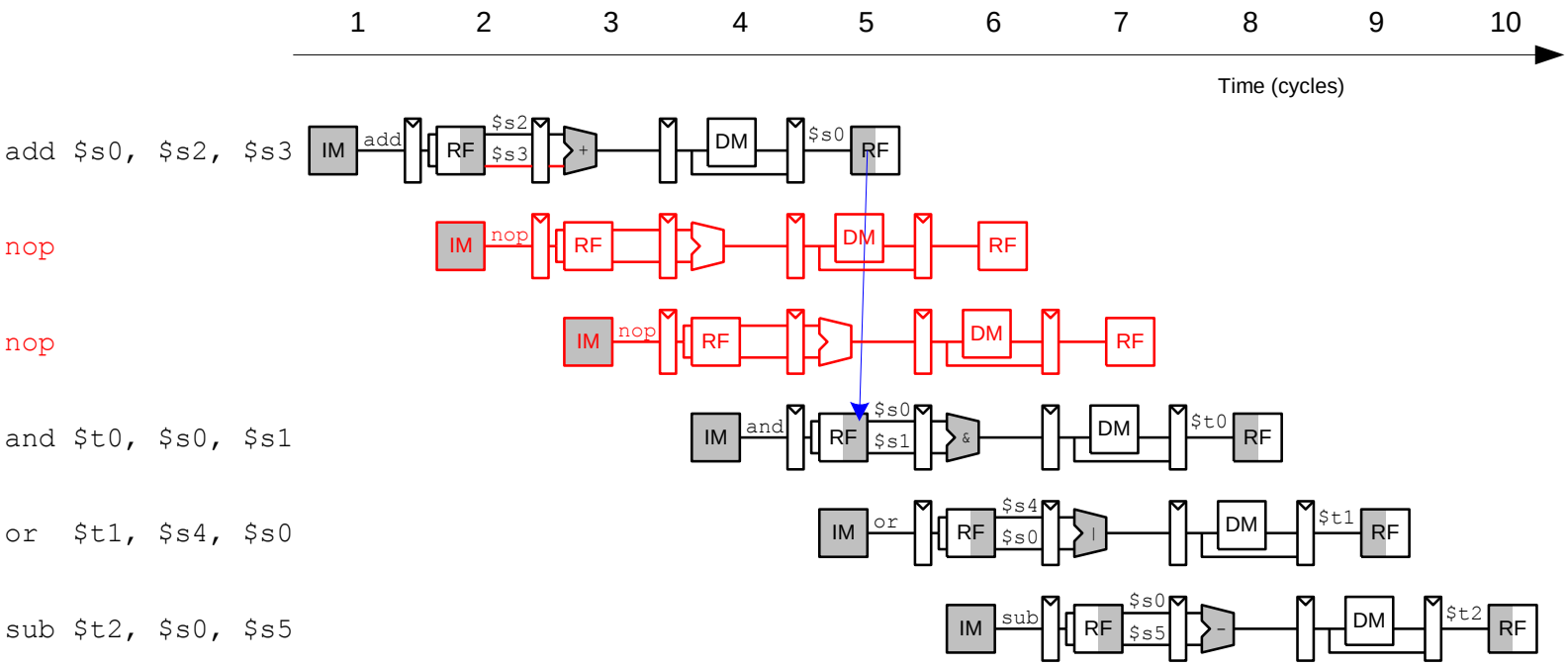
- Вкарване в кода на празни операции (nops) по време на компилирането.
- Преаранжиране на кода по време на компилирането.
- Препращане на данните (Forward data) по време на изпълнението.
- Спиране (Задържане) (Stall) на процесора по време на изпълнението.

КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Data Hazards).

Пример:

- Вкарване на достатъчно nops до получавне на желания резултат.
- Или изместване на друга (независеща) полезна инструкция напред.

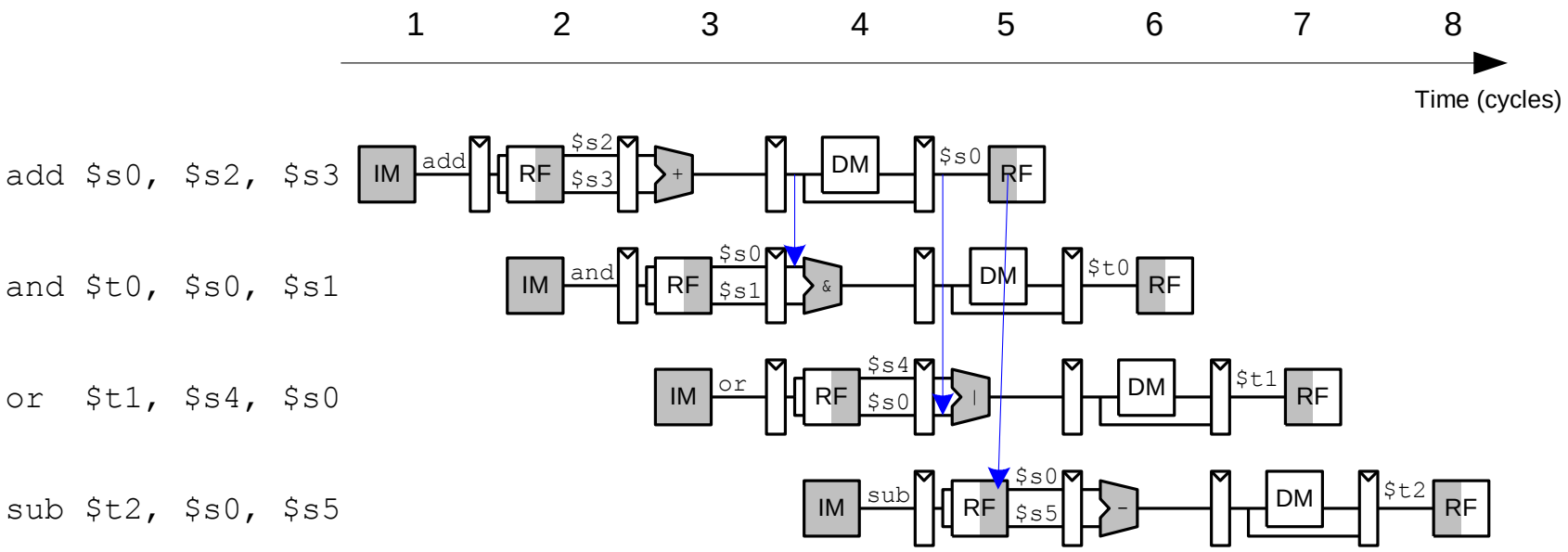


КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Data Hazards).

Пример:

- Препращане на данни (Data Forwarding).

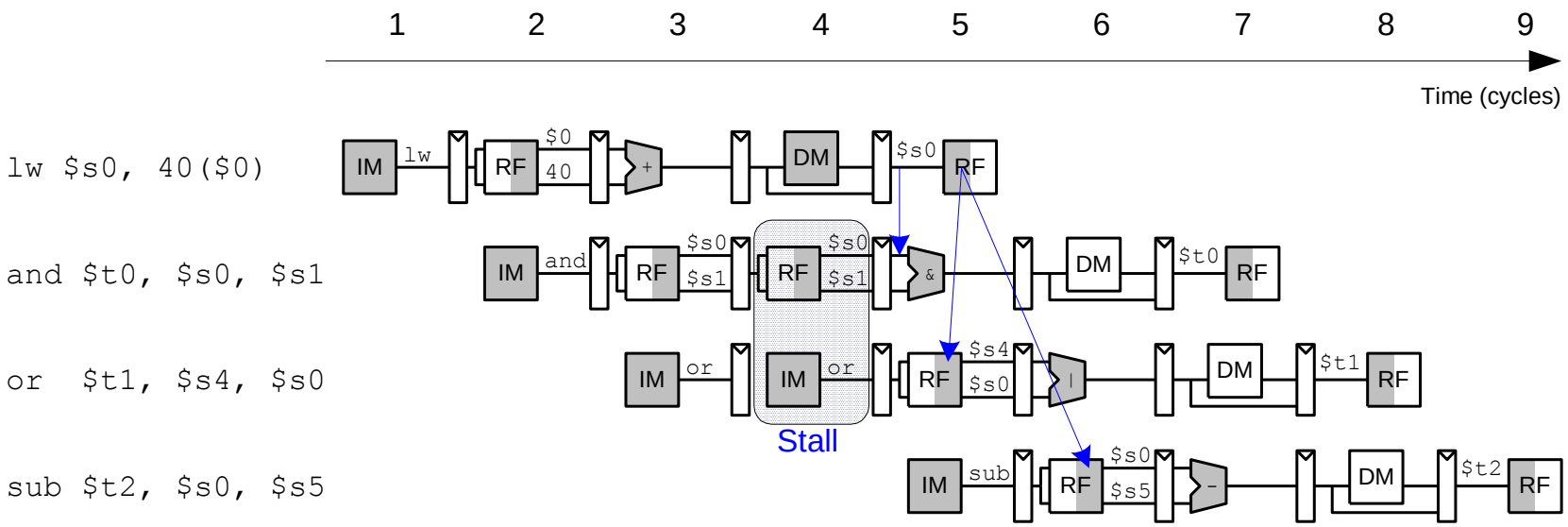


КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Data Hazards).

Пример:

- Задържане на процесора (Stalling).



КАРХ: Тема_11: Микроархитектура

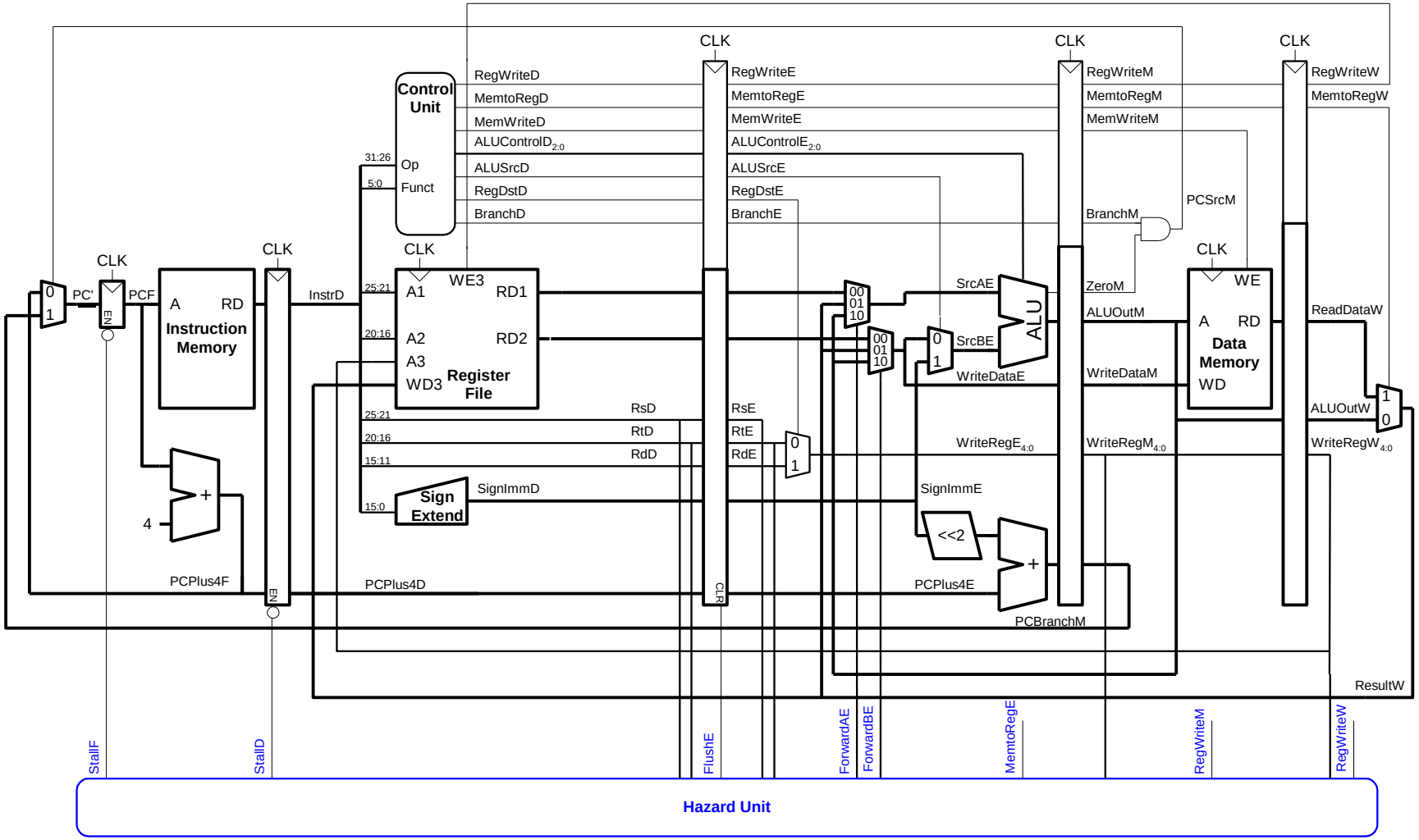
Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Control Hazards).

- **Control Hazards**
- **beq:**
  - Преходът е неопределен до 4-тото стъпало на конвеера
  - Инструкциите, след тази за преход, се извличат (fetched) преди преходът да се определи
  - Тези инструкции трябва да „изгорят“ (**be flushed**), ако преходът се осъществи
- **Загуби при непредсказан преход (Branch misprediction penalty)**
  - Брой на пропадналите инструкции когато преходът е осъществен
  - Могат да се намалят като се определи преходът по-рано

КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Control Hazards).

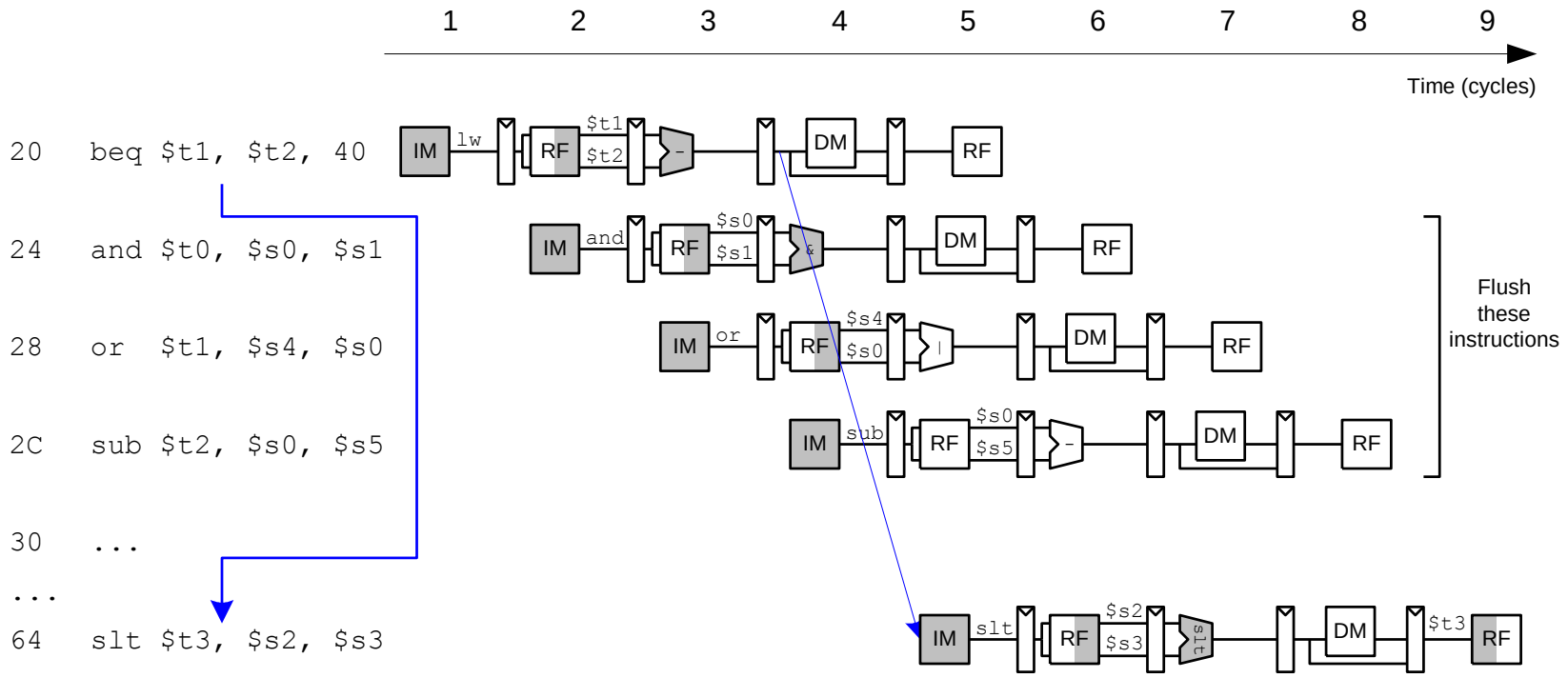
- **Control Hazards**



КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Control Hazards).

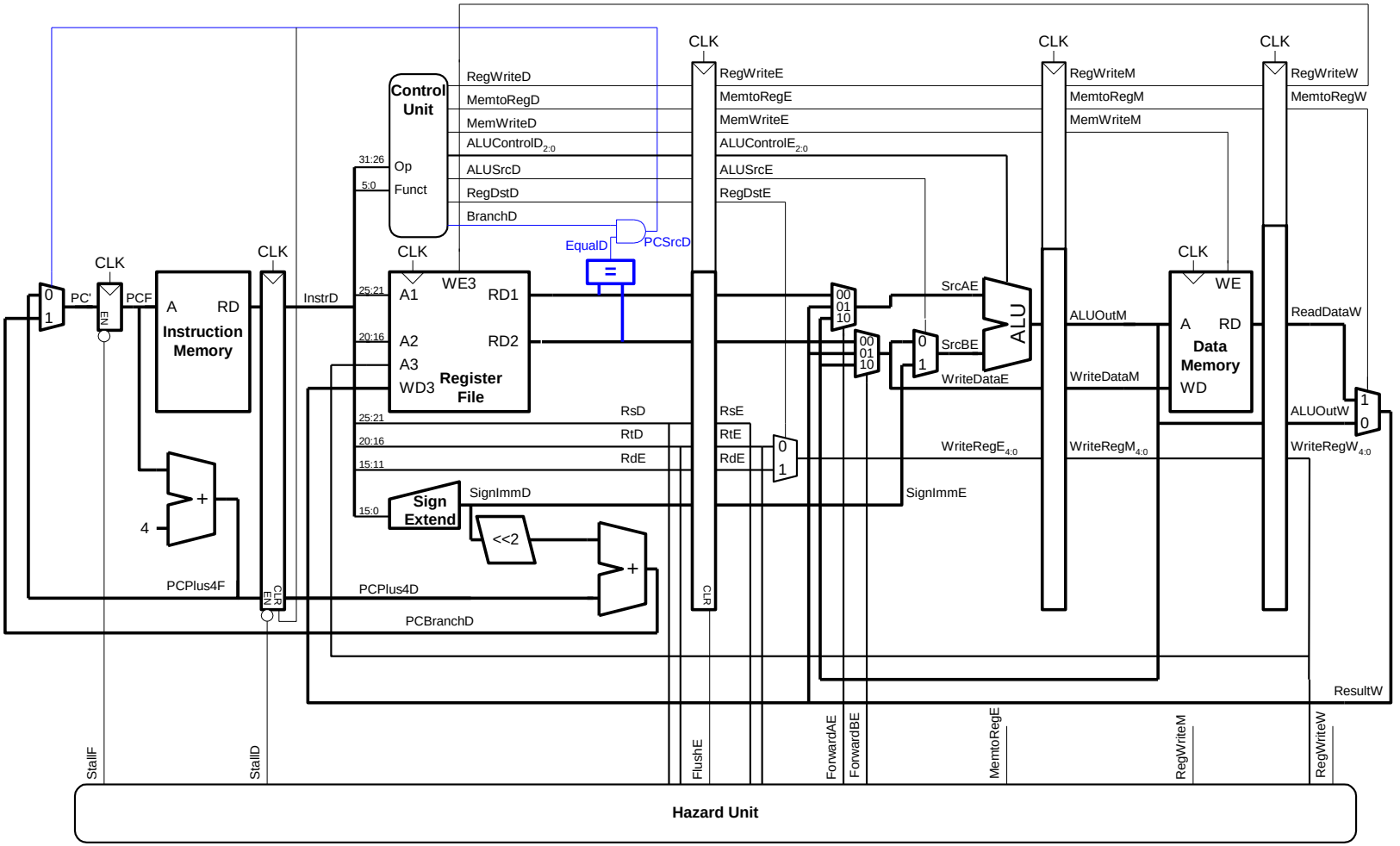
- **Control Hazards**



КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Control Hazards).

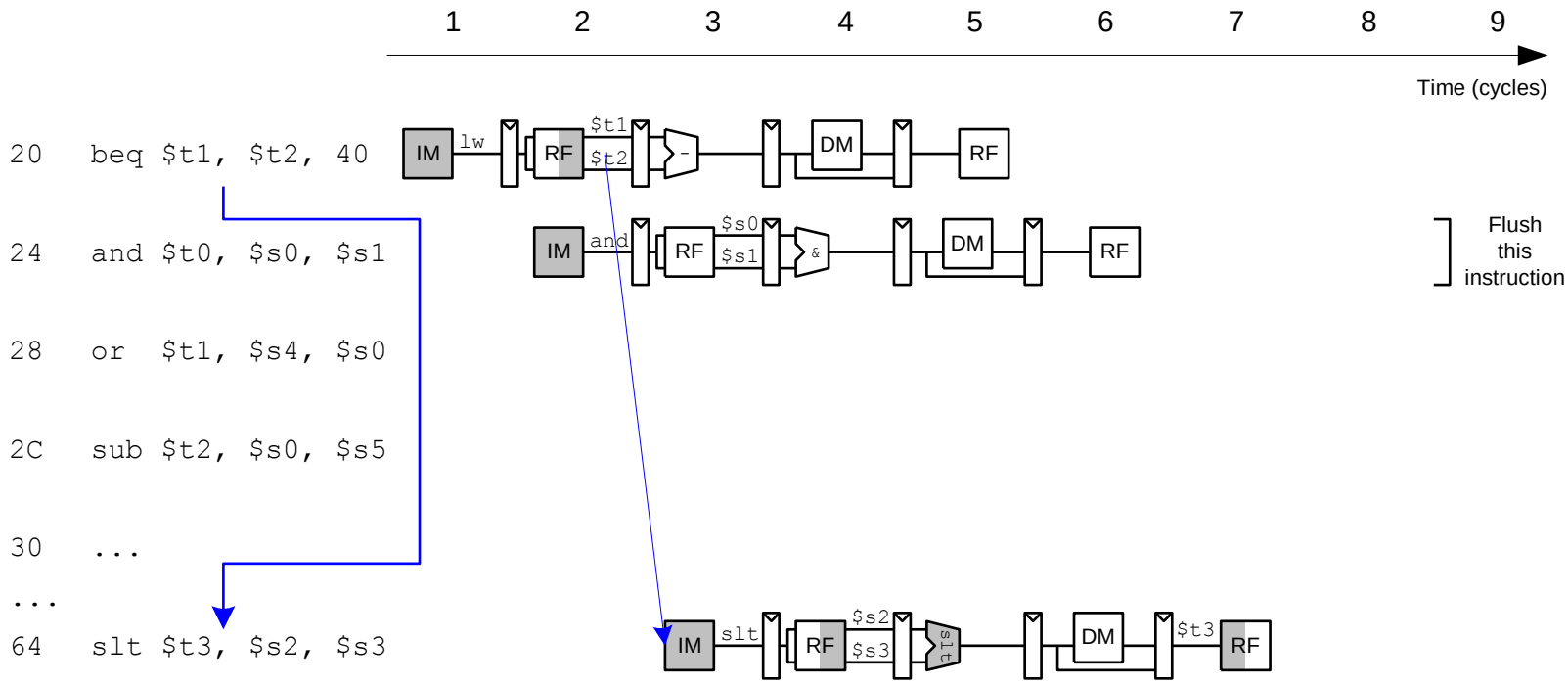
- **Control Hazards**
- Ранно определяне на прехода. **Introduced another data hazard in Decode stage**



КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Control Hazards).

- **Control Hazards**
- Ранно определяне на прехода.



КАРХ: Тема_11: Микроархитектура

Разрешаване на опасностите (рисковете) при конвеерната организация (Handling Control Hazards).

- **Control Hazards**
- Предсказване на преходите.
  - Преходите назад обикновено се изпълняват (loops).
  - Отчитането на предишното поведение подобрява предсказването.
- Доброто предположение намалява загубите от **flush**.

КАРХ: Тема_11: Микроархитектура

Примерна илюстрация за производителност (Pipelined Processor Performance).

- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction
  - 25% of branches mispredicted
  - All jumps flush next instruction
- **What is the average CPI?**

КАРХ: Тема_11: Микроархитектура

Примерна илюстрация за производителност (Pipelined Processor Performance).

- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type
- Suppose:
  - 40% of loads used by next instruction
  - 25% of branches mispredicted
  - All jumps flush next instruction
- **What is the average CPI?**
  - Load/Branch CPI = 1 when no stalling, 2 when stalling
  - $CPI_{lw} = 1(0.6) + 2(0.4) = 1.4$
  - $CPI_{beq} = 1(0.75) + 2(0.25) = 1.25$

Average CPI = (0.25)(1.4) + (0.1)(1) + (0.11)(1.25) + (0.02)(2) + (0.52)(1)

= 1.15

КАРХ: Тема_11: Микроархитектура

Сравнение на производителността на трите микроархитектури (Processor Performance Comparison).

Processor	Execution Time (seconds)	Speedup (single-cycle as baseline)
Single-cycle	92.5	1
Multicycle	133	0.70
Pipelined	63	1.47

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Използвани технологии:

- Многостъпален конвеер (Deep Pipelining)
- Предсказване на преходите (Branch Prediction)
- Суперскаларни процесори (Superscalar Processors)
- Неподредени процесори (Out of Order Processors)
- Преименуване на регистрите (Register Renaming)
- SIMD
- Многонишковост (Multithreading)
- Многоядреност (Multiprocessors)

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Многостъпален конвеер (Deep Pipelining)

- 10-20 стъпала (типично)
- Броят на стъпалата се ограничава от:
  - Рисковете (Pipeline hazards)
  - Силно усложняване на структурата
  - Мощността (Power)
  - Цената (Cost)

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Предсказване на преходите (Branch Prediction)

- За идеалният конвеерен процесор: CPI = 1
- Неточното предсказване при преходите увеличава CPI
- **Статично предсказване при преходи (Static branch prediction):**
  - Проверка посоката на прехода (напред или назад)
  - Ако посоката е назад се предполага преход (predict taken)
  - В противен случай не се предполага преход (predict not taken)
- **Динамично предсказване при преходи (Dynamic branch prediction):**
  - Пази се историята на последните (няколко стотин) прехода записана в *branch target buffer*:
    - Посока на прехода
    - В кои случаи е осъществен преход

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Предсказване на преходите (Branch Prediction)

Пример:

```
add  $s1, $0, $0       # sum = 0
  add  $s0, $0, $0       # i   = 0
  addi $t0, $0, 10       # $t0 = 10
for:
  beq  $s0, $t0, done    # if i == 10, branch
  add  $s1, $s1, $s0     # sum = sum + i
  addi $s0, $s0, 1       # increment i
  j    for
done:
```

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Предсказване на преходите (Branch Prediction)

1-Bit Branch Predictor

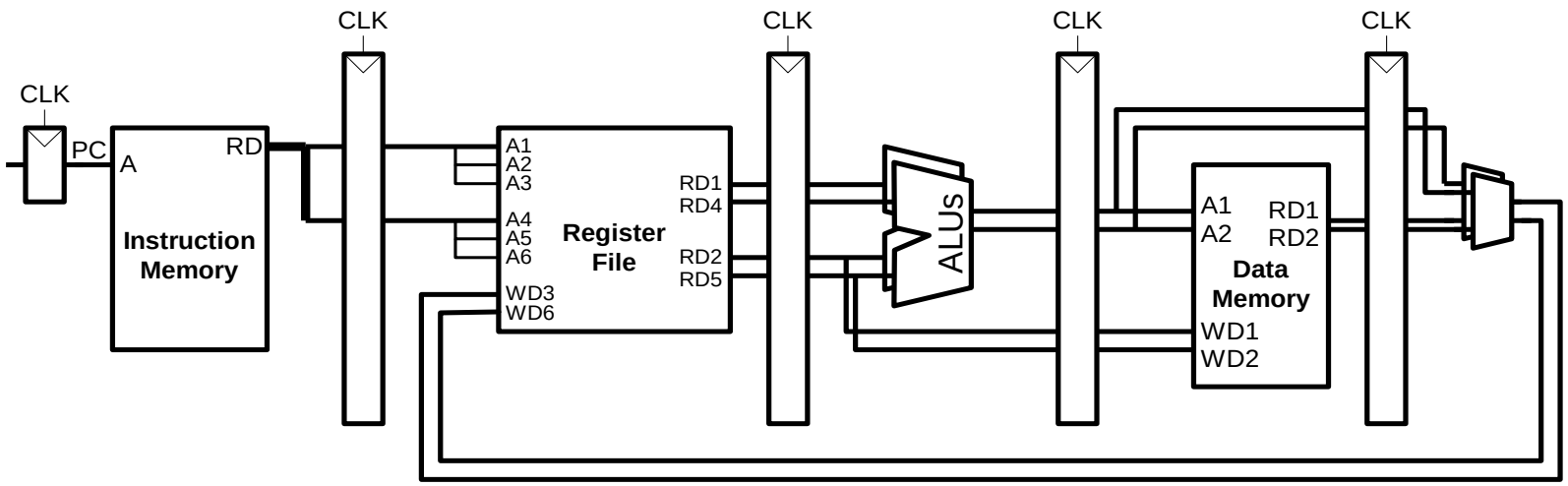
- Помни само последното състояние на прехода и предполага неговото повторение.
- Не познава първия и последния преход в цикъла.

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Суперскаларни процесори (Superscalar Processors)

- Имат няколко (М-на брой) еднакви **datapath,** което им позволява да изпълняват няколко (М-на брой) инструкции едновременно.
- Наличието на зависимости прави трудно изпълнението на няколко инструкции едновременно.



КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

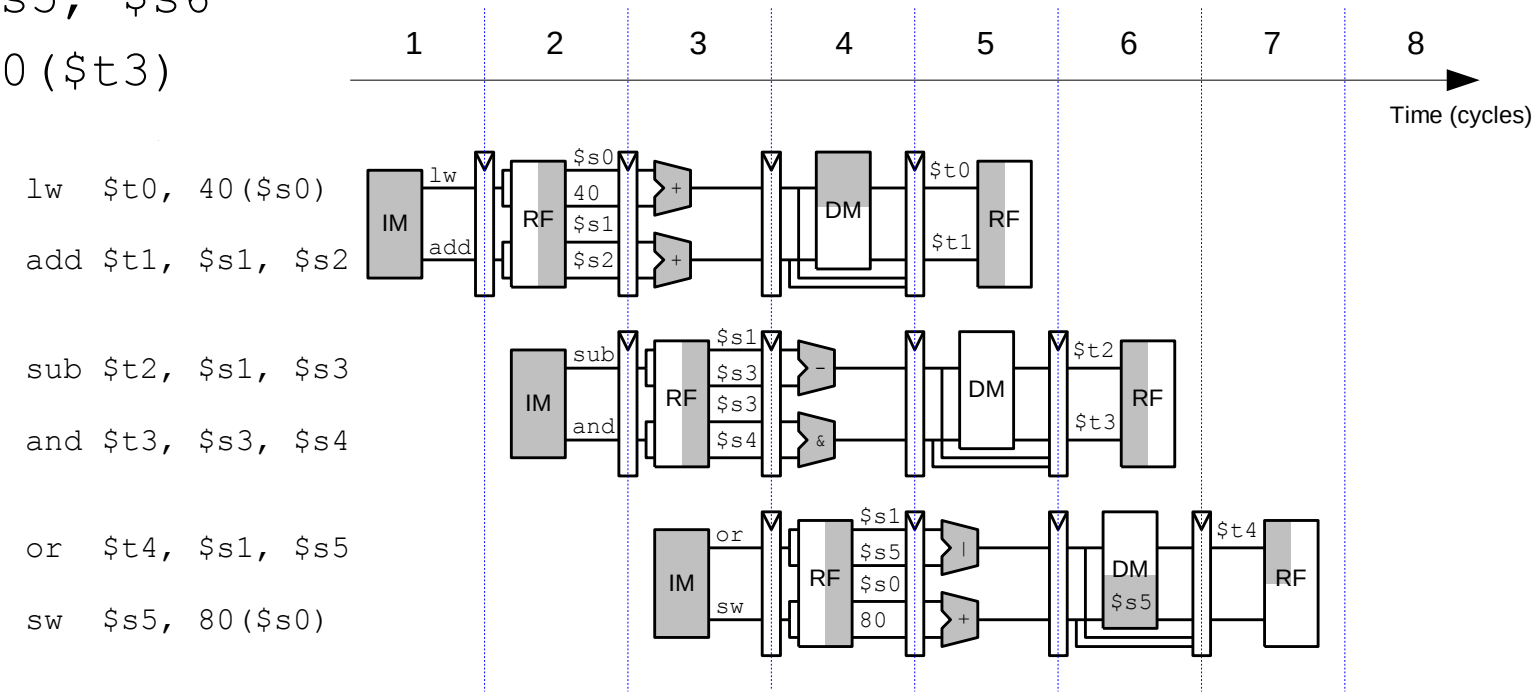
Суперскаларни процесори (Superscalar Processors)

Пример:

```
lw   $t0, 40($s0)
add  $t1, $t0, $s1
sub  $t0, $s2, $s3
and  $t2, $s4, $t0
or   $t3, $s5, $s6
sw   $s7, 80($t3)
```

Ideal IPC: 2

Actual IPC: 2



КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

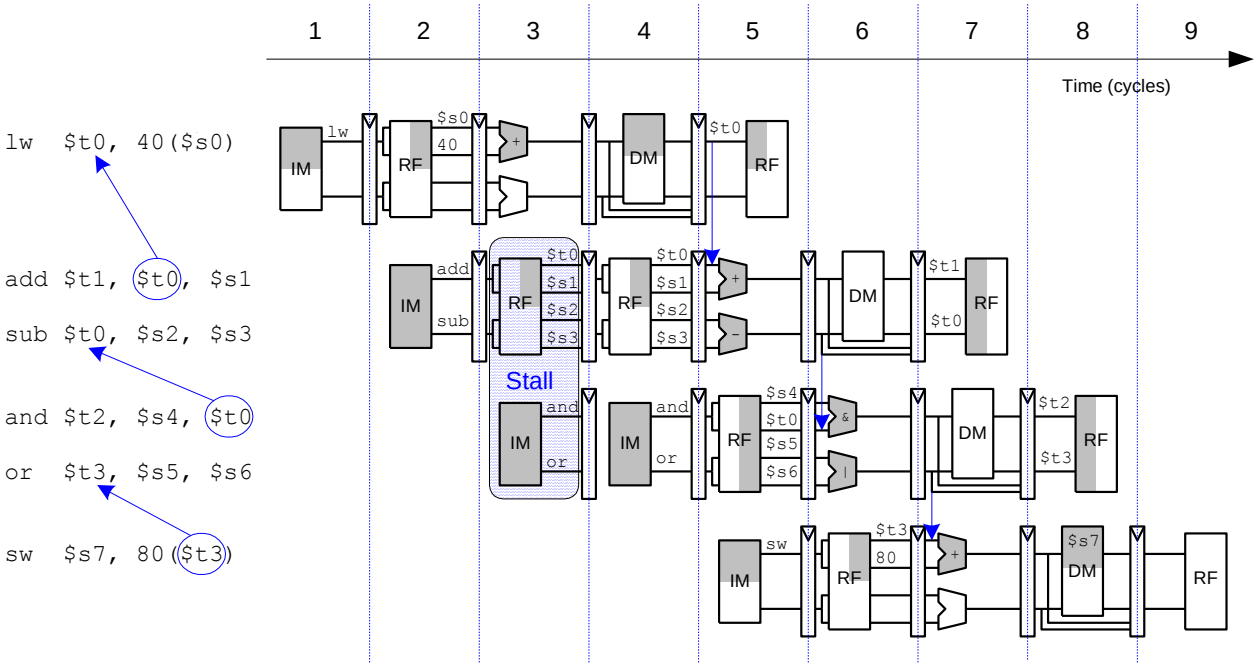
Суперскаларни процесори (Superscalar Processors)

Пример при наличие на зависимости:

```
lw   $t0, 40($s0)
add  $t1, $t0, $s1
sub  $t0, $s2, $s3
and  $t2, $s4, $t0
or   $t3, $s5, $s6
sw   $s7, 80($t3)
```

Ideal IPC: 2

Actual IPC: 6/5 = 1.2



КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Неподреден процесор (Out of Order Processor)

- Преглеждат се няколко инструкции напред
- Едновременно се изпълняват тези, за които това е възможно
- Изпълняват се инструкции не по реда на тяхното следване (out of order) (докато не се появят зависимости)
- **Зависимости (Dependencies):**
  - **RAW** (read after write): една инструкция записва в регистър, следваща инструкция чете от регистъра
  - **WAR** (write after read): една инструкция чете от регистър, следваща инструкция записва в регистъра
  - **WAW** (write after write): една инструкция записва в регистър, следваща инструкция записва в регистъра

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Неподреден процесор (Out of Order Processor)

- **Instruction level parallelism (ILP):** броят на инструкциите, които могат да се изпълнят едновременн**о** (средно < 3)
- **Scoreboard:** таблица, която съдържа запис на:
  - Инструкции чакащи изпълнение
  - Свободни функционални блокове (хардуер)
  - Зависимости (Dependencies)

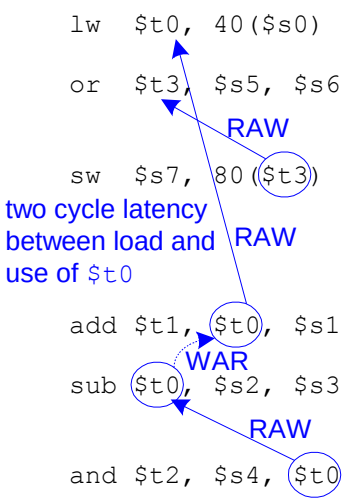
КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Неподреден процесор (Out of Order Processor)

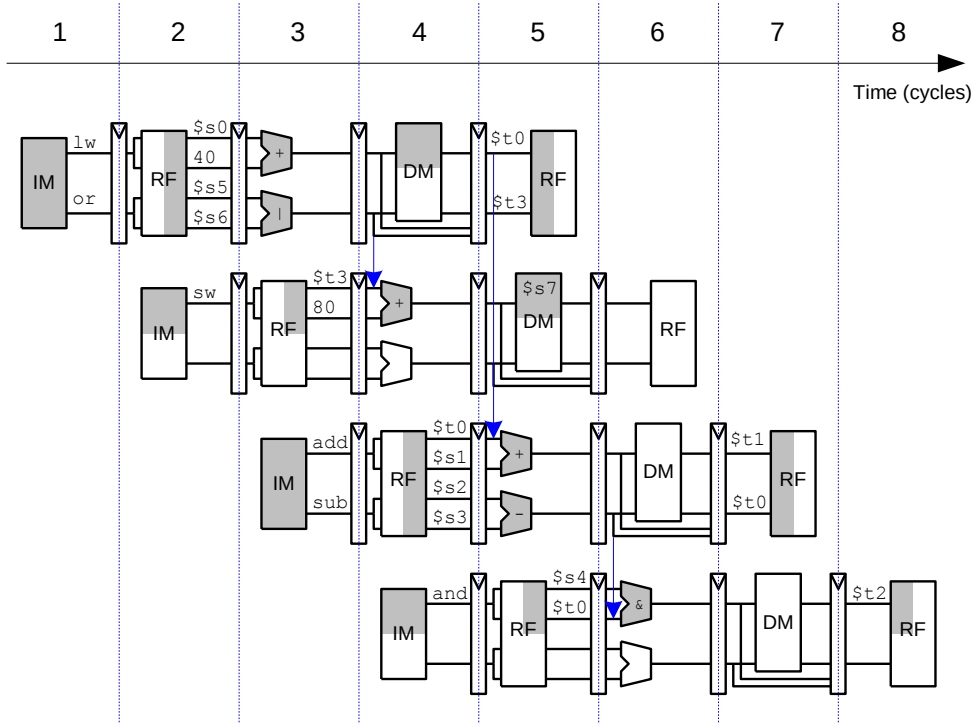
Пример:

```
lw  $t0, 40($s0)
add $t1, $t0, $s1
sub $t0, $s2, $s3
and $t2, $s4, $t0
or  $t3, $s5, $s6
sw  $s7, 80($t3)
```



Ideal IPC: **2**

Actual IPC: **6/4 = 1.5**



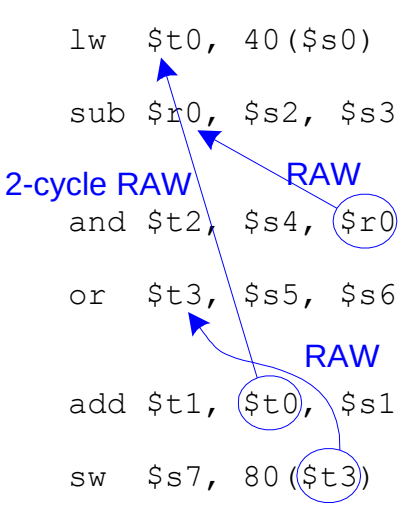
КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Преименуване на регистри (Register Renaming)

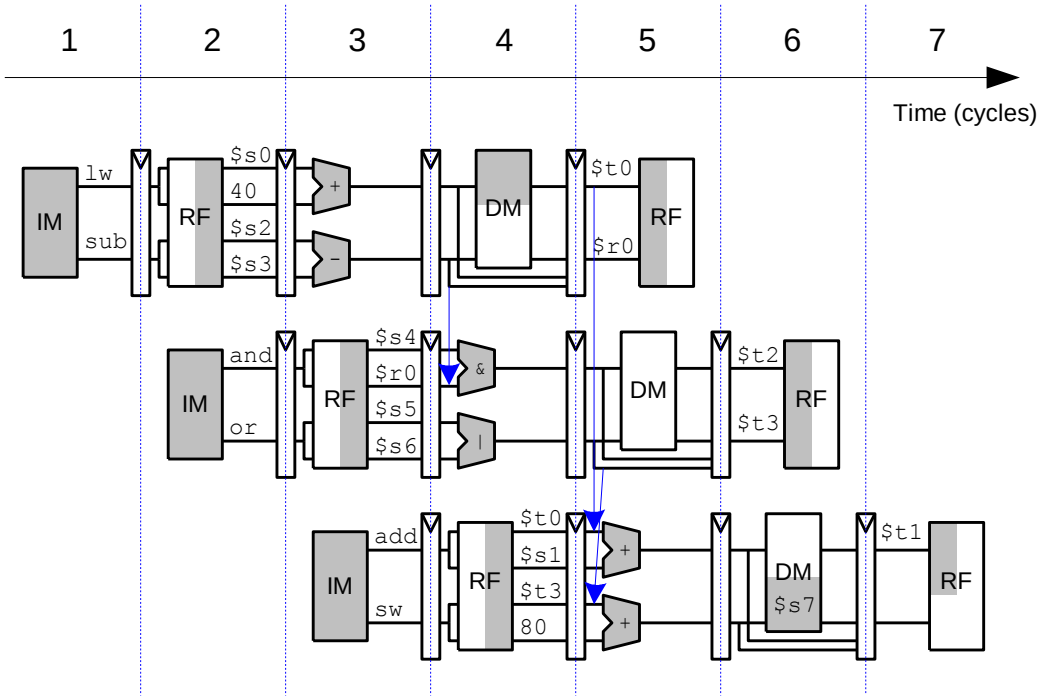
Пример:

```
lw   $t0, 40($s0)
add  $t1, $t0, $s1
sub  $t0, $s2, $s3
and  $t2, $s4, $t0
or   $t3, $s5, $s6
sw   $s7, 80($t3)
```



Ideal IPC: 2

Actual IPC: 6/3 = 2

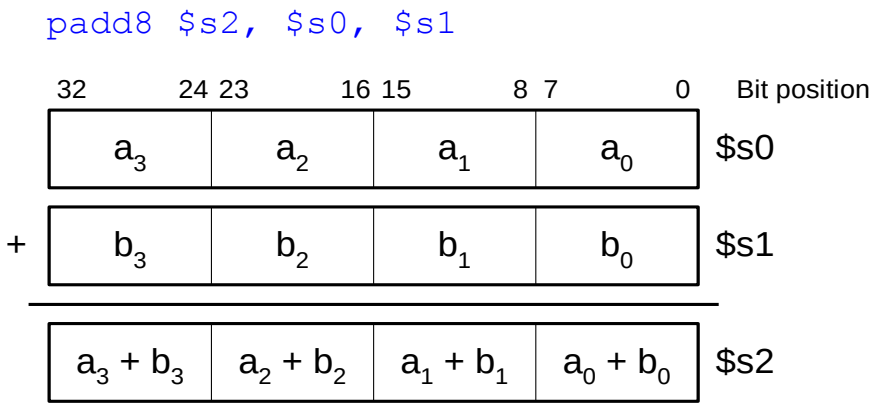


КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

SIMD

- **S**ingle **I**nstruction **M**ultiple **D**ata (SIMD)
  - Една инструкция действа върху много на брой данни (части от данни) едновременно
  - Най-общо приложение: графиките (изображенията)
  - Извършват къси аритметични операции (наричани също *packed arithmetic*)
- Например, събиране на четири 8-bit елементи



КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Висши архитектурни технологии:

- Многонишковост (**Multithreading**)
  - Wordprocessor: нишки (процеси) (thread) за писане, проверка на правопис (spell checking), печатане (printing) на документ

- Многоядреност (Multiprocessors)
  - Много процесори (ядра)(cores) в една интегрална схема (on a single chip)

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Многонишковост (**Multithreading**)

Определения:

- **Процес (Process):** програма, текущо изпълнявана от компютъра
  - Много процеси могат да вървят едновременно: например, сърфиране в Web, възпроизвеждане на музика, писане на статия и др.
- **Нишка (Thread):** част от програма
  - Всеки процес има много нишки: например, word processor може да има нишки за набиране (typing), проверка на правопис (spell checking), печатане (printing)

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Многонишковост (**Multithreading**)

Нишки в конвенционален процесор

- Само една нишка върви в даден момент
- Когато тази нишка спре (stalls) (например, чакайки достъп до паметта):
  - Състоянието ѝ (Architectural state) се записва (запазва)
  - Състоянието (Architectural state) на чакаща нишка се зарежда в процесора и се стартира
  - Този процес се нарича **контекстно превключване (context switching)**
- За потребителя изглежда, че всички нишки вървят едновременно.

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Многонишковост (**Multithreading**)

- Има няколко копия на състояния (architectural state)
- Няколко нишки **активни едновременно**:
  - Когато една нишка спре (stalls), друга тръгва веднага
  - Ако една нишка не използва всички изпълняващи блокове, други нишки могат да ги използват
  - Това не повишава instruction-level parallelism (ILP) на единичната нишка, но повишава производителноста (throughput)

Intel нарича това “hyperthreading”

КАРХ: Тема_11: Микроархитектура

Висша (Авангардна) микроархитектура (Advanced Microarchitecture).

Висши архитектурни технологии:

- Многоядреност (Multiprocessors)
  - Много процесори (ядра)(cores) в една интегрална схема (on a single chip)
  - Много процесори (ядра) с метод за комуникация между тях
- Видове:
  - **Хомогенни (Homogeneous)**: няколко ядра със споделена памет
  - **Хетерогенни (Heterogeneous):** отделни ядра за решаване на различни задачи (например, DSP и CPU в клетъчните телефони)
  - **Клъстерни (Clusters):** всяко ядро има собствена системна памет