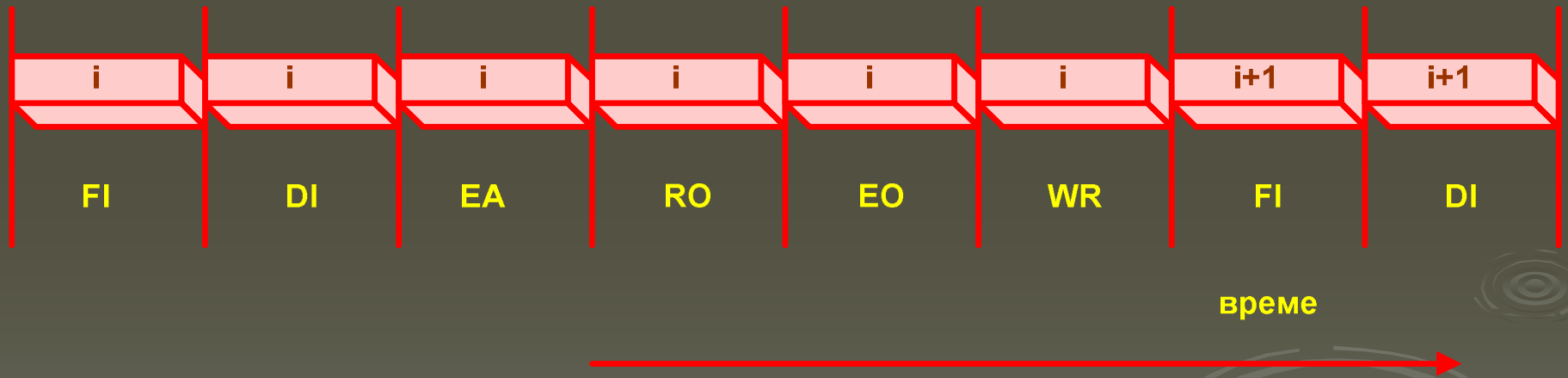
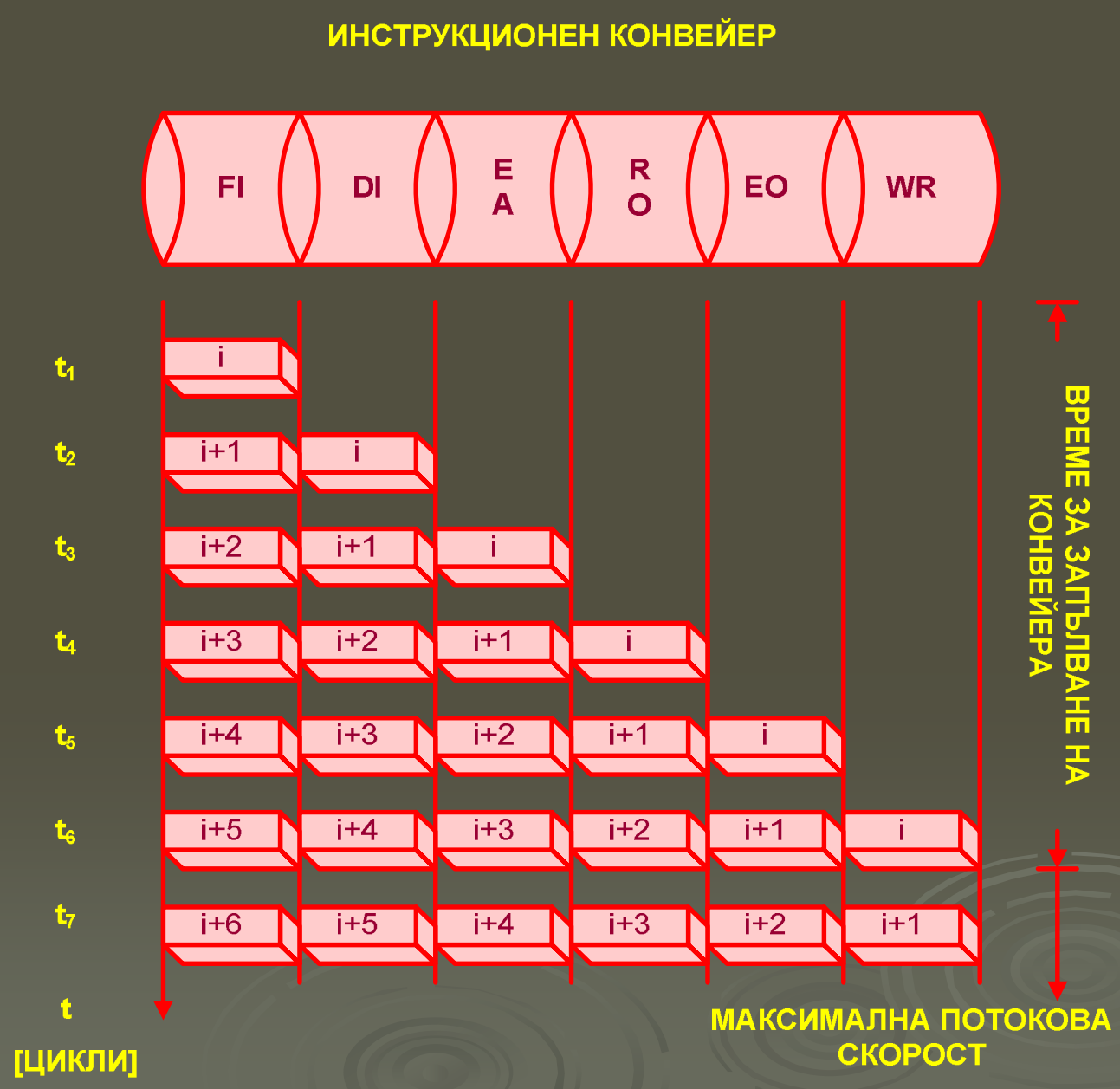
АРХИТЕКТУРА НА ЦЕНТРАЛНИЯ ПРОЦЕСОР

Последователно изпълнение на инструкциите



ИНСТРУКЦИОННИ КОНВЕЙЕРИ



Производителност на инструкционни конвейери

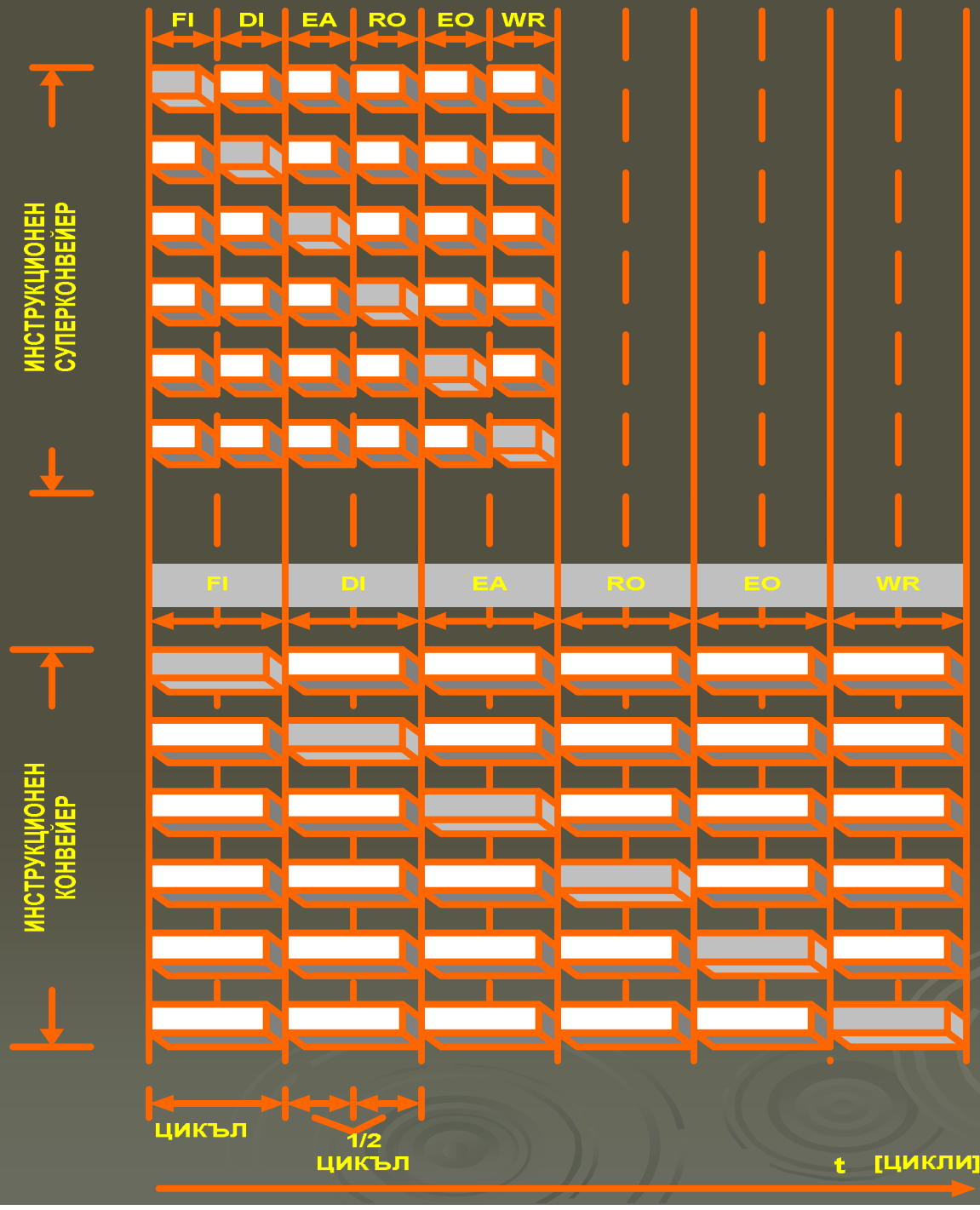
$$T_{init\_pipeline} = (n-1)\tau$$

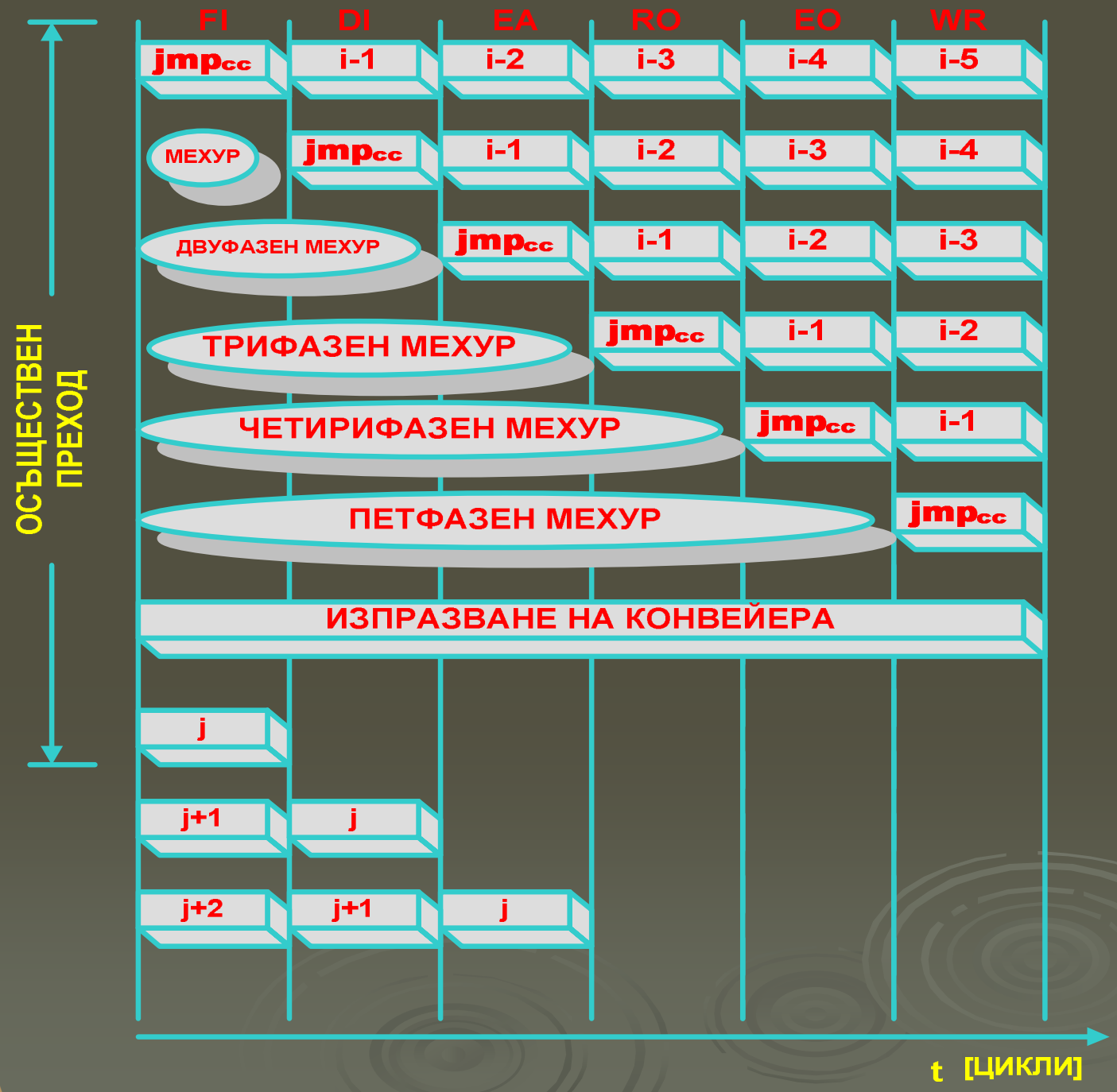
$$S_{up} = \frac{T_{seq}}{T_{pipeline}} = \frac{sn}{n+(s-1)}$$

$$Efficiency = S_{up} / n$$

Ускорението на инструкционния конвейер се определя от броя на фазите му

- *n*-фазен конвейер
- *s* - # инструкциите в конвейера
- τ - закъснение във фазата на конвейера (цикли)
- $T_{init\_pipeline}$ – време за инициализация на конвейера
- T_{seq} – време за последователна обработка на инструкциите
- $T_{pipeline}$ – време за паралелна обработка на инструкциите
- S_{up} - ускорение


FI
DI
EA
RO
EO
WR
ИНСТРУКЦИОНЕН СУПЕРКОНВЕЙЕР
FI
DI
EA
RO
EO
WR
ИНСТРУКЦИОНЕН КОНВЕЙЕР
ЦИКЪЛ
1/2 ЦИКЪЛ
t [ЦИКЛИ]


FI
DI
EA
RO
EO
WR
jmpcc
i-1
i-2
i-3
i-4
i-5
МЕХУР
ДВУФАЗЕН МЕХУР
ТРИФАЗЕН МЕХУР
ЧЕТИРИФАЗЕН МЕХУР
ПЕТФАЗЕН МЕХУР
ИЗПРАЗВАНЕ НА КОНВЕЙЕРА
ОСЪЩЕСТВЕН ПРЕХОД
j
j+1
j+2
t [ЦИКЛИ]

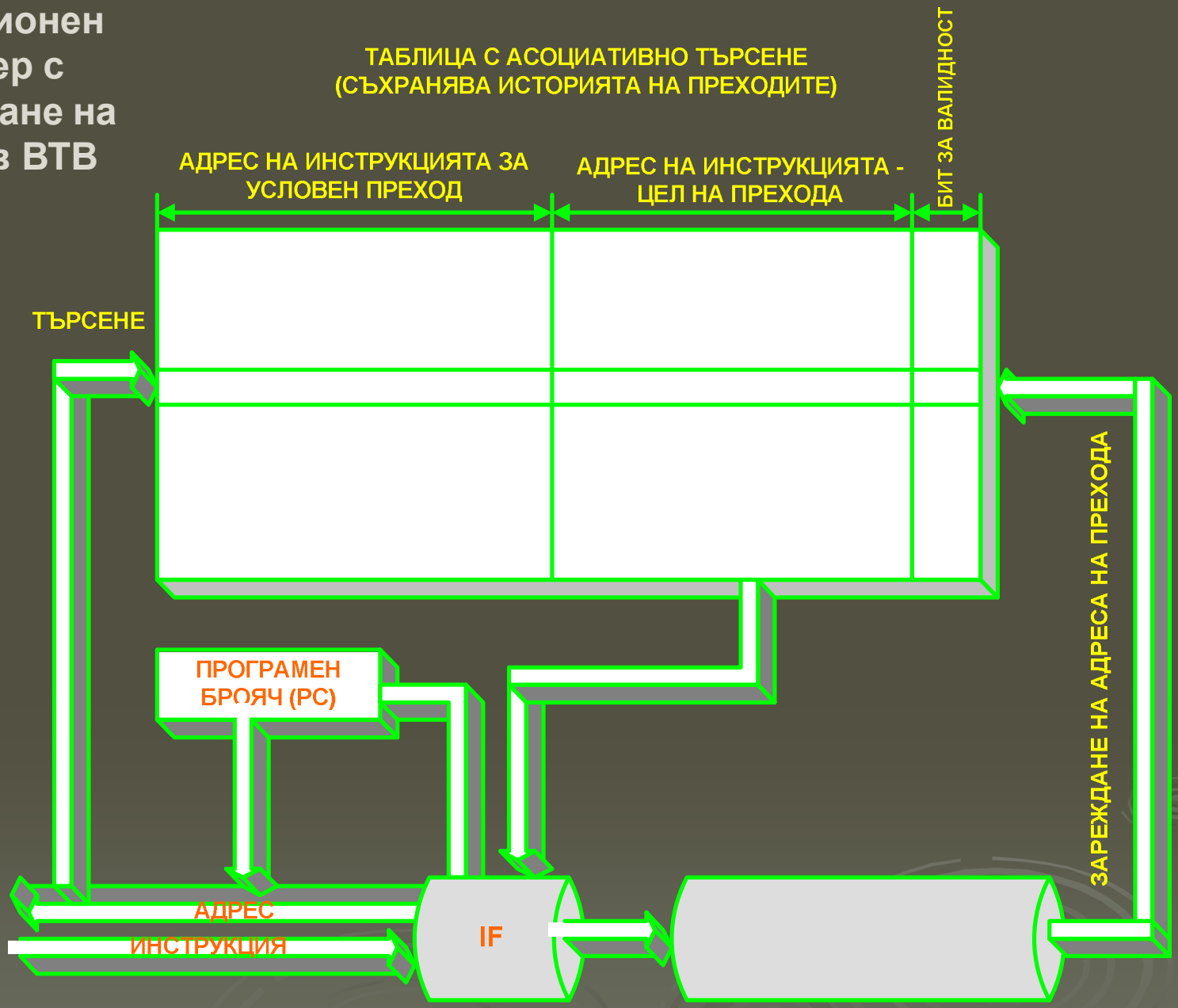
Стратегии

- Дублиране на инструкционните буфери за извличане на двете възможни инструкции – множествено предварително извличане на инструкции или branch bypassing
- Логика за динамично предсказване на прехода – извличане на най-вероятната инструкция след инструкция branch
- Статично предсказване на прехода
- Инструкции за отложен преход (Delayed branch instructions)

Логика за предсказване на прехода и история на преходите (Prediction Logic and Branch History)

- *Динамично предсказване базирано на историята в хода на изпълнение на програмата (run-time execution history)*
- *Prediction Look-up Table (таблица за предсказване на преходите) – адрес на инструкцията на прехода, адрес на прехода, бит за предходно предсказване*
- *Branch Target Buffer (Буфер за историята на преходите) – имплементира се с кеш (напълно асоциативен или асоциативен с множества)*

Инструкционен конвейер с предсказване на прехода в BTB



ХАЗАРТИ В ИНСТРУКЦИОННИТЕ КОНВЕЙЕРИ

1. *Ресурсни конфликти*.
2. *Процедурни зависимости* (дължащи се предимно на инструкции за преход - branch).
3. *Зависимости по данни*.

Ресурсни конфликти

- *Едновременни заявки* за едни и същи ресурси (памет или функционални устройства)
- Разрешават се чрез *дублиране на ресурсите*
- Елиминиране на *конфликтите за памет* :

1. Специализирано функционално устройство за четене на операндите или запис на резултатите в паметта (load/store)
2. Осигуряване на отделни кешове за инструкции и данни на ниво L1

Процедурни зависимости

- *Редът на изпълнение* не е известен преди изпълнението на инструкциите
- Основна причина – *инструкции за безусловен и условен преход - branch*
- *Инструкции за безусловен преход* → *всички инструкции в конвейера се игнорират и конвейерът се изпразва (the pipeline stalls)*
- *Инструкции за условен преход* → *същият ефект ако преходът се осъществи*

Статично предсказване

- Прогнозата се прави преди изпълнението
- Форматът на инструкцията за преход съдържа *допълнителен бит* за предсказване на прехода
- Добавя се мнжество от нови инструкции за преход: *избери програмния брояч PC или избери адреса на прехода*
- *Отговорност на компилатора*
- *Атрактивен подход за RISC архитектури*

Инструкции за отложен преход
Delayed Branch Instructions

- Запълване на мрехурите в конвейера с инструкции, които не зависят от това, дали ще се осъществи прехода (или поне мехурите се запълват с NOP)

Запълване на мехурите

```
        ADD R3, R4, R5
        SUB R2, R2, 1
        BEQ R2, R0, L1
                ...
                ...
L1: ....................
                ...
```

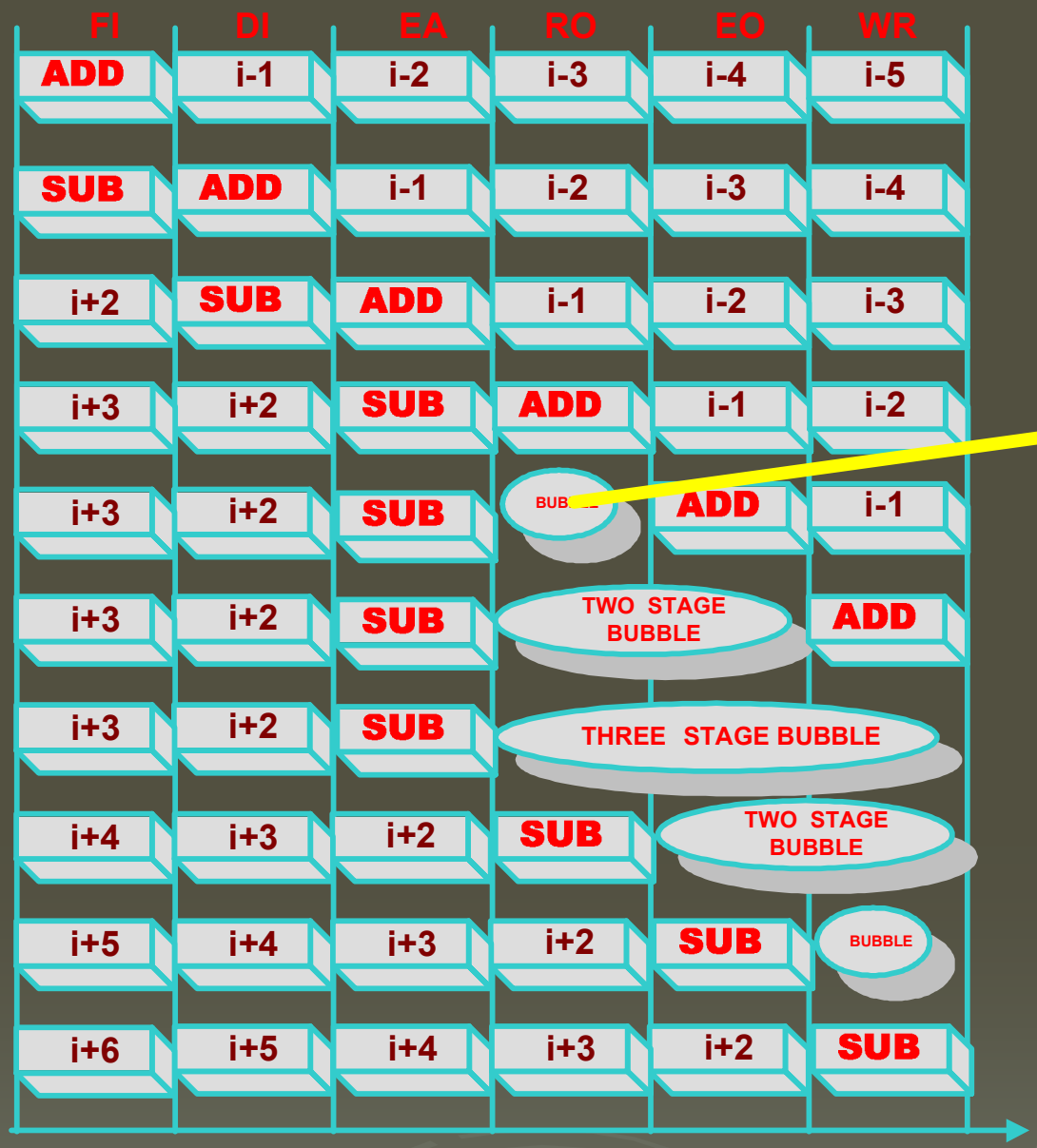
```
        SUB R2, R2, 1
        BEQD R2, R0, L1
        ADD R3, R4, R5
                ...
                ...
L1: ....................
                ...
```

Зависимости по даннни
Data Dependencies
(read-after-write hazard)

- Резултатът на *i-mama* инструкция се използва за операнд на *(i+1)-ma* инструкция
- ***Типове зависимости по данни***:

1. ***Потокови (flow) зависимости по данни***
2. ***Антизависимости (Antidependency)***
3. ***Зависимости по изход (Output dependency)***

FLOW DATA DEPENDENCIES



PIPELINE STALLS

ADD R3, R2, R1 ; R3 = R2 + R1

SUB R4, R3, 1 ; R4 = R3 - 1

Antidependencies (write-after-read hazards)

- Инструкцията осъществява запис в регистър (или адрес в паметтта), съдържанието на който се чете от предходна инструкция

1. ***ADD R3, R2, R1 ; R3 = R2 + R1***
2. ***SUB R2, R3, 1 ; R2 = R3 - 1***

Output Dependencies (write-after-write hazard)

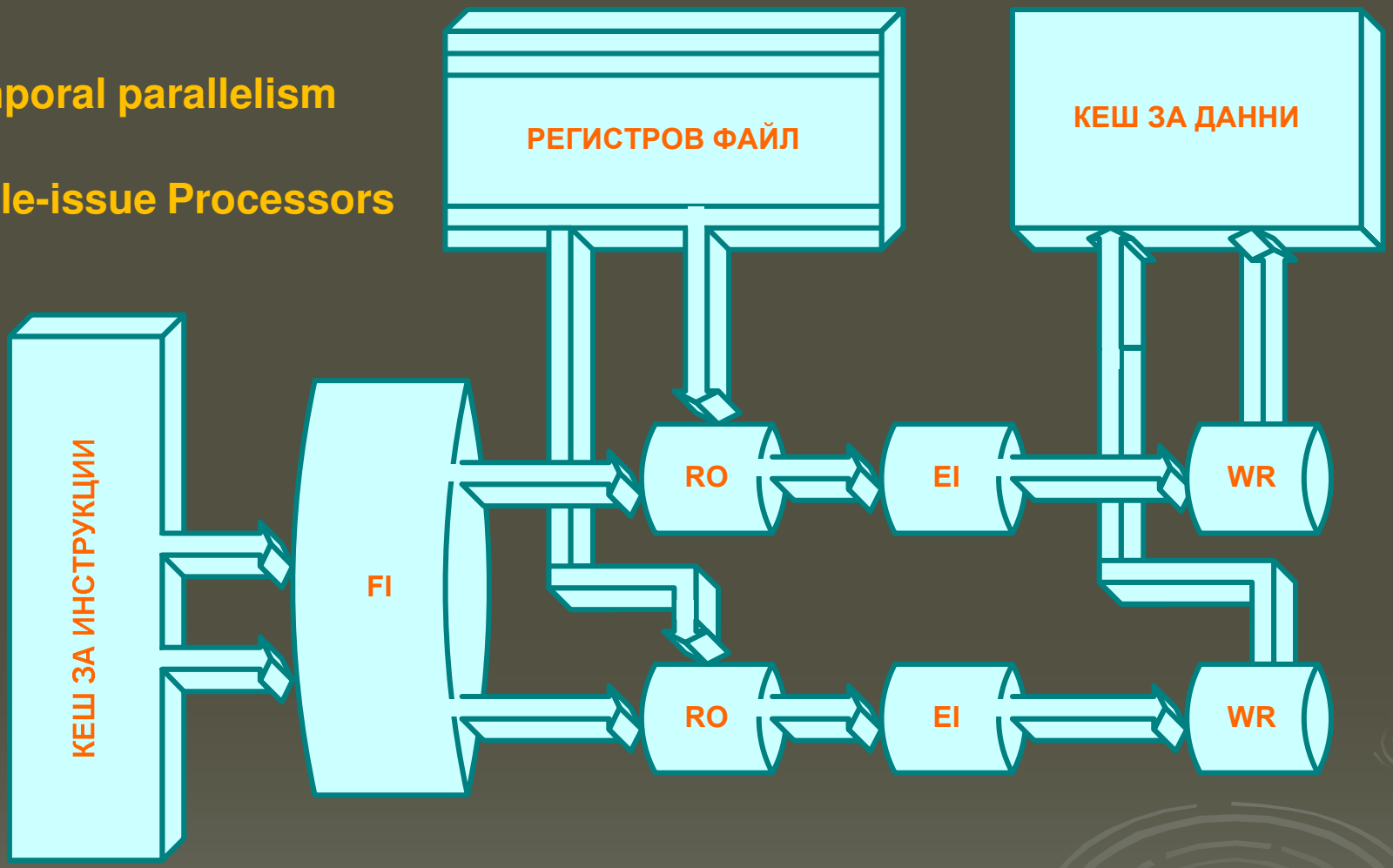
1. ADD R3, R2, R1 ; R3 = R2 + R1
2. SUB R2, R3, 1 ; R2 = R3 - 1
3. ADD R3, R2, R5; R3 = R2 + R5

- *По своята същност е вид ресурсен конфликт*
- *Регистър R3 се използва от няколко последователни инструкции*
- *Елиминиране на този хазарт: да се използва друг регистър*

СУПЕРСКАЛАРНИ ПРОЦЕСОРИ

temporal parallelism

Multiple-issue Processors



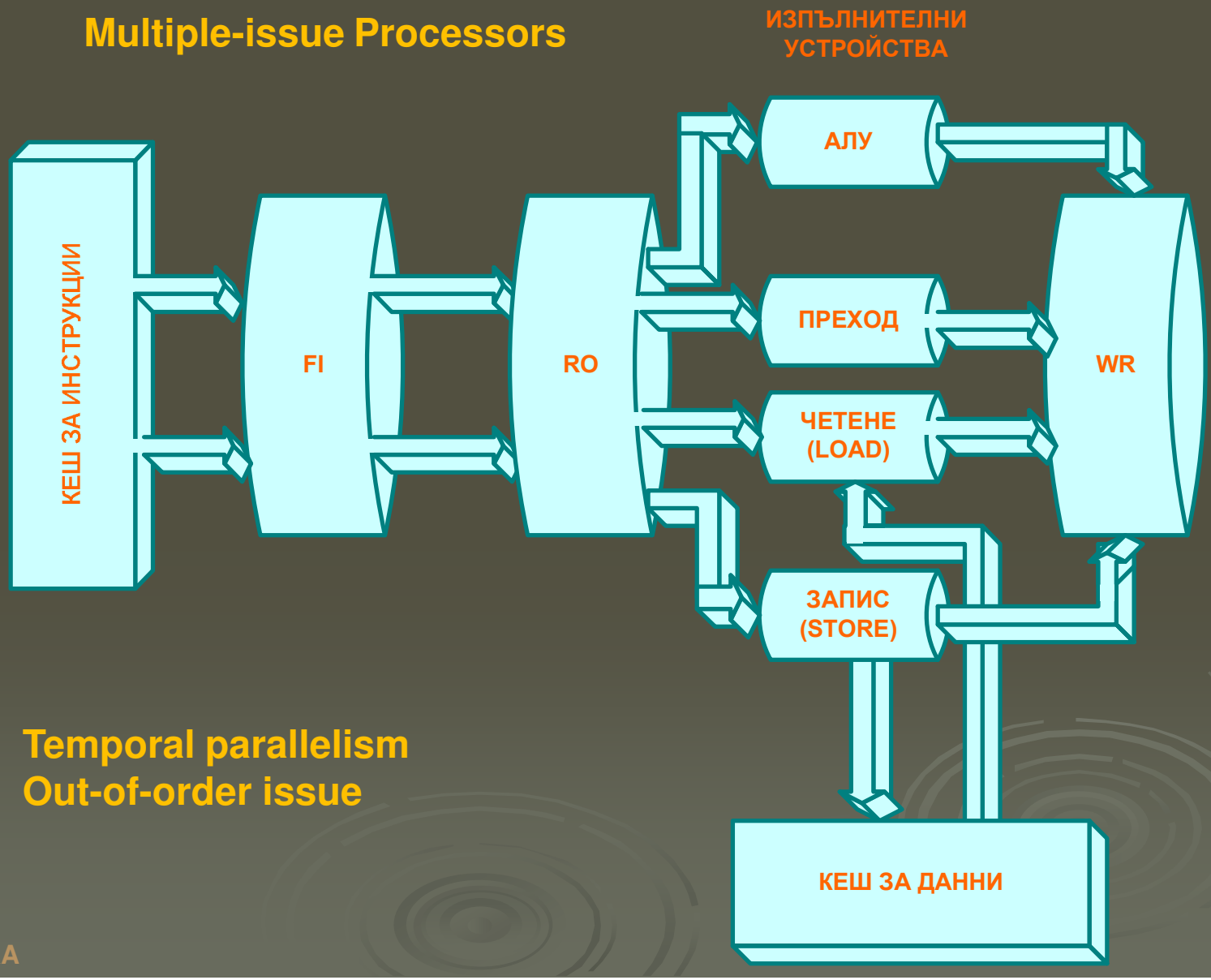
Структура на суперскаларен процесор с два инструкционни конвейера

Изпълнение с пренареждане на инструкциите Out-of-order Instruction Issue

- Множество функционални устройства
- Могат да се имплементират множество АЛУ-та
- Инструкционен буфер, наречен *инструкционен прозорец* (*instruction window),* се имплементира между фазата за извличане на инструкциите и изпълнителната фаза
- Инструкциите се избират за изпълнение от инструкционния прозорец, когато операндите им са готови и съответното функционално устройство е свободно (*основен принцип на data flow компютрите*)
- Инструкционният прозорец може да се имплементира по 2 начина:
  1. Централизиран инструкционен прозорец
  2. Децентрализиран инструкционен прозорец

Суперскаларен процесор със специализирани изпълнителни устройства

Multiple-issue Processors



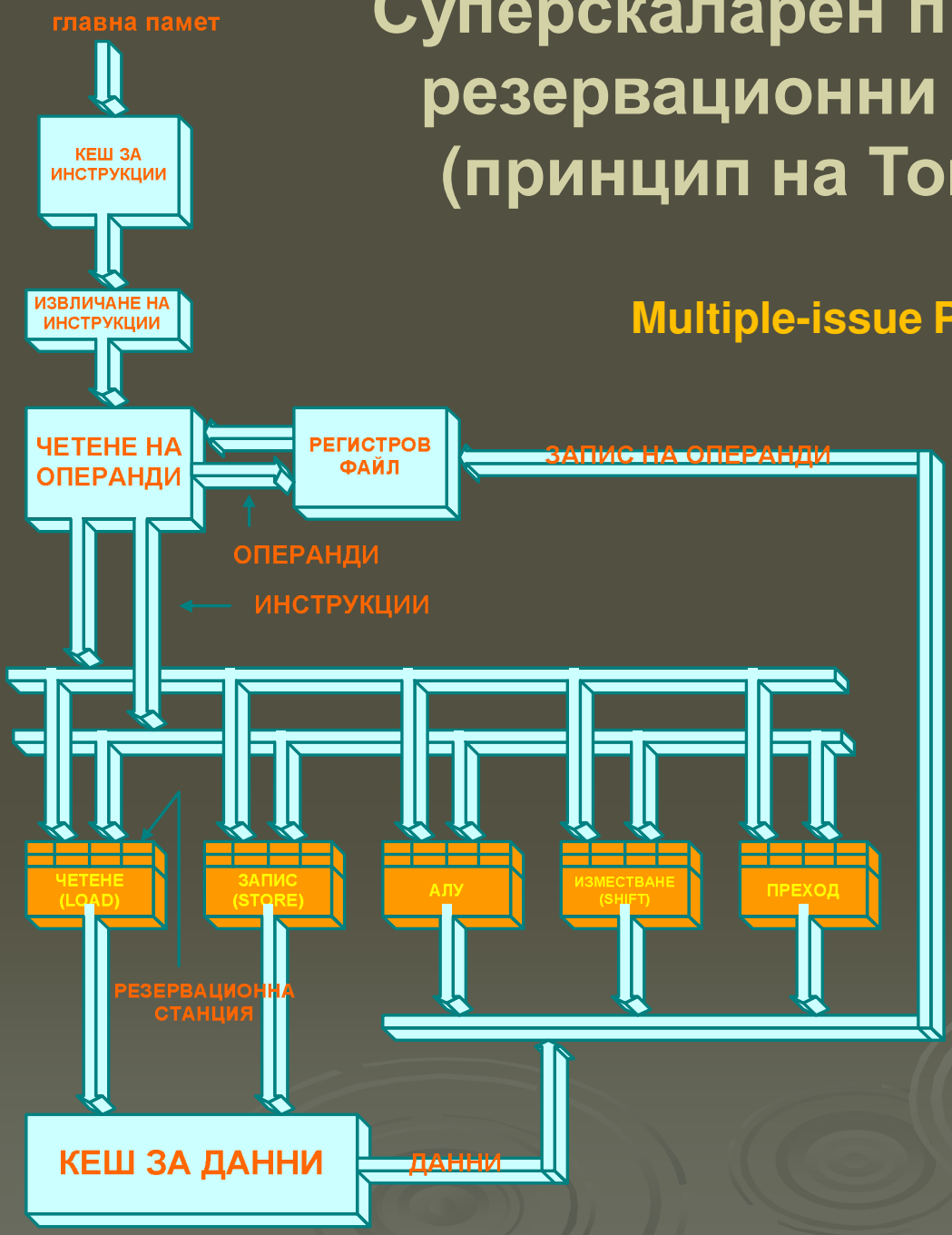
Temporal parallelism
Out-of-order issue

Суперскаларен процесор с централизиран инструкционен прозорец

Multiple-issue Processors

Суперскаларен прозорец с резервационни станции (принцип на Томасуло)

Multiple-issue Processors



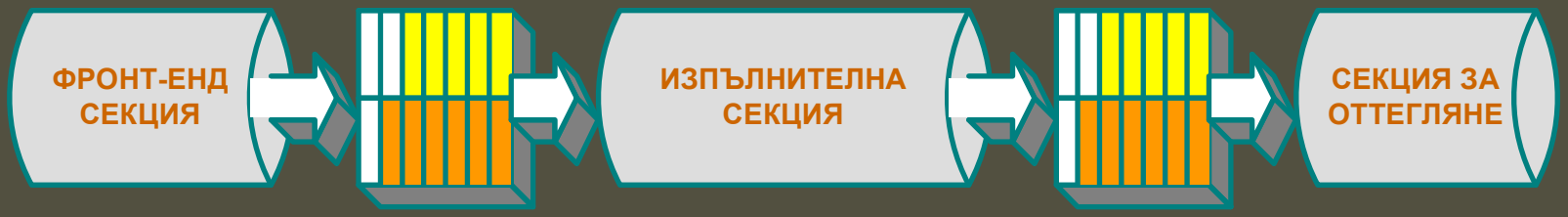
Съдържание на инструкционния прозорец
Instruction Window Contents

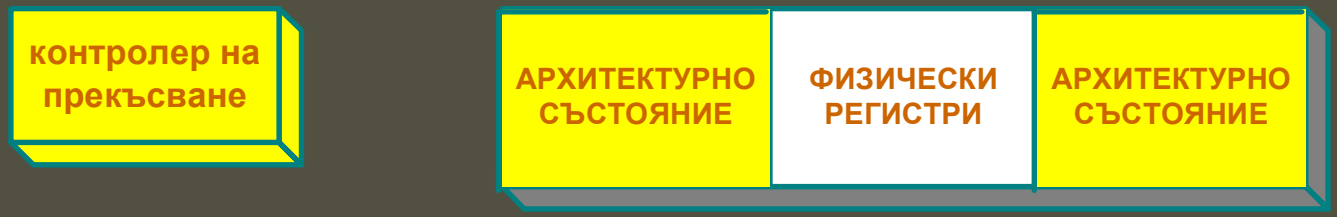
Instruction	Opcode	Register Destination ID	Operand 1	Register Operand 1 ID	Operand 2	Register Operand 2 ID
1	Opcode 1	ID 1	VALUE			ID 1A
2	Opcode 2	ID 2	VALUE			ID 2A
3	Opcode 3	ID 3	VALUE		VALUE	
4	Opcode 4	ID 4			VALUE	
5	Opcode 5	ID 5	VALUE	ID		ID 5A
...	...	...	...	...	...	...
...	...	...	...	...	...	...

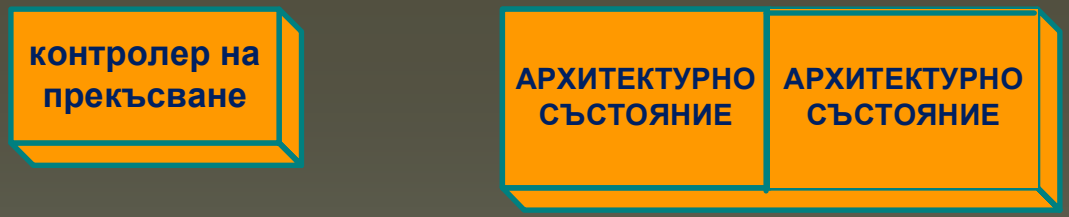
ГОТОВА ЗА ИЗПЪЛНЕНИЕ –
ИЗПРАЩА СЕ КЪМ ФУНКЦИОНАЛНОТО УСТРОЙСТВО

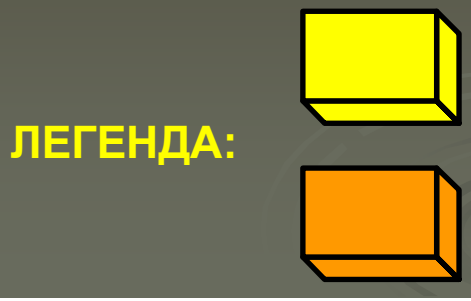
DATA FLOW PRINCIPLE

АРХИТЕКТУРА НА ПРОЦЕСОРИ С ХИПЕРНИШКОВА ТЕХНОЛОГИЯ PENTIUM 4

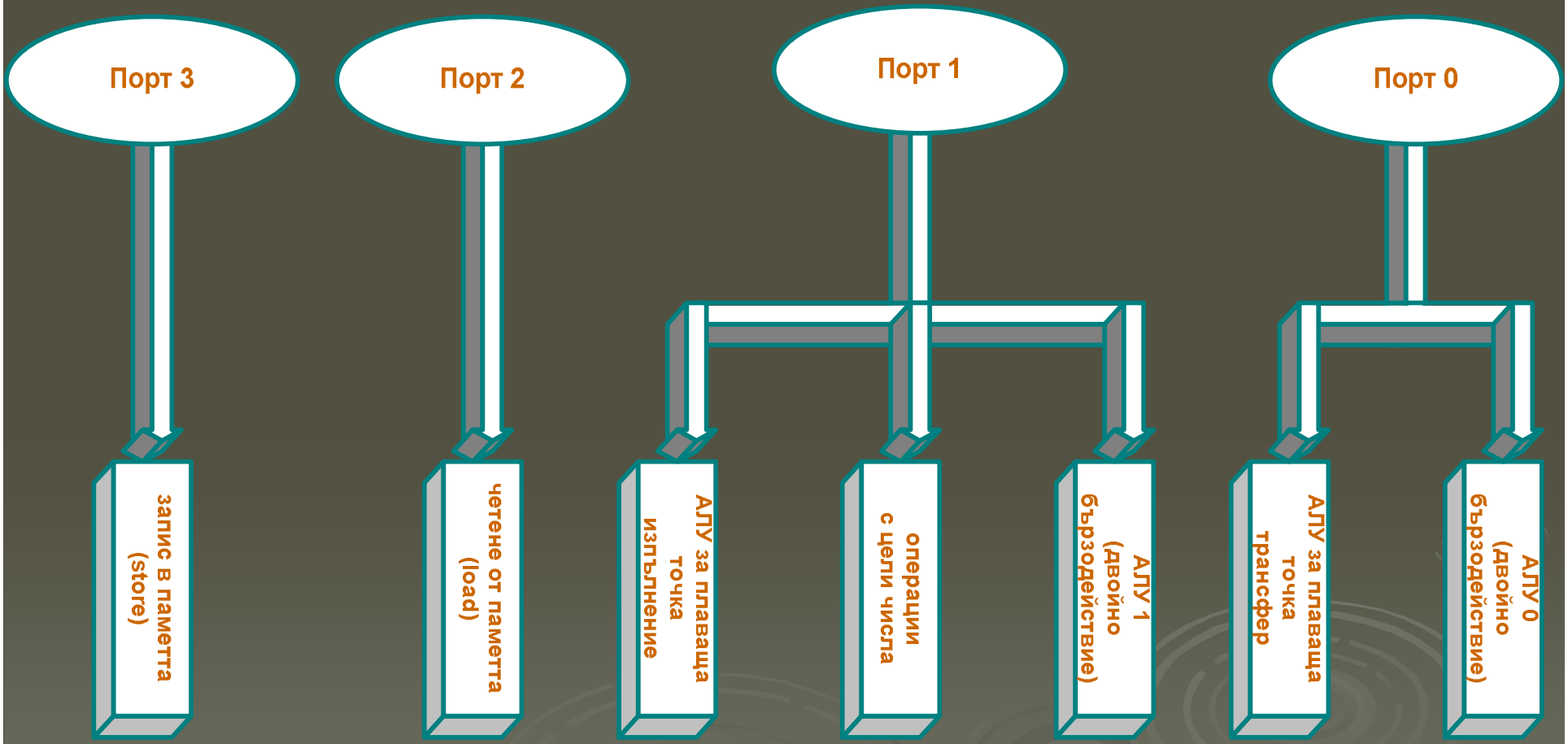








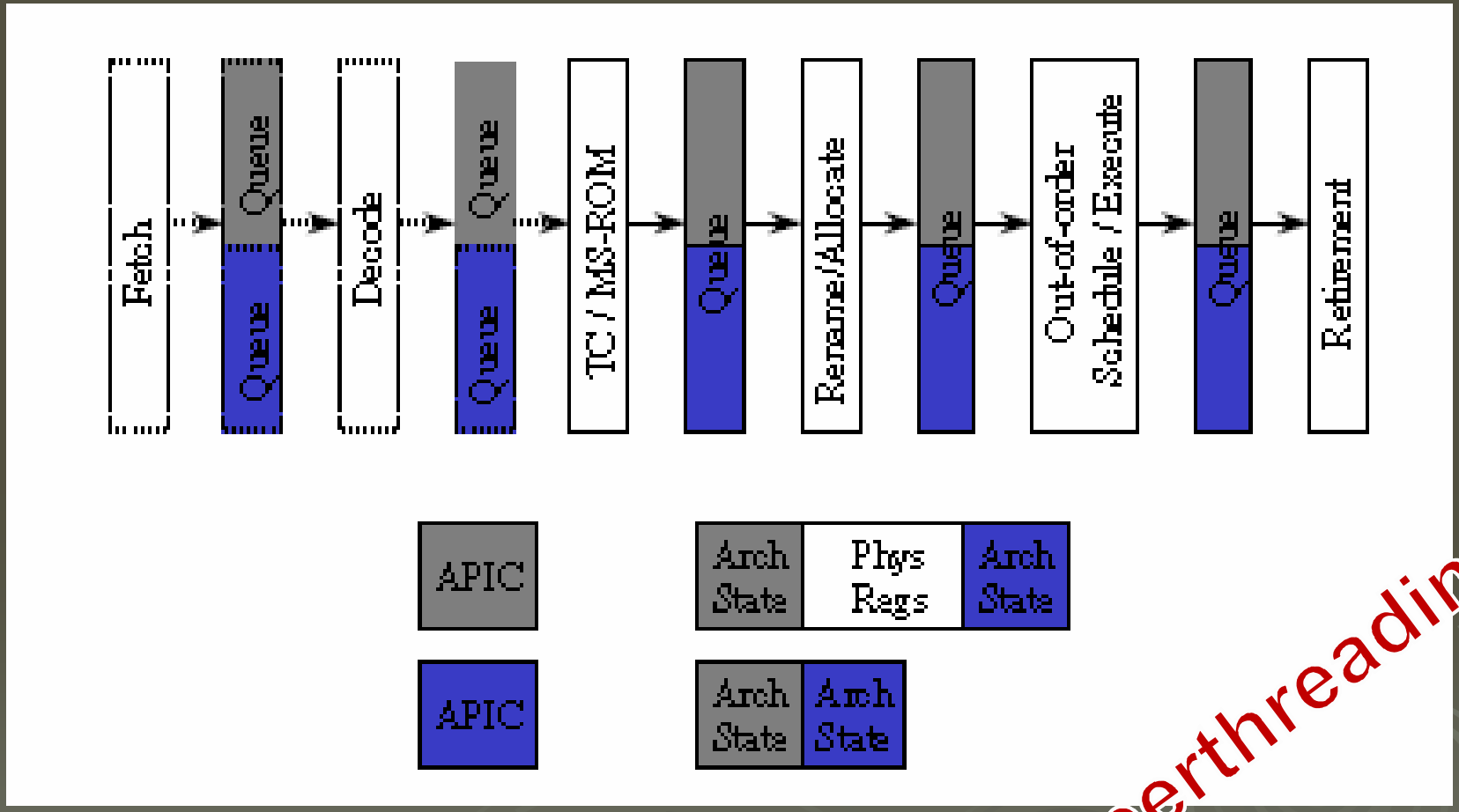
Изпълнителни устройства в Intel® NetBurst™ Микроархитектура



HYPERTHREADING

- Подход за повишаване на производителността – кеширане на декодираните микро-операции, така че процесорът да ги извлича от специален кеш, вместо да ги декодира отново.
- Execution Trace Cache на Intel's NetBurst Microarchitecture (Pentium 4) представлява най-популярната имплементация на този подход.

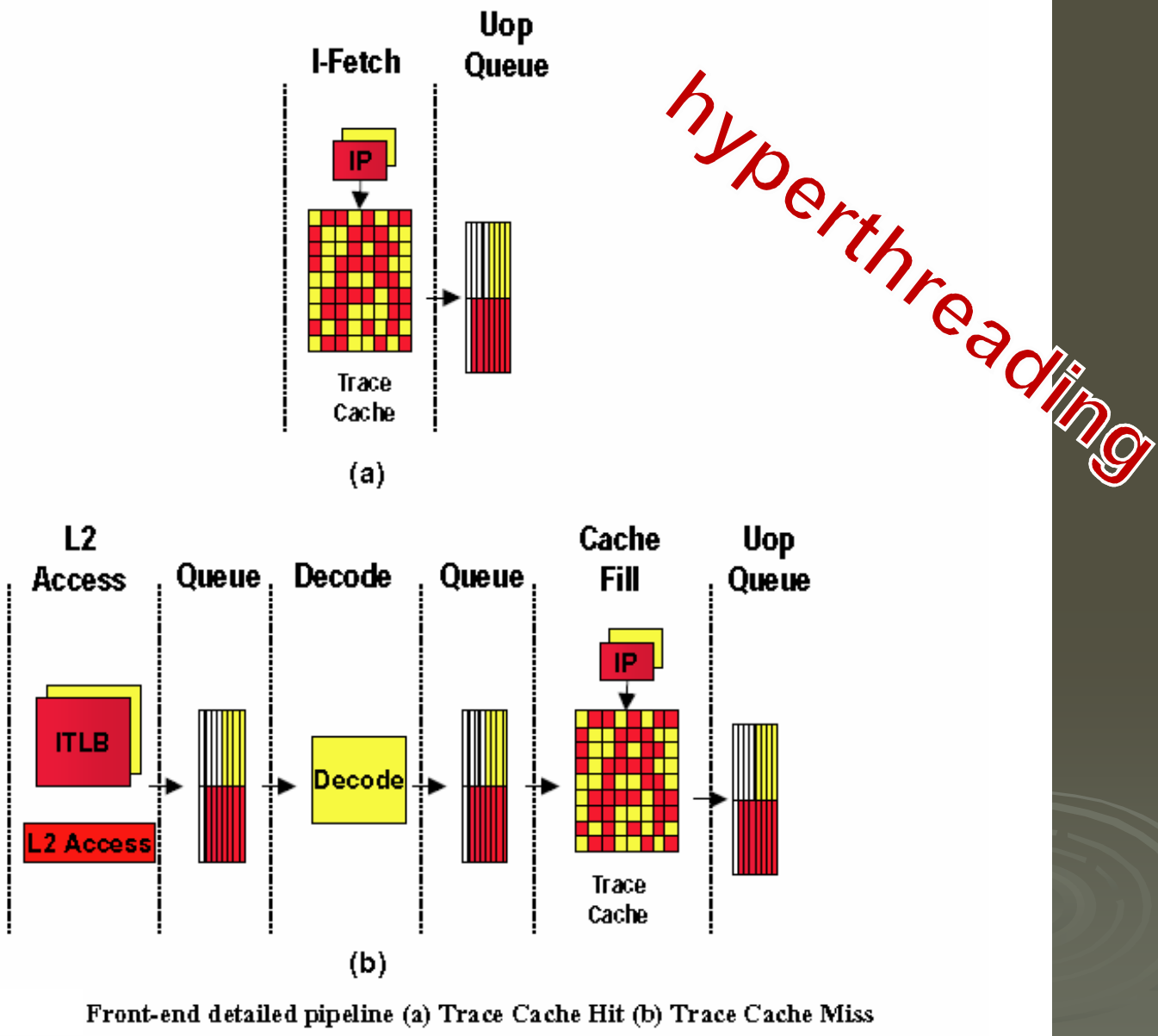
Инструкционен конвейер на Intel Xeon



Изпълнителен кеш за инструкции Execution trace cache (TC)

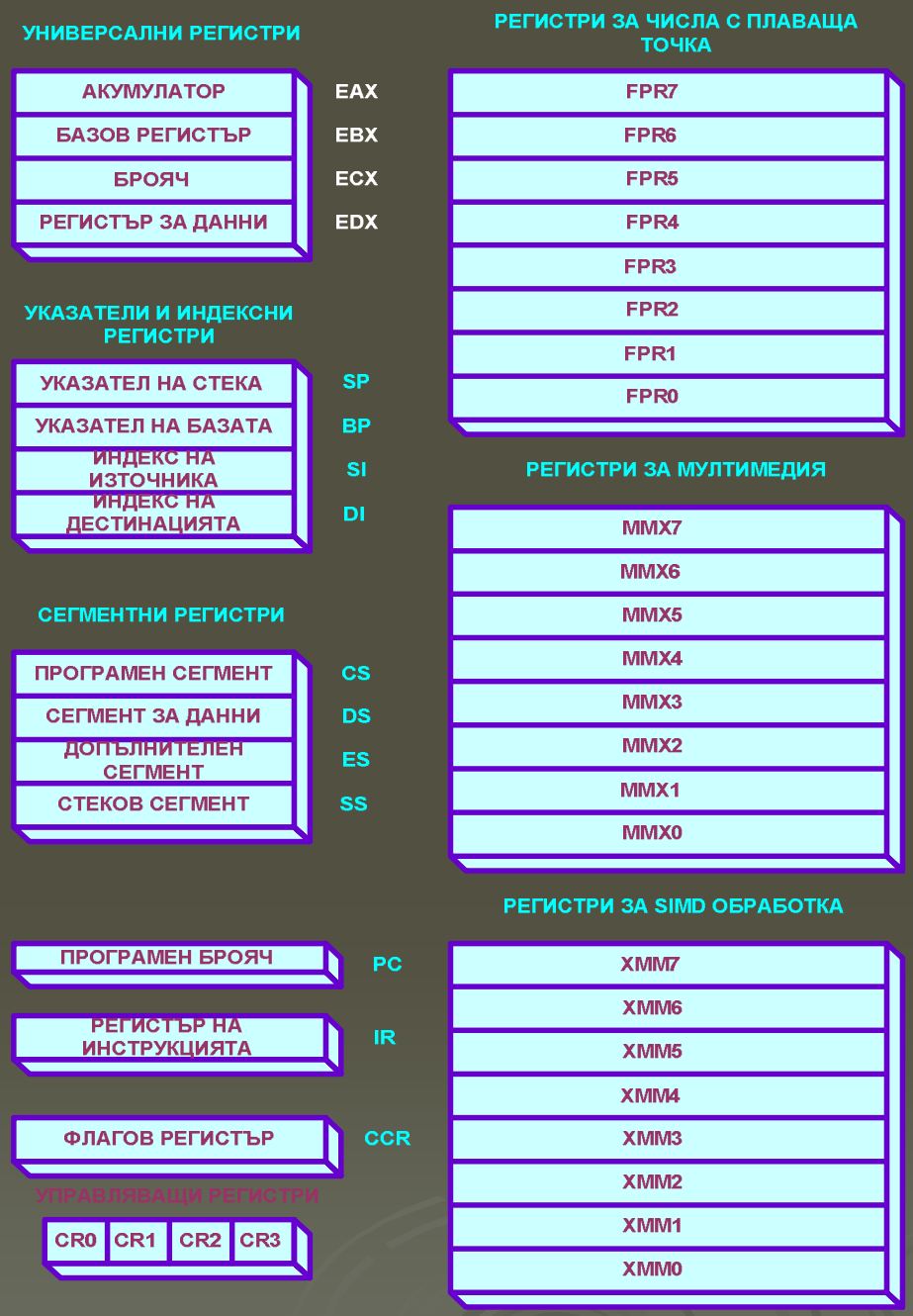
- Съхранява декодираните инструкции, наречени микрооперации или “*uops*”
- Две множества от указатели на инструкции независимо проследяват изпълнението на двете програмни нишки
- TC е споделен кеш за двата логически процесора
- През всеки машинен цикъл се прави арбитраж между двата логически процесора за достъпа до TC - алтернативно
- Всяка линия в TC има таг с информация за нишката
- TC е 8-пътно асоциативен, актуализацията е на базата на алгоритъма LRU

Execution trace cache

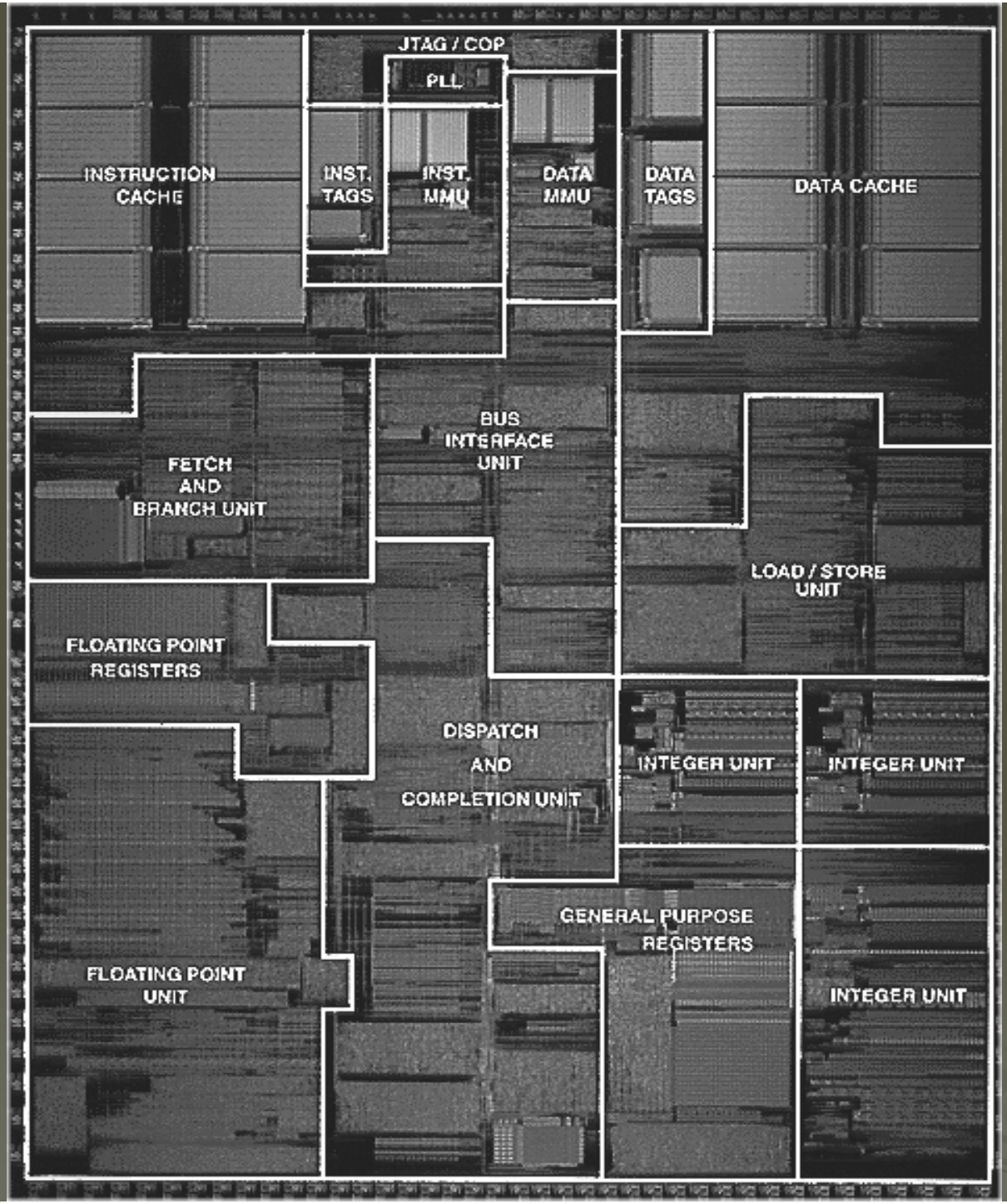


Front-end detailed pipeline (a) Trace Cache Hit (b) Trace Cache Miss

hyperthreading



ПРИМЕРНА РЕГИСТРОВА СТРУКТУРА НА СЪВРЕМЕНЕН ПРОЦЕСОР


JTAG / COP
PLL
INSTRUCTION CACHE
INST. TAGS
INST. MMU
DATA MMU
DATA TAGS
DATA CACHE
BUS INTERFACE UNIT
FETCH AND BRANCH UNIT
LOAD / STORE UNIT
FLOATING POINT REGISTERS
DISPATCH AND COMPLETION UNIT
INTEGER UNIT
INTEGER UNIT
GENERAL PURPOSE REGISTERS
FLOATING POINT UNIT
INTEGER UNIT

Power PC