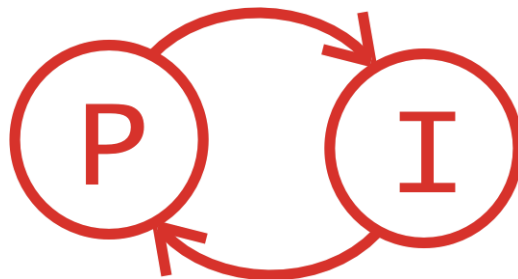


## Podrobniji dijagram stanja PRIBAVI – IZVRŠI

- Interpretacijski dijagram
- Protočna struktura
- Ganttov dijagram
- Usporedba pogodnosti CISC i RISC za protočnu izvedbu
- Modeli I. i II. generacije RISC procesora
- Hazardi u protočnoj strukturi

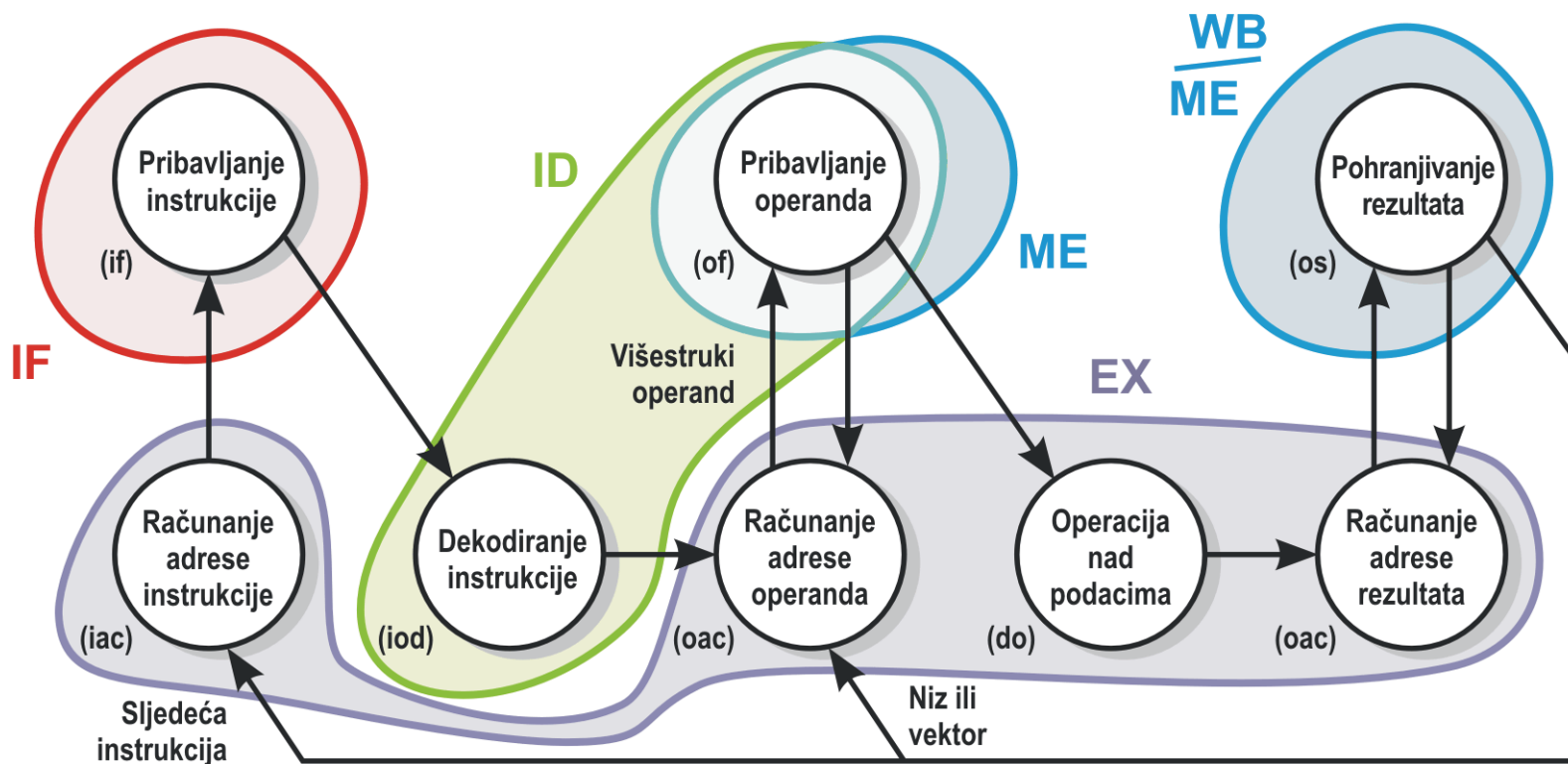
## Dijagram stanja PRIBAVI (P) – IZVRŠI (I)





- Pribavljanje instrukcije (if – instruction fetch)
  - čitanje instrukcije iz memorije i njeno premještanje u CPU
- Dekodiranje instrukcije (iod – instruction operation decoding)
  - analiza i dekodiranje instrukcije u cilju određivanja tipa operacije koja će se izvesti te određivanja tipa operanada
- Računanje adrese operanada (oac – operand address calculation)
  - ako se operacija referencira na operand koji je pohranjen u memoriji ili raspoloživ u U/I podsustavu određuje se efektivna adresa operanda
- Dohvat operanda (of – operand fetch)
  - dohvat operanada iz memorije ili U/I podsustava

- Operacija na podacima/operandima (do – data operation)
  - izvodi se operacija (specificirana operacijskim kodom instrukcije)
- Pohranjivanje rezultata (os –operand store)
  - upisuje se rezultat u memoriju ili u I/U podsustav



Protočna izvedba:

<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>ME</b>	<b>WB</b>
-----------	-----------	-----------	-----------	-----------

Protočni segmenti:

**IF (Instruction Fetching):** Pribavljanje instrukcije. Upotrebljava se programsko brojilo PC za dohvat sljedeće instrukcije. Instrukcije se obično nalaze u priručnoj memoriji (engl. cache) koja se čita tijekom faze PRIBAVI.

**ID (Instruction decoding and operand fetching):** Dekodiranje instrukcije i dohvat operandi (!?) – **istodobno** dekodira operacijski kod instrukcije i dohvaćaju operandi iz skupa registara

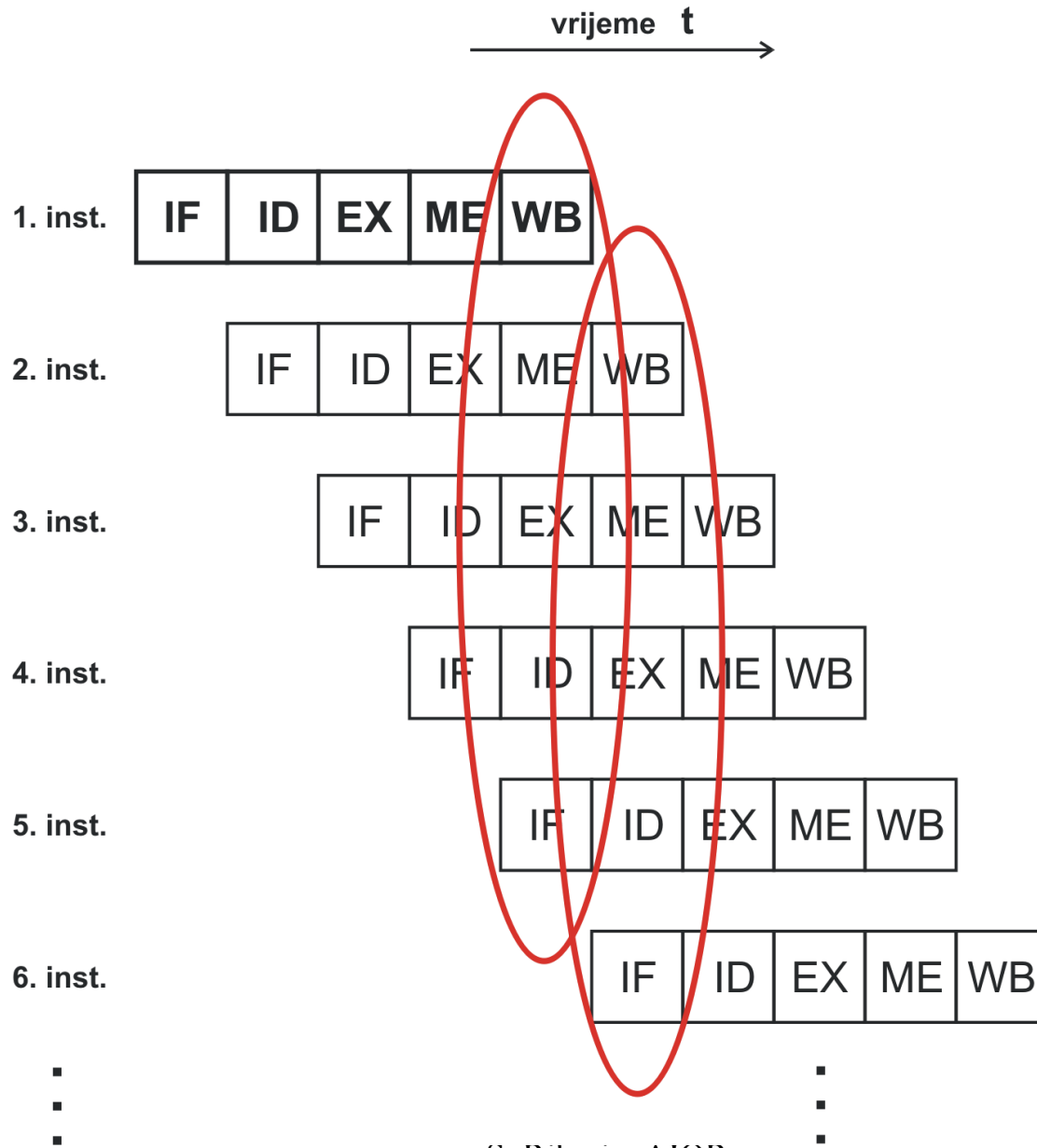
**istodobno (?) prije dekodiranja operacijskog koda !**

**EX (Instruction execution):** Izvođenje (obavljanje) instrukcije – obavlja se operacija specificirana operacijskim kodom. Za instrukcije koje naslovljavaju memoriju (*load*, *store*) u ovom se protočnom segmentu **računa efektivna adresa-**

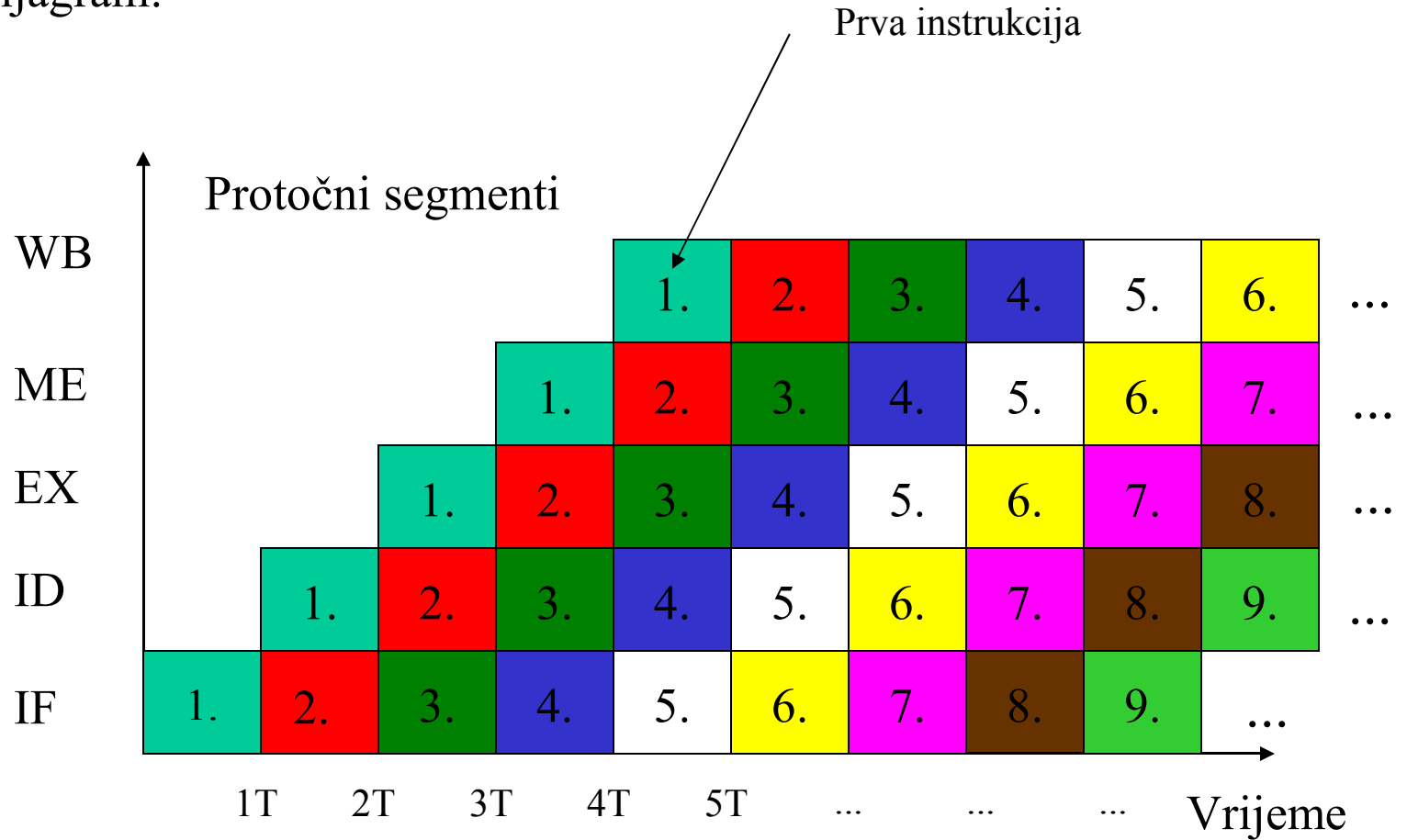
**ME (Memory access):** Pristup memoriji. Izvode se instrukcije *load* i *store*. Obično se upotrebljava priručna memorija.

**WB (Result write-back):** Upis rezultata. Rezultat operacije se upisuje natrag skup registara.





Ganttov dijagram:



**Hazard** – situacija u protočnoj strukturi koja izaziva poremećaje i kašnjenje u “glatkom” protoku zadataka kroz nju

Hazardi sprečavaju da sljedeća instrukcija u nizu bude izvedena u za nju predviđenoj periodi taktnog signala.

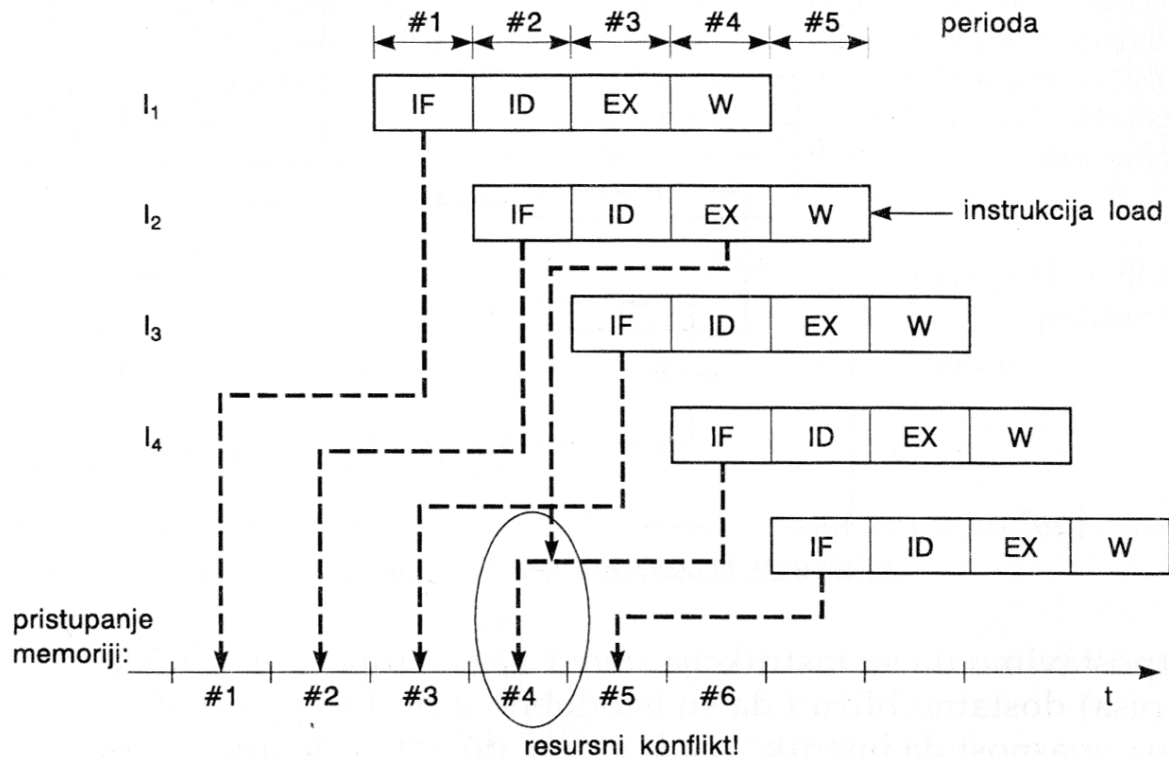
Tri razreda hazarda:

- strukturni hazard
- podatkovni hazard
- upravljački hazard

/S. Ribarić, Arhitektura računala RISC i CISC, Školska knjiga, Zagreb, 1996./

- Ako se neka kombinacija instrukcija ne može izvesti zbog sukobljavanja oko sredstava (resursa) – **strukturni hazard**

**resursni konflikti**



## Podatkovni hazard

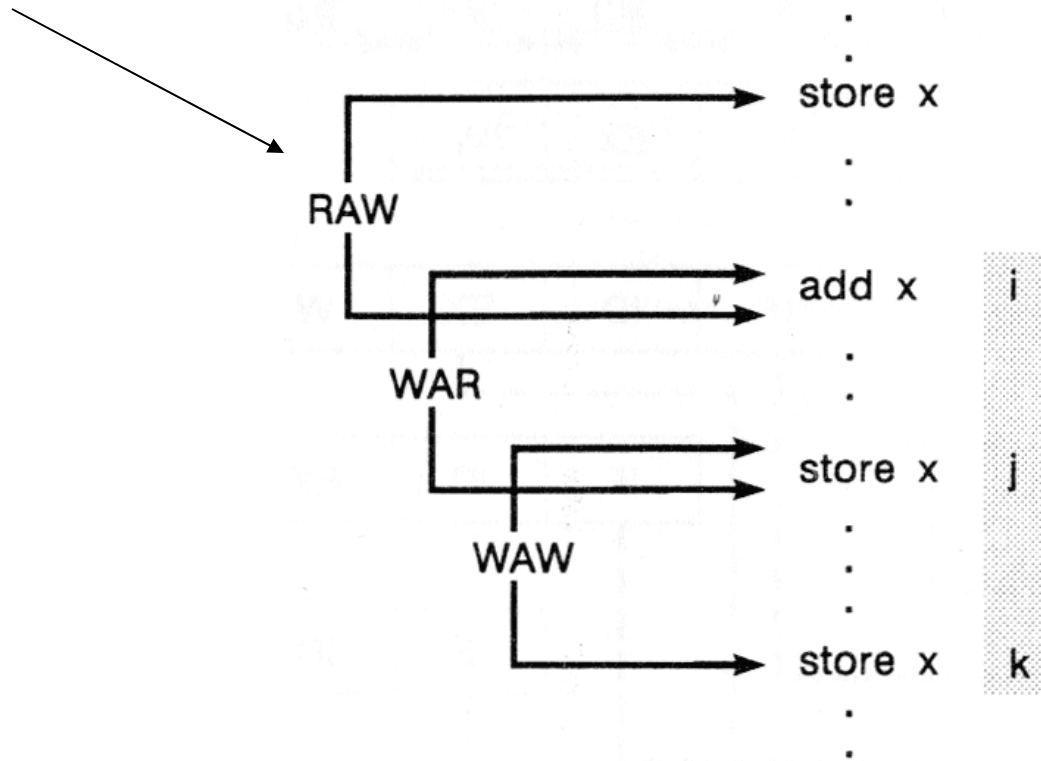
Podatkovni hazard nastupa zbog *međuzavnosti podataka*

- nastaje kad dvije ili više instrukcija, koje se nalaze u protočnoj strukturi, pristupaju istom podatku ili modificiraju isti podatak

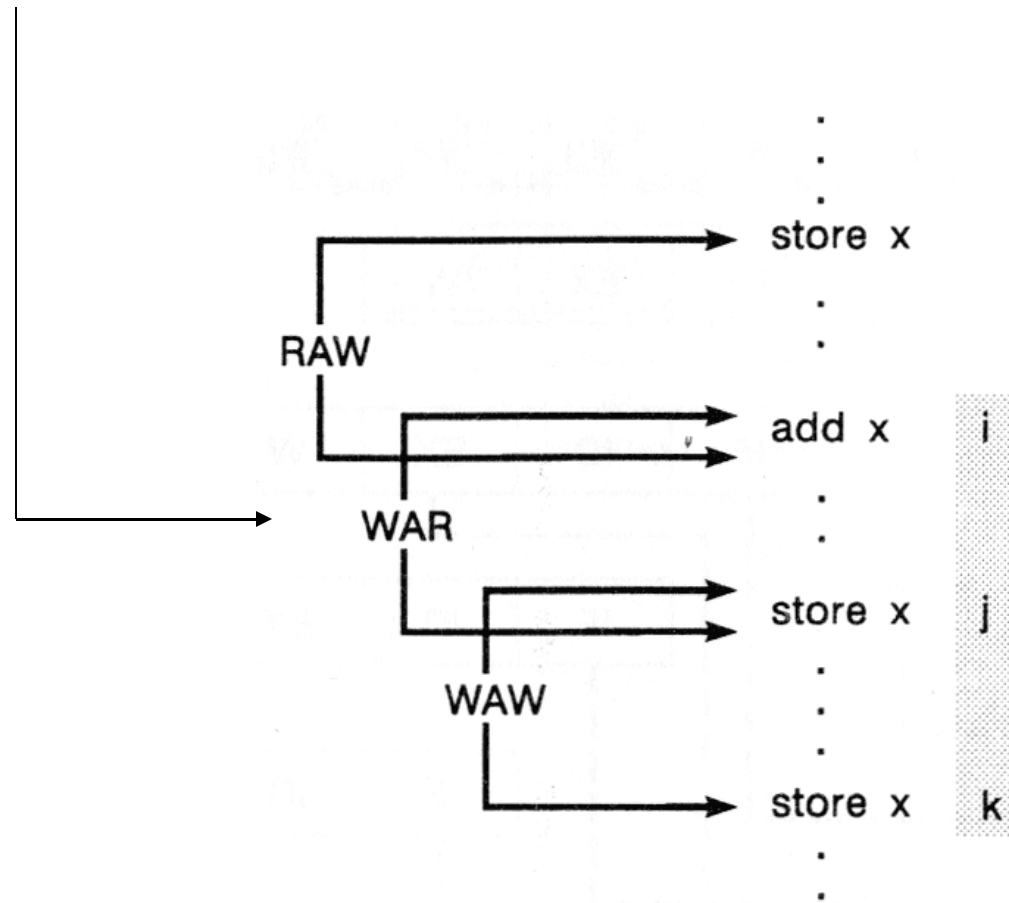
Tri vrste podatkovnih hazarda:

1. RAW – read after write /čitanje poslije upisa/
2. WAR – write after read /pisanje poslije čitanja/
3. WAW – write after write /pisanje poslije pisanja/

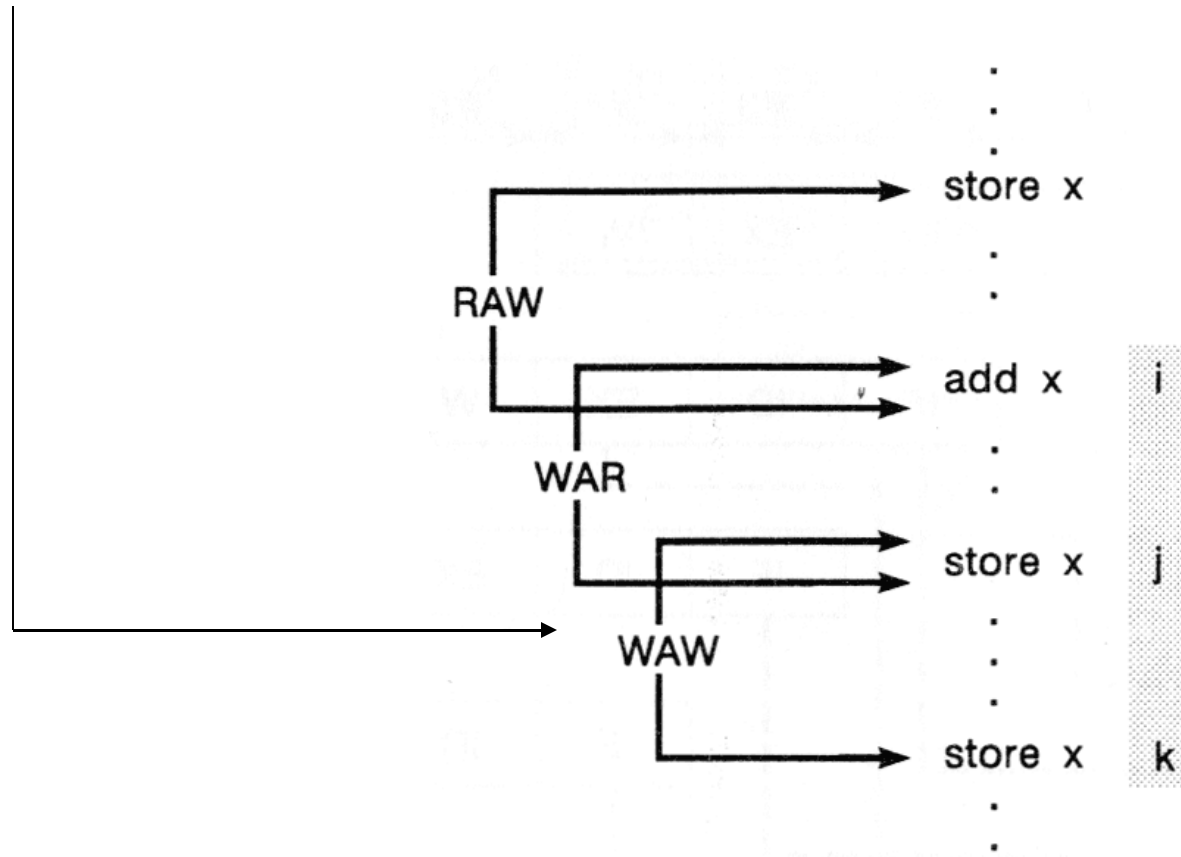
RAW – postoji opasnost da instrukcija *add x* dohvati prije operand s lokacije *x* negoli instrukcija *store x* upiše novu vrijednost na lokaciji *x*



WAR – instrukcija  $j$  (*store x*) koja logički slijedi instrukciji  $i$  **želi promijeniti podatak** na lokaciji  $x$  (koju čita instrukcija  $i$ )

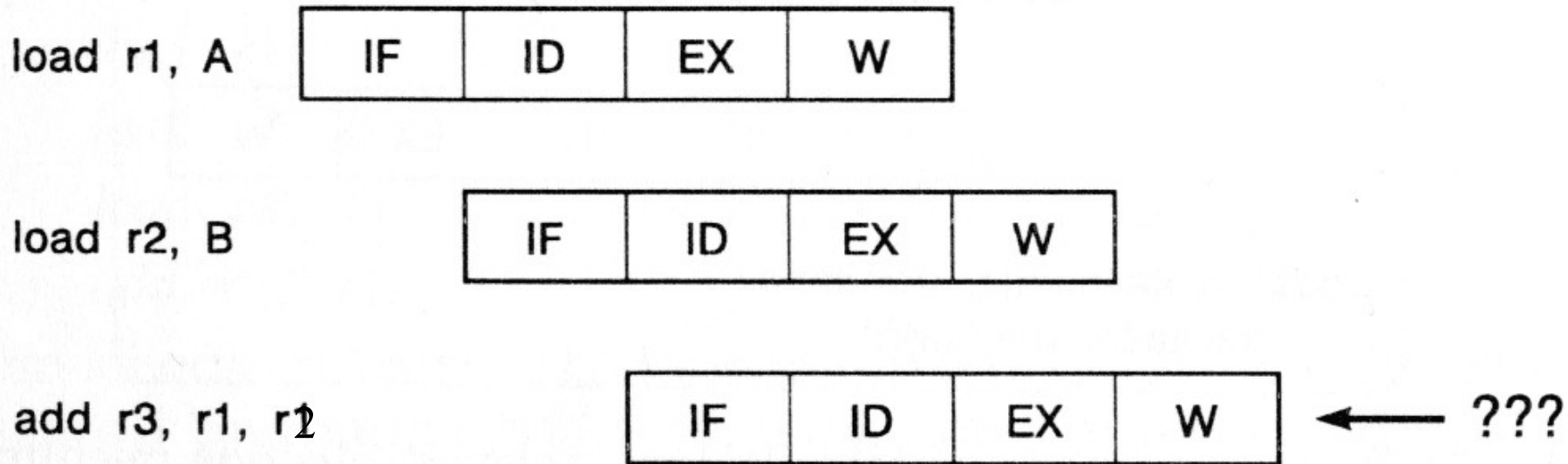


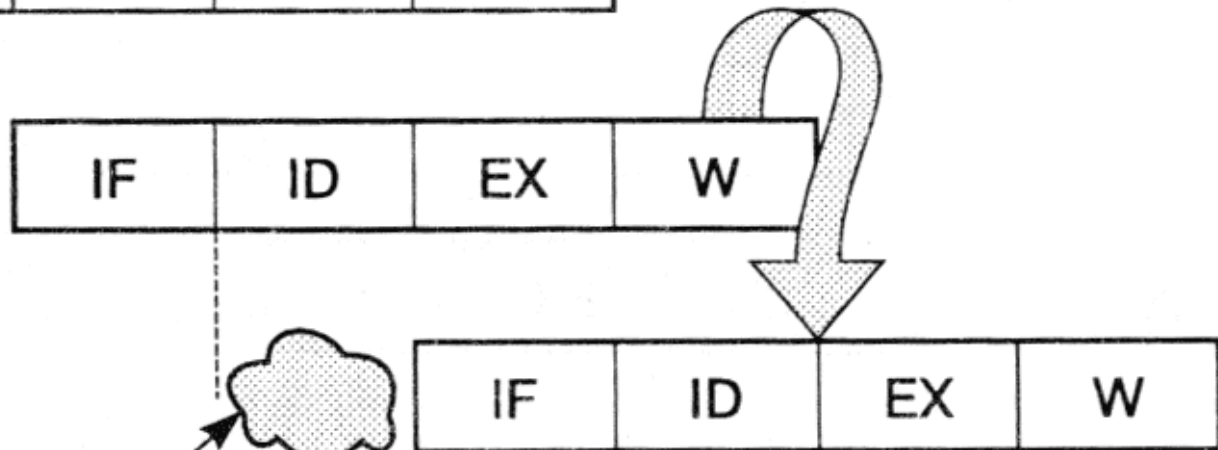
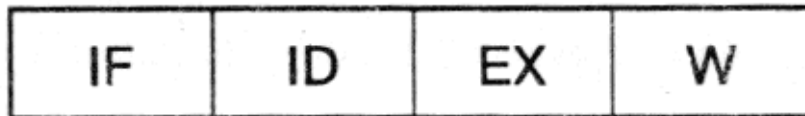
WAW – obje instrukcije  $j$  i  $k$  žele obnoviti vrijednost podatka na memorijskoj lokaciji  $x$  – ako se instrukcija  $j$  izvede **poslije** instrukcije  $k$



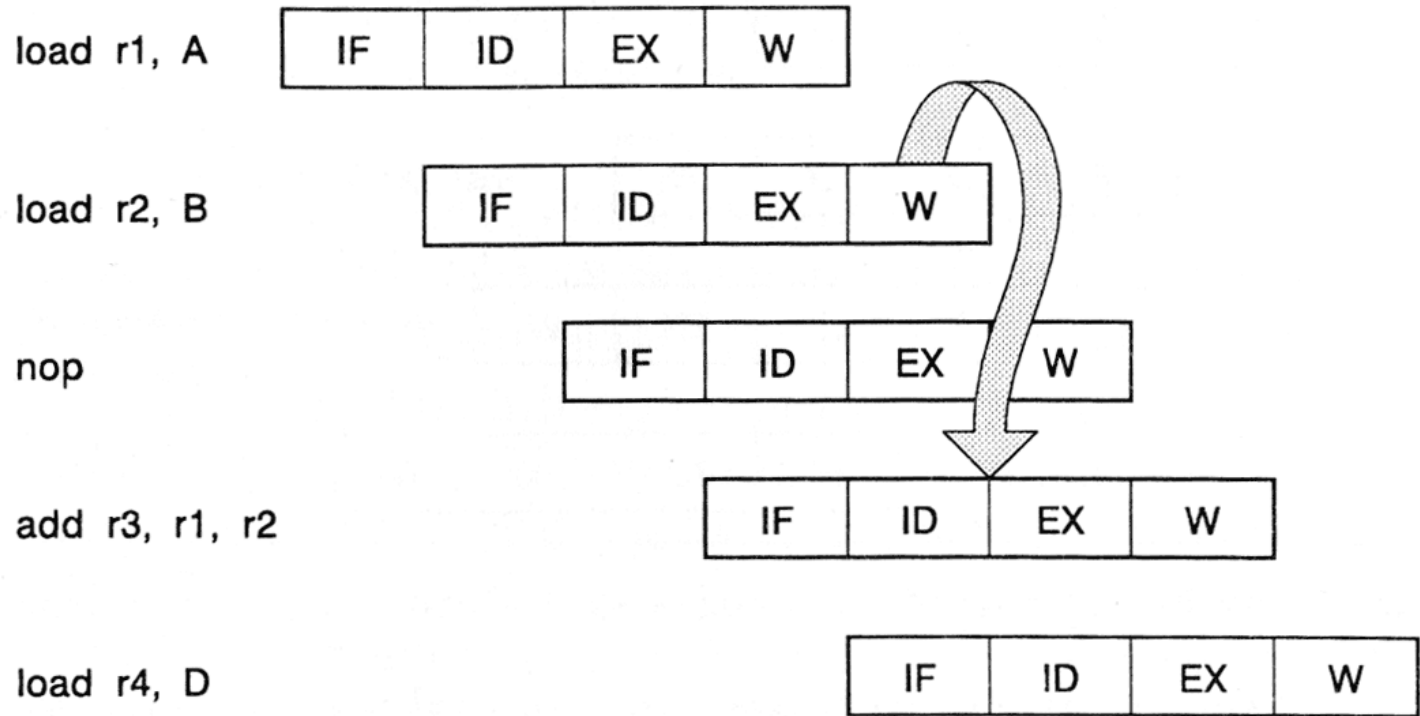


Hazard vrste RAW prisutan je u arhitekturi RISC tijekom izvođenja instrukcije *load*





dodatno kašnjenje, odnosno  
"protočni mjehurić"



Mjesto instrukcije u slijedu instrukcija **neposredno nakon** instrukcije *load* naziva se “**priključak load za kašnjenje**” (engl. Load-delay slot)

Primjer:

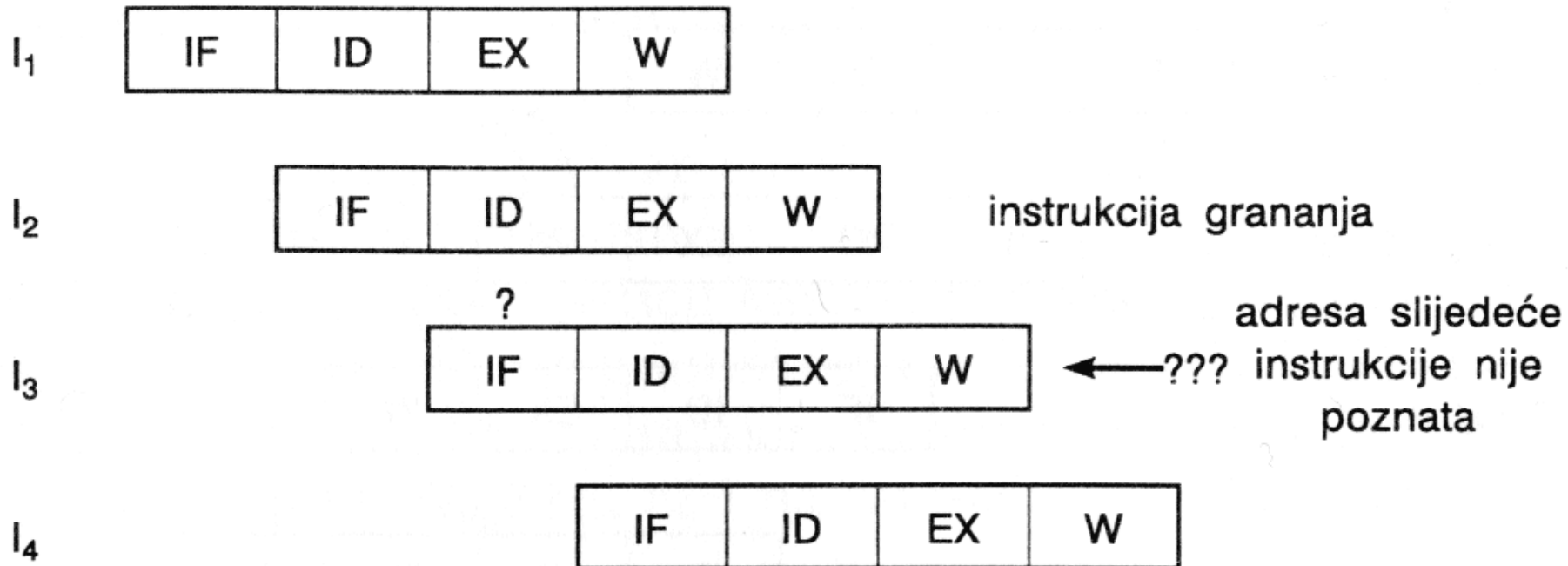
$C := A + B, E := D$

*load r1, A*  
*load r2, B*  
*add r3, r1, r2* ← priključak  
*load r4, D*

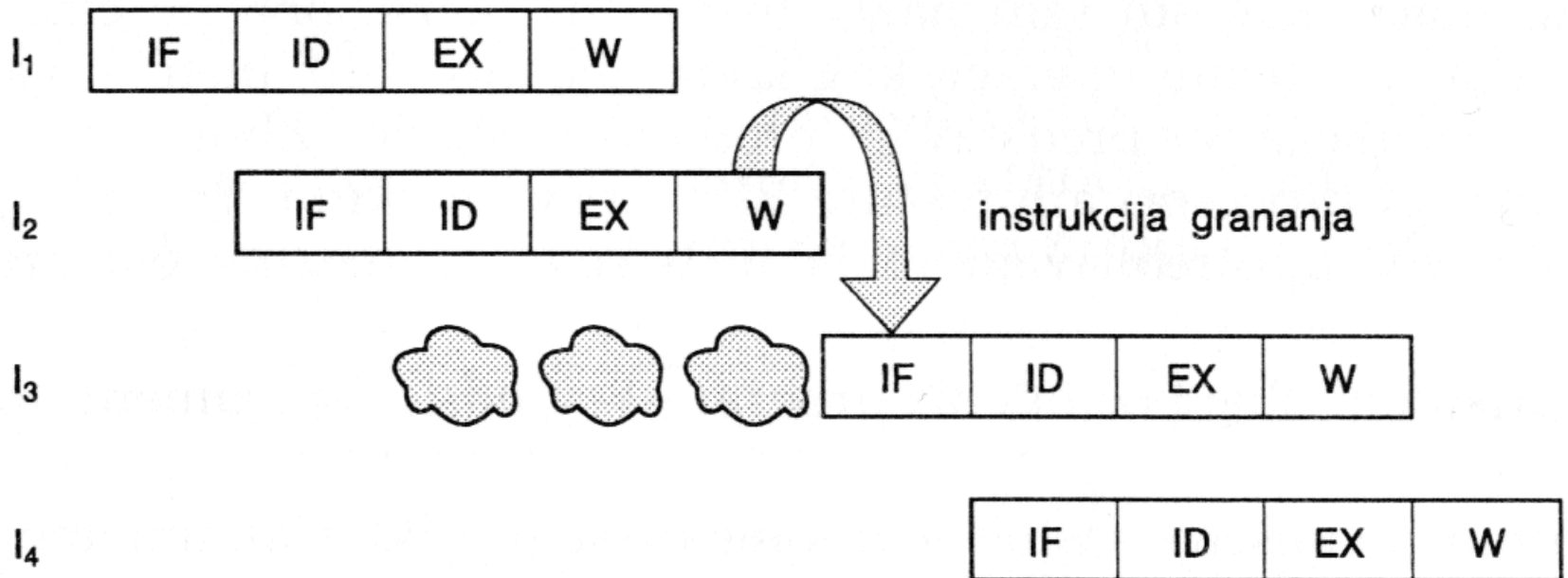
*load r1, A*  
*load r2, B*  
*load r4, D*  
*add r3, r1, r2*

## Zakasnjene instrukcije grananja – **upravljajući hazard**

Primjer:



Umetnuti tri (!!)



Nepovoljno utječe na performansu procesora!

Smanjenje kašnjenja može se postići tako da se računanje ciljne adrese grananja i upis te adrese u PC obavi ranije (umjesto u protočnom segmentu W ili EX) - u protočnom segmentu ID

Aktivnost segmenta ID:

$A \leftarrow Rs\ 1, B \leftarrow Rs\ 2;$

$CAG \leftarrow PC + \text{pomak}$

ako cond  $PC \leftarrow CAG$

CAG – ciljna adresa grananja

Kašnjenje samo s jednim mjehurićem!

## Primjer:

```
move r4, r3  
move r1, r2  
jal x ; bezuvjetno grananje  
C: add r5, r5, 1
```

Prevodilac će preinačiti kod:

```
move r4, r3  
move r1, r2  
jal x ; bezuvjetno grananje  
nop  
C: add r5, r5, 1
```

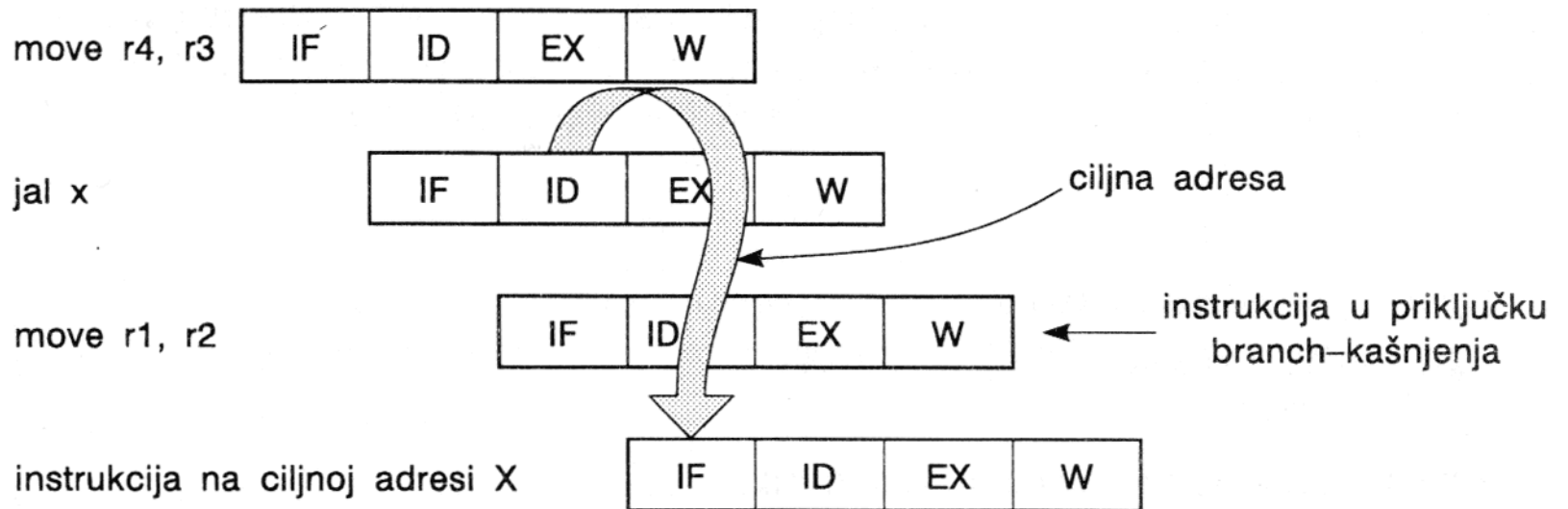


Optimizirajući prevodilac će preinačiti kod:

```
    move r4, r3
    move r1, r2
    jal x      ; bezuvjetno grananje
C: add r5, r5, 1
```

u:

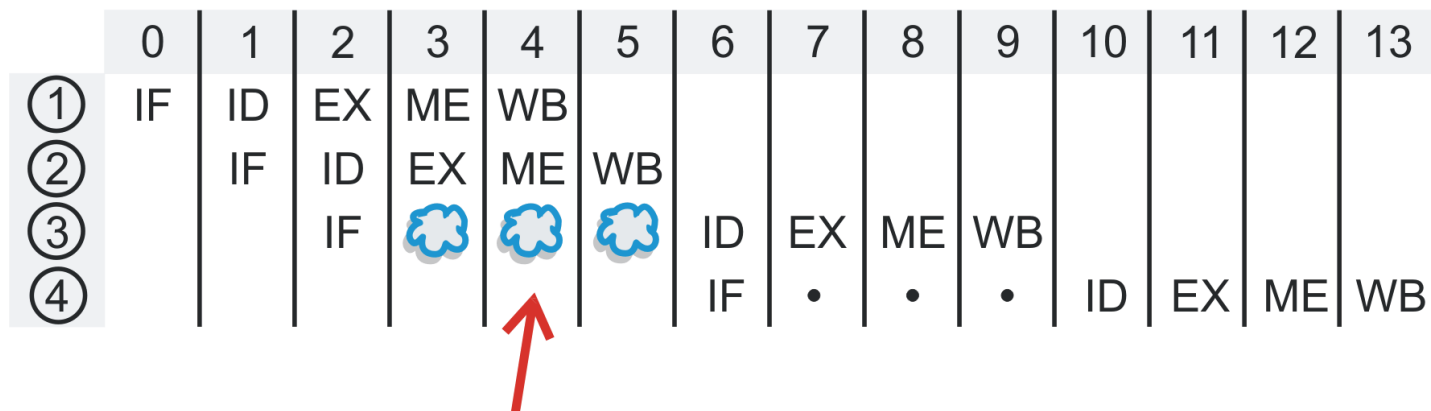
```
    move r4, r3
    jal x      ; bezuvjetno
granje
    move r1, r2
C: add r5, r5, 1
```



## Primjer:

*loadf*     $fp1 = mem(r1 + r2)$   
*loadf*     $fp2 = mem(r3 + disp)$   
*multf*     $fp3 = fp1, fp2$   
*storef*     $mem(r1 + r2) = fp3$

•



**protočni mjehurići**

## Usporedba pogodnosti CISC i RISC za protočnu izvedbu

Kraća perioda signala vremenskog vođenja → povećanje broja perioda za instrukciju.

“posao” jedne instrukcije



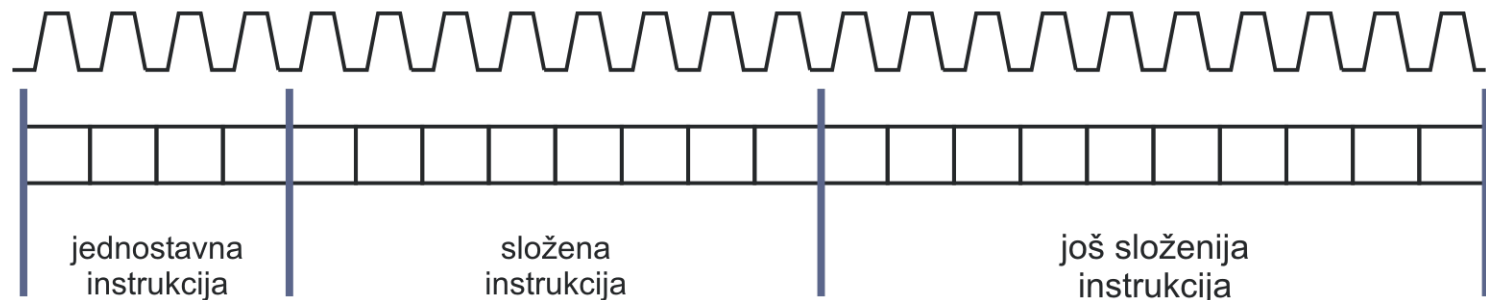
Kraća perioda/veći broj perioda



Dulja perioda/manji broj perioda

**Uobičajeni pristup:** vrijeme trajanja periode tako da dopusti izvođenje najjednostavnijih operacija u jednoj periodi  
- izvođenje složenijih instrukcija/operacija u više perioda

Tipičan slijed instrukcija za CISC procesor:



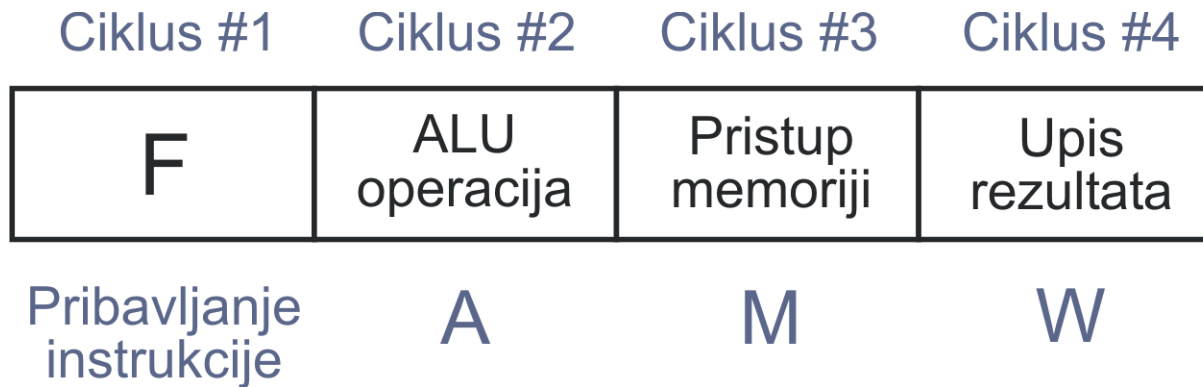
-svaka instrukcija ima dodijeljeno vrijeme upravo onoliko koliko joj je potrebno

Težnja u arhitekturi RISC → jedna perioda po instrukciji

Tehnike koje dopuštaju ostvarivanje težnje:

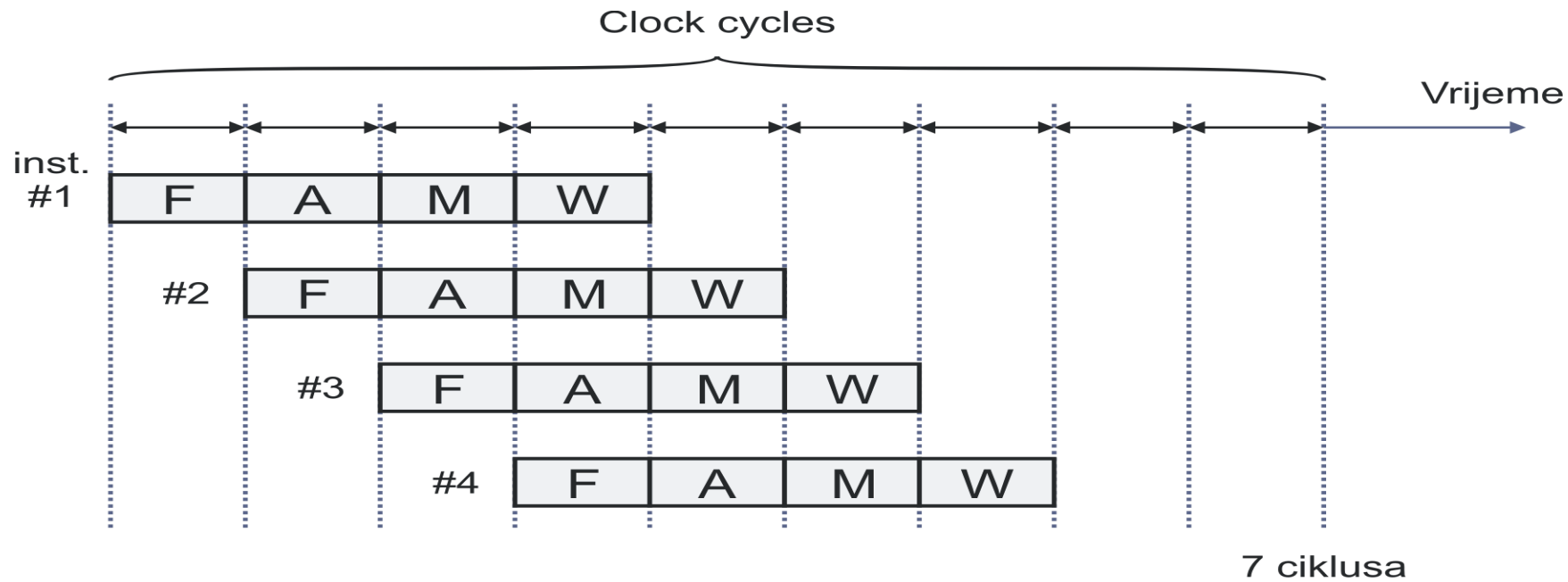
- protočnost
- arhitektura load/store
- “zakašnjele” load instrukcije
- “zakašnjele” branch instrukcije

Protočni segmenti:



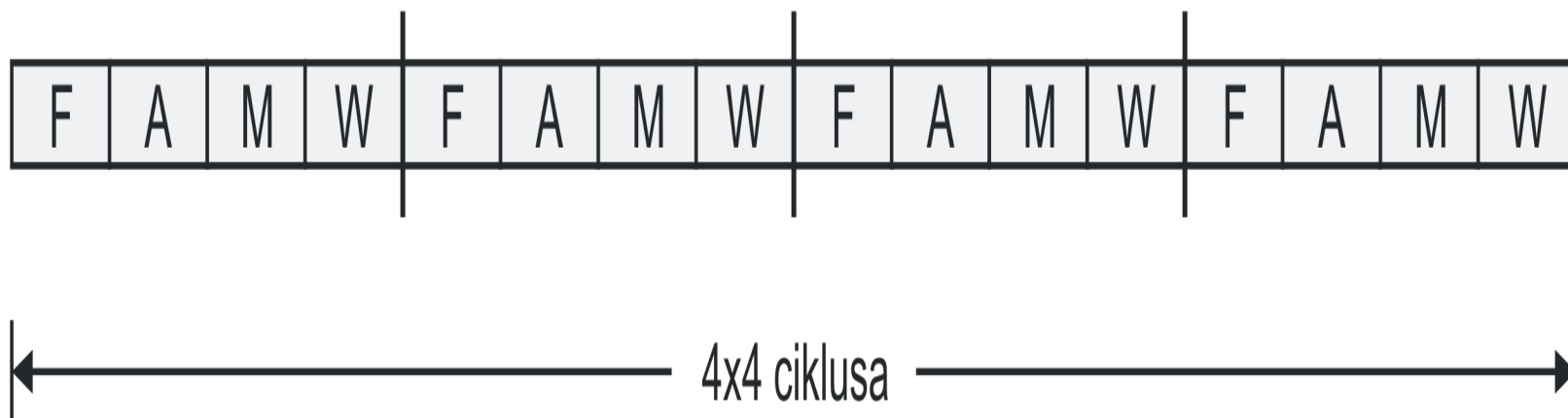
ili







Izvedba u neprotlačnoj strukturi:

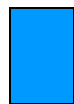
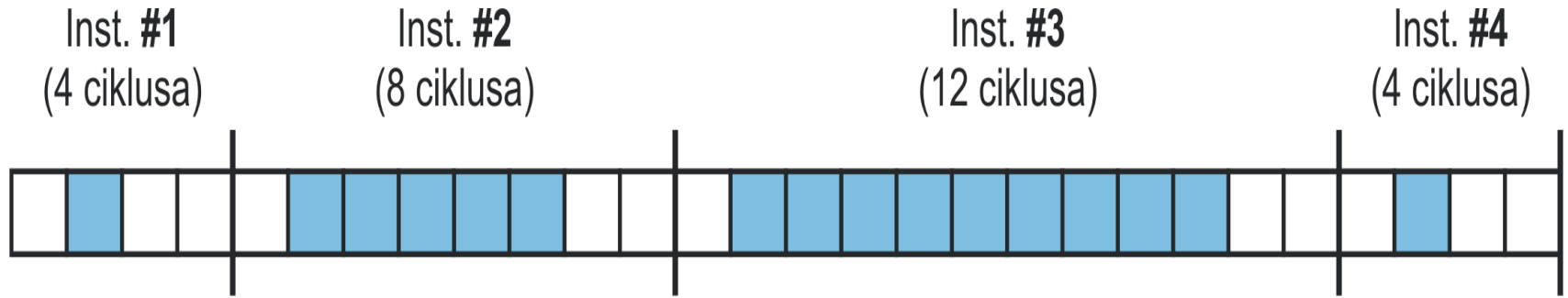


Vrijeme izvođenja u **protočnoj** strukturi: 7 perioda signala vremenskog vođenja

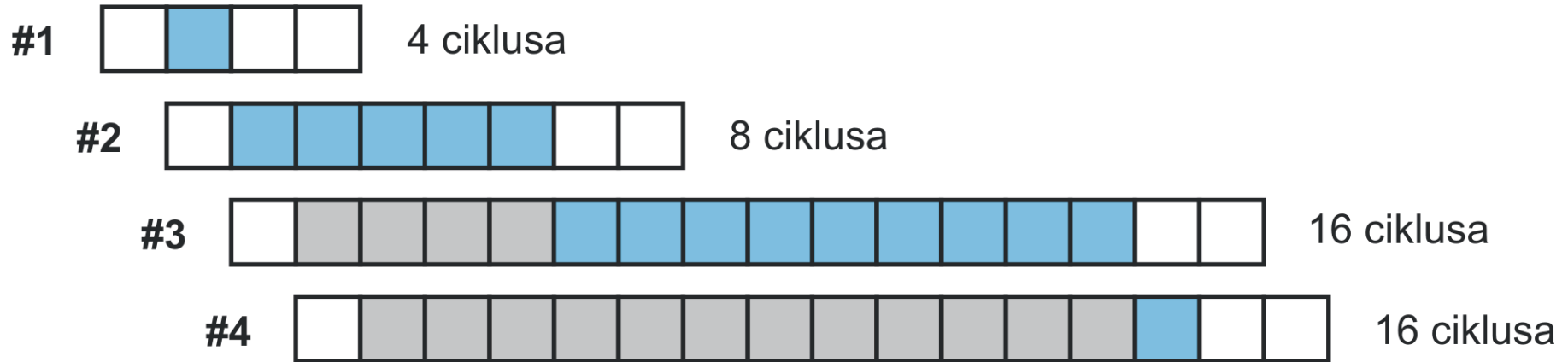
Vrijeme izvođenja u **neprotočnoj** strukturi: 16 perioda



Protočna struktura potencijalno smanjuje broj perioda po instrukciji za faktor jednak “dubini” protočne strukture

Primjer protočnog izvođenja za CISC instrukcije:



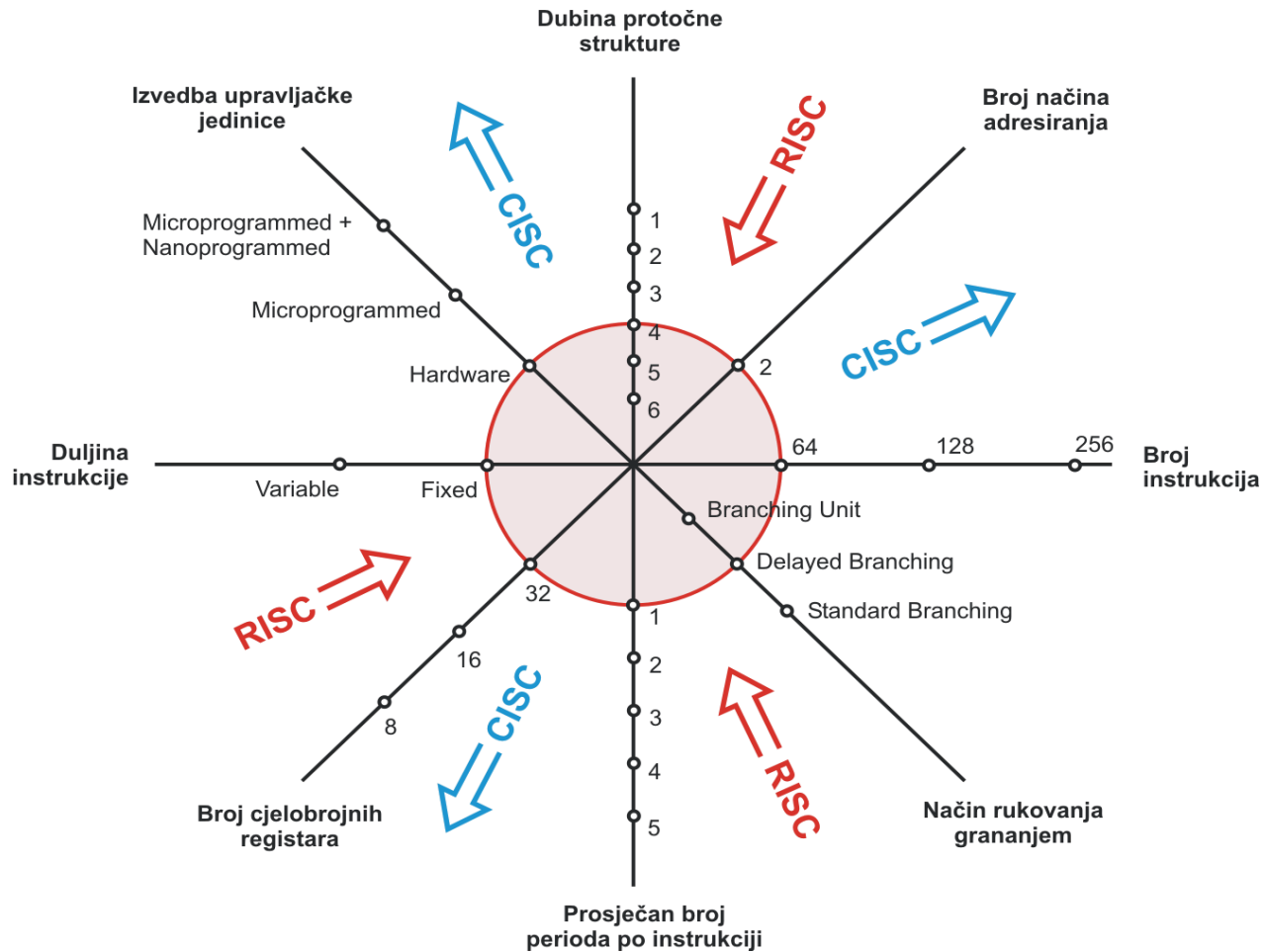
- ekskluzivni zahtjevi za resursom  
(ALU, registri, sklop za posmak,...)



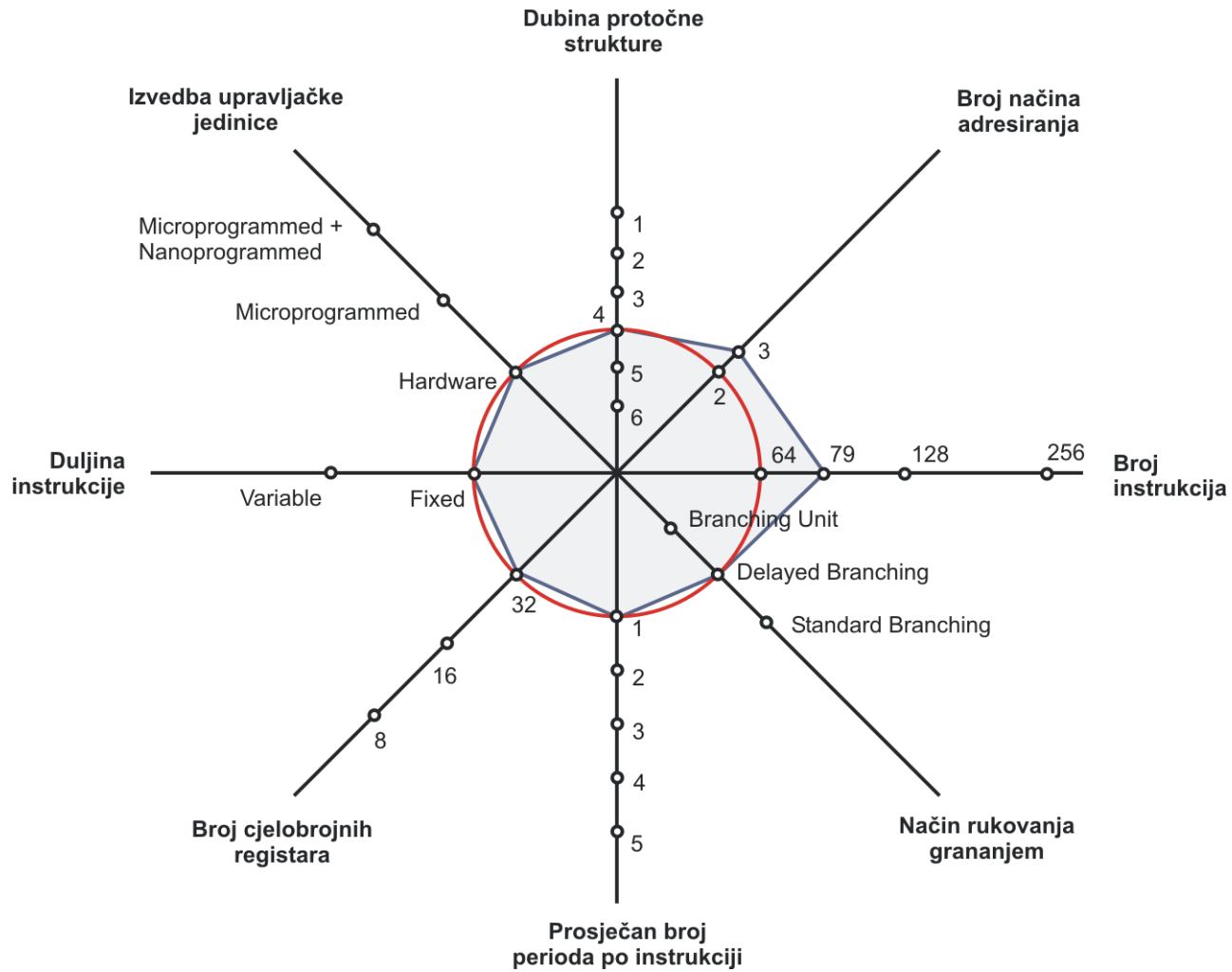
-  - periode kašnjenja
-  - ekskluzivni zahtjevi za resursom (ALU, registri, sklop za posmak,...)

Negativan utjecaj promjenjive duljine trajanja instrukcija u CISC arhitekturi!

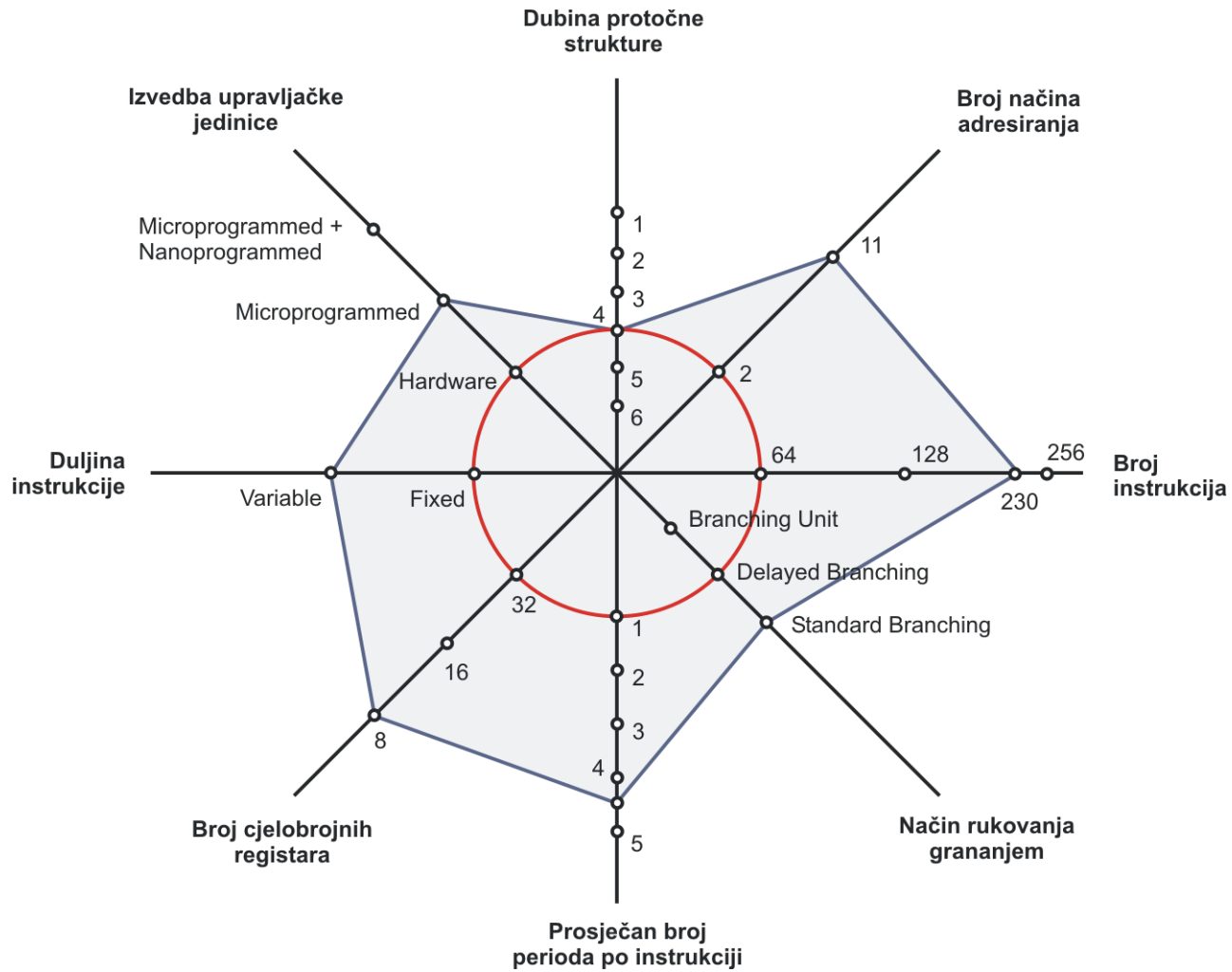
## Kivijat grafovi:



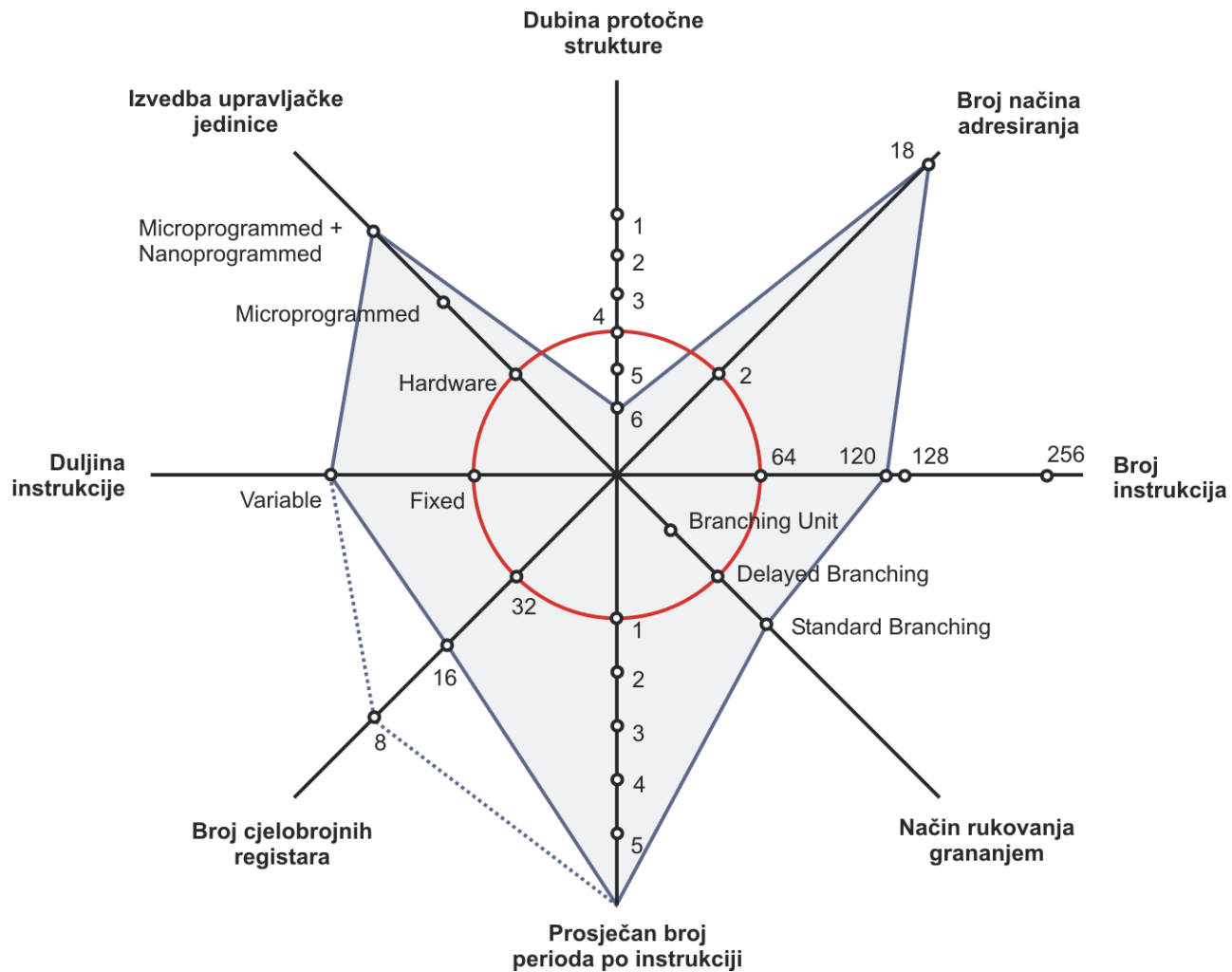
i 860



i486

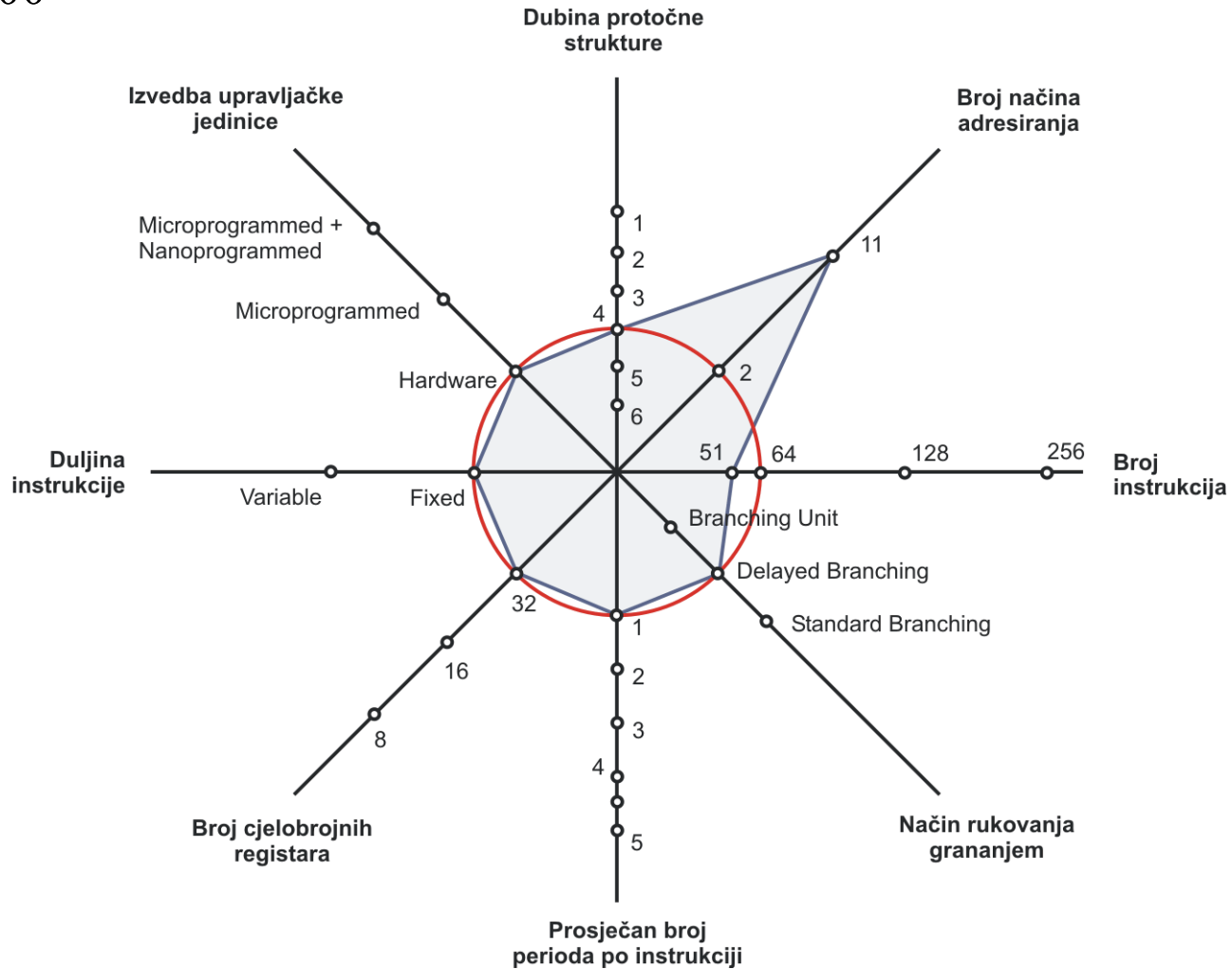


# MC 68040

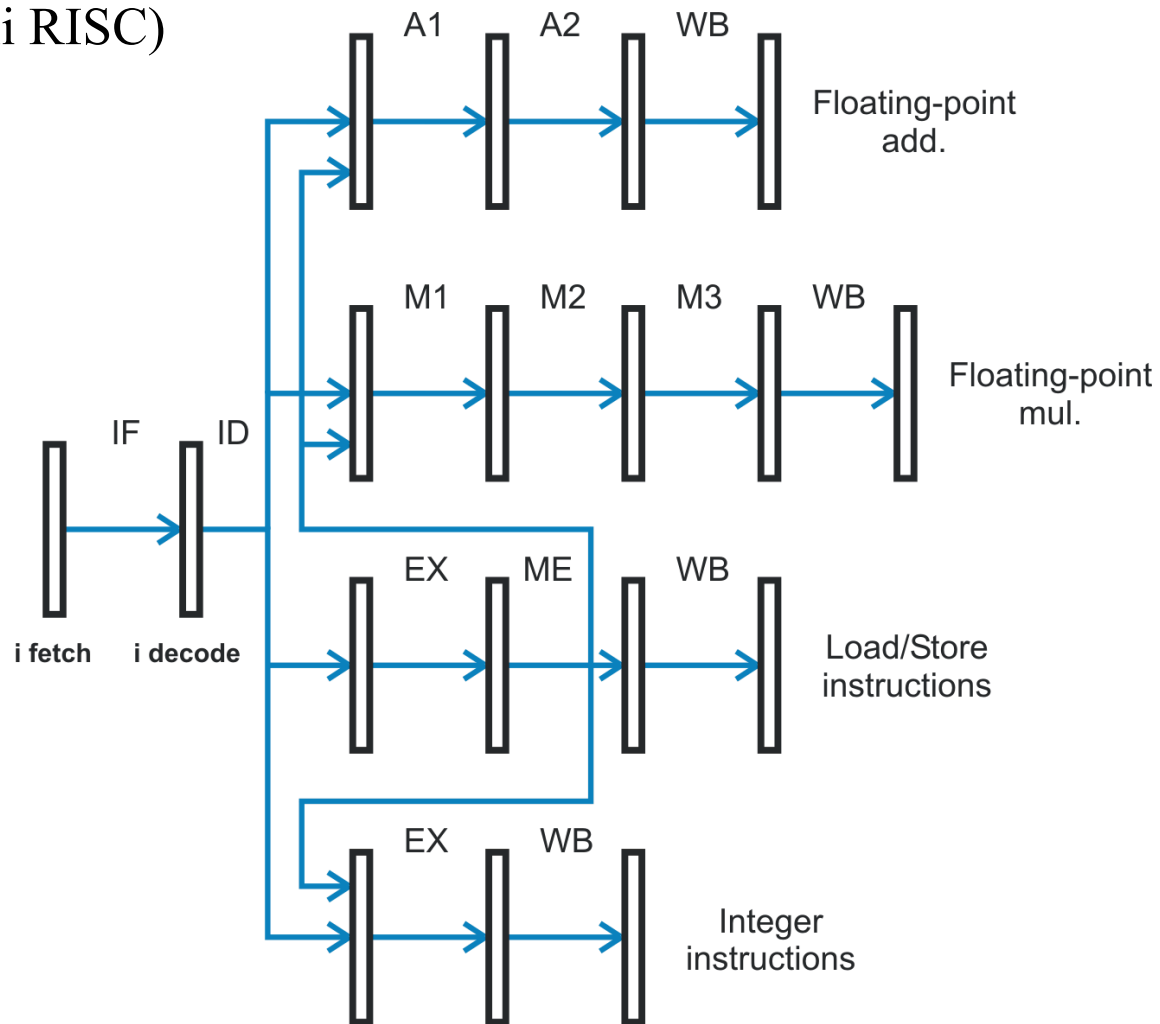




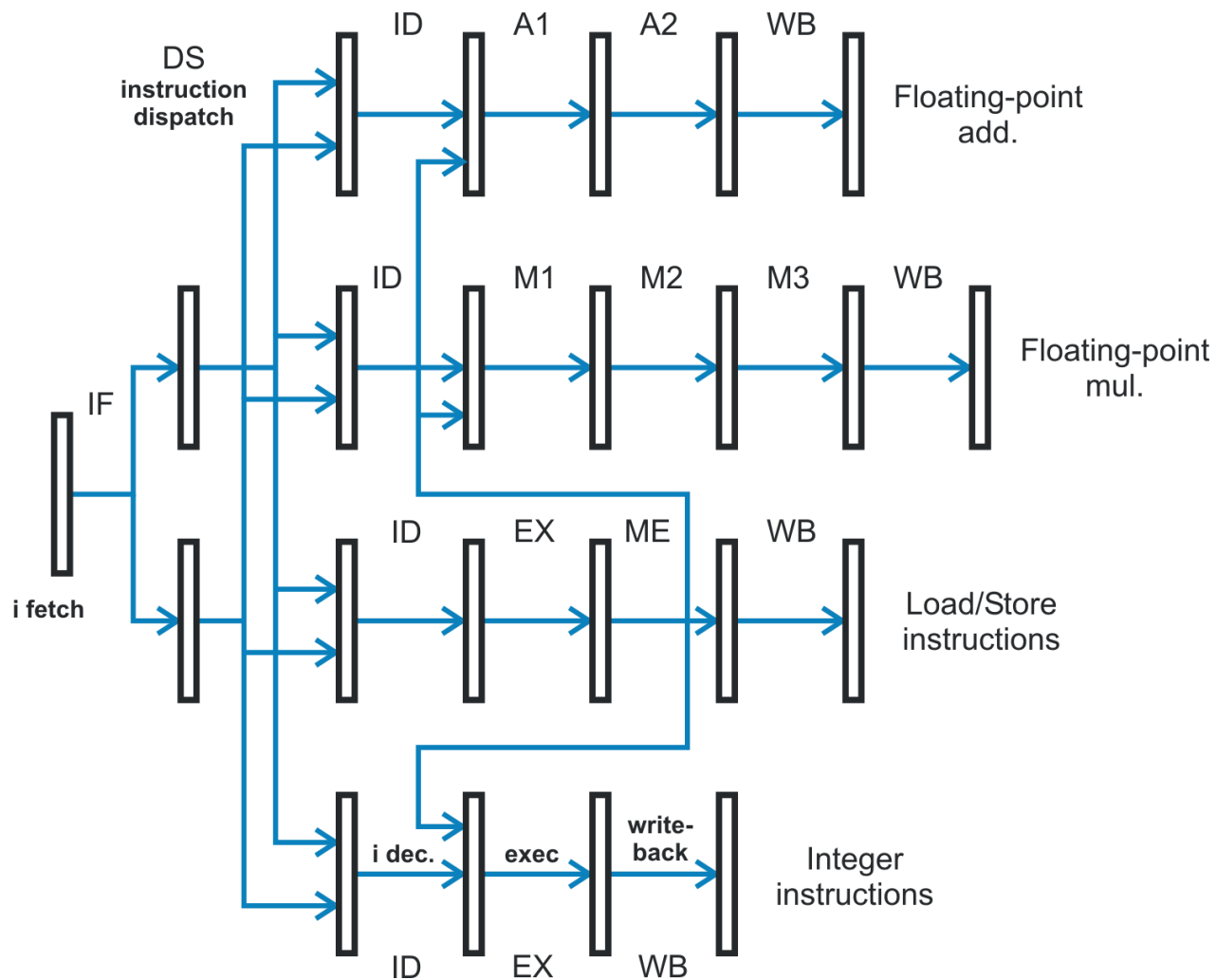
# MC 88100



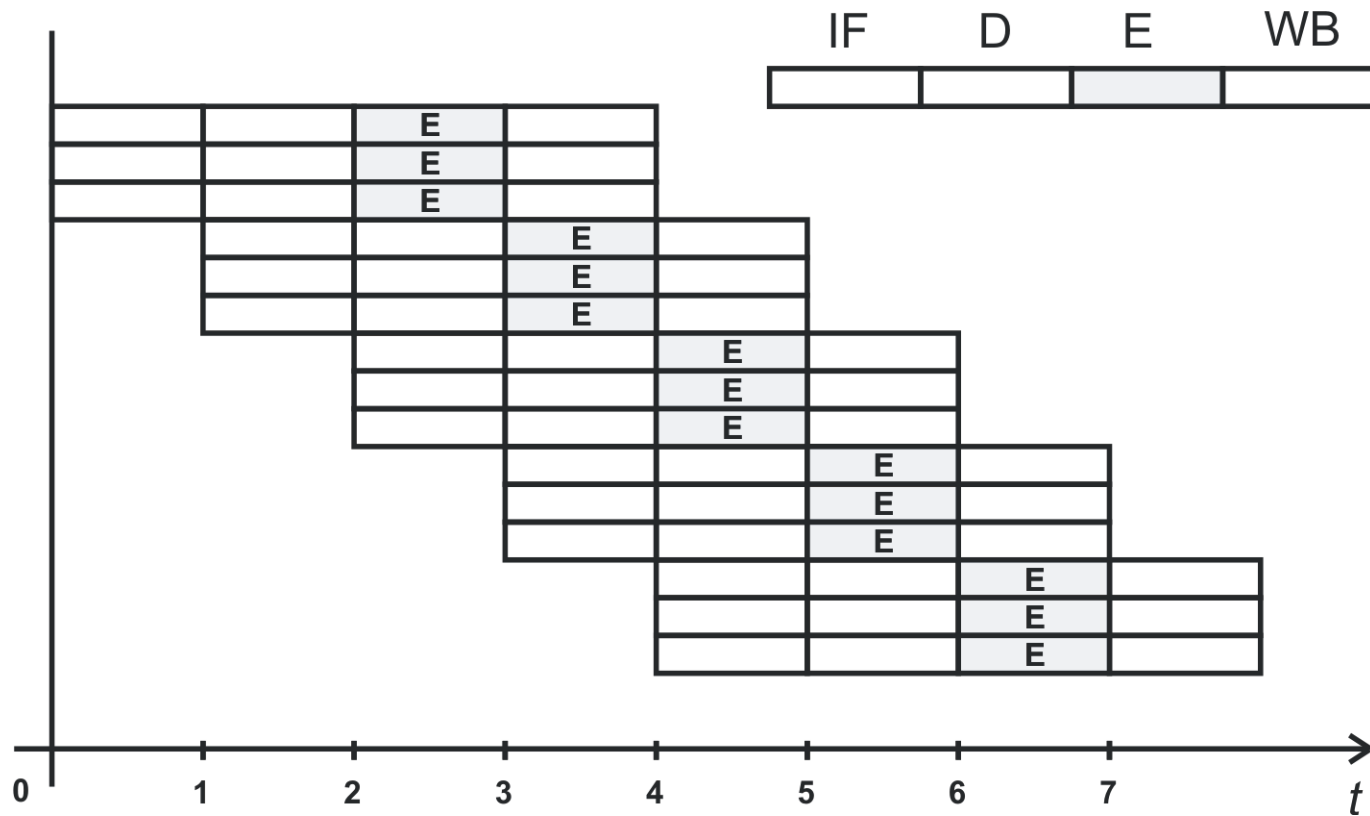
## 2. generacija RISC procesora (superskalarni RISC)



## Superskalarna izvedba RISC procesora



## Superskalarna izvedba



## **VAŽNO:**

Gradivo koje se treba samostalno obraditi (bit će uključeno u ispitno gradivo završnog ispita):

1. Aritmetičko-logička jedinica

(S. Ribarić, Naprednije arhitekture mikroprocesora, Element, Zagreb, 2002., str. 73-91)

2. Obrada iznimaka – mikroprocesor MC 68000

(S. Ribarić, Naprednije arhitekture mikroprocesora, Element, Zagreb, 2002., str. 149-160)

3. Priručna memorija

(S. Ribarić, Arhitektura računala RISC i CISC, Školska knjiga, Zagreb, 1996., str. 265-285.)