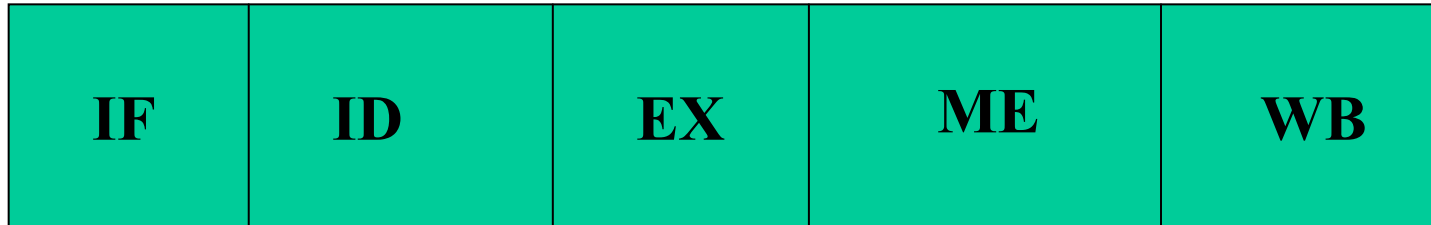


Protočna struktura procesora

- Protočna struktura
- Ganttov dijagram
- Hazardi u protočnoj strukturi
- Usporedba pogodnosti CISC i RISC za protočnu izvedbu

Protočna izvedba:



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.

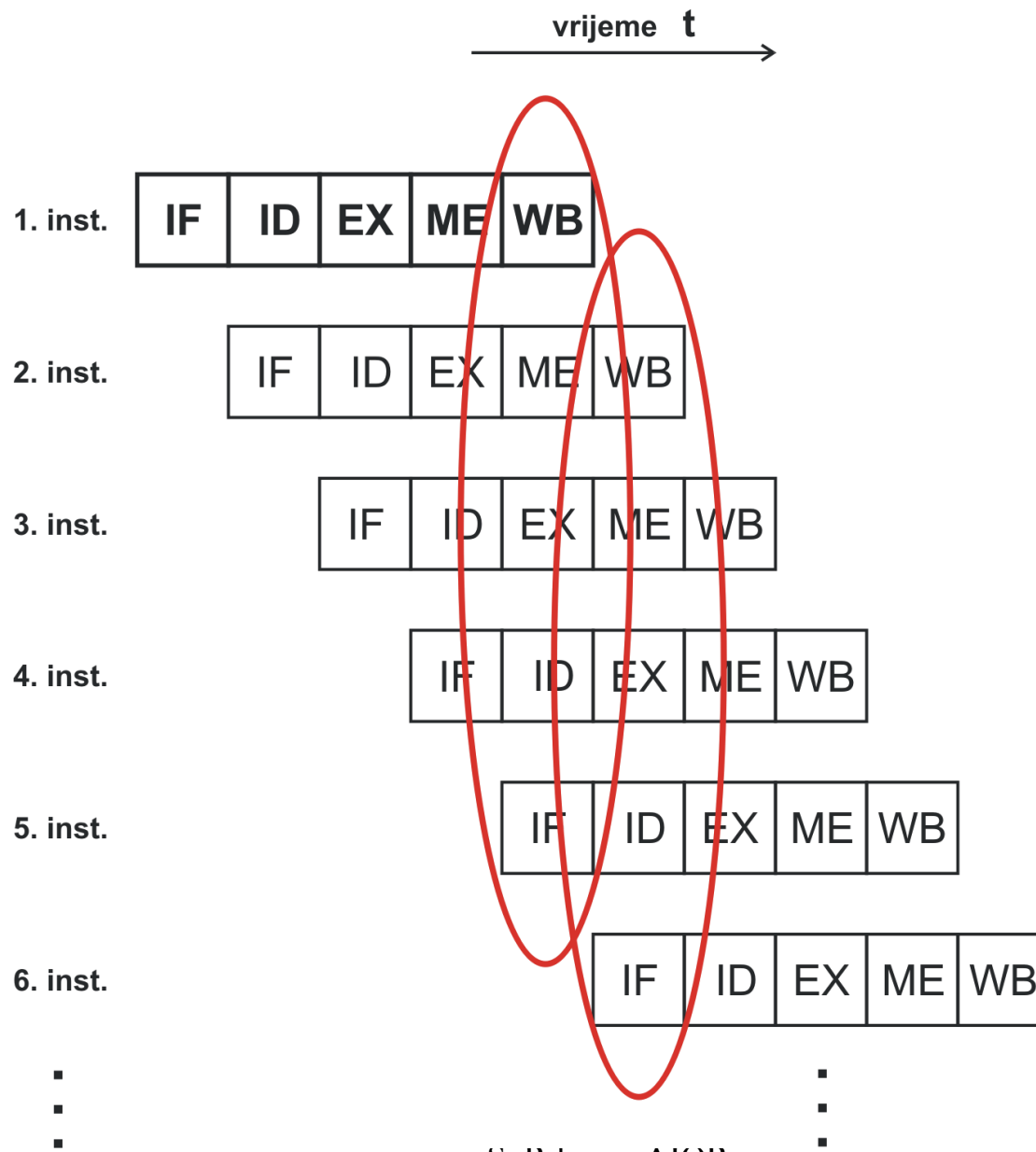
ID (Instruction decoding and operand fetching): Dekodiranje instrukcije
i dohvat operandata – **istodobno** dekodiranje operacijskog koda
i dohvat operandata iz skupa registara

istodobno \Rightarrow prije dekodiranja operacijskog koda !

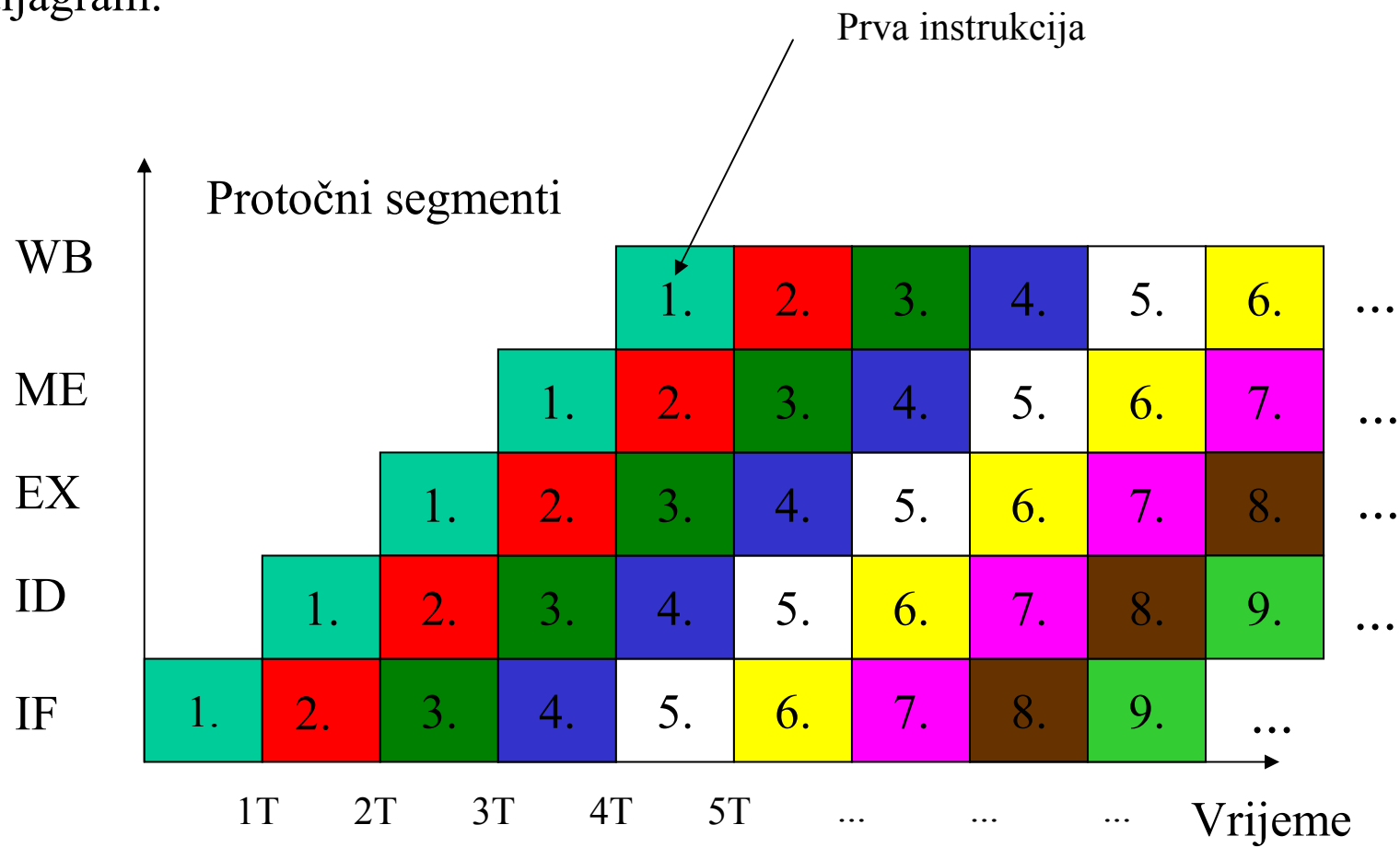
EX (Instruction execution): Izvođenje (obavljanje) instrukcije – obavlja se operacija specificirana operacijskim kodom. Za instrukcije koje naslovljavaju memoriju (*load*, *store*) u ovom se protočnom segmentu računa **efektivna adresa**

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.

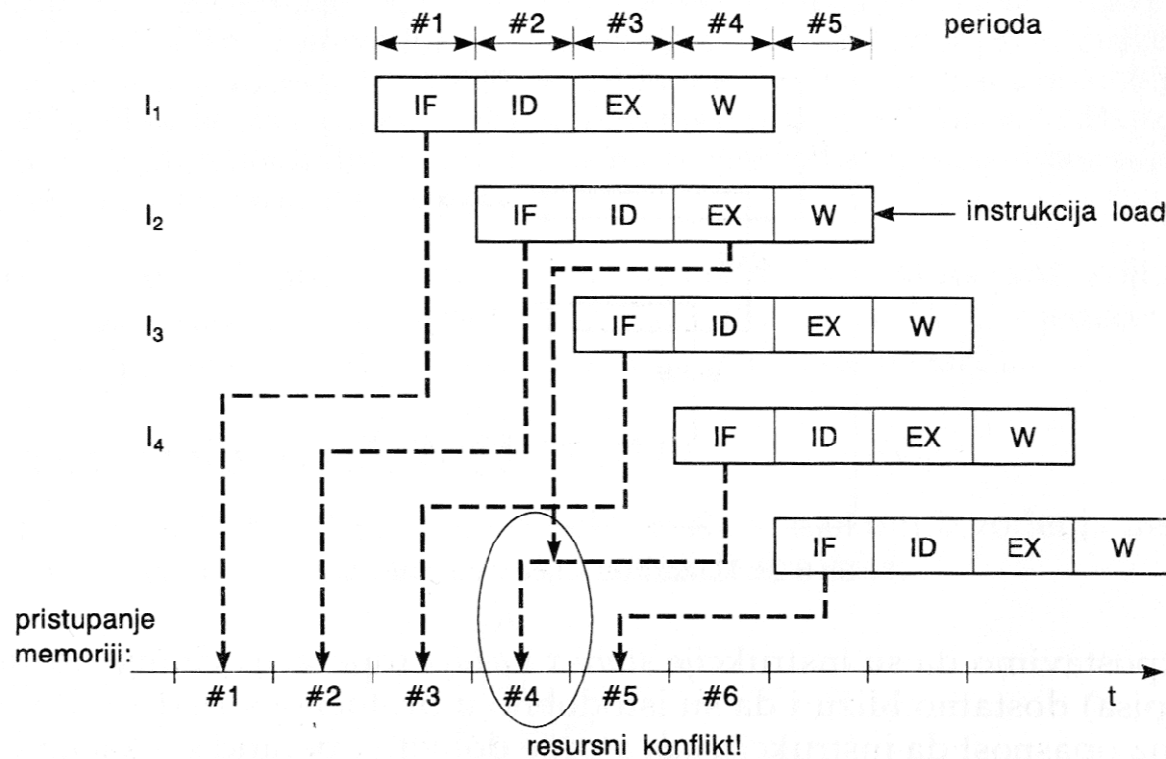
Tri razreda hazarda:

- strukturni hazard
- podatkovni hazard
- upravljački hazard

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

Strukturni hazard (resursni konflikt)

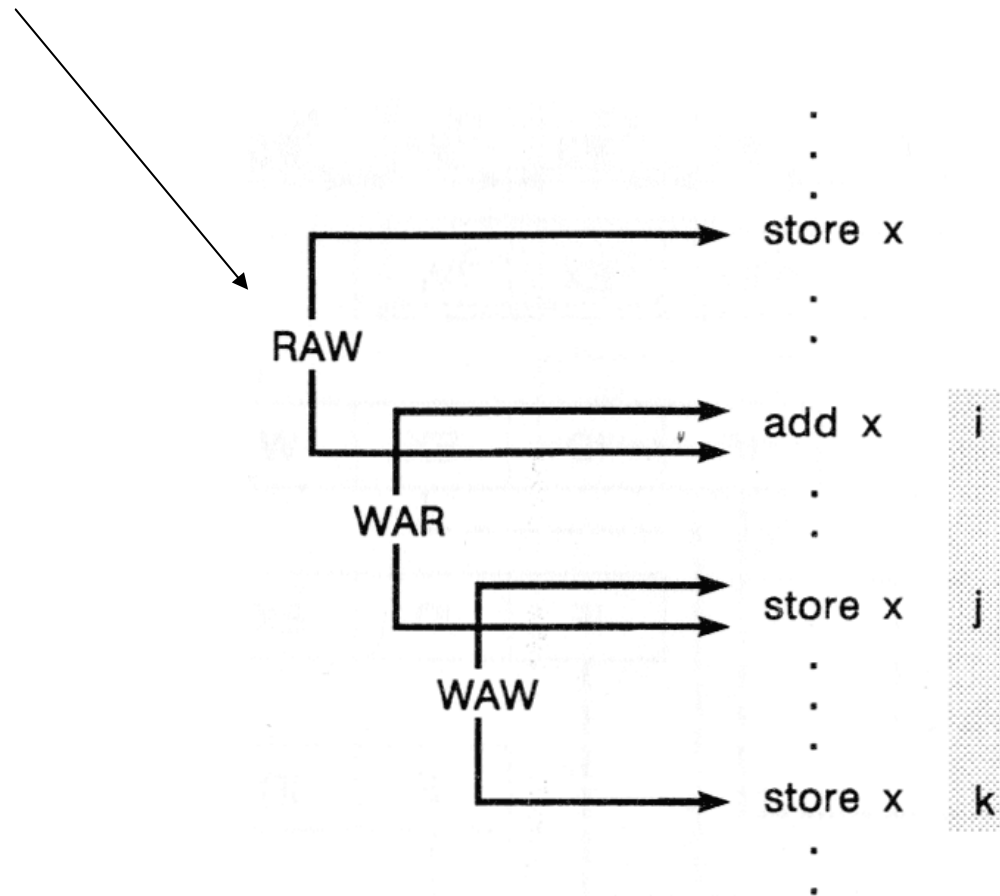
Nastupa kad se instrukcija ne može izvesti zbog sukobljavanja oko sredstava (resursa)



Podatkovni hazard

- Podatkovni hazard nastupa zbog *međuzavnosti podataka*
- Nastaje kad dvije ili više instrukcija 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/
- WAR i WAW nastaju isključivo u naprednim superskalarnim arhitekturama s dinamičkim raspoređivanjem i izvođenjem izvan redosljeda

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



ekvivalentni registarski primjer:

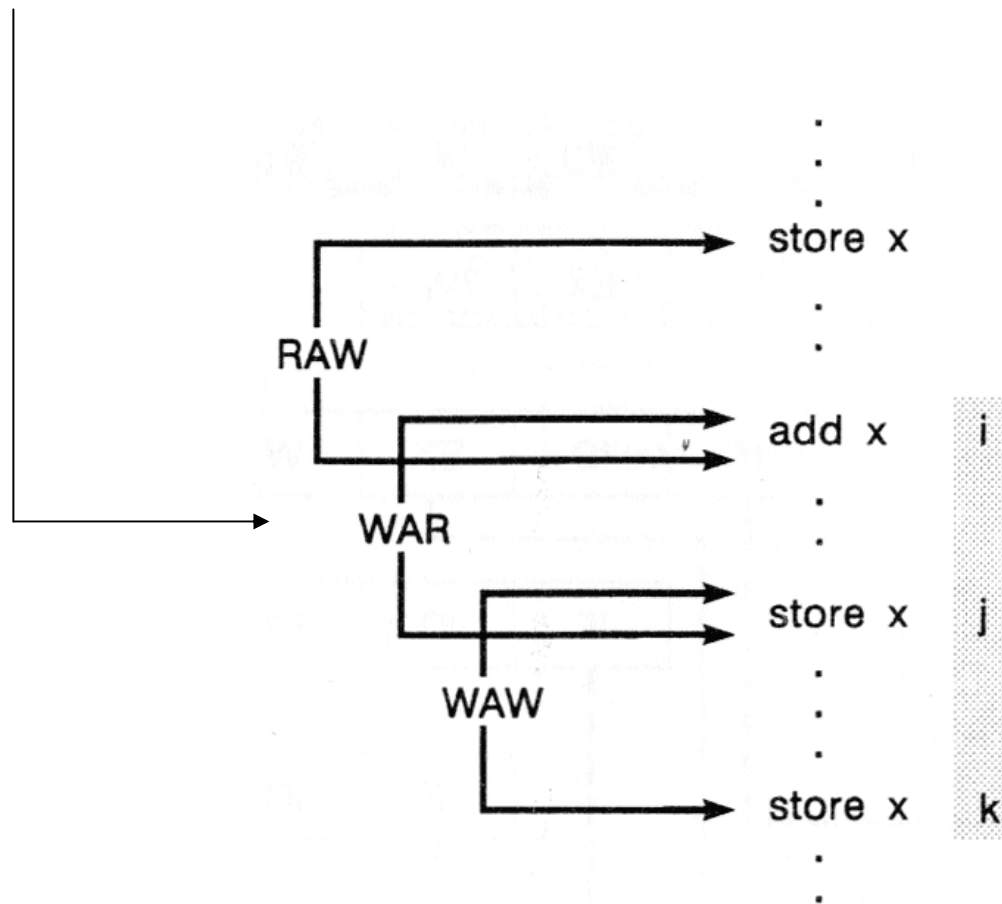
`add $r1,$r2,$r3`

`add $r4,$r1,$r5 #raw $r1`

`add $r5,$r6,$r7 #war $r5`

`lw $r5,$r0,40 #waw $r5`

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



registarski primjer:

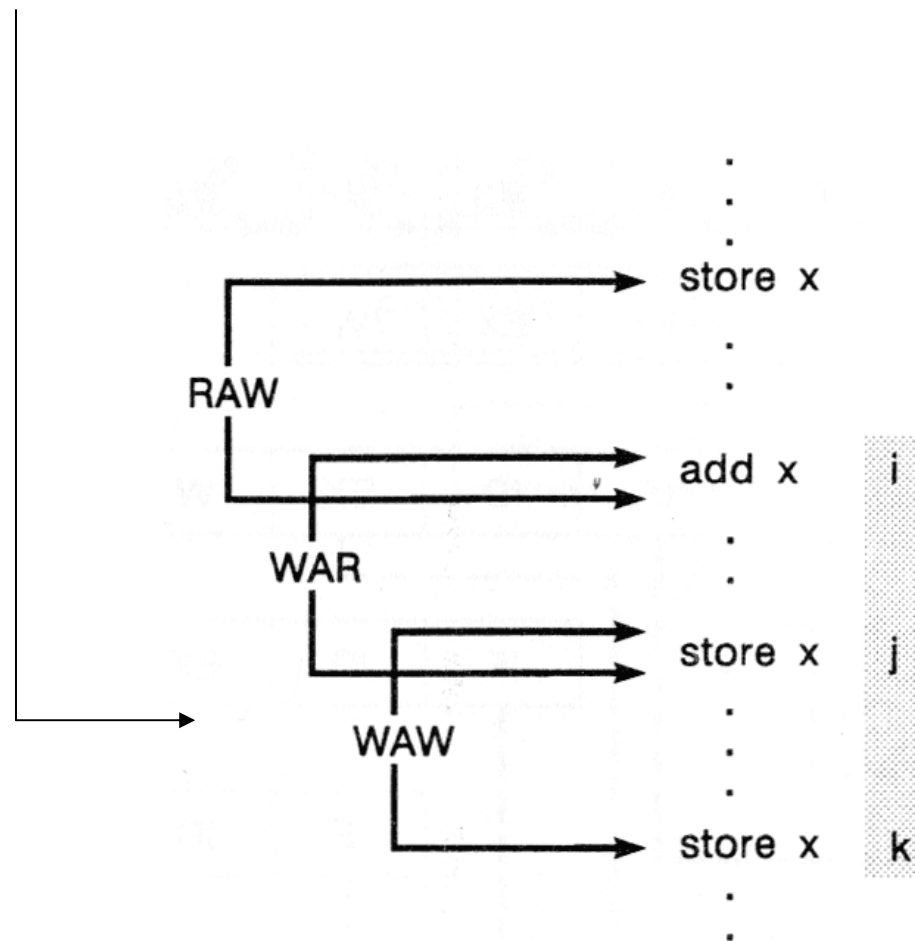
add \$r1,\$r2,\$r3

add \$r4,\$r1,\$r5 #raw \$r1

add \$r5,\$r6,\$r7 #war \$r5

lw \$r5,\$r0,40 #waw \$r5

WAW – obje instrukcije j i k žele upisati podatak na memorijskoj lokaciji x (problem nastaje ako se instrukcija j izvede **poslije** instrukcije k)



registarski primjer:

```
add $r1,$r2,$r3
```

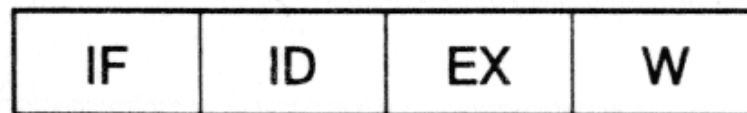
```
add $r4,$r1,$r5 #raw $r1
```

```
add $r5,$r6,$r7 #war $r5
```

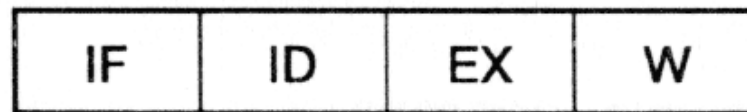
```
lw $r5,$r0,40 #waw $r5
```

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

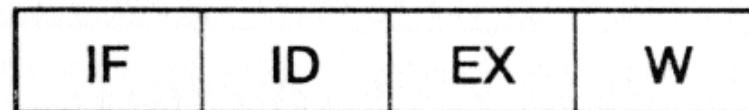
load r1, A



load r2, B



add r3, r1, r2



← ???

Saniranje hazarda RAW nakon instrukcije load:

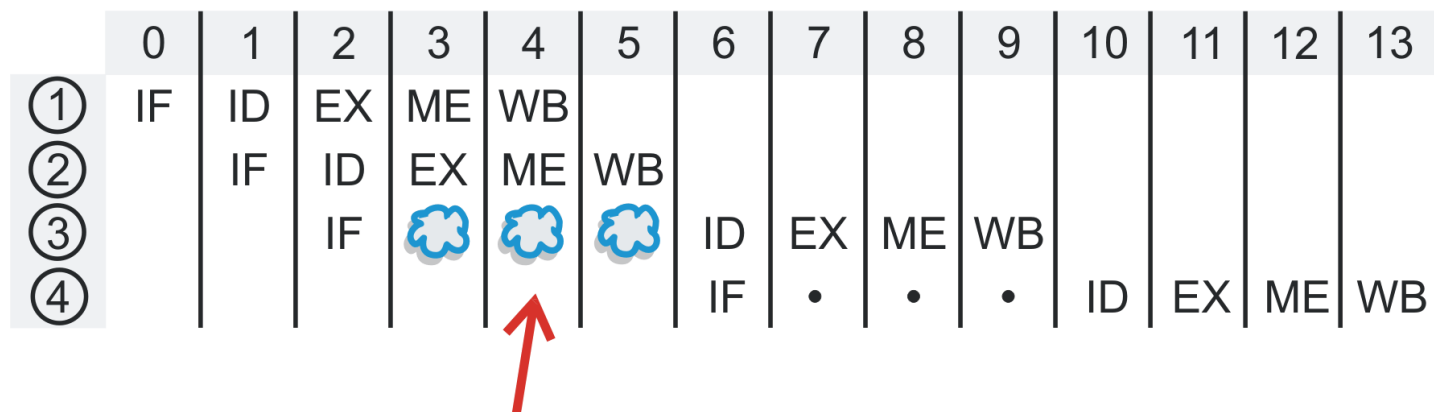
- usporavanjem protočnosti (**protočni mjehurići**, vidi dolje)
- **prosljeđivanjem** npr, $\text{MEM}[i] \rightarrow \text{ID}[i+1]$ (ipak jedan mjehurić)
- specifičnom definicijom instrukcije load (**zakašnjelo čitanje**)

loadf $fp1 = \text{mem}(r1 + r2)$

loadf $fp2 = \text{mem}(r3 + \text{disp})$

multf $fp3 = fp1, fp2$

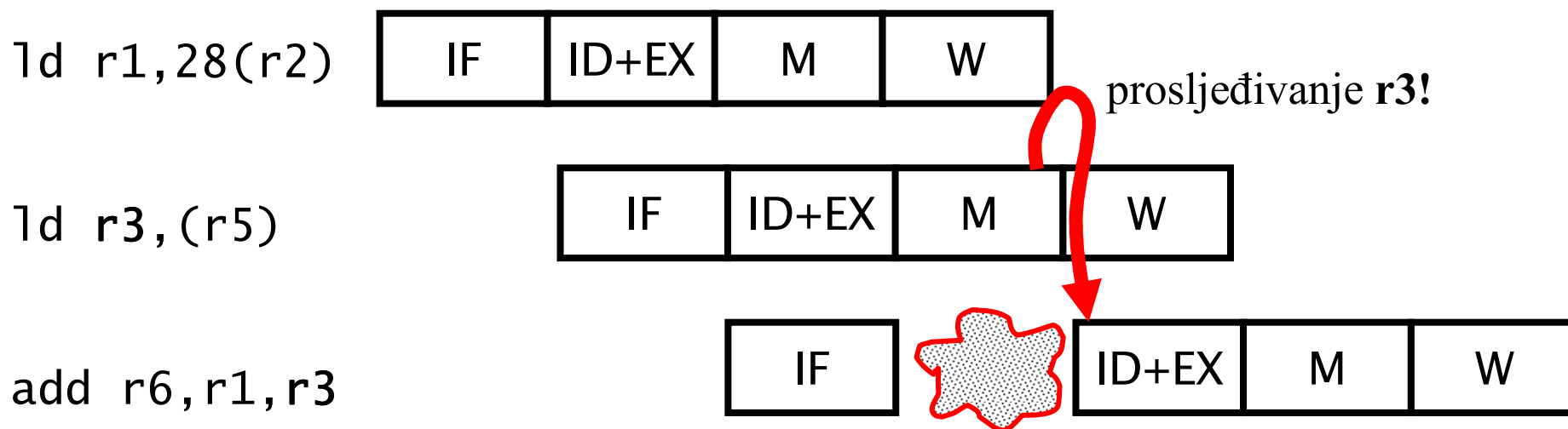
storef $\text{mem}(r1 + r2) = fp3$.



protočni mjehurići

Kod jednostavnih instrukcija, najčešće možemo proći samo s jednim mjehurićem

- prosljeđivanje: prospajanje (vodovi + MUX) rezultata prethodne operacije natrag prema ulazu procesne jedinice
- primjer za arhitekturu Berkeley RISC:

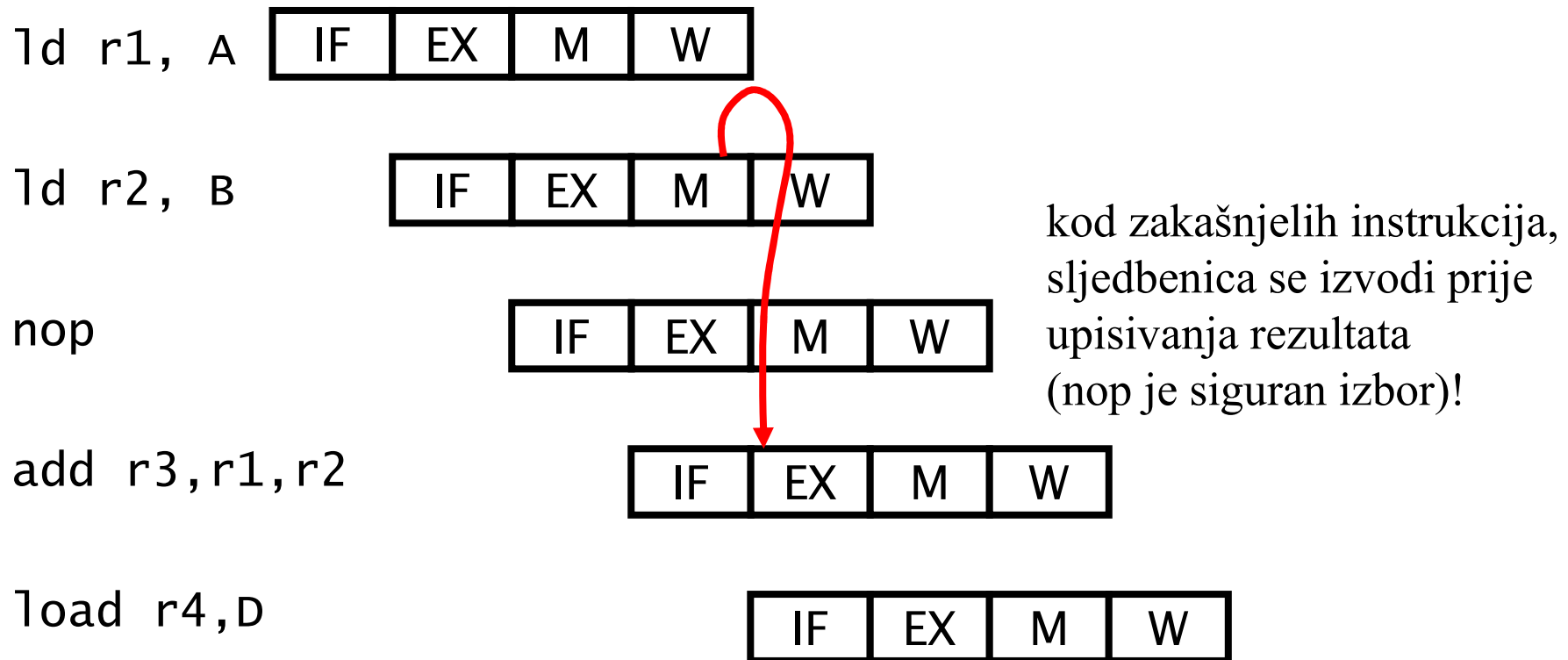


protočni mjehurić ekvivalentan
instrukciji `nop`: gubi se 1 ciklus

Ideja: potpuno otkloniti potrebu za mjehurićima zakašnjelom definicijom problematičnih instrukcija (`load`, `jump`)

Prevodioc ima priliku **iza** zakašnjelih instrukcija staviti korisnu instrukciju, koja ne ovisi o rezultatu zakašnjele instrukcije

Logički, dodana instrukcija se izvršava **istovremeno** sa zakašnjelom instrukcijom.



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

Ako prevodioc u priključak za kašnjenje ne uspije ubaciti korisnu instrukciju, ubacuje se instrukcija *nop*

Instrukcija iz priključka za kašnjenje ne vidi rezultate izvođenja prethodne instrukcije: logički, dvije instrukcije se izvršavaju **istovremeno**

Primjer (zakašnjelo čitanje):

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

load r1, A

load r2, B

add r3, r1, r2

load r4, D

← hazard RAW

load r1, A

load r2, B

load r4, D

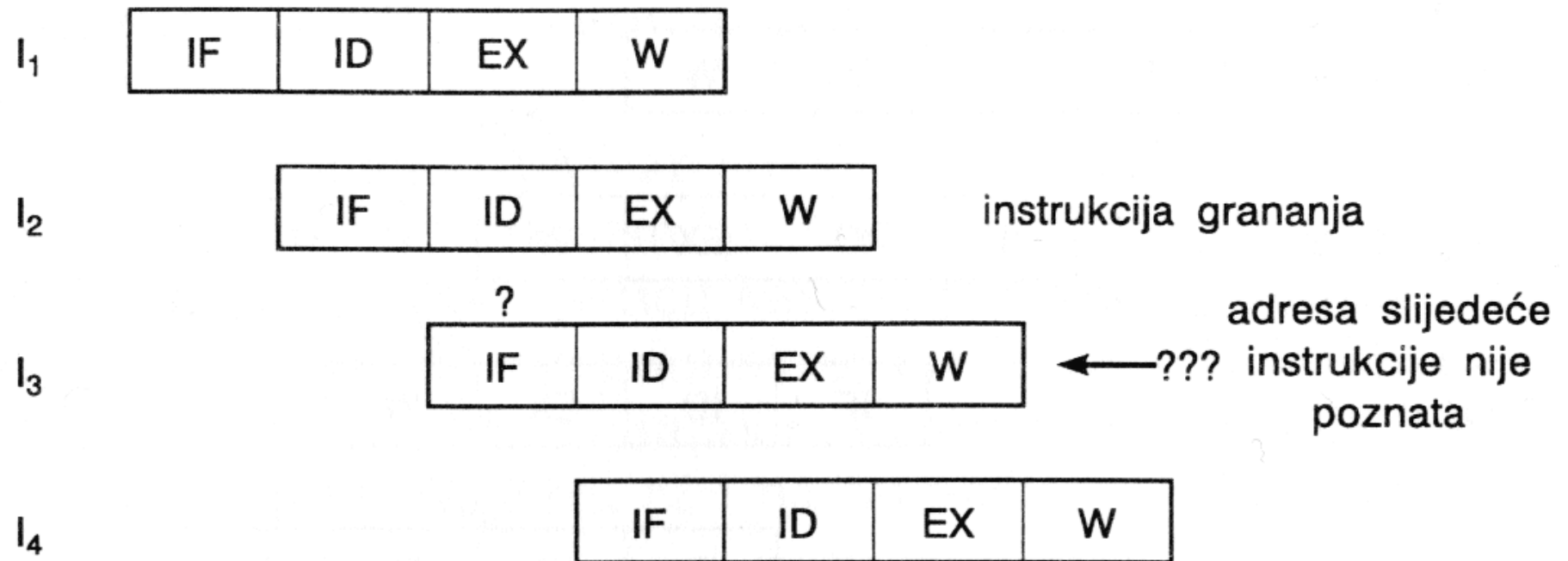
add r3, r1, r2

← priključak

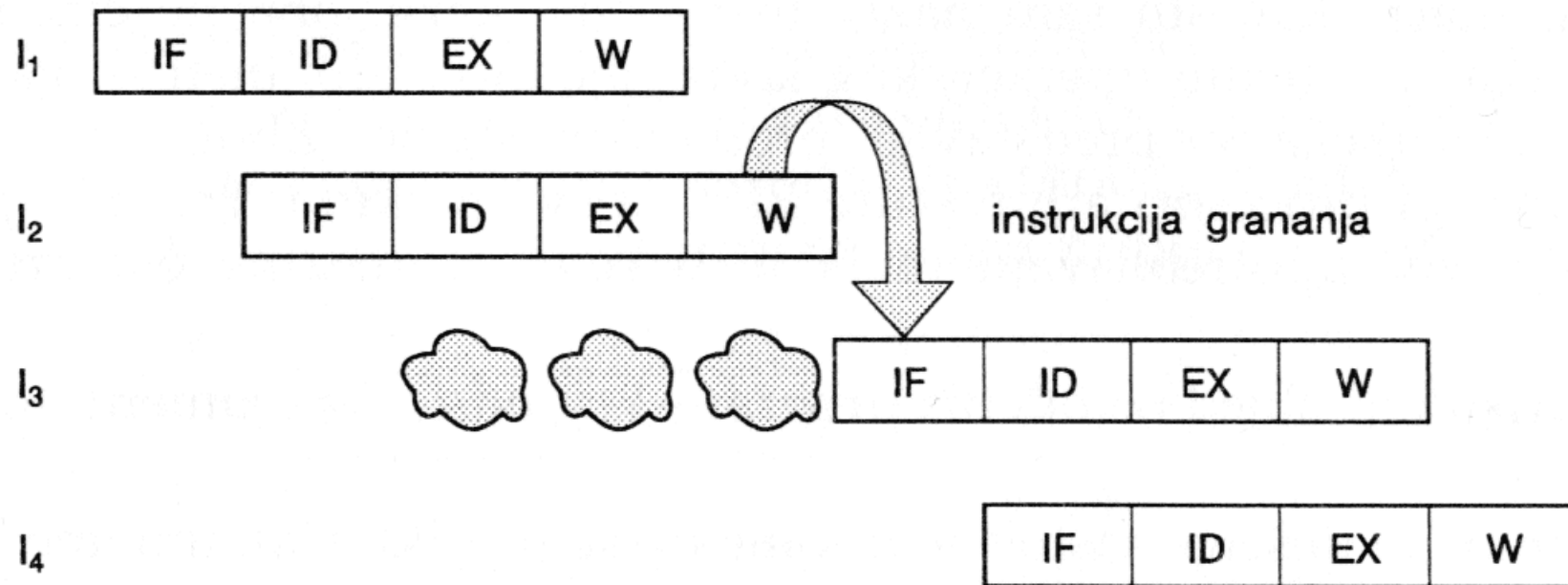
Upravljački hazard

Nastupa kad adresu sljedeće instrukcije nije moguće izračunati prije dohvata sljedeće instrukcije

Primjer:



Umetnuti tri (!!)



Nepovoljno utječe na performansu procesora!

Smanjenje kašnjenja može se postići tako da se računanje i upis ciljne adrese grananja obavi u segmentu **ID** (umjesto u W ili EX)

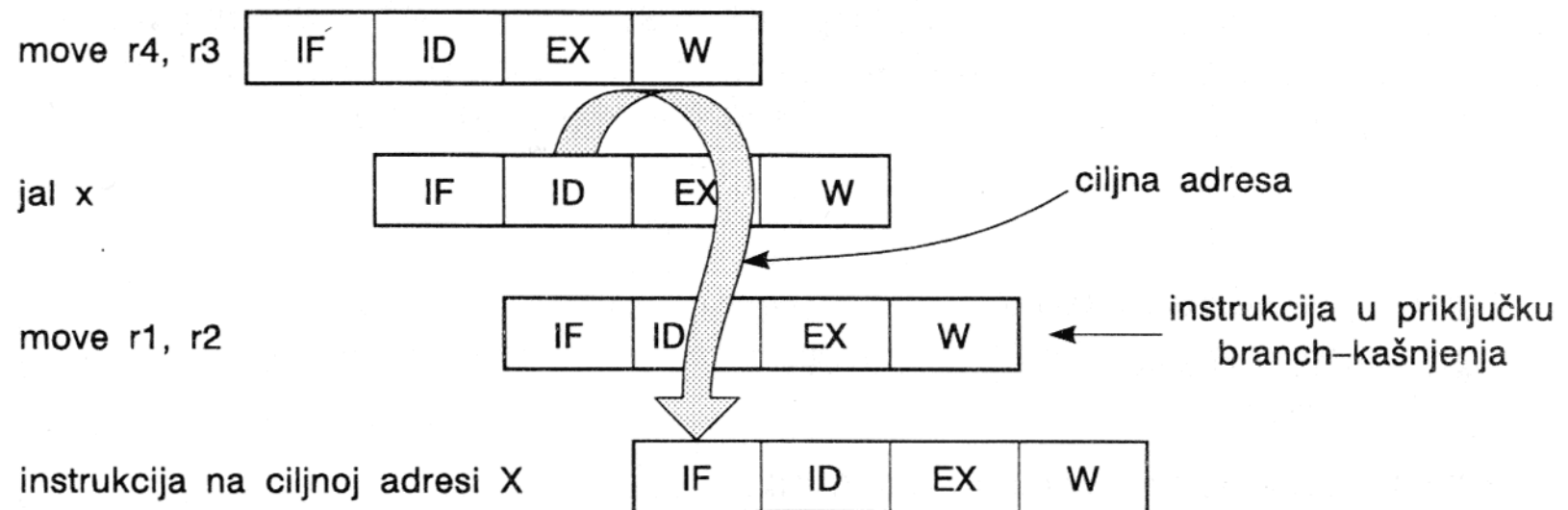
Koristi se **prosljeđivanje** $ID[i] \rightarrow IF[i+1]$: relativno odredište računa se u zasebnom zbrajalu, u okviru sklopa za upravljanje grananjem (glavno zbrajalo u to vrijeme računa rezultat prethodne operacije!)

Spekulativna operacija zbrajanja započinje dok instrukcija još nije dekodirana!

Rezultat dekodiranja utječe na izlazni multiplekser koji konačno odabire između $PC+4$ i $PC+4+i \cdot mm$

Kašnjenje samo s jednim mjehurićem!

Mjehurići se mogu potpuno zaobići **zakašnjelim grananjem**



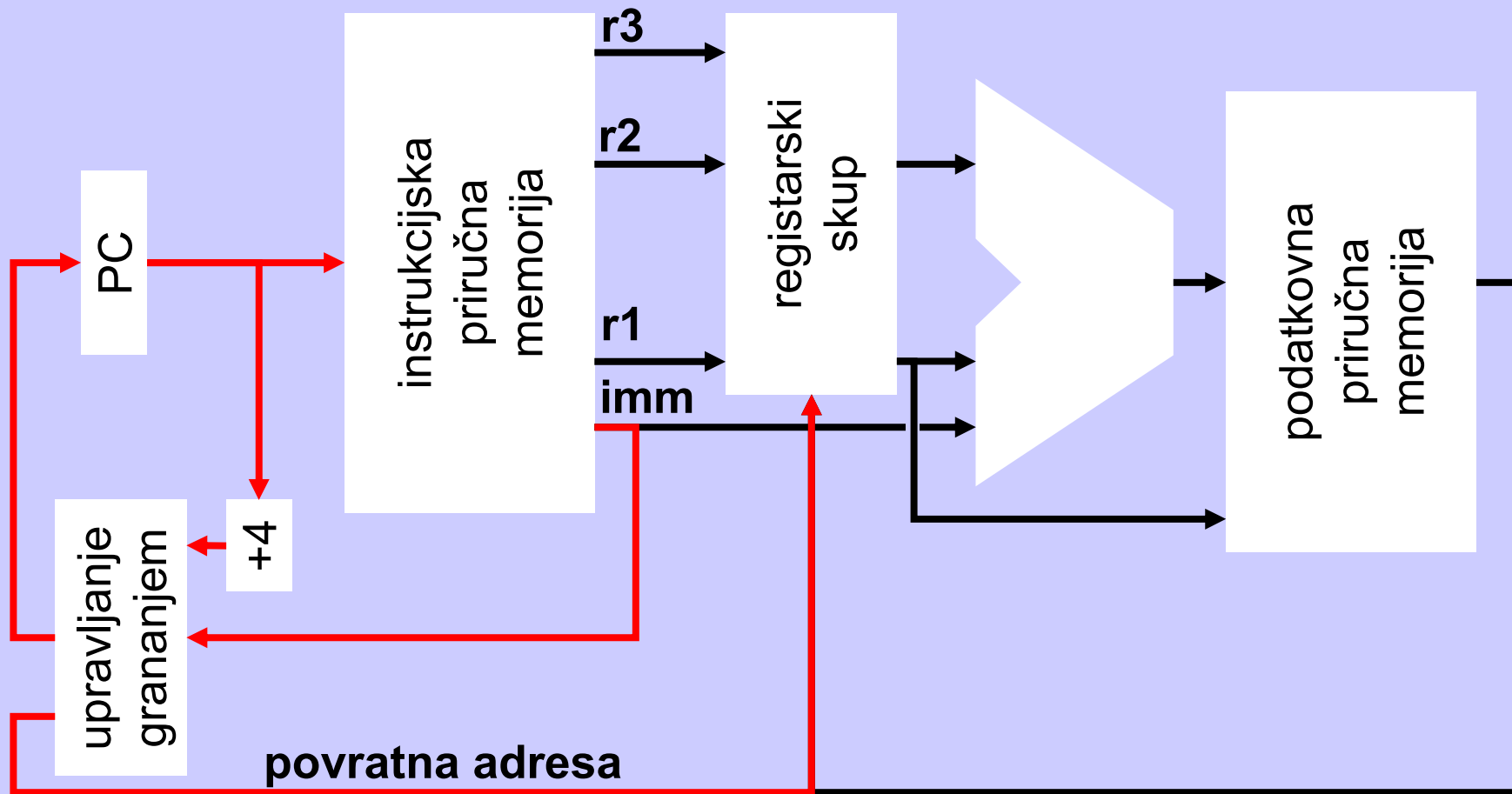
Primjer za računalo MIPS: instrukcija grananja

```
jal $10000 # $pc<-$10000; $r31<-$pc+8
```

- IF: pribaviti instrukciju, PC+=4 (uvijek isto)
- ID: dekodirati, proslijediti \$10000
- EX: ništa
- MEM: ništa
- WB: upisati povratnu adresu u \$r31

Pažnja: koristi se **zakašnjelo** grananje, instrukcija neposredno nakon instrukcije grananja se također izvodi!

jal \$10000 # \$pc<-\$10000; \$r31<-\$pc+8



Primjer (zakašnjelo grananje):

“Običan” prevodioc generirao bi standardni (neispravan) kod:

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

RISC prevodioc bi generirao:

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

Optimizirajući RISC prevodioc bi umjesto (neoptimizirane) izvedbe:

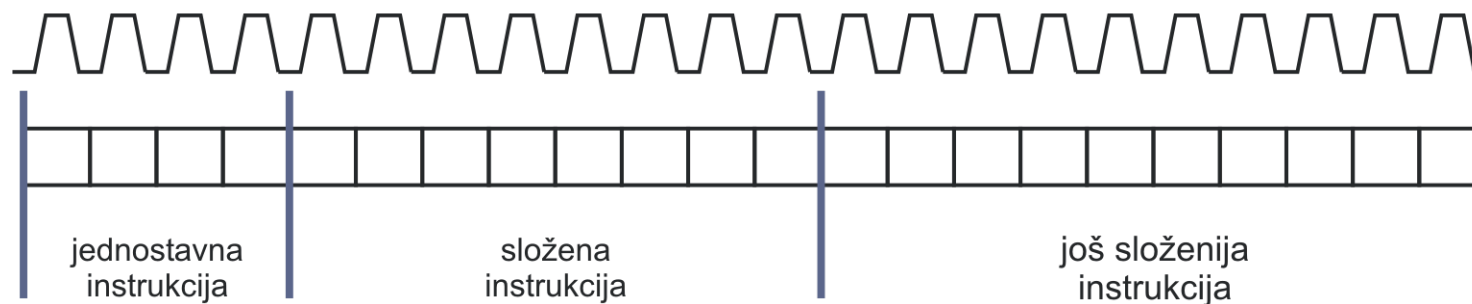
```
move r4, r3  
move r1, r2  
jal x           ; bezuvjetno grananje  
nop  
C: add r5, r5, 1
```

... proizveo konačnu varijantu:

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


Usporedba pogodnosti CISC i RISC za protočnu izvedbu

Tipičan slijed instrukcija za CISC procesor:



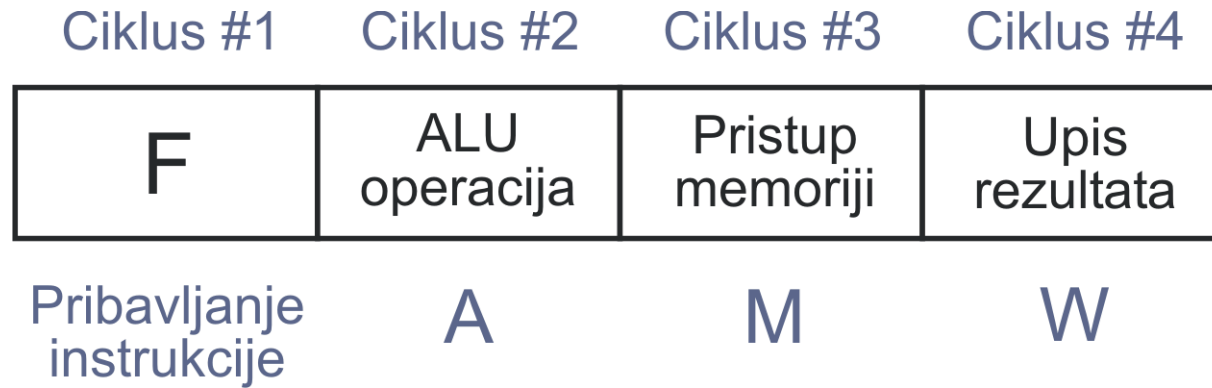
Svaka instrukcija koristi onoliko perioda signala takta koliko joj je potrebno

Težnja u arhitekturi RISC → jedan period po instrukciji

Tehnike koje dopuštaju ostvarivanje težnje:

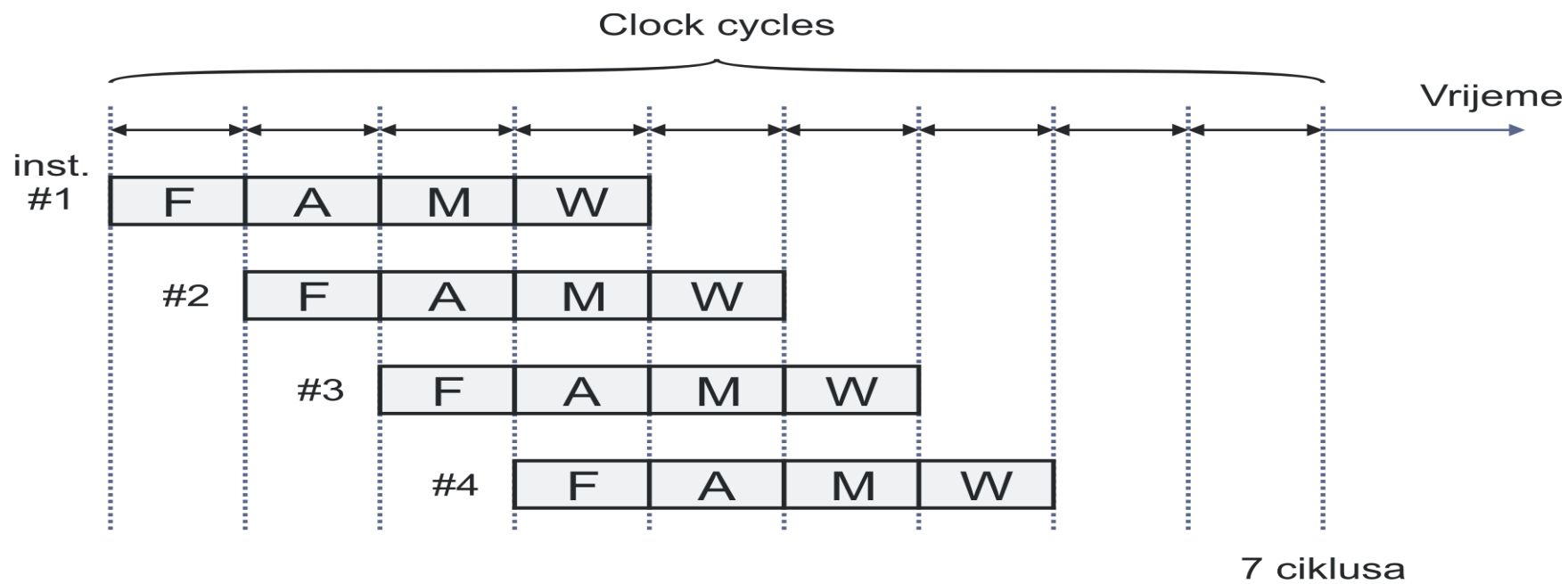
- jednostavan i pravilan instrukcijski skup
- plitka, dobro popunjena protočnost
- arhitektura load/store
- “zakašnjele” load instrukcije
- “zakašnjele” branch instrukcije

Protočni segmenti:

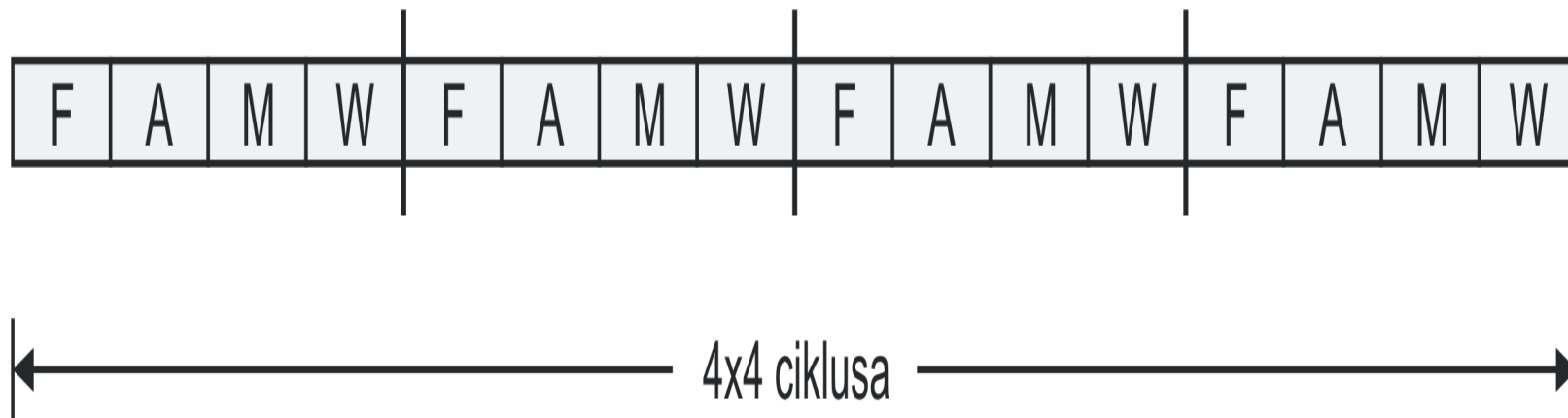


ili





Izvedba u neprotočnoj strukturi:

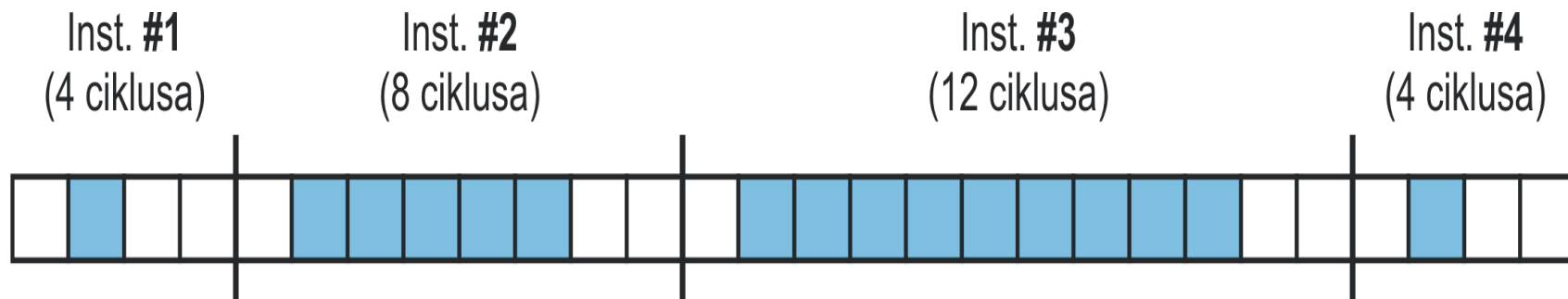


Vrijeme izvođenja u **protočnoj** strukturi: 7 perioda signala vremenskog vođenja

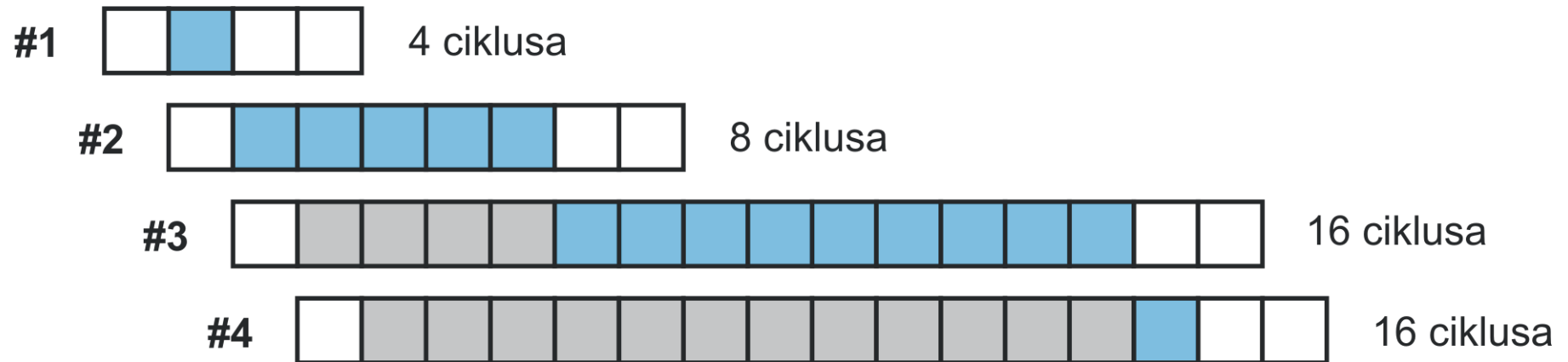
Vrijeme izvođenja u **neprotočnoj** strukturi: 16 perioda



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

Primjer protočnog izvođenja za CISC instrukcije:



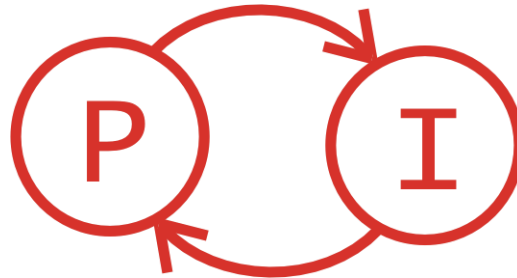
- ekskluzivni zahtjevi za resursom
(ALU, registri, sklop za posmak,...)



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

Negativan utjecaj promjenjive duljine trajanja instrukcija u CISC arhitekturi!

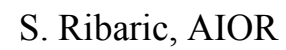
Dijagram stanja procesora na najgrubljoj razini apstrakcije
(Von Neumannovo računalo):



Detaljniji dijagram stanja za tipičan procesor arhitekture RISC:

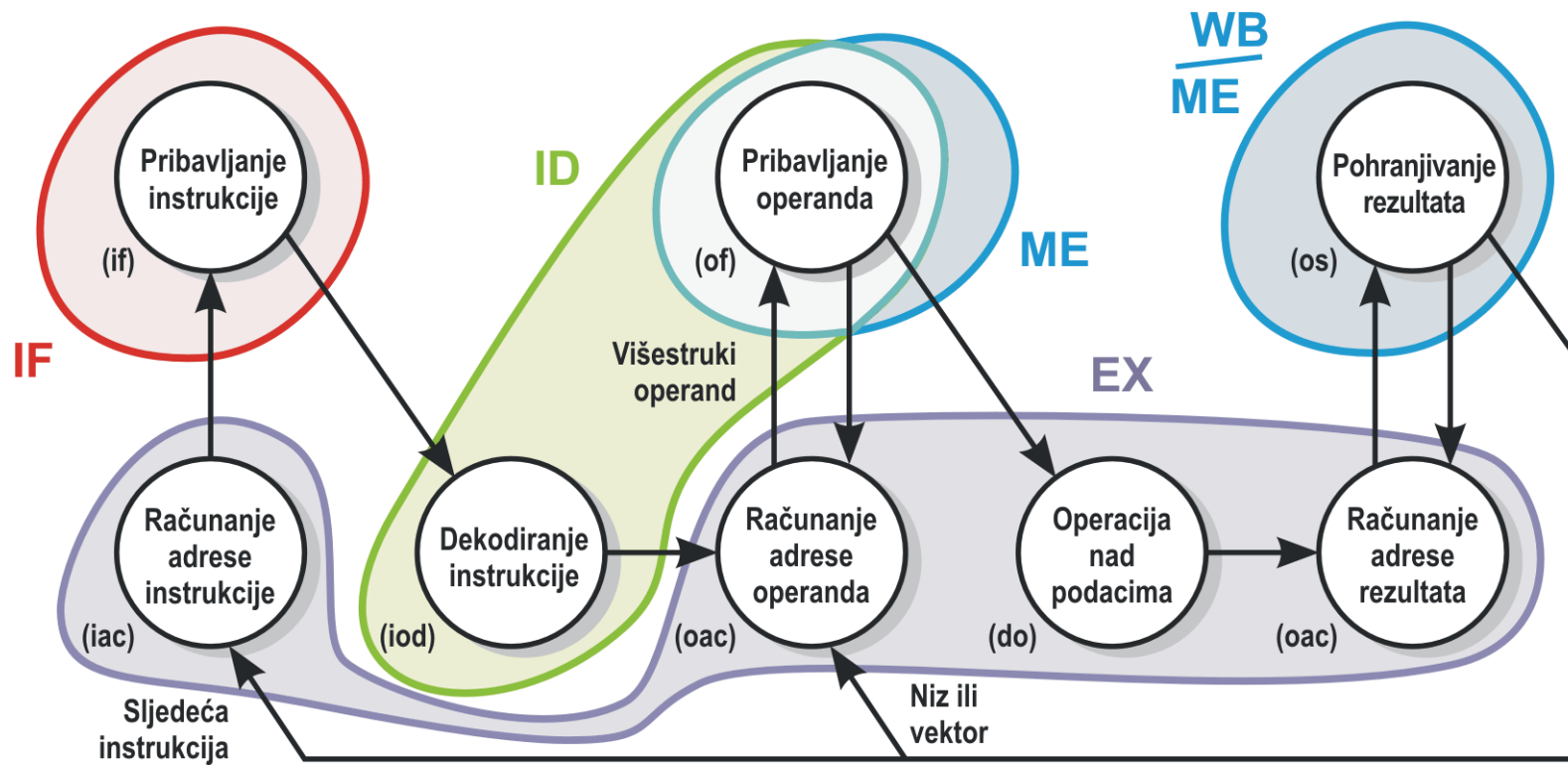
IF → ID → EX → MEM → WB

Kakav bi bio detaljniji dijagram stanja za procesor instrukcijske
arhitekture CISC?

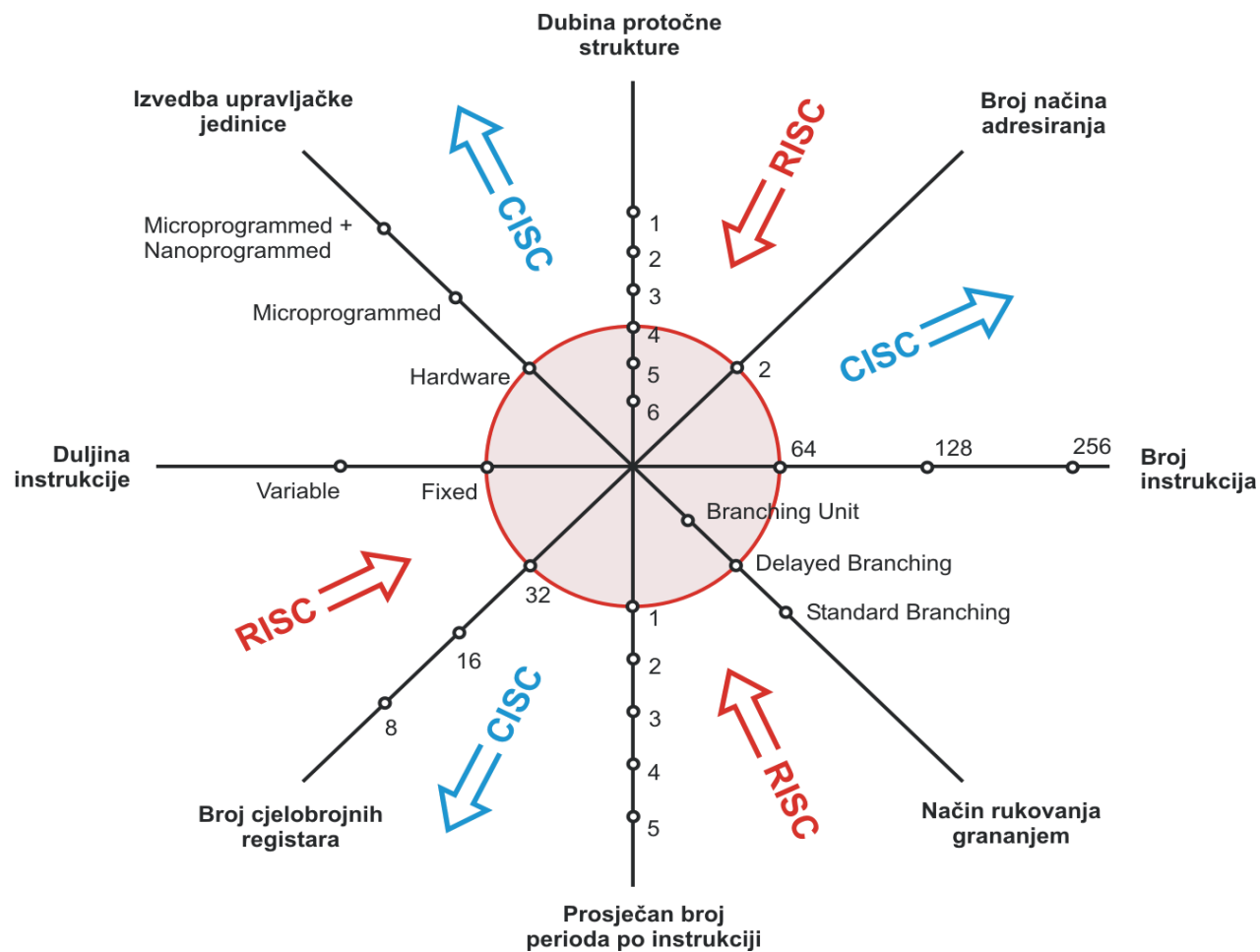


- Pribavljanje instrukcije (if – instruction fetch)
 - čitanje instrukcije iz memorije i njeno premještanje u CPU
- Dekodiranje instrukcije (iod – instruction operation decoding)
 - analiza i dekodiranje instrukcije u cilju određivanja tipa operacije koja će se izvesti te određivanja tipa operanada
- 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

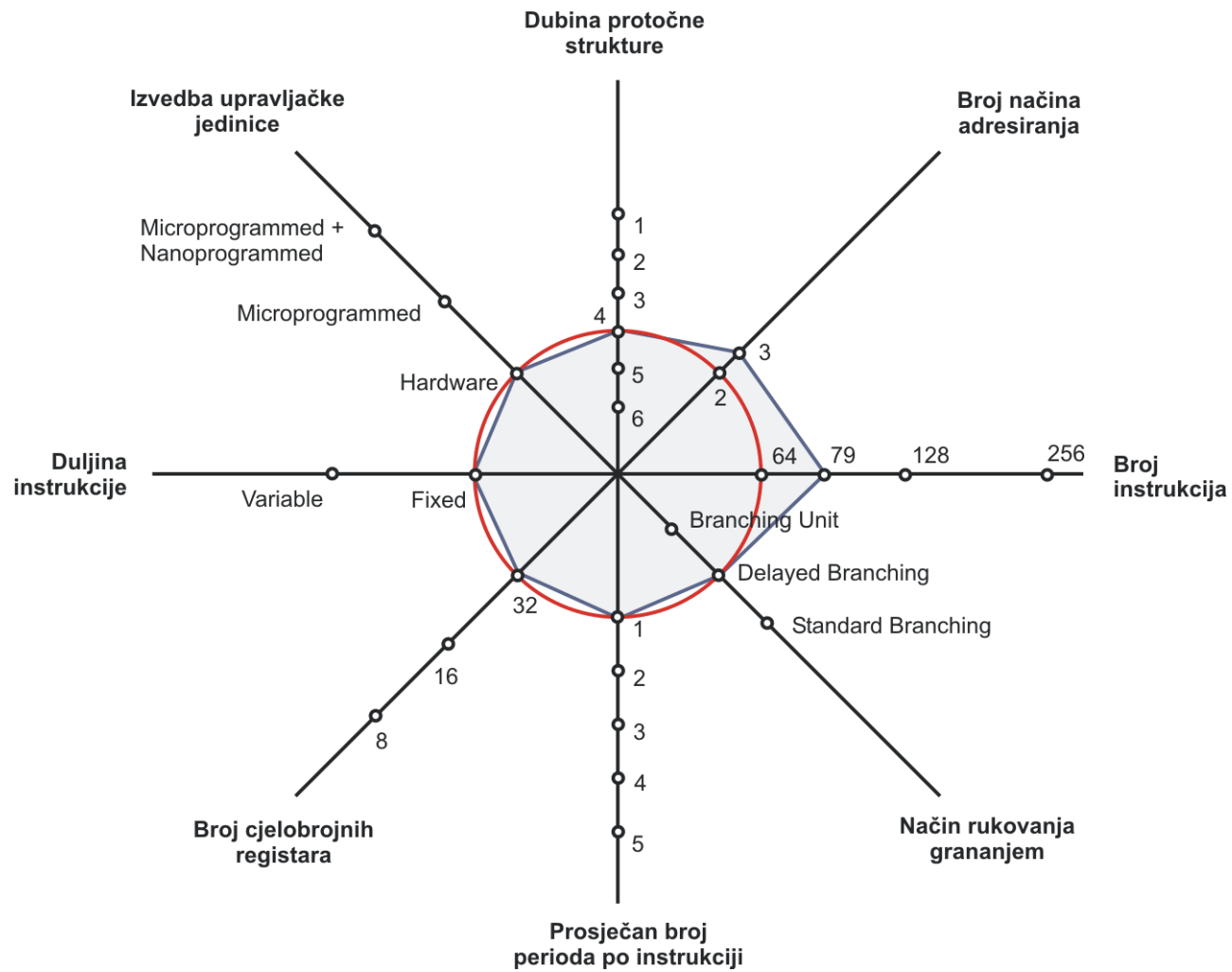
- Operacija na podacima/operandima (do – data operation)
 - izvodi se operacija (specificirana operacijskim kodom instrukcije)
- Pohranjivanje rezultata (os –operand store)
 - upisuje se rezultat u memoriju ili u I/U podsustav



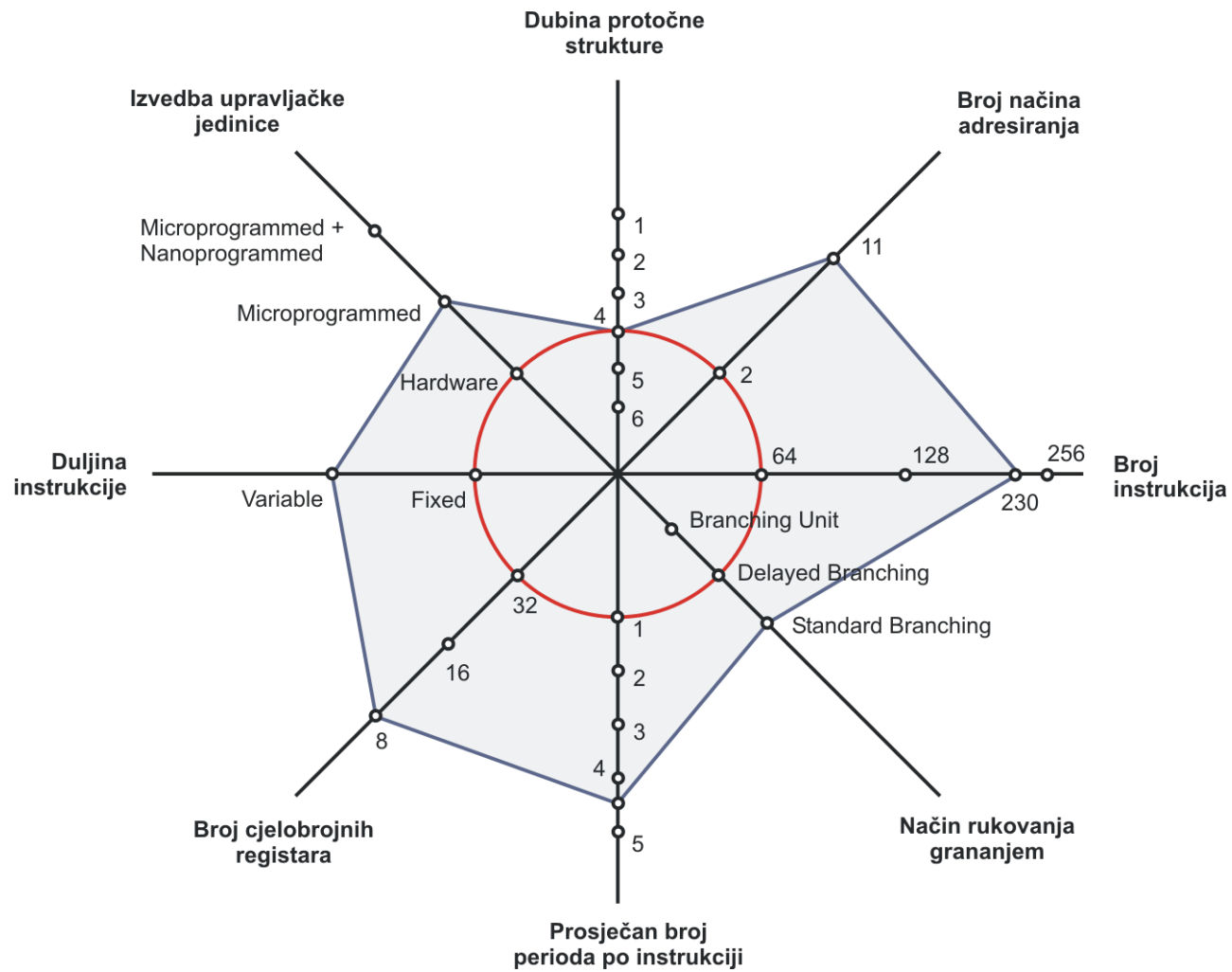
Zvezdasti (kiviat) grafovi:



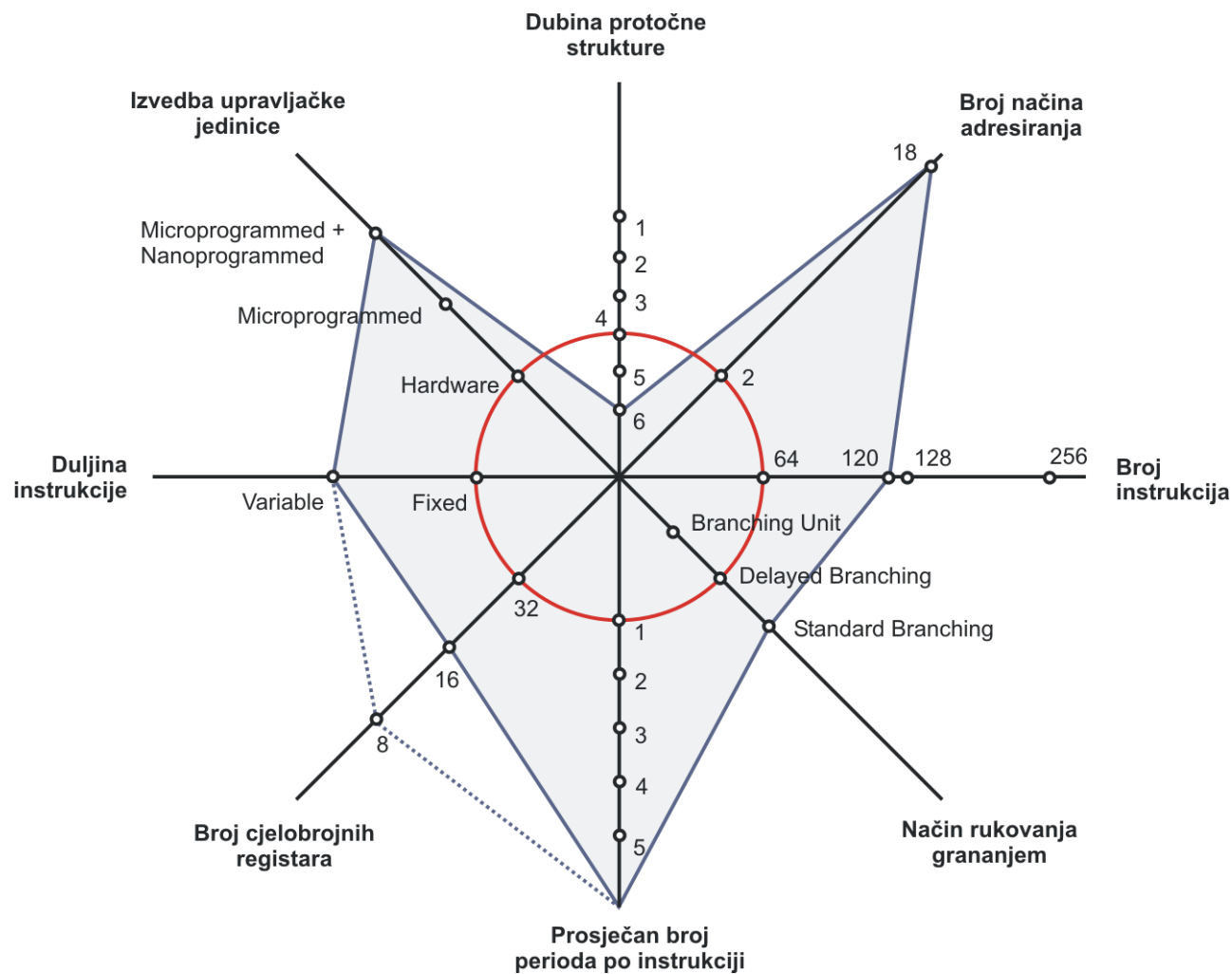
i 860



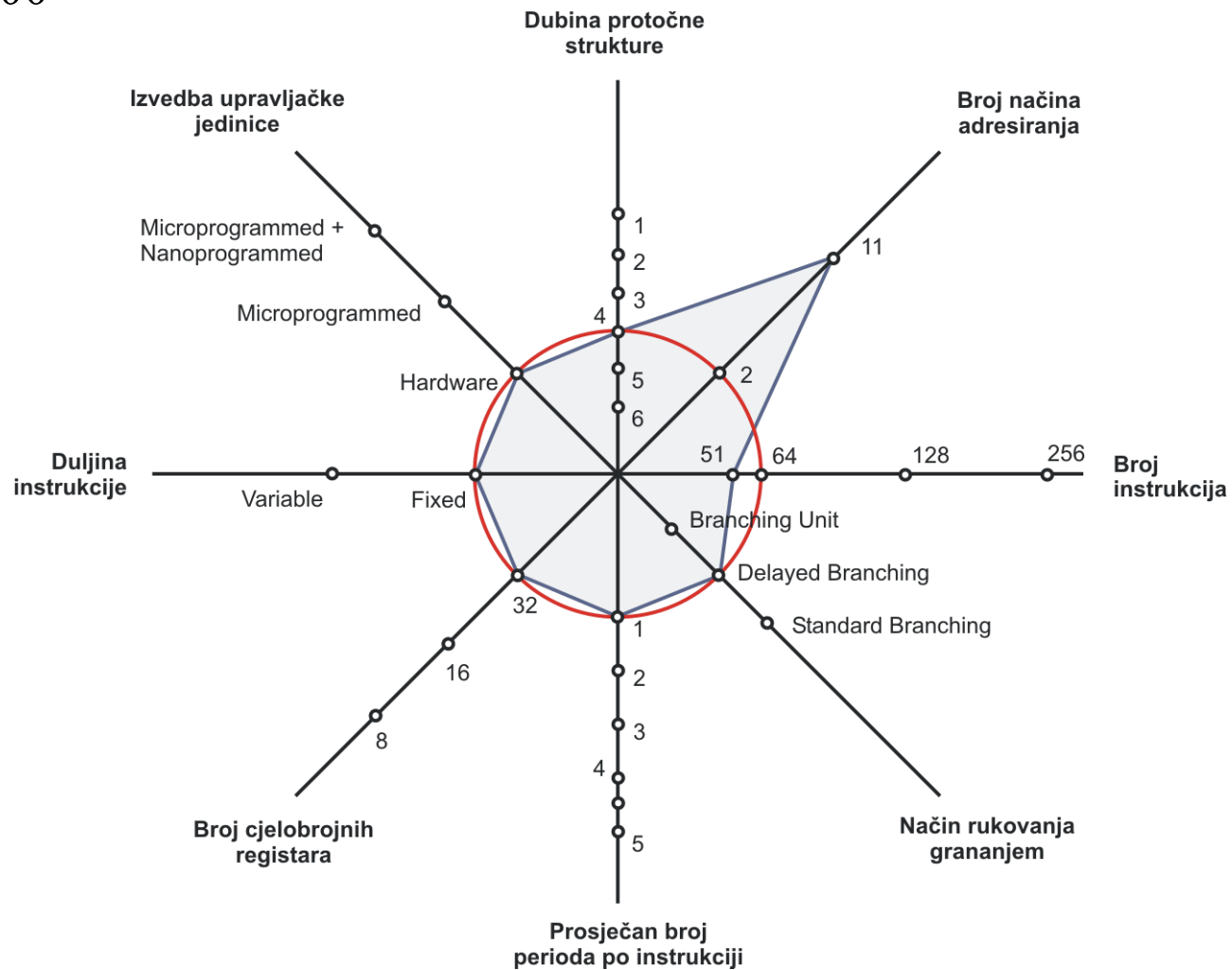
i486



MC 68040



MC 88100



Protočnost, sažetak

- moćan koncept koji omogućava višestruko povećanje performanse
- optimalna protočna struktura:
 - svaki segment odgovoran za približno jednako zahtjevan dio puta podataka
 - po jedna instrukcija izlazi iz cjevovoda u svakom periodu
 - prosječna performansa znatno bolja od neprotočne izvedbe
- nužne pretpostavke:
 - pažljivo odabran (ortogonalan) instrukcijski skup
 - μ operacije po segmentima jednako traju

Protočnost, sažetak (2):

- protočnost omogućava efikasno iskorištavanje instrukcijskog paralelizma
- faktor ubrzanja $\times 4$ i više!
- glavni izazov su hazardi:
 - tehnika prosljeđivanja obično pomaže (ali ne uvijek dovoljno)
 - zakašnjelo grananje i zakašnjelo učitavanje ponekad pomaže (prevodioc ne uspijeva uvijek naći zamjenu!)
 - hazardi će smetati još i više kod **superskalarnih** izvedbi