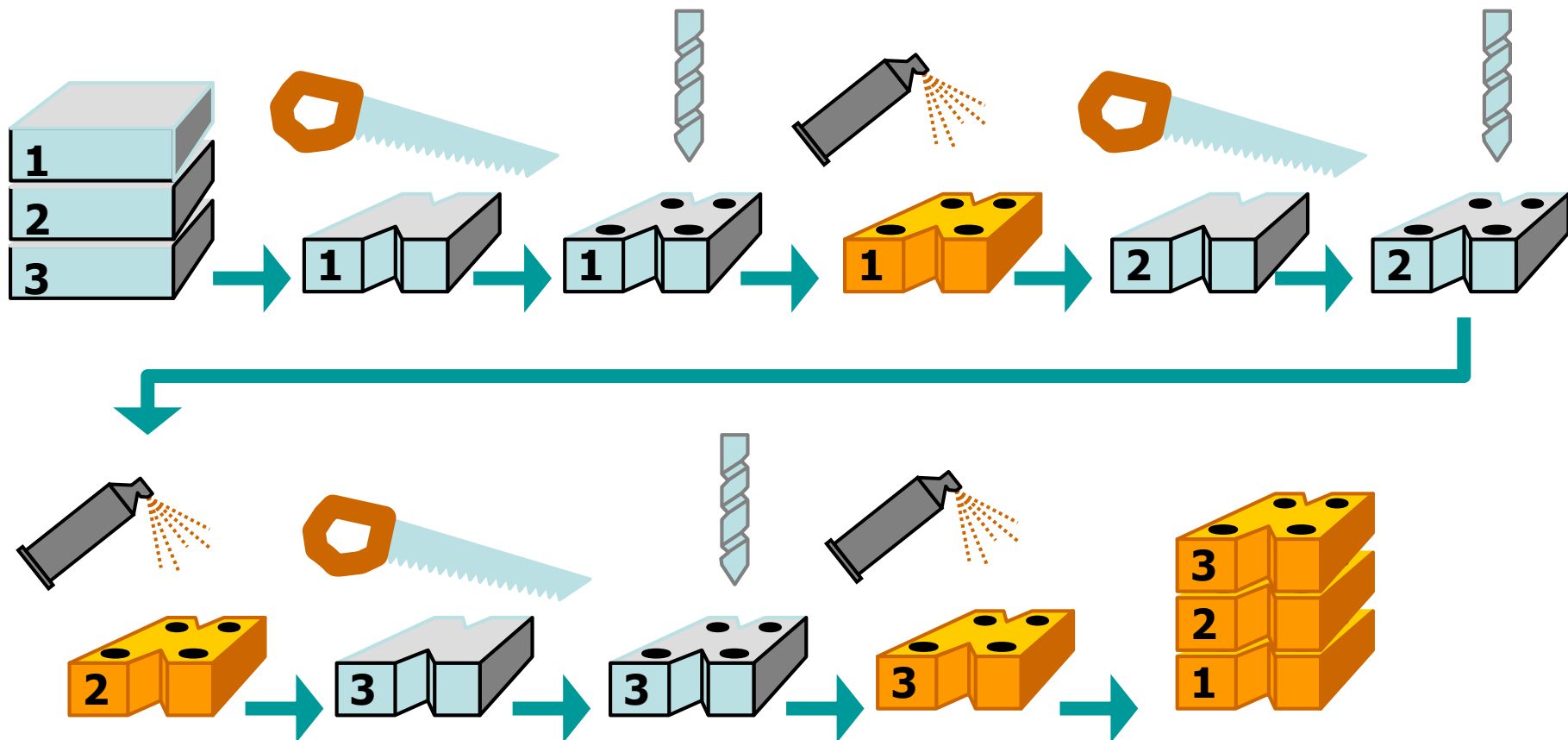


# ***Izvođenje naredaba i protočna struktura***

# Efikasno izvođenje poslova

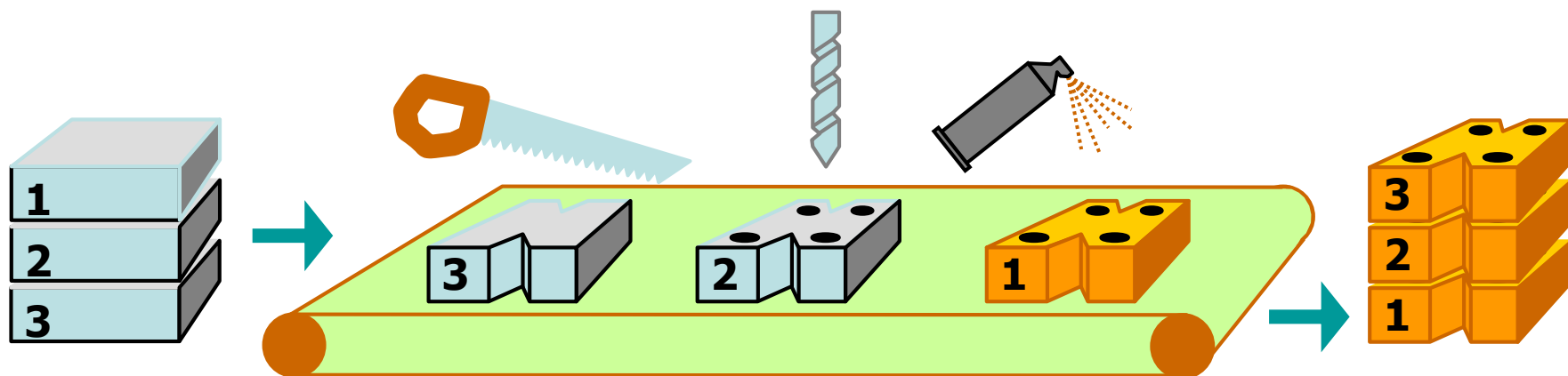
- Prije diskusije o izvođenju naredaba na procesoru pogledajmo jedan neračunarski primjer: izrada proizvoda koje treba izrezati, izbušiti i lakirati.



**Za izradu 3 proizvoda treba ukupno 9 koraka.**

# Efikasno izvođenje poslova

- Puno brže i efikasnije je upotrijebiti pokretnu traku i obavljati sva tri koraka istodobno (bitno je da su koraci međusobno NEOVISNI i da jedan drugom ne "smetaju"):



**Za izradu 3 proizvoda ukupno treba 5 koraka.**

**Nakon što prvi proizvod bude napravljen u 3 koraka, svi sljedeći proizvodi izlaze u jednom koraku !!**

**Uvjet koji ne smijemo zaboraviti: za ovakvu proizvodnju treba nam više radne snage: 3 radnika !!! -> Cijena rada sustava je veća.**

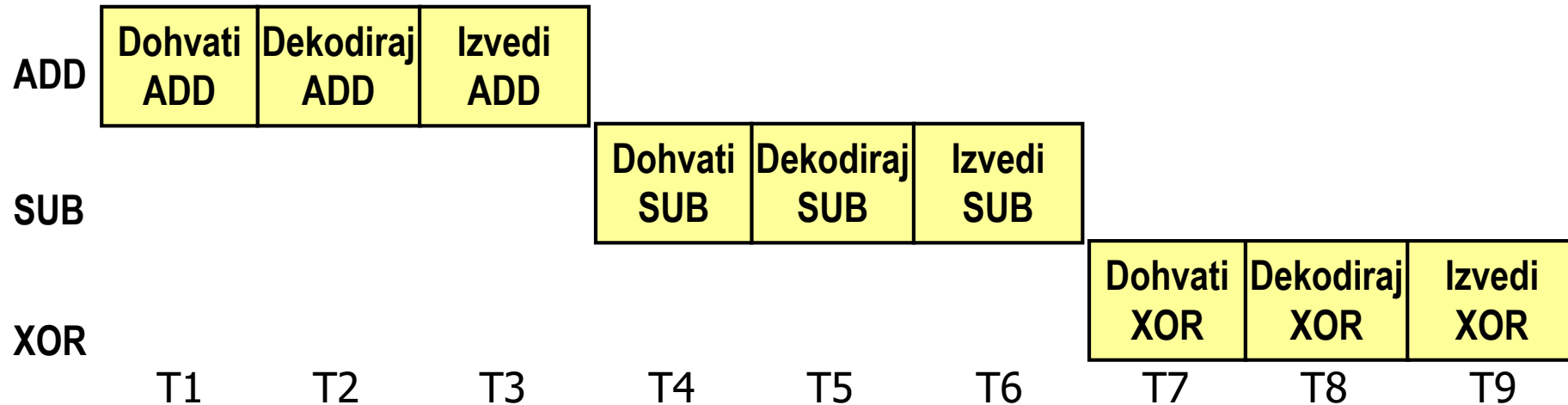
# ***Izvođenje naredaba u procesoru***

---

- **Podsjetnik** - tipične faze u izvođenju naredbe su:
  - dohvat naredbe iz memorije (fetch)
  - dekodiranje naredbe (decode)
  - izvođenje naredbe (execute)
- Kao što ćemo vidjeti kasnije, kompleksniji procesori imaju puno više koraka...

# Izvođenje naredaba u procesoru

- Npr. neka imamo redom naredbe ADD, SUB i XOR koje se trebaju izvesti:



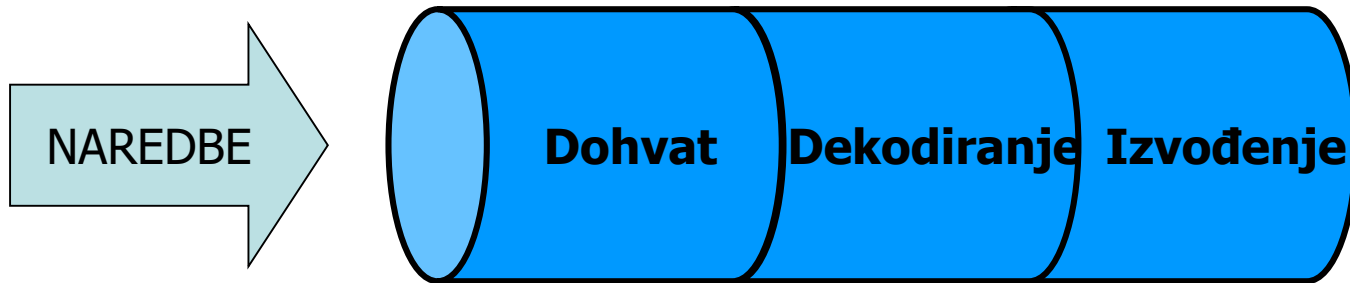
- Ako bi se naredbe izvodile slijedno po fazama, tada bi za svaku naredbu bila potrebna 3 vremenska perioda (uz pretpostavku da svaka faza traje jedan period).
  - Ovo predstavlja STARI način izvođenja ---- NEEFIKASNO
  - Još uvijek se koristi kod nekih jednostavnih procesora.



# ***Izvođenje naredaba i protočna struktura***

---

- Podijelimo li faze tako da su međusobno neovisne, možemo primijeniti načela pokretne trake, pa se sve tri faze mogu izvoditi istodobno:

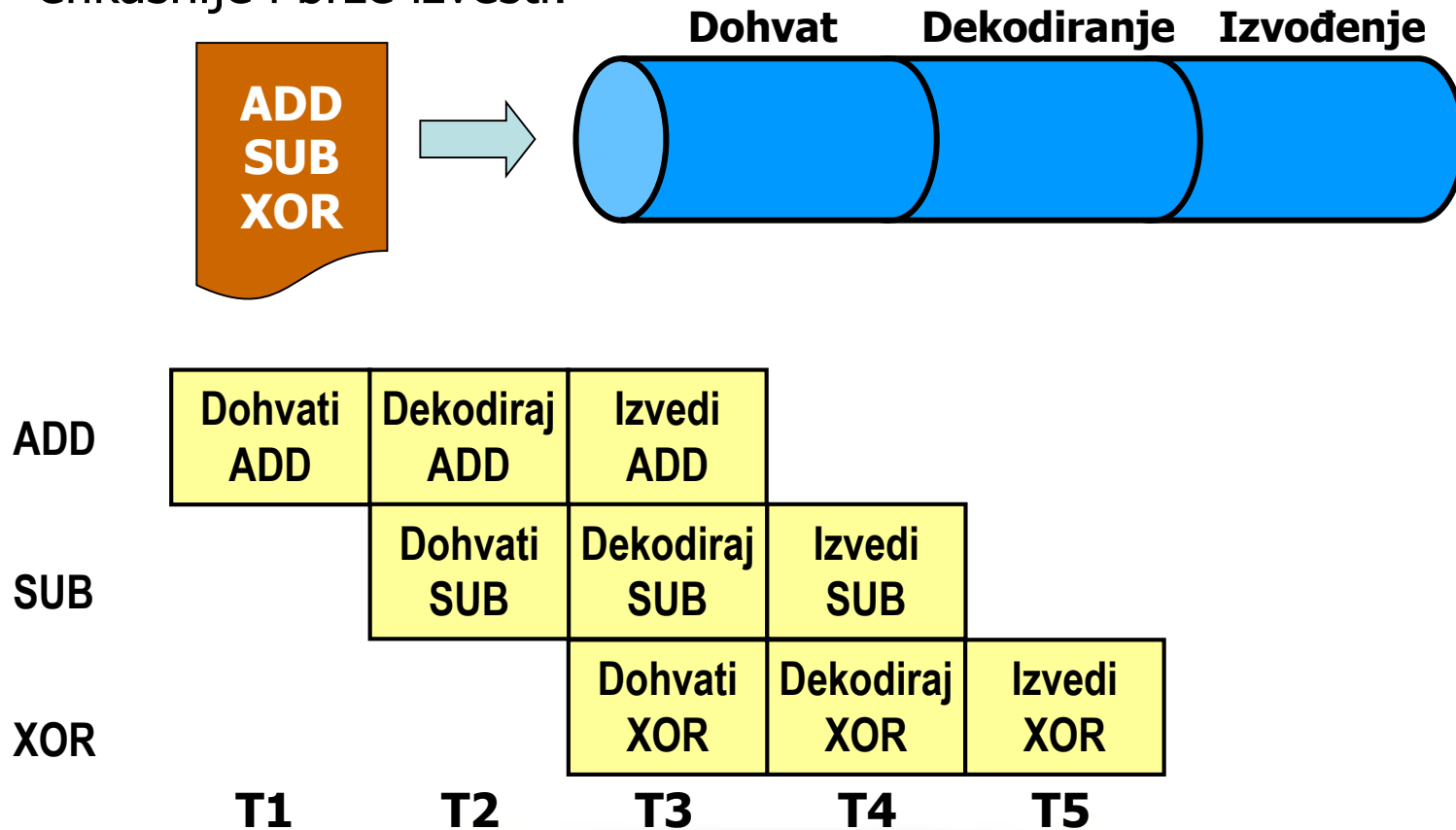


- Ovakva organizacija izvođenja se u literaturi naziva cjevovod (engl. pipeline), a mi ga nazivamo **protočna struktura**.
- Svaka **razina** protočne strukture (engl. pipeline stage) izvodi jednu fazu.
- Glavna namjena je **ubrzanje izvođenja**



# ***Izvođenje naredaba i protočna struktura***

- Sada se prije spomenute tri naredbe ADD, SUB i XOR mogu puno efikasnije i brže izvesti:



# ***Izvođenje naredaba i protočna struktura***

---

- Podijelimo li naredbu na **N faza**, onda možemo izračunati ubrzanje:
  - u slijednom izvođenju:
    - za svaku naredbu treba N vremenskih perioda
    - za M naredaba slijedno izvođenje traje:  $M * N$
  - u protočnom izvođenju:
    - prva naredba traje N perioda, a svaka sljedeća traje 1 period
    - za M naredaba protočno izvođenje traje:  $N + (M-1)$
- za  $M \gg N$  vrijedi:  $(M*N) / (N+M-1) \approx (M*N) / (M) = N$
- **Protočno izvođenje u N razina je u prosjeku N puta brže od izvođenja korak-po-korak** (uz pretpostavku linearnog programa bez hazarda)





# ***Izvođenje naredaba i protočna struktura***

---

- Radi većeg ubrzanja, dobro bi bilo naredbu podijeliti na što više faza (tj. protočnu strukturu na što više razina)
- Brzina cijele protočne strukture ovisi o brzini najsporije razine, tj. najsporija razina predstavlja ograničenje (usko grlo)
- Zato se pokušavaju odabrati faze tako da svaka traje čim kraće, ali i tako da sve imaju podjednako trajanje



# ***Izvođenje naredaba i protočna struktura***

---

- RISC arhitekture upravo koriste navedena načela da se izvodi puno brzih i jednostavnih naredaba čime se vrijeme izvođenja skraćuje
- Zato se protočna struktura i počela koristiti s pojavom procesora RISC arhitekture
- Danas se, zbog napretka tehnologije, protočnost koristi u većini procesora bez obzira jesu li RISC ili CISC
- U nastavku ćemo definirati protočnu strukturu procesora FRISC

# **Izvođenje naredaba i protočna struktura procesora FRISC**

---

# Protočna struktura FRISC-a

---

- Kako bi zadržali jednostavnost našeg procesora, za efikasnije izvođenje naredaba uvest ćemo najjednostavniju moguću protočnu strukturu
- Odabiremo **protočnu strukturu FRISC-a** sa samo **dvije razine**, koje ćemo nazvati:
  - **razina za dohvat**
  - **razina za izvođenje**



# ***Protočna struktura FRISC-a***

---

- Podjela operacija između razina je ovakva:
  - **Razina za dohvat**
    - Dohvat naredbe
    - Dekodiranje naredbe
    - Dohvat operanada
  - **Razina za izvođenje**
    - Izvođenje AL operacije
    - Spremanje rezultata

# ***Podjela naredaba po brzini izvođenja***

---

- Trajanje operacija u svakoj razini je jedan period signala vremenskog vođenja CLOCK-a (uz pretpostavku da memorija nije spora)
- Naredbe procesora FRISC razlikuju se po načinu kako se izvode:
  - **Faza dohvata svih naredaba** traje jedan period
  - **Faza izvođenja** također traje jedan period, ali kod nekih naredaba **nije moguće preklapanje** s fazom dohvata sljedeće naredbe



# Jednociklusne naredbe

---

- Kod jednostavnih naredaba moguće je preklapanje faze izvođenja sa fazom dohvata sljedeće naredbe.
  - Ove naredbe zovu se **jednociklusne** jer im efektivno vrijeme izvođenja u protočnoj strukturi iznosi jedan period (ciklus) CLOCK-a.
  - Ove naredbe efikasno koriste protočnu strukturu.
- U jednociklusne naredbe spadaju:
  - Aritmetičko-logičke naredbe
  - Registarske naredbe
  - Upravljačke naredbe kod kojih uvjet nije zadovoljen

# ***Jednociklusne naredbe***

---

- Izvođenje slijeda jednociklusnih naredaba:

	Dohvat	Izvođenje
T1	ADD	
T2	SUB	ADD
T3	XOR	SUB
T4	AND	XOR
T5		AND

# ***Dvociklusne naredbe***

---

- Kod složenijih naredaba nije moguće preklapanje faze izvođenja s fazom dohvata sljedeće naredbe
  - Takve naredbe zovu se **dvociklusne** naredbe jer im efektivno vrijeme izvođenja u protočnoj strukturi iznosi dva perioda (ciklusa) CLOCK-a.
  - Ove naredbe ne koriste efikasno protočnu strukturu
- U dvociklusne naredbe spadaju:
  - Memorijske naredbe
  - Upravljačke naredbe kod kojih je uvjet za izvođenje zadovoljen

# ***Dvociklusne naredbe***

---

- Izvođenje slijeda dvociklusnih naredaba:

	Dohvat	Izvođenje
T1	LOAD	
T2		LOAD
T3	JP	
T4		JP
T5	STORE	
T6		STORE

# ***Jednocyklusne naredbe: dohvat***

---

## Prva polovica perioda:

- Postavljanje adrese na adresnu sabirnicu       $PC \rightarrow AR$
- Aktiviranje signala rd \*\*

## Druga polovica perioda \*:

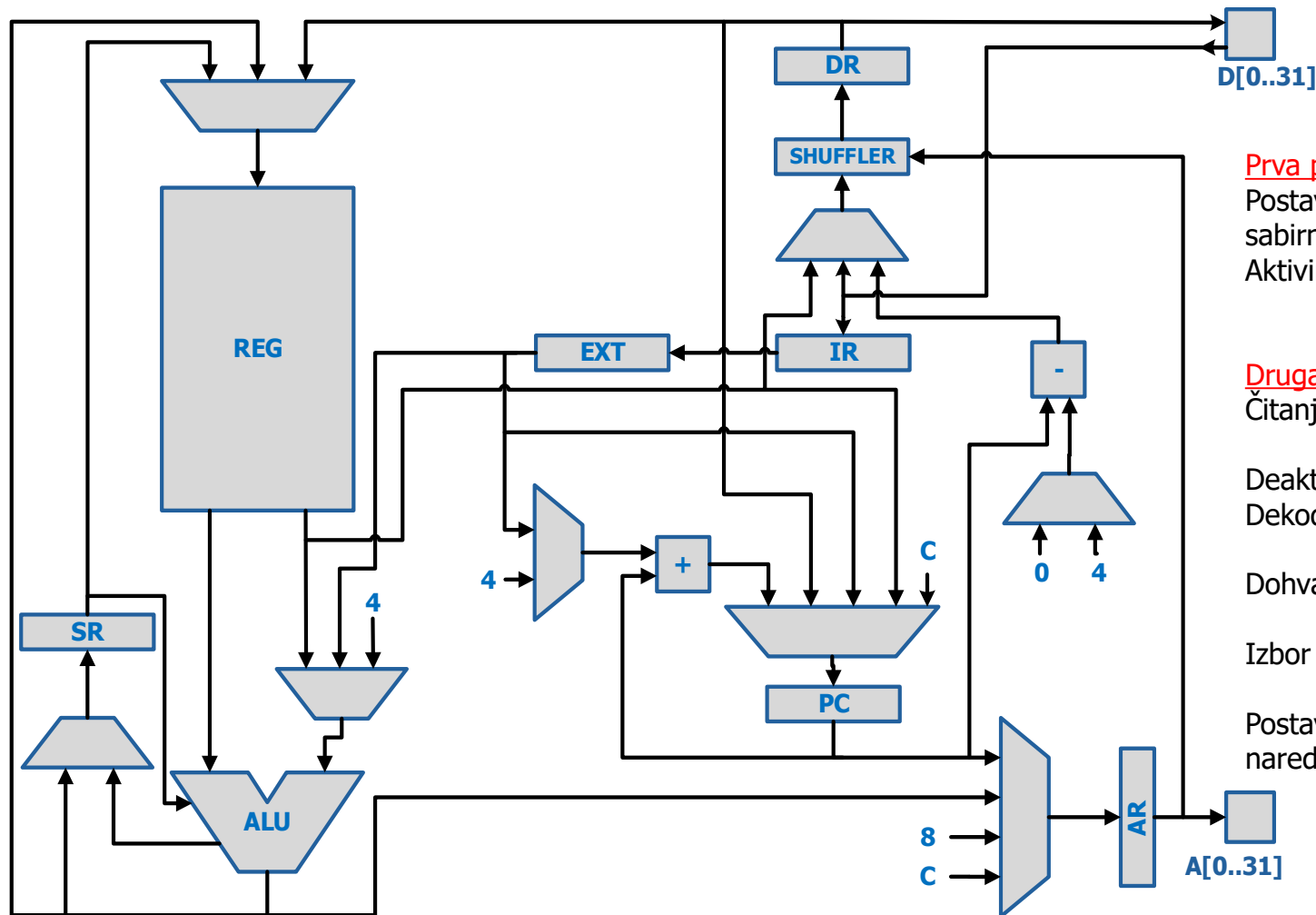
- Čitanje naredbe       $(AR) \rightarrow IR$
- Deaktiviranje signala rd \*\*
- Dekodiranje naredbe      dekodiranje
- Dohvat operandi i slanje u ALU      operandi  $\rightarrow$  ALU
- Izbor i pokretanje ALU operacije
- Postavljanje PC za sljedeću naredbu       $PC+4 \rightarrow PC$

---

\* Ovi koraci izvode se u slučaju da je memorija dovoljno brza i da nije bio aktiviran signal WAIT

\*\* Aktiviranje i deaktiviranje signala rd i wr se neće više navoditi

# Jednociklusne naredbe: dohvat



## Prva polovica perioda:

Postavljanje adrese na adresnu sabirnicu  $PC \rightarrow AR$

Aktiviranje signala rd \*\*

## Druga polovica perioda \*:

Čitanje naredbe

$(AR) \rightarrow IR$

Deaktiviranje signala rd \*\*

Dekodiranje naredbe

dekodiranje

Dohvat operandi i slanje u ALU

operandi  $\rightarrow ALU$

Izbor i pokretanje ALU operacije

Postavljanje PC za sljedeću naredbu  
 $PC+4 \rightarrow PC$



# ***Dvociklusne naredbe: dohvat***

---

- Do trenutka dekodiranja sve naredbe imaju isti način dohvata naredbe

# ***Jednocyklusne naredbe: izvođenje***

---

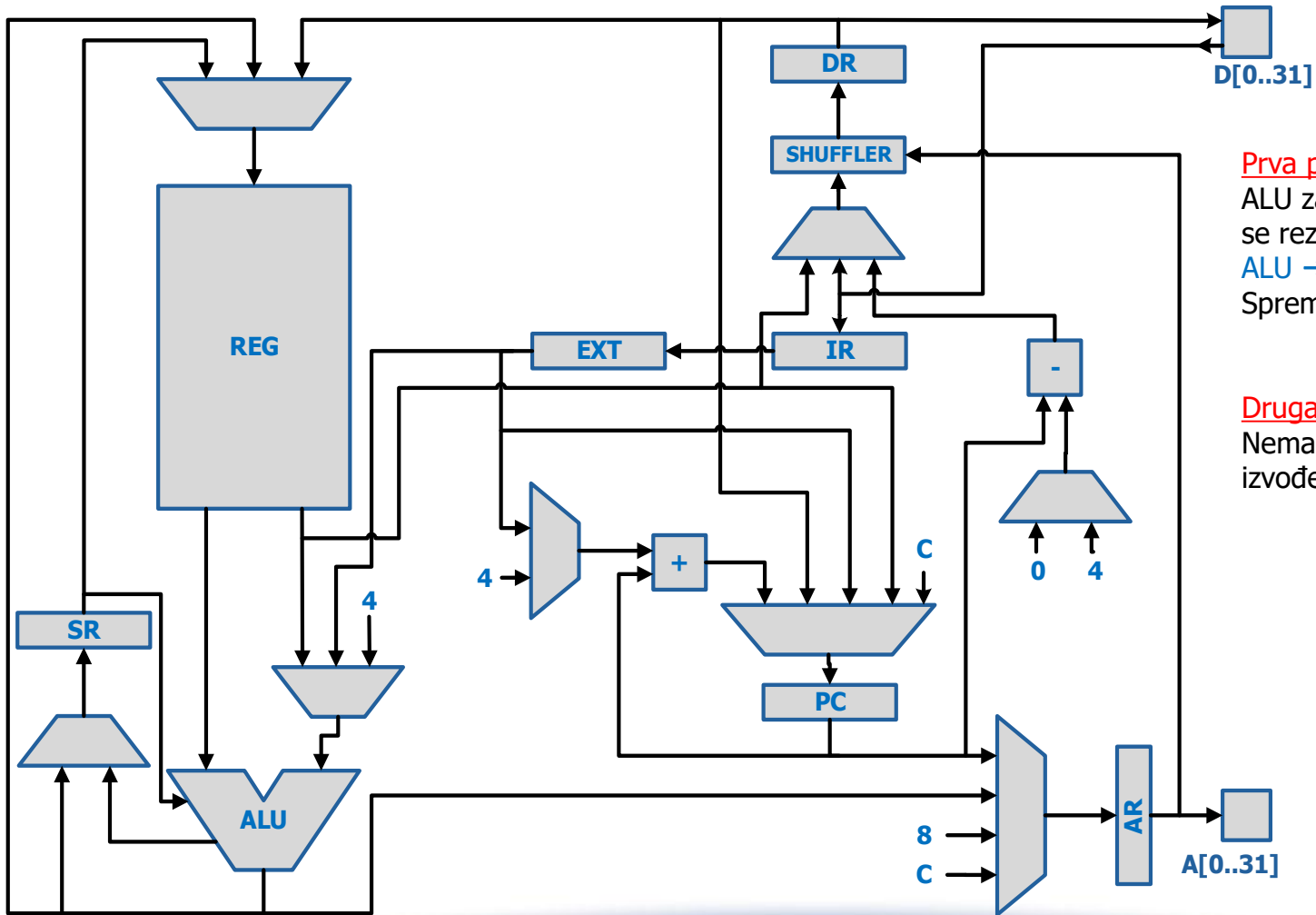
## Prva polovica perioda:

- ALU završava operaciju i sprema se rezultat (osim kod CMP)  
ALU → Reg
- Spremaju se zastavice u SR

## Druga polovica perioda:

- Nema aktivnosti vezano za izvođenje naredbe

## ***Jednociklusne naredbe: izvođenje***



Prva polovica perioda:

ALU završava operaciju i sprema se rezultat (osim kod CMP)

ALU → Reg

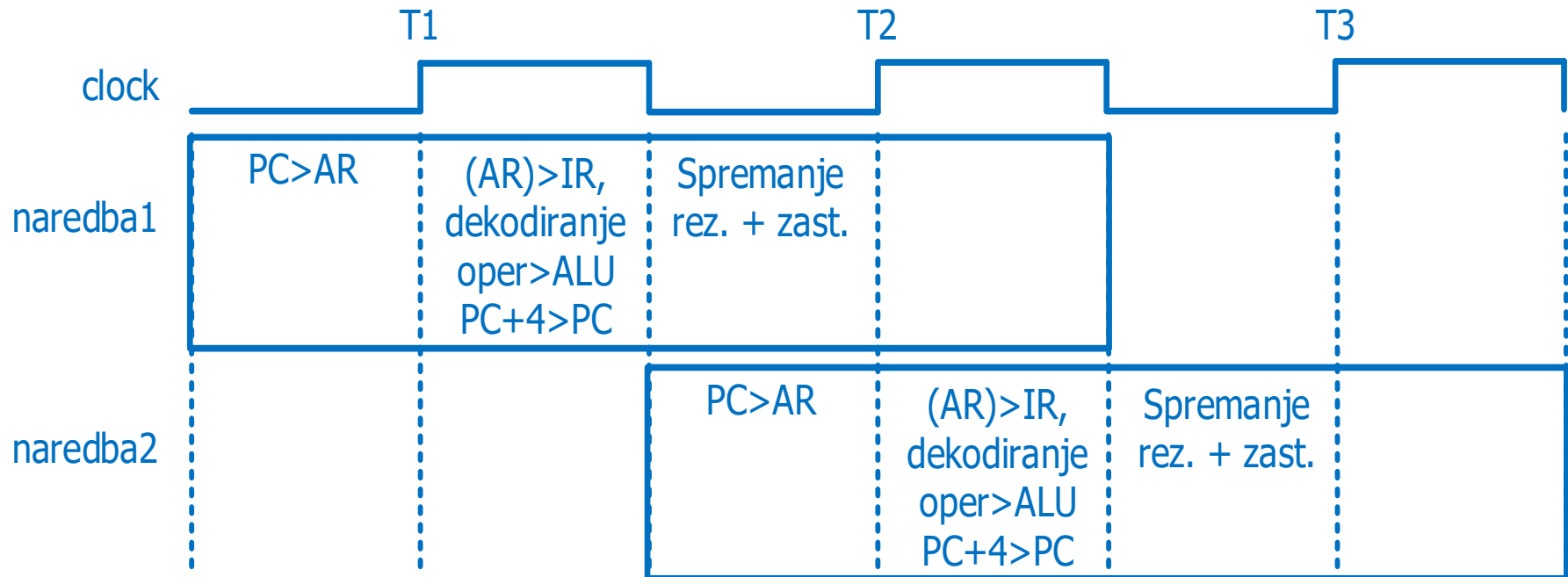
Spremaju se zastavice u SR

Druga polovica perioda \*:

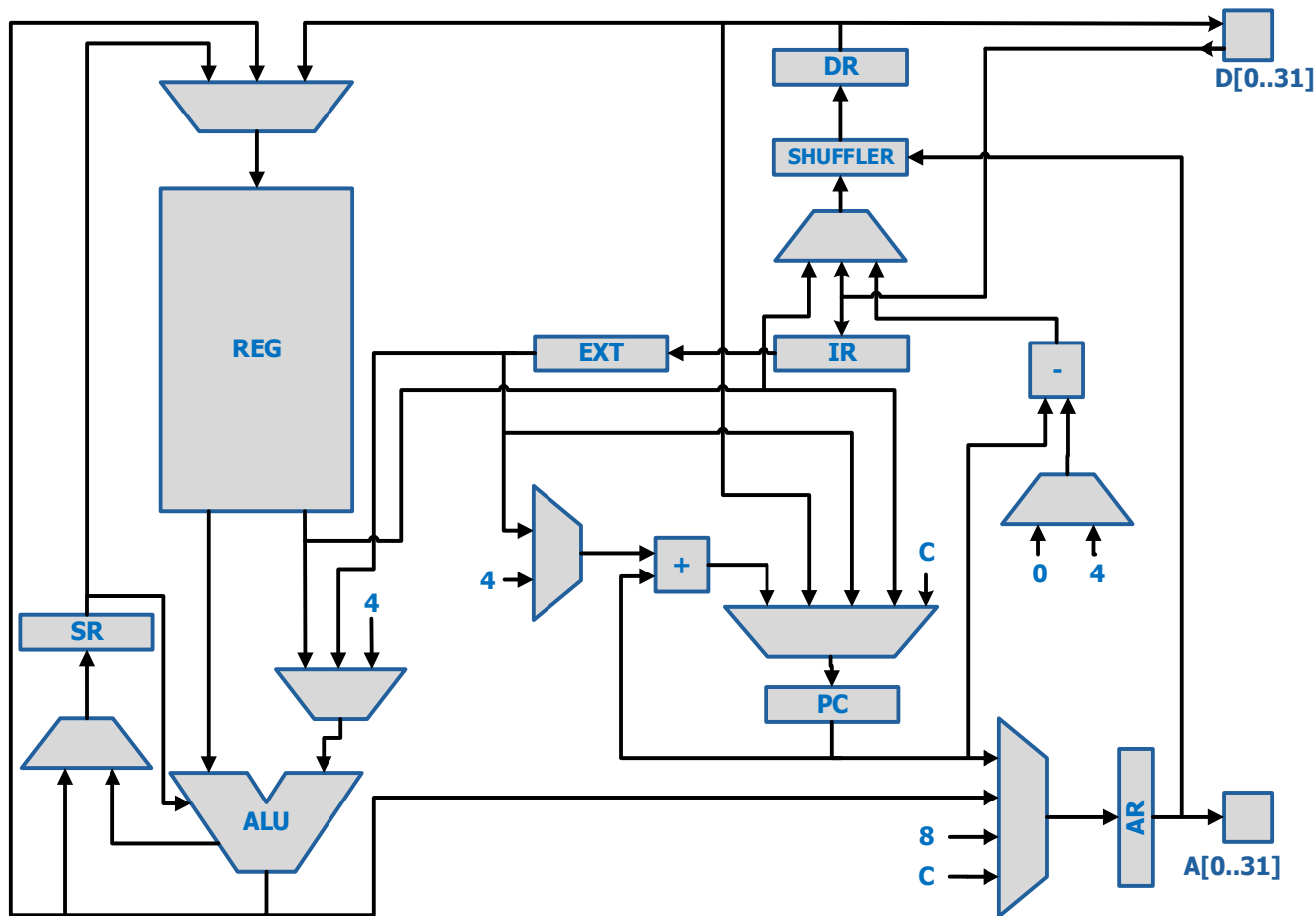
Nema aktivnosti vezano za izvođenje naredbe

# ***Izvođenje dvije AL naredbe u protočnoj strukturi***

- Uočite da mikroarhitektura dozvoljava preklapanje koraka dovata i izvođenja



## ***Jednociklusne naredbe: izvođenje***



Prva polovica perioda:

Postavljanje adrese na adresnu  
sabirnicu      PC → AR

Druga polovica perioda \*:

## Čitanje naredbe

$$(AR) \rightarrow IR$$

## Dekodiranje naredbe

dekodiranje

## Dohvat operandata i slanje u ALU

operandi → ALU

## Izbor i pokretanje ALU operacije

Postavljanje PC za sljedeću naredbu **PC+4 → PC**

Prva polovica perioda:

ALU završava operaciju i sprema se rezultat (osim kod CMP)

ALU → Reg

Spremaju se zastavice u SR

Druga polovica perioda \*:

Nema aktivnosti vezano za izvođenje naredbe

# ***Hazardi***

---

- Do sada objašnjeni rad protočne strukture za jednociklusne naredbe bio je **idealan**, no u stvarnom radu postoje situacije koje uzrokuju da protočna struktura djelomično gubi svoju efikasnost
- U nekim situacijama protočna struktura ne može obraditi sljedeću naredbu odmah u sljedećem periodu
- Takve situacije nazivaju se **hazardi**
- Postoje tri osnovna tipa hazarda:
  - **Strukturni**
  - **Upravljački**
  - **Podatkovni**



# ***Strukturni hazard***

---

- Strukturni hazard je pojava kad procesor u određenom trenutku ne može izvesti sve faze onih naredaba koje se nalaze u protočnoj strukturi, jer **struktura** (tj. sklopovlje) procesora ne omogućuje istodobno izvođenje svih tih faza
- Jednostavan primjer strukturnog hazarda je izvođenje naredbe npr. LOAD ili npr. STORE kod procesora s Von Neumannovom arhitekturom. Struktura memorijskog sučelja (Von Neumannova arhitektura) ne dozvoljava istovremeno dva pristupa memoriji:
  - jedan pristup treba za izvođenje naredbe (npr. kod naredbe STORE za spremanje podatka)
  - drugi pristup je dohvat strojnog kôda sljedeće naredbe

# ***Strukturni hazard***

---

- Strukturni hazard rješava se tako da procesor odgodi izvođenje naredaba koje se nalaze u prethodnim razinama protočne strukture (to su naredbe koje su kasnije ušle u protočnu strukturu) dok se uzrok hazarda ne ukloni.
- Odgoda izvođenja naredaba u prethodnim razinama u literaturi se naziva **mjehurić (bubble)**. Slikovito se promatra kao da je u protočnu strukturu ušao mjehurić koji prolazi kroz razine i uzrokuje njihovu neaktivnost.
- Kada uzrok hazarda nestane, procesor nastavlja izvoditi naredbe na normalan način.

# ***Naredbe FRISC-a i strukturni hazard***

---

- Zbog Von Neumannove arhitekture memorijskog sučelja, sve memorijske naredbe procesora FRISC (LOAD, STORE, LOADH, STOREH, LOADB, STOREB, PUSH, POP) uzrokovat će strukturni hazard.
- Te naredbe imat će fazu izvođenja koja se ne može preklapati s fazom dohvata sljedeće naredbe te će njihovo izvođenje efektivno trajati dva perioda. Zato ćemo te naredbe svrstati u grupu **dvociklusnih** naredaba.

# ***Izvođenje memorijskih naredaba***

---

- **Memorijske naredbe LOAD i STORE:**

- Naredba LOAD prilikom izvođenja mora pročitati podatak iz memorije, a naredba STORE ga mora upisati u memoriju (sve isto vrijedi i za naredbe LOADH,LOADB, STOREH, STOREB)
- Pristup memoriji pri izvođenju LOAD/STORE **ne može** se odvijati istodobno s dohvatom sljedeće naredbe iz iste memorije
- Zato se, nakon prepoznavanja naredbe LOAD/STORE, **onemogućuje** rad razine dohvata u sljedećem ciklusu (mjehurić) te će biti aktivna samo razina izvođenja
- Na kraju izvođenja se zato **omogućuje** rad razine za dohvat u sljedećem ciklusu
- Međutim, za vrijeme tog (odgođenog) dohvata bit će neaktivna razina za izvođenje (mjehurić prelazi iz razine dohvata u razinu izvođenja), jer nema dohvaćene naredbe koja bi se mogla izvoditi

# Primjer






- Analizirajmo broj perioda potrebnih za izvođenje programa za zbrajanje dva broja:

LOAD R0, (100)

LOAD R1, (200)

ADD R0,R1,R2

STORE R2, (300)

	Dohvat	Izvođenje
T1	LOAD R0, (100)	
T2		LOAD R0, (100)
T3	LOAD R1, (200)	
T4		LOAD R1, (200)
T5	ADD R0,R1,R2	
T6	STORE R2,(300)	ADD R0,R1,R2
T7		STORE R2,(300)

- $T = 7$

# Dvociklusne naredbe: dohvat *STORE*

---

Prva polovica perioda:

- Postavljanje adrese na adresnu sabirnicu PC → AR

Druga polovica perioda \*:

- Citanje naredbe (AR) → IR
- Dekodiranje naredbe dekodiranje
- Dohvat adrese\*\* i slanje prema ALU (ext → ALU) ili (Rx, ext → ALU)
- Prosljeđivanje adrese ili izračun adrese\*\* (ALU prosljeđuje) ili (ALU zbraja)
- Postavljanje PC za sljedeću naredbu PC+4 → PC
- onemogućići dohvat u sljedećem ciklusu

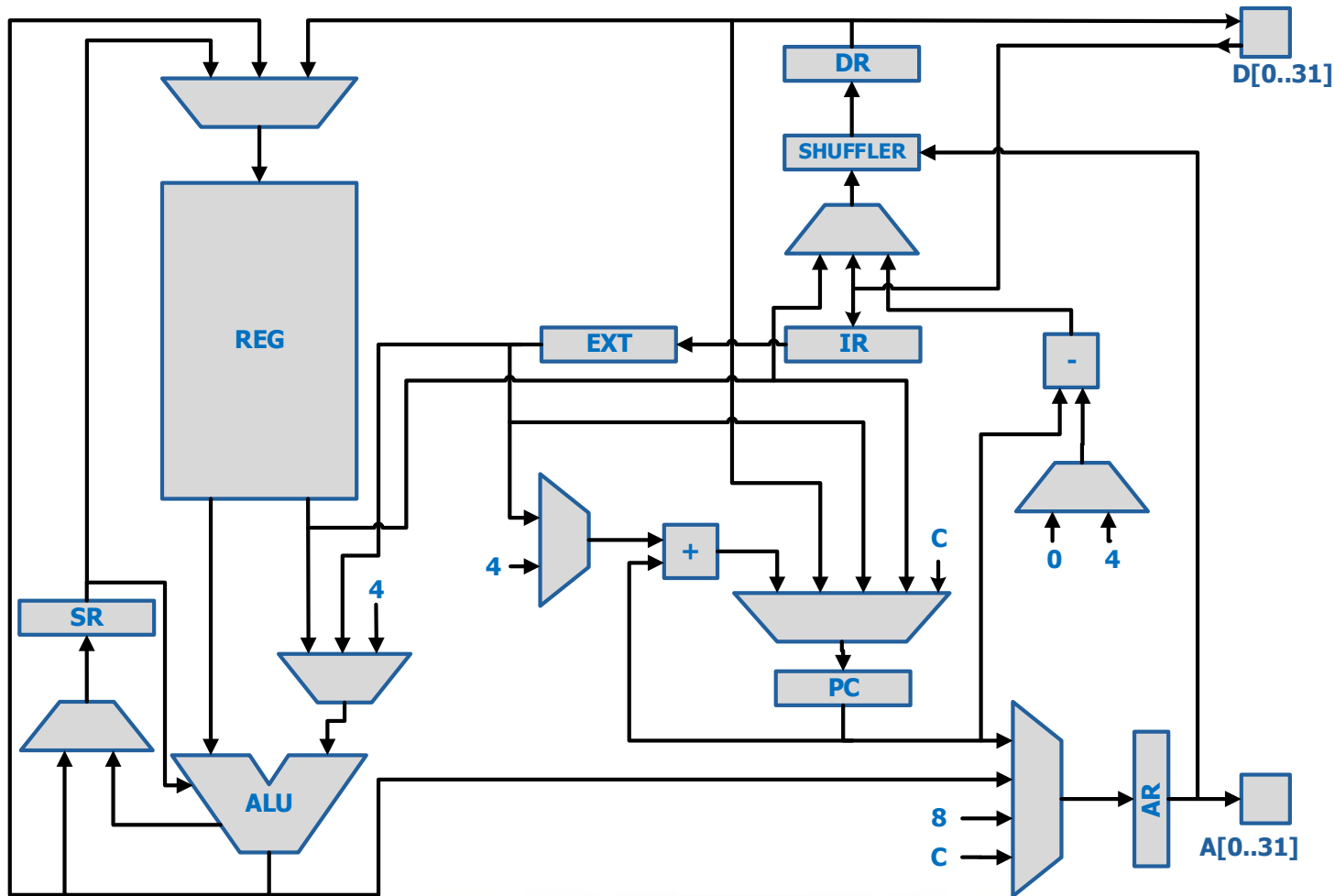
---

\* Ovi koraci izvode se u slučaju da je memorija dovoljno brza i da nije bio aktiviran signal WAIT

\*\* Ovisno o načinu adresiranja u naredbi



# Dvociklusne naredbe: dohvat *STORE*



# ***Dvociklusne naredbe: izvođenje STORE***

---

Prva polovica perioda:

- Postavljanje adrese na adr. sabirnicu       $ALU\_OUT \rightarrow AR$
- Podatak na sab. podataka (uz shuffle)       $REG\_B \rightarrow DR$

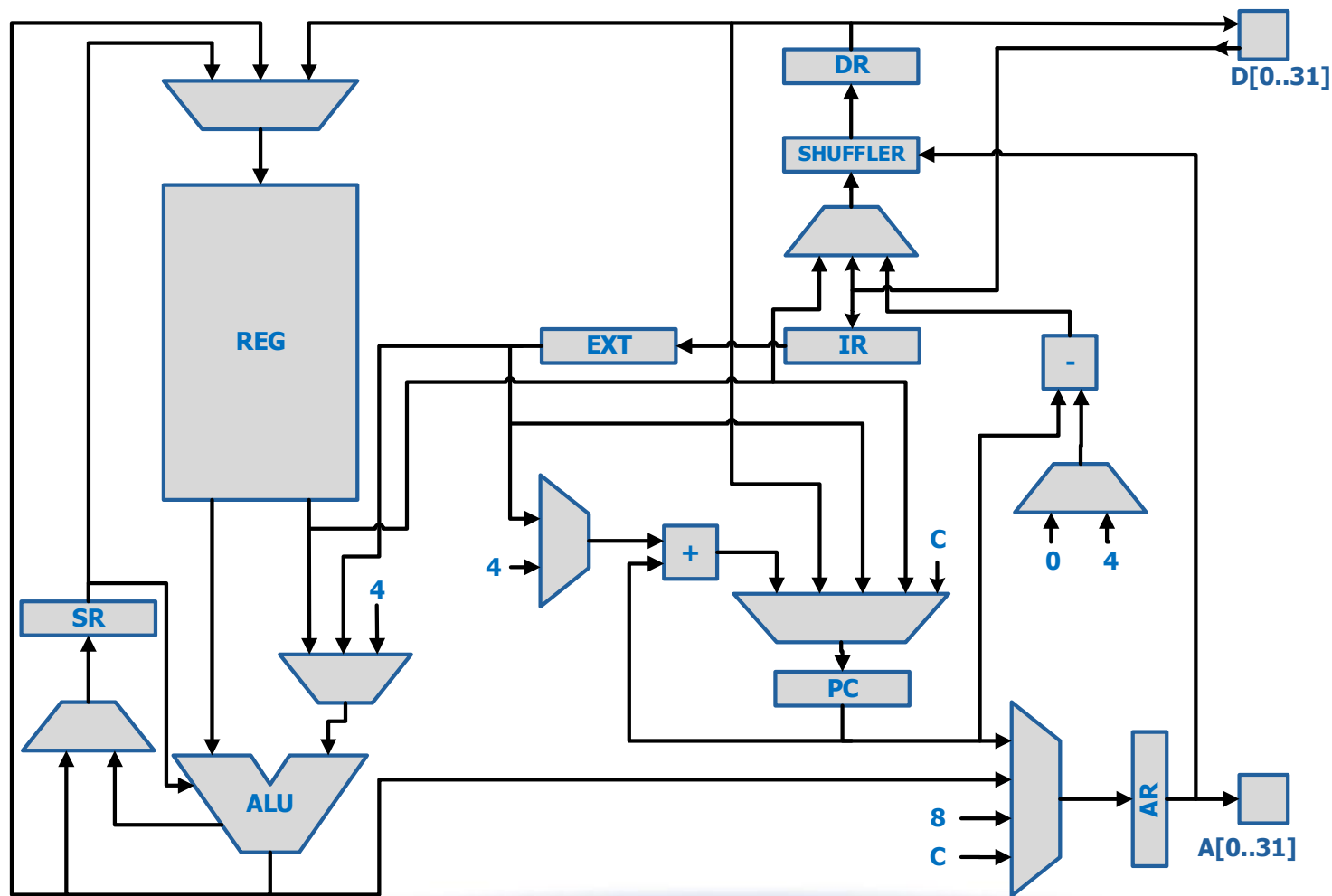
Druga polovica perioda \*:

- omogući dohvat u sljedećem ciklusu

\*Ovi koraci izvode se u slučaju da je memorija dovoljno brza i da nije bio aktiviran WAIT signal

**Na WEBu se nalazi ispravak opisa izvođenja naredbe STORE iz knjige !!!  
(u knjizi je pogrešno  $ALU\_OUT \rightarrow AR$  bilo napisano u ciklusu dohvata)**

# Dvociklusne naredbe: izvođenje



# ***Primjer: STOREB R1, (R2+3ABC)***

---

Razina dohvata:

Prva polovina periode CLOCK-a:

PC -> AR

Druga polovina periode CLOCK-a:

(AR) -> IR,

dekodiranje

ext 3ABC i R2 -> ALU

ALU: izvodi zbrajanje

PC +4 -> PC

onemogućí dohvat u sljedećem ciklusu

Razina izvođenja:

Prva polovina periode CLOCK-a:

ALU -> AR

R1 -> DR (uz shuffle)

Druga polovina periode CLOCK-a:

omogućí dohvat u sljedećem ciklusu

# ***Primjer: LOAD R0, (R1+300)***

---

Razina dohvata:

Prva polovina periode CLOCK-a:

PC -> AR

Druga polovina periode CLOCK-a:

(AR) -> IR,

dekodiranje

ext 300 i R1 -> ALU

ALU: izvodi zbrajanje

PC +4 -> PC

onemogućí dohvat u sljedećem ciklusu

Razina izvođenja:

Prva polovina periode CLOCK-a:

ALU -> AR

Druga polovina periode CLOCK-a:

(AR) -> DR

DR -> R0

omogućí dohvat u sljedećem ciklusu

# ***Dvociklusne naredbe: dohvat PUSH***

---

(Dohvat je sličan kao i kod LOAD/STORE samo se za adresu uzima R7-4)

Prva polovica perioda:

- Postavljanje adrese na adresnu sabirnicu PC → AR

Druga polovica perioda \*:

- Čitanje naredbe (AR) → IR
- Dekodiranje naredbe dekodiranje
- Dohvat adrese i slanje prema ALU R7, 4 → ALU
- ALU ALU oduzimanje
- Postavljanje PC za sljedeću naredbu PC+4 → PC
- onemogućiti dohvat u sljedećem ciklusu

---

\* Ovi koraci izvode se u slučaju da je memorija dovoljno brza i da nije bio aktiviran signal WAIT

# ***Dvociklusne naredbe: izvođenje PUSH***

---

## Prva polovica perioda:

- Postavljanje adrese na adresnu sabirn.
- Osvježavanje SP
- Podatak na sabirnicu podataka

ALU\_OUT → AR

ALU\_OUT → R7

REG\_B → DR

## Druga polovica perioda \*:

- omogućiti dohvat u sljedećem ciklusu

---

\* Ovi koraci izvode se u slučaju da je memorija dovoljno brza i da nije bio aktiviran signal WAIT



# ***Izvođenje memorijskih naredaba***

---

- **Memorijske naredbe PUSH i POP:**

- Prilikom izvođenja naredaba PUSH i POP također se pristupa memoriji kao i kod naredba LOAD i STORE, pa se naredbe izvode na sličan način
- Dodatno, naredba PUSH smanjuje registar SP, a POP povećava SP
- DZ: Proučiti iz knjige način izvođenja naredbe POP. Razmisliti koje su razlike između izvođenja LOAD I STORE, te koje su razlike između izvođenja POP i PUSH
- Napredno gradivo: proučiti tablicu izvođenja svih naredaba procesora FRISC (osim za prekide)



# Upravljački hazard

---

- Drugi od tri ranije spomenuta hazarda je **upravljajući hazard**. Ovaj hazard dešava se kad naredba koja se nalazi u protočnoj strukturi i spremna je za izvođenje nije naredba koja se u stvari treba izvesti
- Ovaj hazard događa se kod izvođenja naredaba grananja kad je procesor već učitao sljedeću naredbu i pripremio se za njeno izvođenje, ali zbog grananja program treba nastaviti s izvođenjem naredbe na nekoj drugoj adresi
- Zbog toga se ovaj hazard naziva još i hazardom grananja
- Naredbe koje uzrokuju ovaj hazard kod FRISC-a su upravljačke naredbe

# Primjer

```
0   SUB R0,R0,R0
4   JP_NZ 14           ; false
8   ADD R0, 2, R0
C   JP_NZ 18           ; true
10  OR  R0,R0,R0
14  AND R0,R0,R0
18  XOR R0,R0,R0
```

	Dohvat	Izvođenje
T1	SUB R0, R0, R0	
T2	JP_NZ 14	SUB R0, R0, R0
T3	ADD R0, 2, R0	JP_NZ 14
T4	JP_NZ 18	ADD R0, 2, R0
T5		JP_NZ 18
T6	XOR R0, R0, R0	
T7	...	XOR R0, R0, R0

# ***Uvjetna upravljačka naredba***

---

- uvjetna upravljačka naredba ponaša se kao:
  - dvociklusna kada je uvjet zadovoljen (jer dolazi do pojave hazarda i umeće se mjehurić u protočnu strukturu),
  - jednociklusna kada uvjet nije zadovoljen

# ***Izvođenje upravljačkih naredaba***

---

- Specifičnosti pojedinih upravljačkih naredaba su:
  - Adresa skoka zadana je brojem ili registrom: zbrajanje nije potrebno za izračun adrese skoka (za razliku od memorijskih naredaba s indirektnim registarskim adresiranjem s odmakom)
  - Izuzetak je naredba JR u kojoj je zadana relativna adresa koja se pribraja registru PC, ali za to se ne koristi ALU, nego zasebni sklop za zbrajanje koji postoji uz registar PC
  - Naredbe CALL i RET su specifične po tome što osim promjene tijeka izvođenja zahtijevaju dodatni pristup memoriji, tj. stogu (po tome su one slične naredbama PUSH i POP)

# *Upravljačke naredbe: dohvat JP*

---

## Prva polovica perioda:

- Postavljanje adrese na adresnu sabirnicu  $PC \rightarrow AR$

## Druga polovica perioda \*:

- Čitanje naredbe  $(AR) \rightarrow IR$
- Dekodiranje naredbe dekodiranje
- Ispitivanje UVJETA
- Postavljanje PC za sljedeću naredbu  $PC+4 \rightarrow PC$
- **Ako je UVJET istinit onemogućiti dohvat u sljedećem ciklusu**

\*Ovi koraci izvode se u slučaju da je memorija dovoljno brza i da nije bio aktiviran WAIT signal

# ***Upravljačke naredbe: izvođenje JP***

---

**OVO SE IZVODI AKO JE UVJET BIO ISTINIT!**

Prva polovica perioda:

- Nema aktivnosti

Druga polovica perioda:

- Adresa skoka u PC      adr → PC
- omogući dohvat u sljedećem ciklusu

**OVO SE IZVODI AKO UVJET NIJE BIO ISTINIT!**

Prva polovica perioda:

- Nema aktivnosti

Druga polovica perioda:

- Nema aktivnosti



# ***Upravljačke naredbe: JR***

---

- Dohvat i izvođenje isto kao i JP osim što se u periodu izvođenja računa adresa skoka:

PC+ext → PC

# ***Upravljačke naredbe: CALL, RET, HALT***

---

- Ove naredbe također su jednociklusne ako je zadan uvjet za izvođenje i on nije istinit, a dvociklusne su ako su bezuvjetne ili je zadani uvjet istinit
- DZ: proučite iz knjige korake pri dohvat i izvođenju naredbe CALL

# Primjer

---

- Treba odrediti ukupno trajanje programa u ciklusima

```
START  ORG 0
        MOVE 50, R0
        SUB R0, 5, R0
        CALL_Z PRIPR
PETLJA SUB R0, 1, R0
        JR_NE PETLJA
        CALL_Z PRIPR
        HALT
PRIPR   ADD R0, 5, R0
        RET
```

# Primjer

- Treba odrediti ukupno trajanje programa u ciklusima

START	ORG 0	0 pseudonaredba !!!
	MOVE 50, R0	1 x 1c
	SUB R0, 5, R0	1 x 1c ( R0 = 4B = 75 <sub>10</sub> !!!)
	CALL_Z PRIPR	1 x 1c (uvjet nije istinit!)
PETLJA	SUB R0, 1, R0	75 <sub>10</sub> x 1c
	JR_NE PETLJA	74 <sub>10</sub> x 2c + 1 x 1c = 149 <sub>10</sub> c
	CALL_Z PRIPR	1 x 2c (uvjet istinit!)
	HALT	1 x 2c (objašnjenje kasnije)
PRIPR	ADD R0, 5, R0	1 x 1c
	RET	1 x 2c

UKUPNO:  $3c + (75 \times 3c - 1c) + (2c + 3c) + 2c = 234 c$

# *Objašnjenje izračuna trajanja programa*

---

Slučaj #1:

- prva naredba jednociklusna
- zadnja naredba jednociklusna:

ADD

SUB

AND

- Trajanje: 4 perioda

DOHVAT	IZVOĐENJE
ADD	
SUB	ADD
AND	SUB
	AND

# *Objašnjenje izračuna trajanja programa*

Slučaj #2A:

- prva naredba jednociklusna
- zadnja naredba dvociklusna:

ADD

SUB

LOAD

- Trajanje: 4 perioda

DOHVAT	IZVOĐENJE
ADD	
SUB	ADD
LOAD	SUB
	LOAD

# Objašnjenje izračuna trajanja programa

Slučaj #2B:

- prva naredba jednociklusna
- zadnja naredba HALT (dvociklusna):

ADD

SUB

HALT

- Trajanje: 4 perioda

DOHVAT	IZVOĐENJE
ADD	
SUB	ADD
HALT	SUB
	HALT



# Objašnjenje izračuna trajanja programa

Slučaj #3:

- prva naredba dvociklusna
- zadnja naredba jednociklusna:

LOAD

SUB

AND

- Trajanje: 5 perioda

DOHVAT	IZVOĐENJE
LOAD	
	LOAD
SUB	
AND	SUB
	AND

# Objašnjenje izračuna trajanja programa

Slučaj #4A:

- prva naredba dvociklusna
- zadnja naredba dvociklusna:

LOAD

SUB

STORE

- Trajanje: 5 perioda

DOHVAT	IZVOĐENJE
LOAD	
	LOAD
SUB	
STORE	SUB
	STORE

# *Objašnjenje izračuna trajanja programa*

---

Slučaj #4B:

- prva naredba dvociklusna
- zadnja naredba HALT(dvociklusna):

LOAD

SUB

HALT

- Trajanje: 5 perioda

DOHVAT	IZVOĐENJE
LOAD	
	LOAD
SUB	
HALT	SUB
	HALT

# *Objašnjenje izračuna trajanja programa*

---

## Sažetak:

- Pri izračunu trajanja programskog odsječka za FRISC najjednostavnije Vam je ne razmišljati o inicijalnom punjenju protočne strukture nego za sve naredbe OSIM ZADNJE računati njeno efektivno trajanje (jednociklusna ili dvociklusna) a za ZADNJU NAREDBU **UVIJEK** UZETI **DVA PERIODA** bez obzira da li je naredba jednociklusna ili dvociklusna.

# ***Podatkovni hazard***

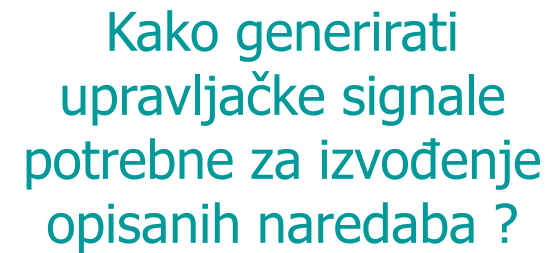
---

- **Podatkovni hazard** javlja se kod izvođenja naredaba u protočnoj strukturi kada se naredba ne može izvesti jer podaci potrebni za njeno izvođenje još nisu spremni
- Kod FRISC-a nema pojave podatkovnog hazarda pa ćemo primjer ovog hazarda proučiti kod procesora ARM

# *Cjelovit rad procesora*

---

- U prethodnim predavanjima proučili smo arhitekturu puta podataka i proučili način izvođenja naredaba
- U nastavku ćemo ukratko objasniti na koji način funkcionira procesor kao cjelina, odnosno koja su načela upravljanja sa svim dijelovima puta podataka
- Prisjetimo se nakratko mikroarhitekture puta podataka procesora FRISC



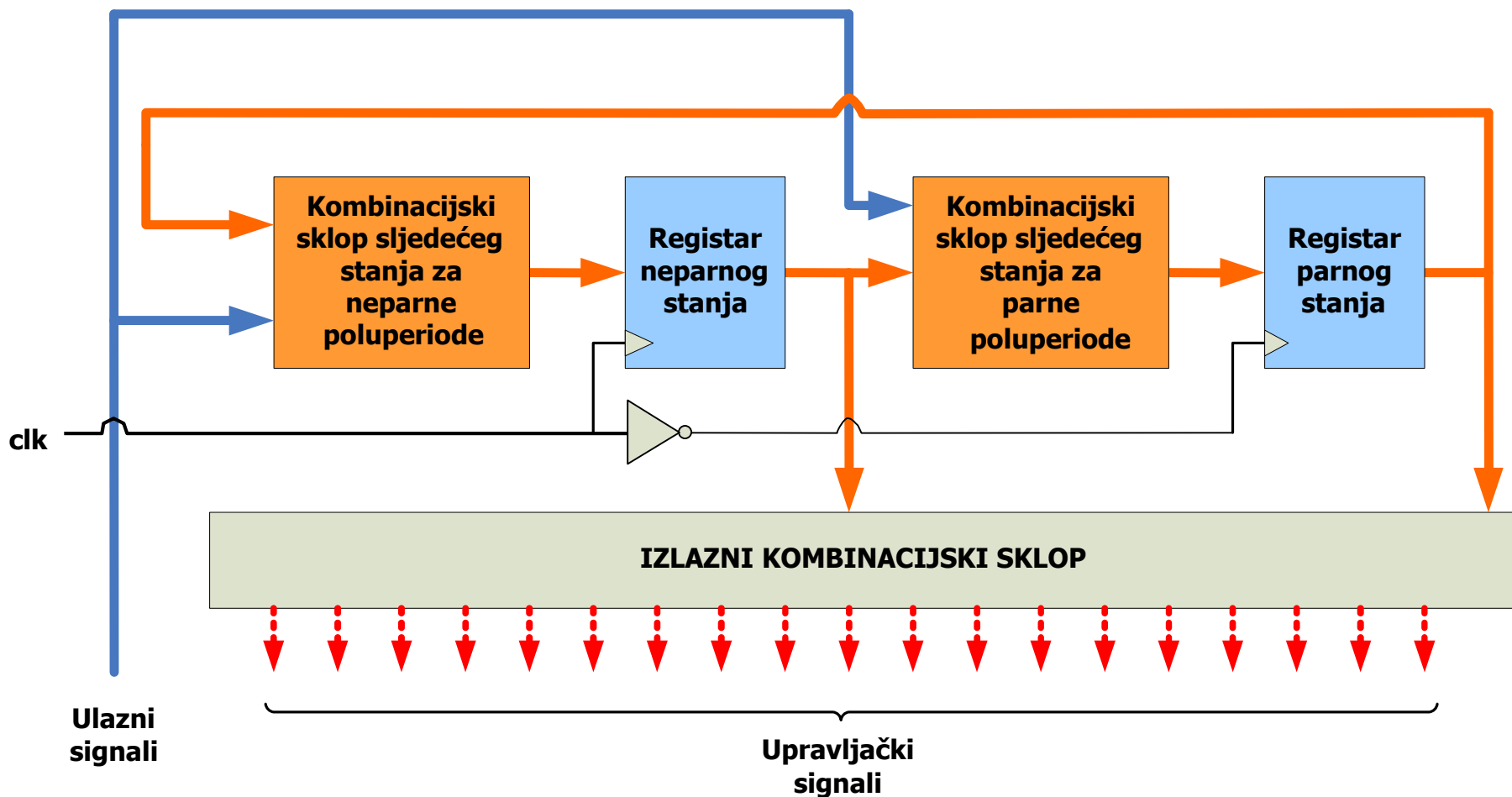


# *Upravljanje putom podataka*

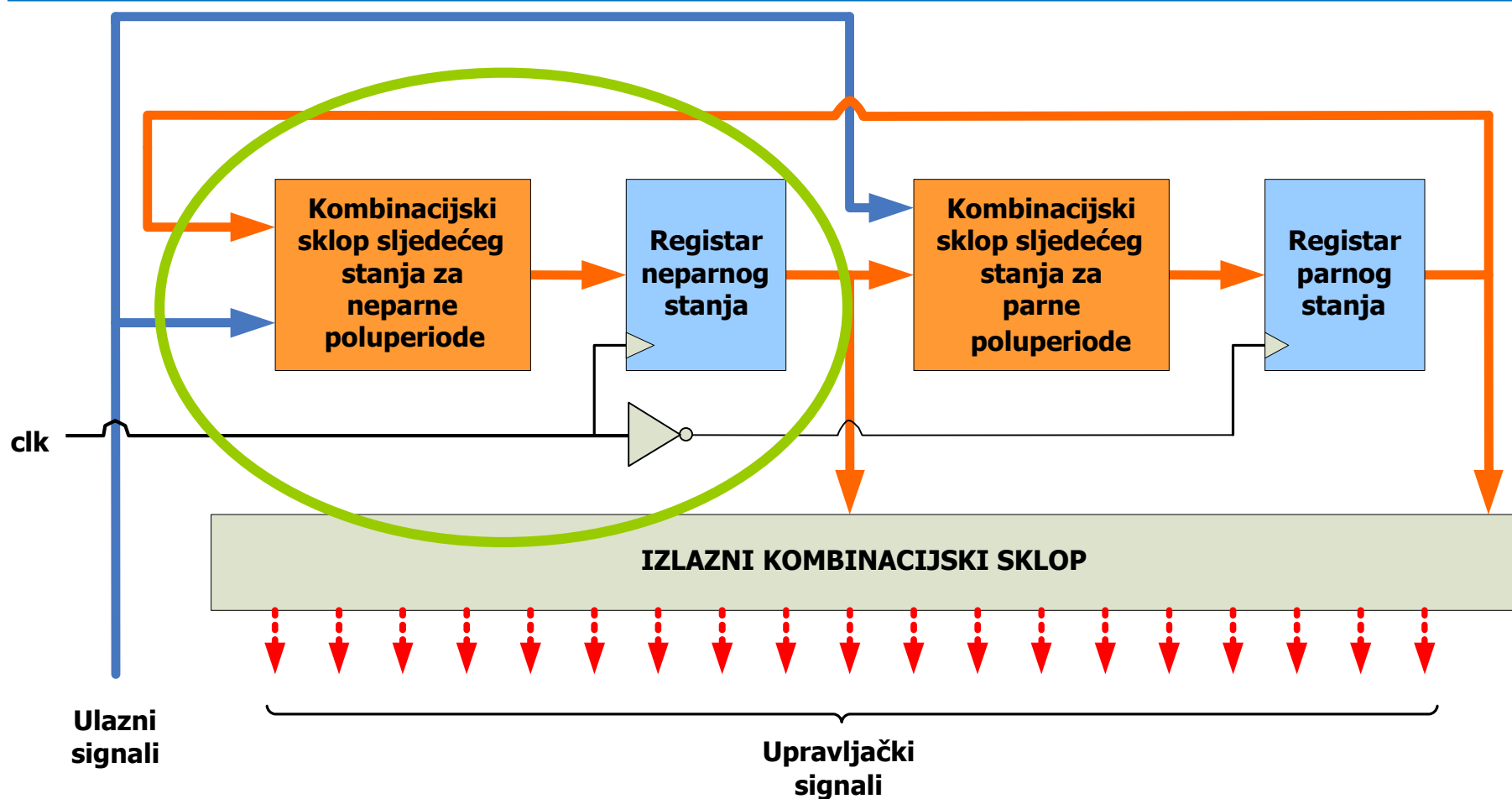
---

- Kao što ste naučili u "Digitalnoj", jednostavan način generiranja upravljačkih signala može se postići strojem s konačnim brojem stanja (finite state machine - FSM)
- Pri izvođenju naredaba treba generirati upravljačke signale na rastući i na padajući brid signala vremenskog vođenja clock
- Klasični FSM generira signale na jedan od bridova (rastući ili padajući) pa upravljačka jedinica FRISC-a koristi dvostruki FSM

# Upravljačka jedinica – stroj s konačnim brojem stanja

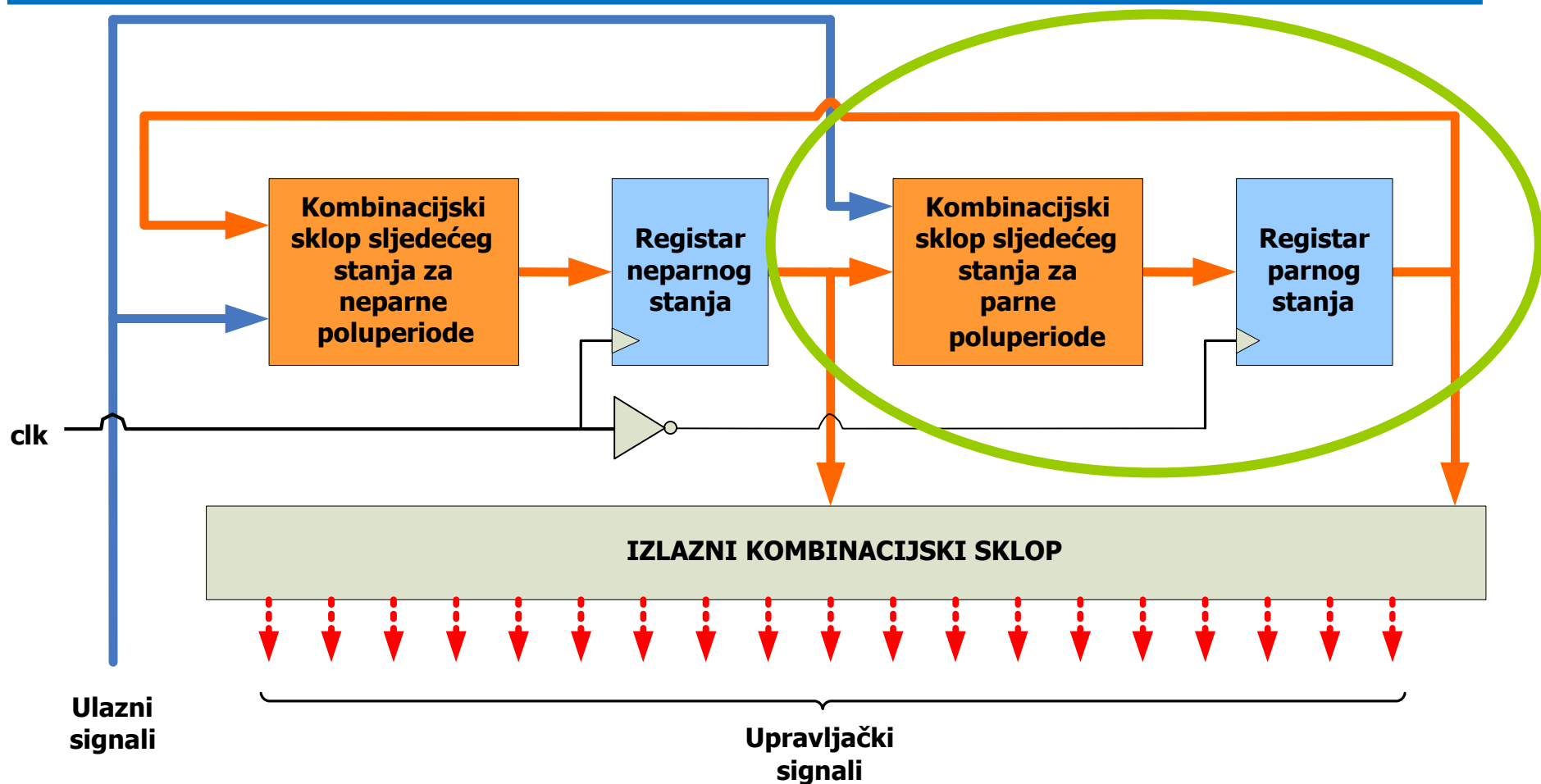


# Upravljačka jedinica – stroj s konačnim brojem stanja



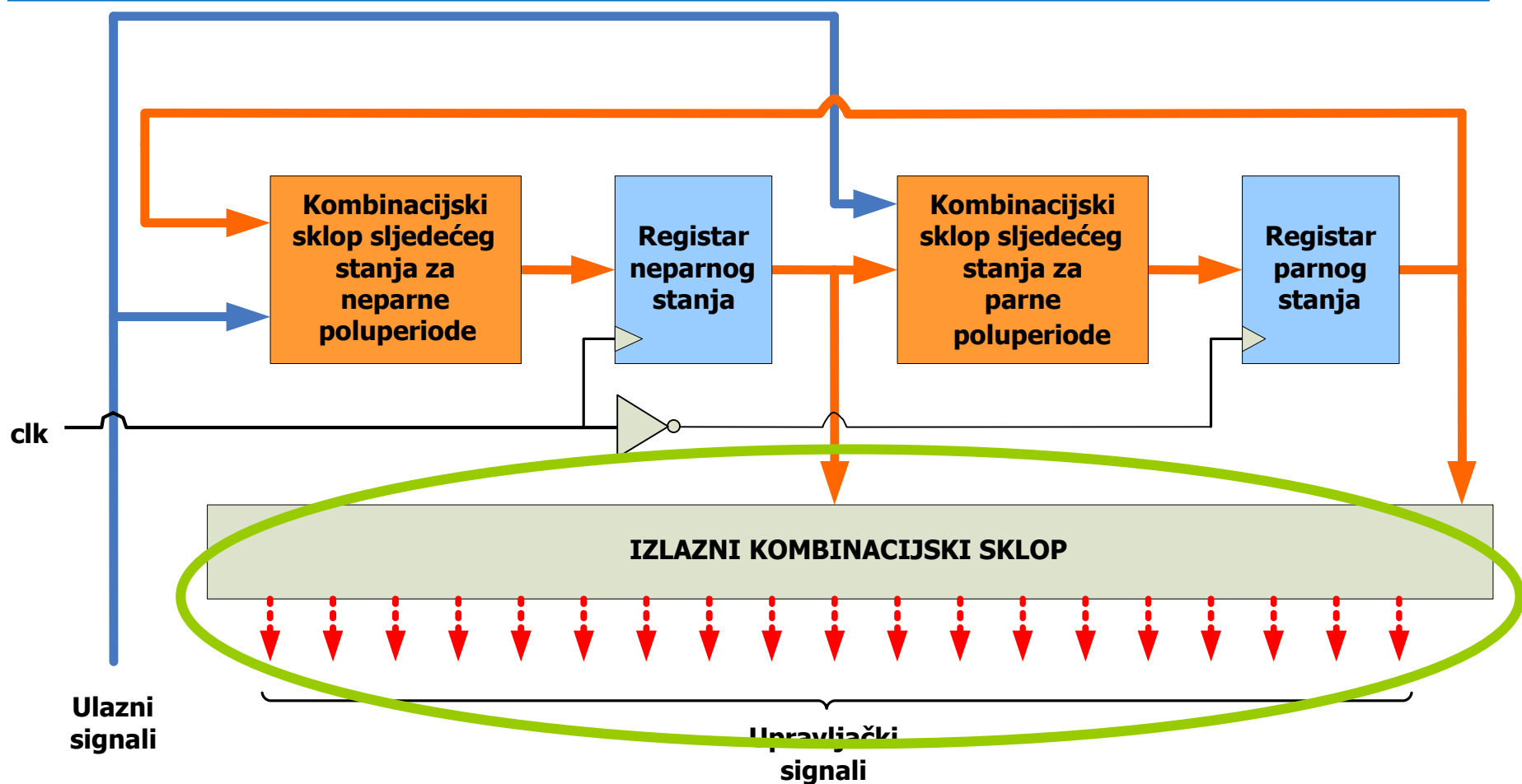
Prvi FSM zadužen je za generiranje svih signala u neparnim poluperiodima

# Upravljačka jedinica – stroj s konačnim brojem stanja



Drugi FSM zadužen je za generiranje svih signala u parnim poluperiodima

# Upravljačka jedinica – stroj s konačnim brojem stanja



Upravljački signali generiraju se na temelju stanja oba stroja