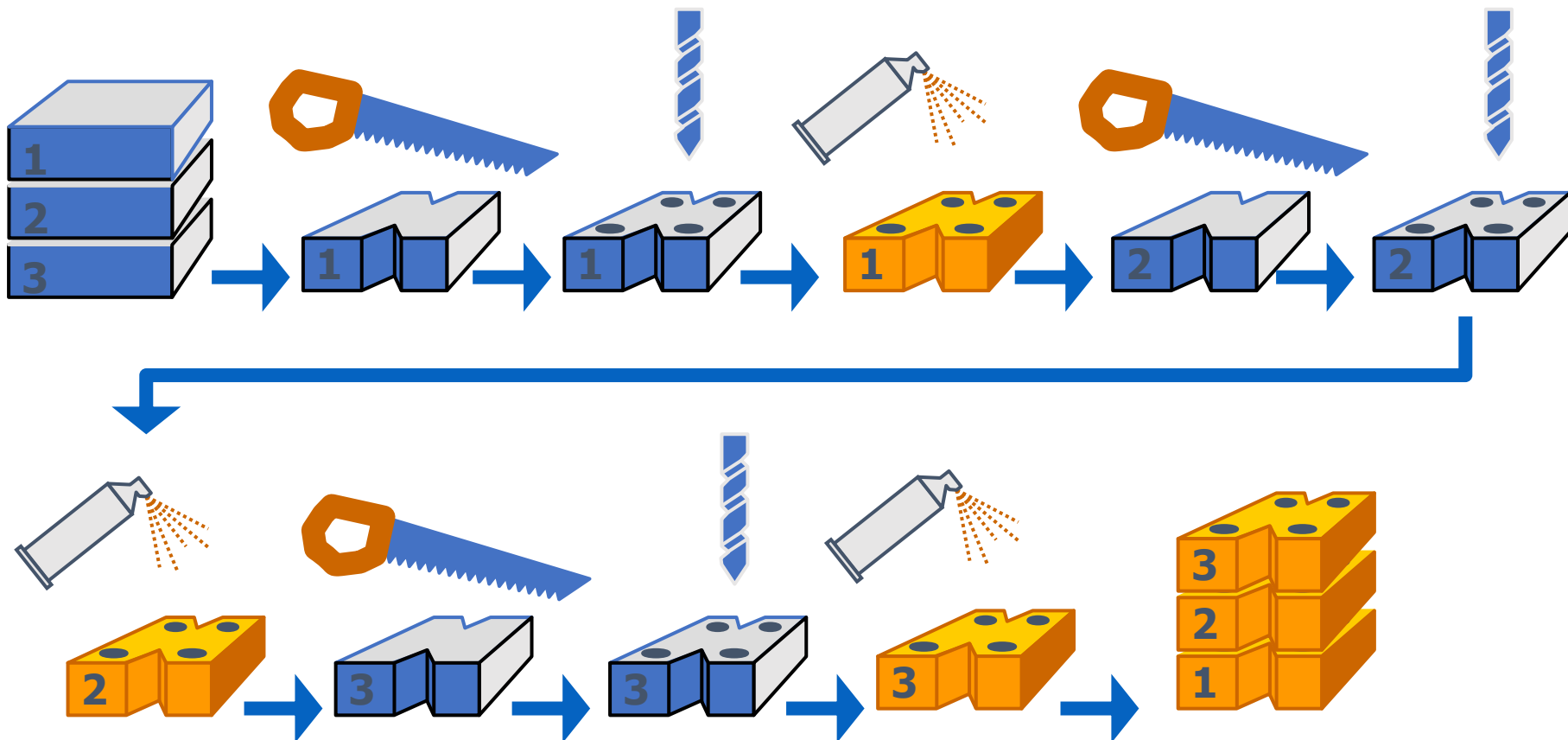


# Izvođenje naredaba i protočna struktura

# Efikasno izvođenje poslova

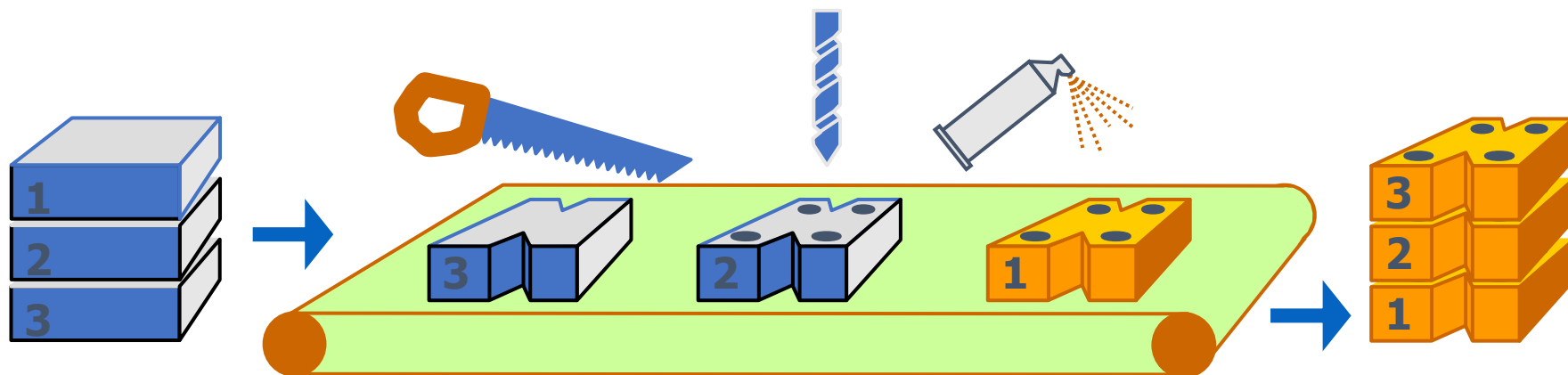
- Prije diskusije o izvođenju naredaba na procesoru pogledajmo jedan neračunarski primjer: izrada proizvoda koje treba izrezati, izbušiti i lakirati.



**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 !!**

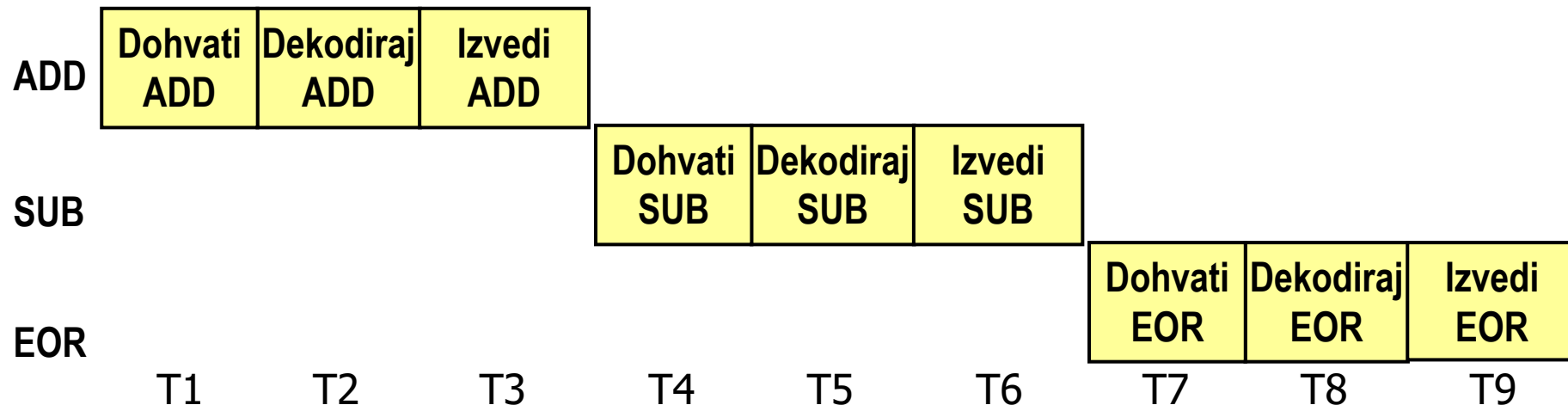
**Uvjet koji ne smijemo zaboraviti: za ovakvu proizvodnju treba nam više radne snage: 3 radnika !!! -> Cijena rada sustava je veća.**

# 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

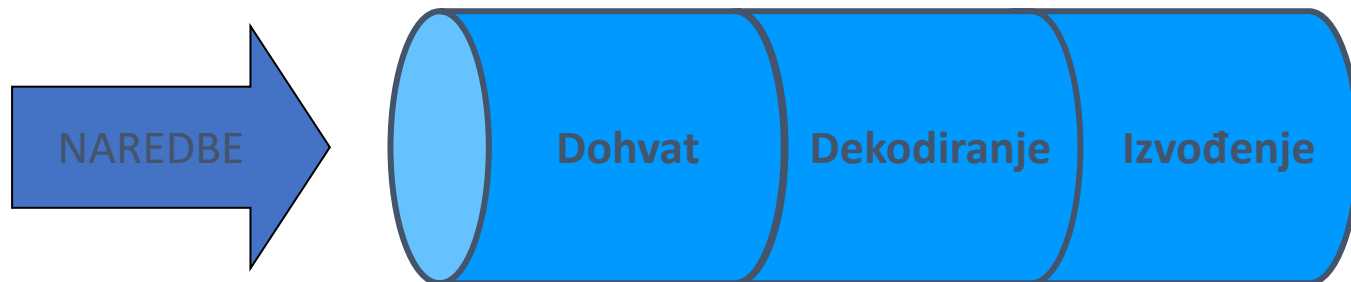
- Npr. neka imamo redom naredbe ADD, SUB i EOR koje se trebaju izvesti:



- 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

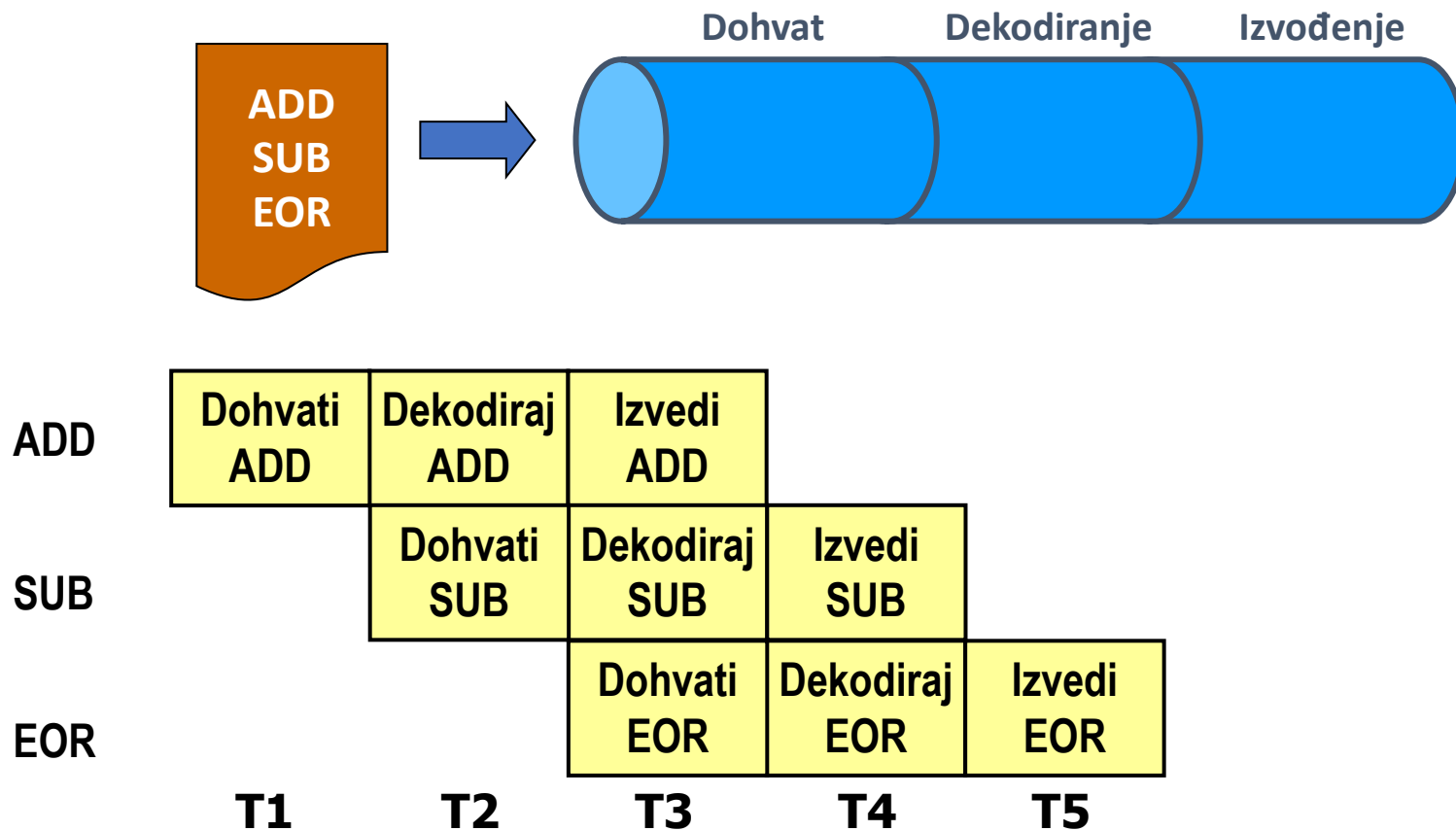
- Podijelimo li faze tako da su međusobno neovisne, možemo primijeniti načela pokretne trake, pa se sve tri faze mogu izvoditi istodobno:



- 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 !)

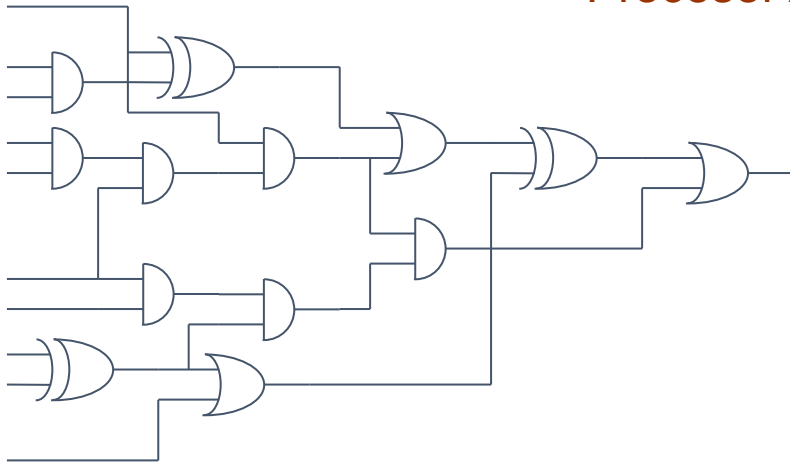


# Izvođenje naredaba i protočna struktura

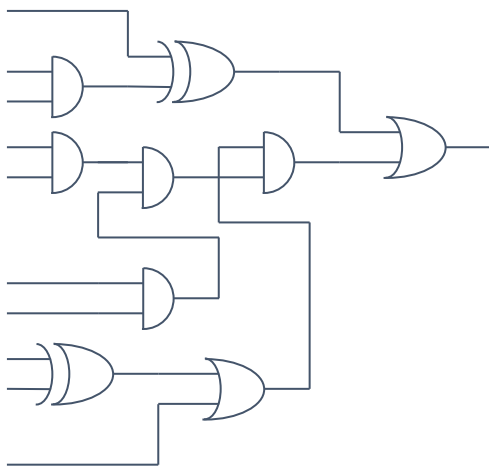
- Radi većeg ubrzanja, dobro bi bilo naredbu podijeliti na što više faza (tj. protočnu strukturu na što više razina)
- Brzina cijele protočne strukture ovisi o brzini najsporije razine, tj. najsporija razina predstavlja ograničenje (usko grlo)
- Zato se pokušavaju odabrati faze tako da svaka traje čim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

# Minimizacija logike ili promjene u izvedbi

Procesor A



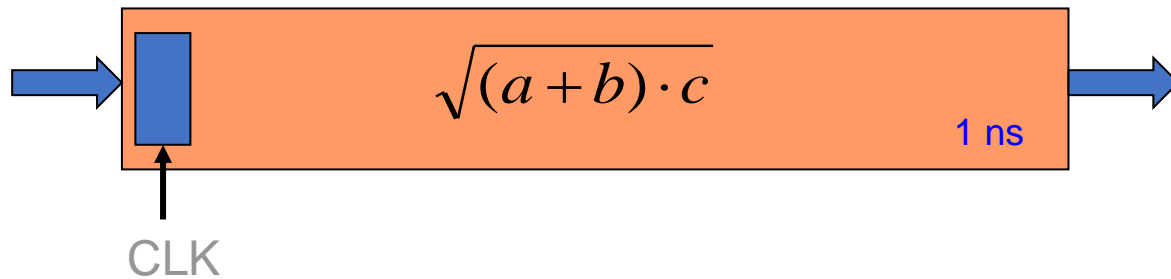
Procesor B



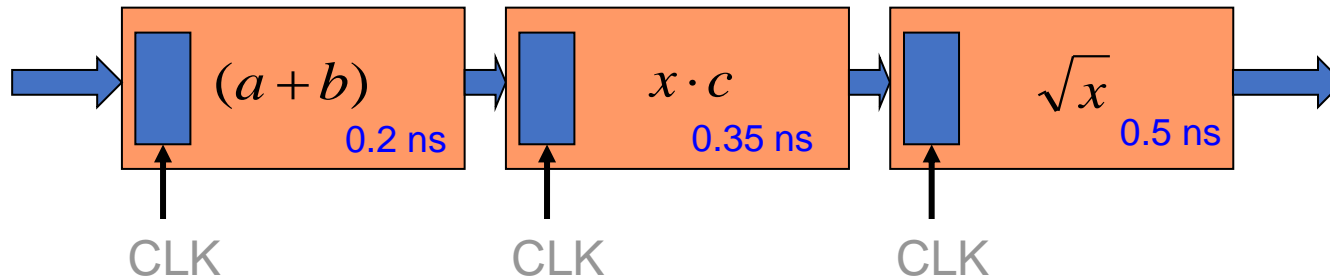
- Primjer:
  - Kašnjenje log. vrata = 0.1 ns
  - Kašnjenje za logički blok
    - Procesor A = 0.6 ns
    - Procesor B = 0.4 ns
  - Max. frekvencija
    - Procesor A = 1.66 GHz
    - Procesor B = 2.5 GHz
- Minimizacijom logike i promjenom izvedbe (npr. carry look-ahead zbrajala umjesto običnih) postižu se značajna ubrzanja bez mijenjanja tehnologije

# Podjela funkcijskih modula na manje cjeline

- Logička funkcija dijeli se na nekoliko dijelova
- Svaki dio mora se izvoditi neovisno (nema povratnih veza koje se koriste u istom vremenskom periodu !!!)
- Ovime se postiže mogućnost rada na brzini najsporijeg dijela (što je još uvijek znatno brže nego brzina rada originalnog sklopa za izvođenje cijele funkcije)



$$f_{\max} = 1 \text{ GHz}$$



$$f_{\max} = 2 \text{ GHz}$$

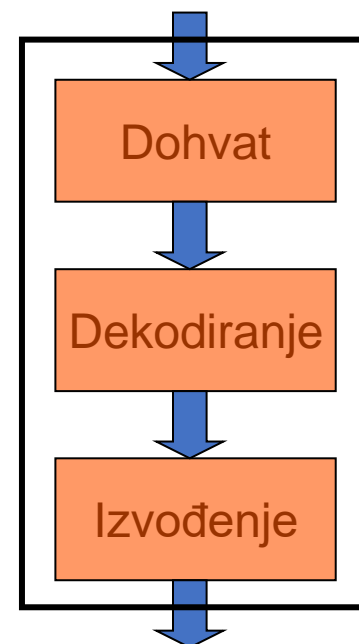
# Izvođenje naredaba i protočna struktura

- 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

# Protočná struktura procesora ARM

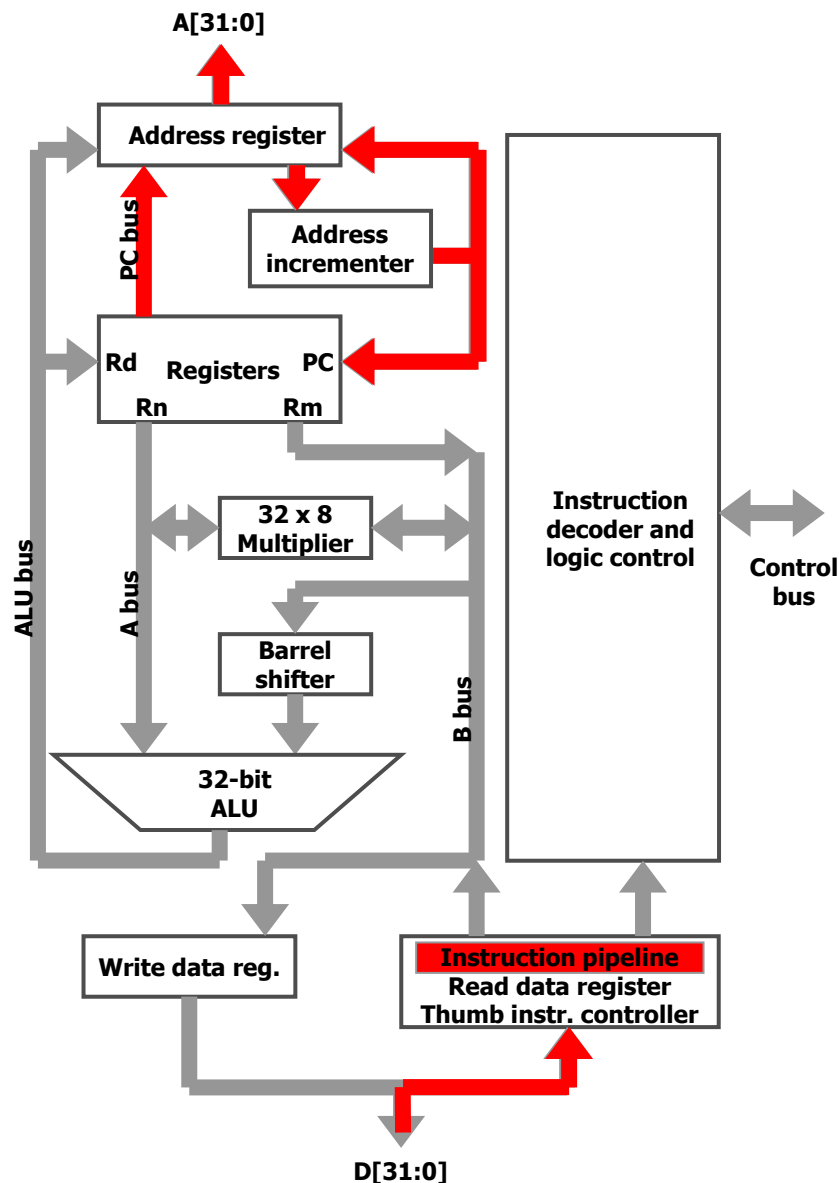
# Protočna struktura procesora ARM

- U okviru AR1R razmatrati ćemo jednostavnu protočnu strukturu Arm v7 koju karakterizira:
  - Protočna struktura za izvođenje naredaba sastavljena od **tri razine**
  - Jedinstveno memorijsko sučelje za naredbe i podatke (Von Neumann)
- Svaka naredba izvodi se u tri koraka:
  - u prvom koraku naredba se dohvaća
  - u drugom koraku naredba se dekodira
  - u trećem koraku naredba se izvodi
- Dok su prva dva koraka slični u svim naredbama, korak izvođenja može se sastojati od više operacija ovisnih o naredbi



# ARM protočna arhitektura: DOHVAT

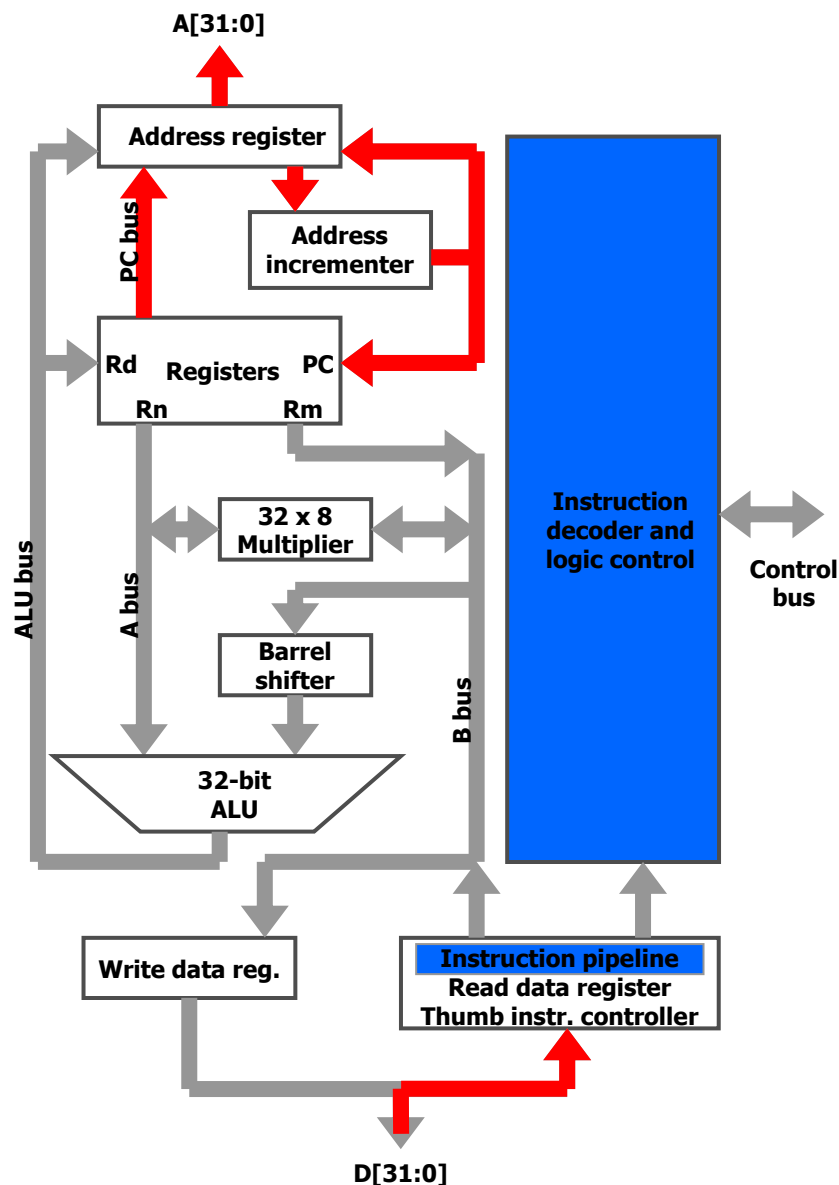
## Dohvat naredbe:



- Gornji dio procesora (na slici) koristi se za generiranje adrese iz PC-a i slanje te adrese na vanjsku adresnu sabirnicu
- Naredba se učitava iz memorije i šalje direktno u protočnu strukturu za učitane naredbe (instruction pipeline). Ova protočna struktura postoji da bi ARM mogao čitati dijelove strojnog koda naredbe sve do trenutka njenog izvođenja

# ARM protočna arhitektura : DEKODIRANJE

## Dekodiranje naredbe:



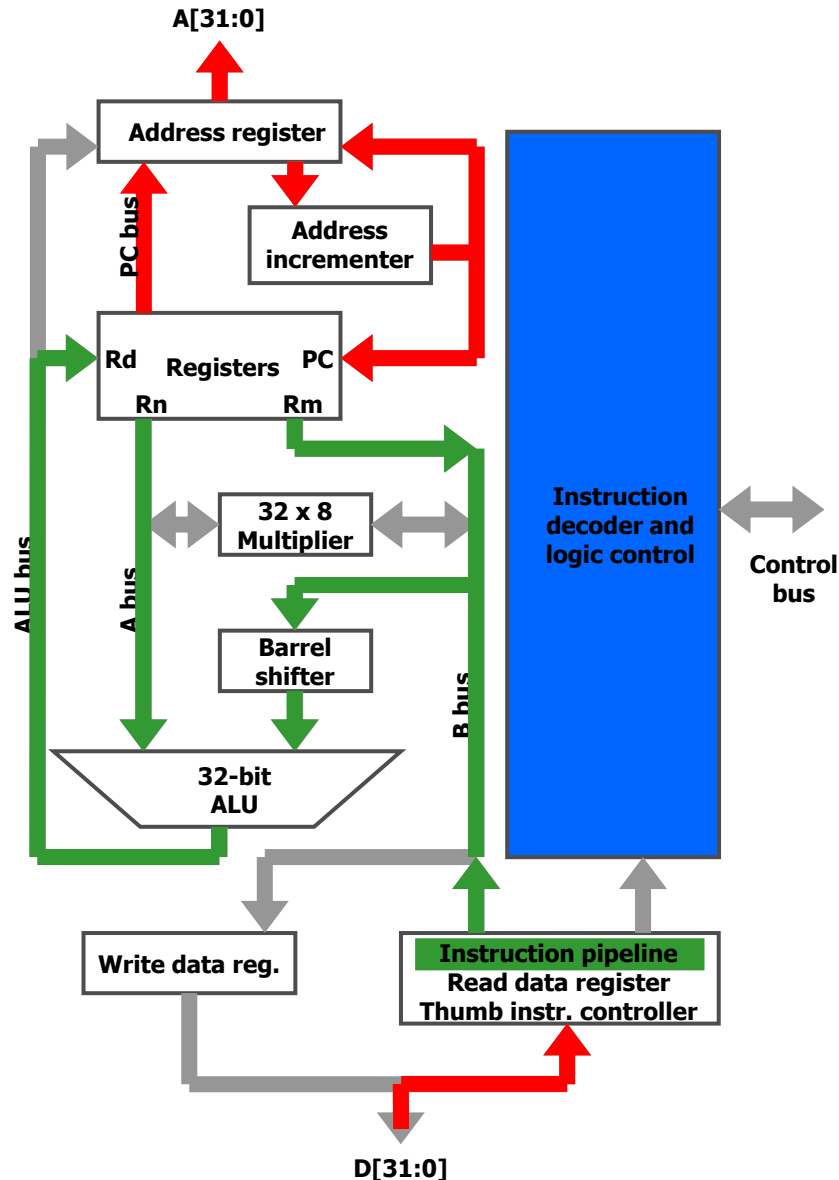
- Naredba pročitana u prethodnom periodu se dekodira
- Ovdje se može vidjeti da ARM tijekom **DEKODIRANJA** ne koristi dio za adresiranje niti dio za čitanje podatka iz memorije tako da se u isto vrijeme može izvoditi **DOHVAT** sljedeće naredbe



# ARM protočna arhitektura: IZVOĐENJE

- Izvođenje naredaba je RAZLIČITO od naredbe do naredbe
- Kod ARM7 postoje tri različite grupe naredaba s obzirom na način kako se naredbe izvode:
  - naredbe za obradu podataka
  - naredbe za prijenos podataka
  - naredbe za grananje
- Analizirat ćemo razinu izvođenja za svaku od gornjih grupa

# IZVOĐENJE: Naredbe za obradu podataka



## Izvođenje naredbe za obradu podataka:

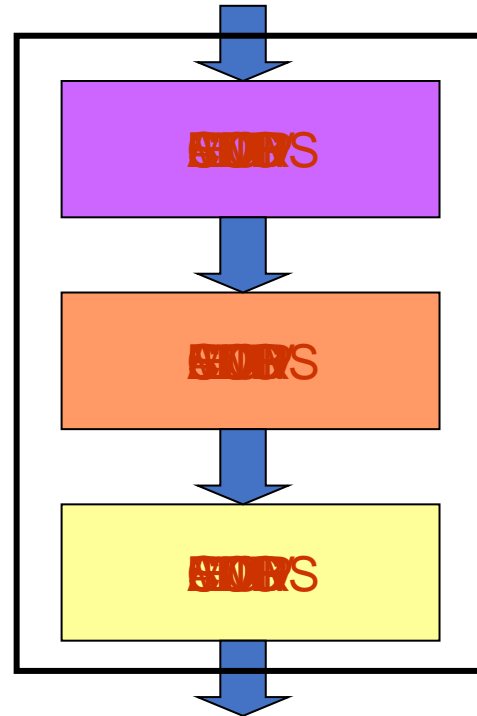
- Naredba dekodirana u prethodnom periodu se izvodi
- Vidimo da **IZVOĐENJE** ne utječe na dio procesora koji obavlja **DOHVAT** naredbe niti na dio procesora koji obavlja **DEKODIRANJE** naredbe te se dohvat, dekodiranje i izvođenje mogu paralelno izvoditi za tri naredbe

# IZVOĐENJE: Naredbe za obradu podataka

→ MOV R4, #25  
→ SUB R4, R4, #1  
→ CMP R4, #1  
→ EOR R3, R3, R2  
→ ADDS R4, R4, #1  
→ BIC R4, R4, #1

Vremenski period: 002

Izvršeno naredbi: 000



Dohvat

Dekodiranje

Izvođenje

- Iz ovog razmatranja vidimo da osnovne naredbe za obradu podataka u prosjeku trebaju samo jedan period za izvođenje

# IZVOĐENJE: Naredbe za obradu podataka

- Na temelju prethodnih razmatranja vidjeli smo da osnovne naredbe za obradu podataka optimalno koriste protočnu arhitekturu
  - izvođenje tih naredaba efektivno traje 1 period
  - (osim naredaba MUL i MLA koje trebaju više perioda - objasniti ćemo kasnije !)
- Međutim, programi se ne sastoje samo od naredaba za obradu podataka
- Kada se u programu pojave naredbe za prijenos podataka ili grananje, one će narušiti ovo efikasno korištenje puta podataka

- Do sada je objašnjeni rad protočne strukture za osnovne naredbe bio **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. LDR ili npr. STR 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 STR za spremanje podatka)
  - drugi pristup je dohvat strojnog kôda sljedeće naredbe

- 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.
  - mjehurić
- Kada uzrok hazarda nestane, procesor nastavlja izvoditi naredbe na normalan način.
- **Posljedica ovog hazarda kod arhitekture Arm7:**
  - Korak izvođenja naredbe STR traje 2 perioda
  - Korak izvođenja naredbe LDR traje 3 perioda
  - Naredbe LDM i STM kao i naredbe kojima se mijenja registar PC imaju još kompleksnije načine izvođenja

- Procesor rješava strukturni hazard tako da odgodi izvođenje naredbe STR za jedan period dok ne dohvati sljedeću naredbu (ali to vrijeme ipak koristi za računanje adrese podatka)
- **Naredba STR:**
  - Korak izvođenja traje 2 perioda
    - Period 1: Računanje adrese za spremanje podatka + dohvrat sljedeće naredbe koja će se kasnije izvesti
    - Period 2: Pisanje podatka u memoriju
- **Naredba STM:**
  - Korak izvođenja traje  $n+1$  period ( $n$  je broj registara koji se spremaju)
    - Period 1: Računanje adrese za spremanje podatka + dohvrat sljedeće naredbe koja će se kasnije izvesti
    - Period 2: Pisanje prvog podatka u memoriju
    - Period 3: Pisanje drugog podatka u memoriju
    - ...



# Objašnjenje izvođenja naredbe STR

Dohvat	Dekodiranje	Izvođenje
Dohvat ADD		
Dohvat STR	Dekodiranje ADD	
Dohvat SUB	Dekodiranje STR	Izvođenje ADD
Dohvat AND		Računanje adrese R6+2 za STR
	Dekodiranje SUB	Spremanje u memoriju STR
Dohvat EOR	Dekodiranje AND	Izvođenje SUB
	Dekodiranje EOR	Izvođenje AND
		Izvođenje EOR

```
ADD R0,R1,R2
STR R0,[R6+2]
SUB R5,R6,R7
AND R8,R9,R0
EOR R1,R2,R3
```

## Napomene:

- Za vrijeme računanja adrese može se pristupati memoriji pa se to koristi za dohvat slijedeće naredbe (ali se prethodna ne dekodira)
- Za vrijeme spremanja u memoriju ne može se dohvaćati iduća naredba

- Naredbe za čitanje podataka (LDR i LDM) zahtjevaju dva dodatna perioda za izvođenje:
  - računanje adrese (kao kod STR i STM)
  - podatak mora proći kroz barrel shifter kako bi bio pravilno smješten u registar (npr. kad se čita BAJT ili POLURIJEČ koji nisu pozicionirani na najnižem dijelu riječi) što zahtjeva dodatno vrijeme (1 period)
- Zbog toga korak izvođenja za naredbe LDR/LDM traje jedan period više nego za naredbe STR/STM
- DZ: proučite korake izvođenja naredaba LDR/LDM iz knjige.

# 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 su upravljačke naredbe

# IZVOĐENJE: naredbe grananja-uvjet nije zadovoljen

CMP R0,R1

BEQ X

ADD R3,R4,R5

MOV R6,R7

...

	Dohvat	Dekodiranje	Izvođenje
t	CMP		
t+1	BEQ	CMP	
t+2	ADD	BEQ	CMP
t+3	MOV	ADD	BEQ (Z=0)
t+4	...	MOV	ADD

} "Grananje"  
traje 1  
period

X SUB R1,R2,R3

MVN R4, R3

EOR R3,R4,R3

...

- Ako uvjet za grananje **nije zadovoljen**, procesor nastavlja izvoditi program
- U tom slučaju naredba grananja efektivno traje jedan period (u stvari, procesor troši taj period ne izvodeći ništa) i protočna struktura radi efikasno

# IZVOĐENJE: naredbe grananja-uvjet je zadovoljen

CMP R0,R1

BEQ X

ADD R3,R4,R5

MOV R6,R7




....

X SUB R1,R2,R3

MVN R4, R3

EOR R3,R4,R3

...

	Dohvat	Dekodiranje	Izvođenje
t	CMP		
t+1	BEQ	CMP	
t+2	ADD	BEQ	CMP
t+3	<del>MOV</del>	<del>ADD</del>	BEQ
t+4	SUB		
t+5	MVN	SUB	
t+6	EOR	MVN	SUB

Grananje traje  
3 perioda

- Ako je uvjet za grananje **zadovoljen**, procesor mora zanemariti naredbe do tada dohvaćene u protočnu strukturu (tzv. pipeline flush) i mora započeti dohvat novih naredaba s adrese skoka
- Zbog toga naredba grananja efektivno traje dodatna 2 perioda (ukupno 3) dok sljedeća naredba ne stigne u razinu Izvođenja
- Dodatno kašnjenje naziva se *branch penalty*

- Upravljački hazard značajno utječe na performanse procesora
- Postoji metoda pisanja programa kojom možemo izbjeći ovaj hazard: uvjetno izvođenje naredaba
- Oznake vezane za primjer na sljedećoj strani:
  - F** (False, uvjet nije zadovoljen)
  - T** (True, uvjet zadovoljen)
- Periodi:
  - S** (sljedni): čita se podatak sa sljedeće adrese (kada ćemo govoriti o memoriji vidjeti ćemo da je ovakav dohvat podatka brz)
  - N** (nesljedni): čita se podatak sa adrese koja nije sljedna. Ovakav dohvat podatka je sporiji jer obično zahtjeva podatak koji nije u priručnoj memoriji (engl. cache)

# Primjer: Izbjegavanje grananja uporabom uvjetnog izvođenja

## Izvođenje odsječka (3 MOV naredbe) korištenjem uvjetnog grananja:

<i>CMP R0, #0</i>	1S
<i>BNE DALJE</i>	F:(1S) T:(1N+2S)
<i>MOV R1, #1; uvjetni dio koda</i>	1S
<i>MOV R2, #2; uvjetni dio koda</i>	1S
<i>MOV R3, #3; uvjetni dio koda</i>	1S

Ukupno trajanje:  
F: 5S  
T: 1N + 3S

DALJE ...

## Izvođenje istog odsječka korištenjem uvjetnog izvođenja naredaba:

<i>CMP R0, #0</i>	1S
<i>MOVEQ R1, #1; uvjetni dio koda</i>	1S
<i>MOVEQ R2, #2; uvjetni dio koda</i>	1S
<i>MOVEQ R3, #3; uvjetni dio koda</i>	1S

Ukupno trajanje:  
F: 4S  
T: 4S

## Zaključak:

- Za vrlo kratke odsječke uvjetno izvođenje je brže bez obzira na statistiku uvjeta (tj. koliko puta je uvjet istinit, a koliko puta je lažan)
- Za dulje odsječke ili za odsječke s memorijskim naredbama uvjetno grananje je brže

# Sabirnički periodi na AHB (samo za informaciju)

nMREQ	SEQ	Tip perioda	Opis
0	0	N-period	Neslijedni period (engl. Nonsequential)
0	1	S-period	Slijedni period (engl. Sequential)
1	0	I-period	Interni period (engl. Internal)
1	1	C-period	Period prijenosa sadržaja koprocesorskog registra (engl. Coprocessor register transfer)

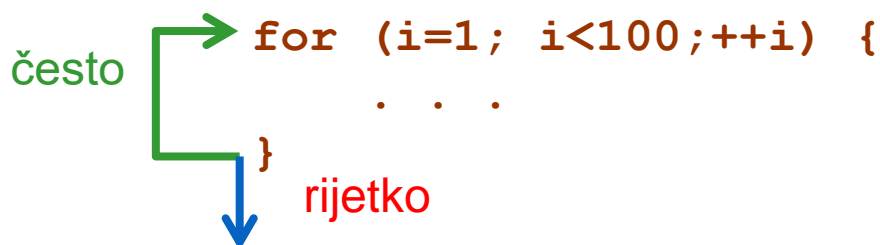


# Predviđanje grananja

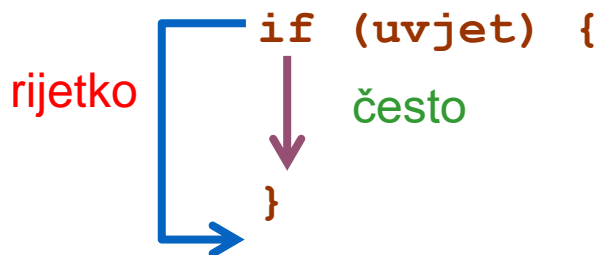
- Neke arhitekture procesora uvode **predviđanje grananja**
  - Načelo predviđanja grananja:
    - Na neki način predvidjeti da li će uvjet grananja biti zadovoljen ili ne
    - Na temelju rezultata predviđanja:
      - ako je grananje vjerojatno: dohvaćati naredbe sa adrese za grananje
      - ako grananje nije vjerojatno: dohvaćati naredbe slijedno
  - Ovime se značajno smanjuje broj slučajeva kad procesor mora prazniti protočnu strukturu
  - Postoje dvije osnovne metode predviđanja:
    - statičko
    - dinamičko
- te kombinacija obje metode

# Statičko predviđanje grananja

- Primjer jednostavnog statičkog predviđanja
  - Ako je **adresa grananja manja od PC**, pretpostavlja se da će doći do grananja pa se dohvaća naredba s adrese grananja



- Ako je **adresa grananja veća od PC**, pretpostavlja se da neće doći do grananja pa se naredbe nastavljaju dohvaćati slijedno



- Jednostavno za izvedbu, a poboljšava performanse

# Dinamičko predviđanje grananja

- Dinamičko predviđanje puno je složenije od statičkog i zasniva se na načelu vođenja statistike o tome da li je pojedino grananje bilo ostvareno ili nije
- Statistika se vodi korištenjem male memorijske tablice izvedene u procesoru u kojoj se zapisuje da li je pojedina naredba grananja zadnji puta bila izvedena ili nije
  - Pretpostavlja se da će se naredba ponašati jednako kao i kad je zadnji puta bila izvedena (tj. da će se skok izvesti odnosno da se neće izvesti)
- Tablica se adresira na temelju nekoliko najnižih bitova adrese naredbe grananja (moguće je da se time ponekad dobiju podaci o krivoj naredbi koja ima iste niže bitove adrese, ali to puno ne utječe na efikasnost predviđanja)
- Kad učitava naredbu grananja, procesor će na temelju stanja u tablici dohvatiti ili naredbe koje slijede ili naredbe s adrese skoka

# 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 Arm7 nema pojave podatkovnog hazarda pa ćemo primjer ovog hazarda proučiti kasnije

- Još neke situacije kad se protočna struktura ne koristi efikasno a ne spadaju u hazarde

# Izvođenje naredaba MUL, MLA

- Podsjetnik:
  - $MUL\{cond\}\{S\} Rd, Rm, Rs$
  - $MLA\{cond\}\{S\} Rd, Rm, Rs, Rn$
- Množenje je kompleksna operacija koja zahtjeva više perioda za izvođenje
- Kako bi se ubrzalo množenje Arm7 koristi specijalno 32x8 množilo koje izvodi množenje jedne riječi i 1 bajta druge riječi
- Broj potrebnih perioda za izvođenje MUL/MLA naredbe može se izračunati formulom:
  - Za MUL:  $1+m$
  - Za MLA:  $2+m$
  - Gdje je
    - $m=1$  ako su bitovi  $Rs[32:8]$  svi nule ili svi jedinice
    - $m=2$  ako su bitovi  $Rs[32:16]$  svi nule ili svi jedinice
    - $m=3$  ako su bitovi  $Rs[32:24]$  svi nule ili svi jedinice
    - $m=4$  inače

# Izvođenje naredaba MUL, MLA

- 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
- Primjer izvođenja naredbe MUL

R2=0x00123456

m = 3

trajanje m+1=4







ADD ...

MUL R1,R1,R2

SUB ...

AND ...

EOR ...

	Dohvat	Dekodiranje	Izvođenje
t	ADD		
t+1	MUL	ADD	
t+2	SUB	MUL	ADD
t+3	AND		MUL
t+4			MUL
t+5			MUL
t+6		SUB	MUL
t+7	EOR	AND	SUB