

Protočne arhitekture procesora

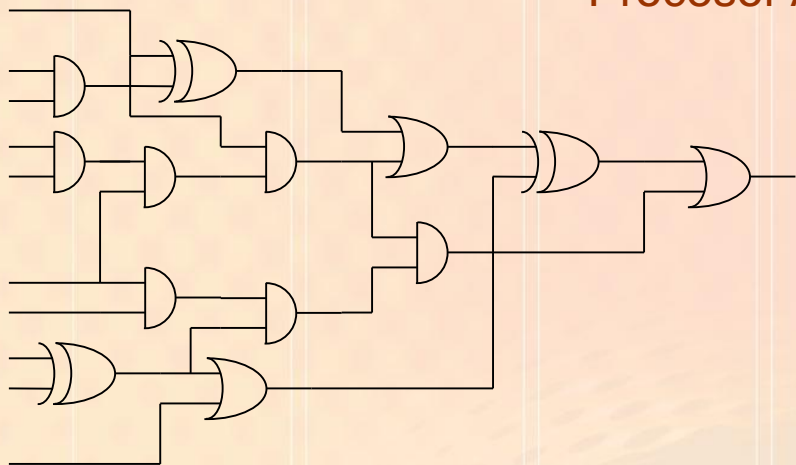


Arhitektura protočne strukture za izvođenje naredaba

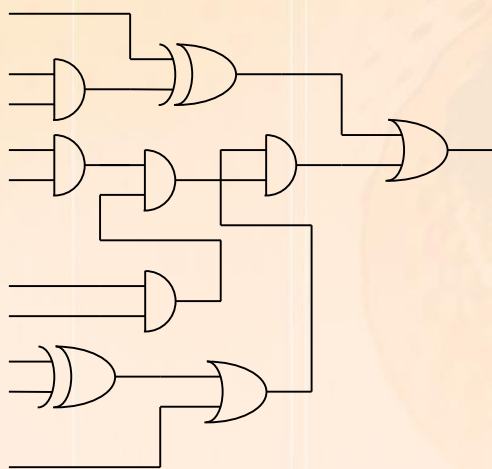
- Neka osnovna načela za ubrzanje rada procesora uz pretpostavku da se brzina signala vremenskog vođenja i tehnologija (proces proizvodnje) ne mijenjaju:
 - Pojednostavljenje naredbe (načelo RISC procesora)
 - Jednostavnije naredbe trebaju manje logike za dekodiranje i izvođenje
 - Podjela sklopovlja za izvođenje naredbe na dijelove koji se mogu izvoditi u isto vrijeme

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

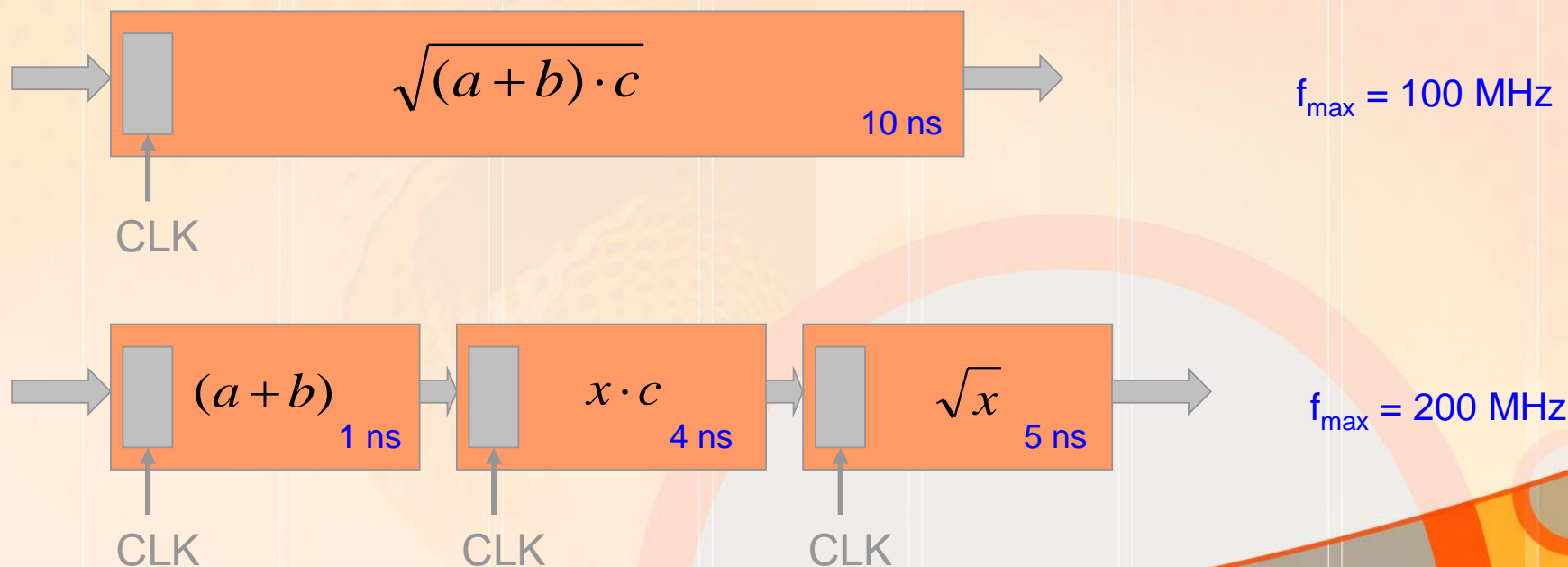
- Procesor A = 1.66 GHz

- Procesor B = 2.5 GHz

- Minimizacijom logike i promjenom izvedbe (npr. carry look-ahead zbrajala umjesto običnih) postižu se značajna ubrzanja bez mijenjanja tehnologije

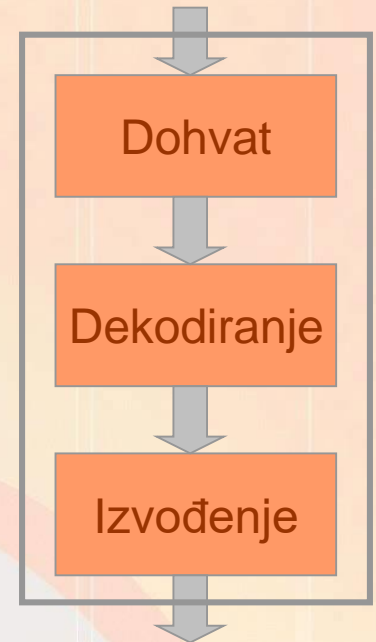
Podjela funkcijskih modula na manje cjeline

- 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)

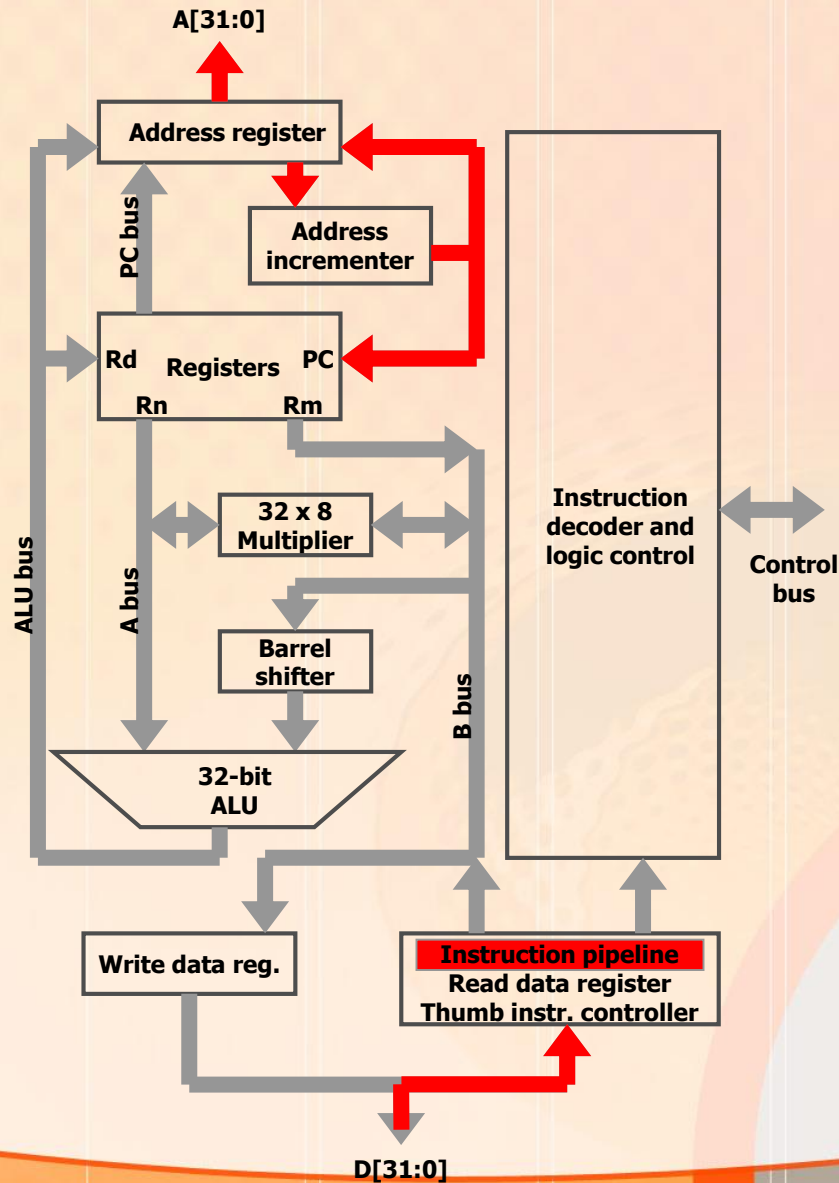


Protočna struktura procesora ARM

- Porodicu ARM7 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



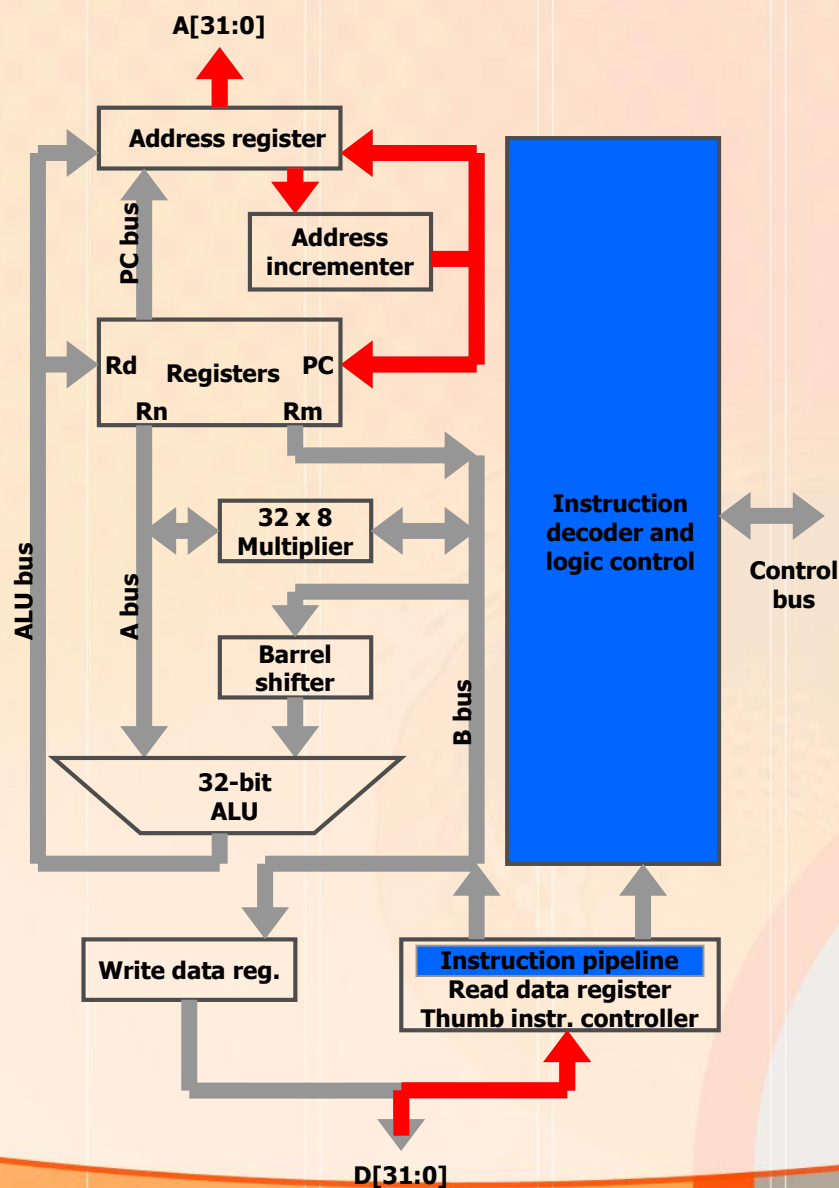
ARM protočna arhitektura: DOHVAT



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 (donji dio na slici). Ova protočna struktura postoji da bi ARM mogao čitati dijelove strojnog koda naredbe sve do trenutka njenog izvođenja

ARM protočna arhitektura : DEKODIRANJE



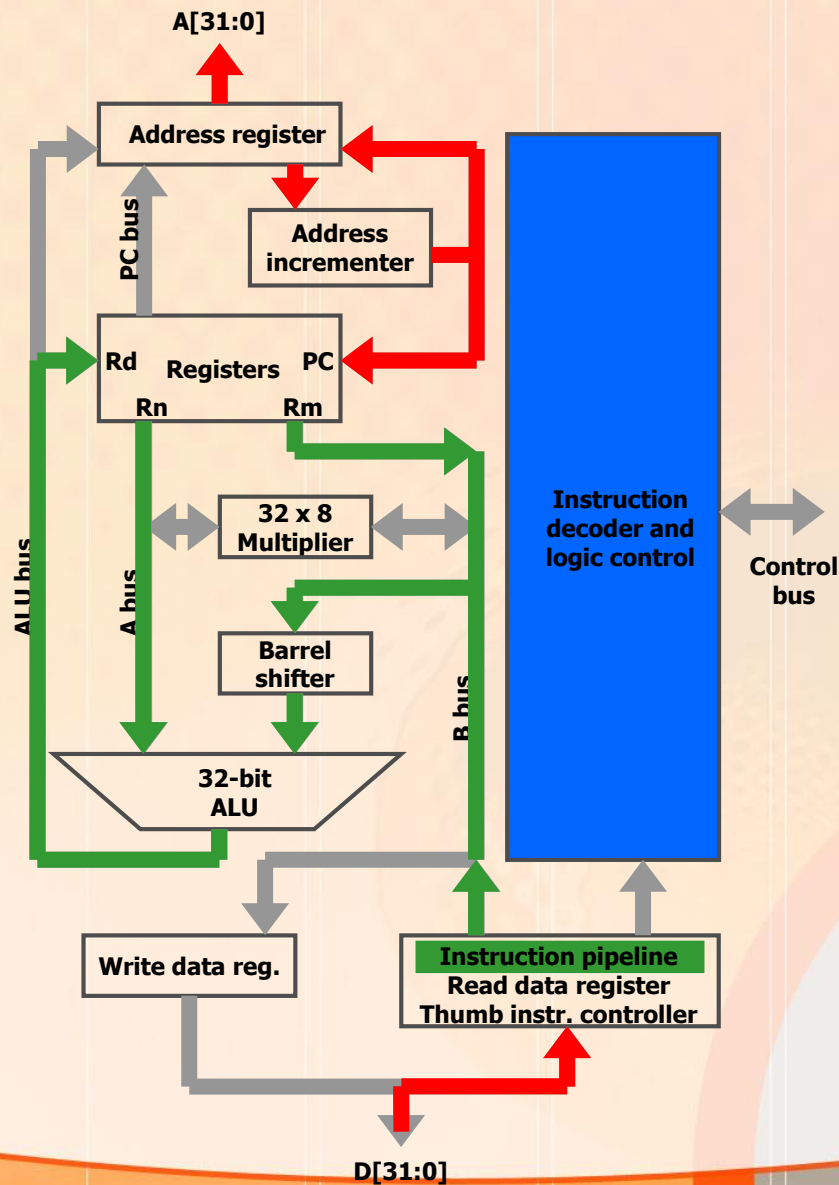
Dekodiranje naredbe:

- 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:

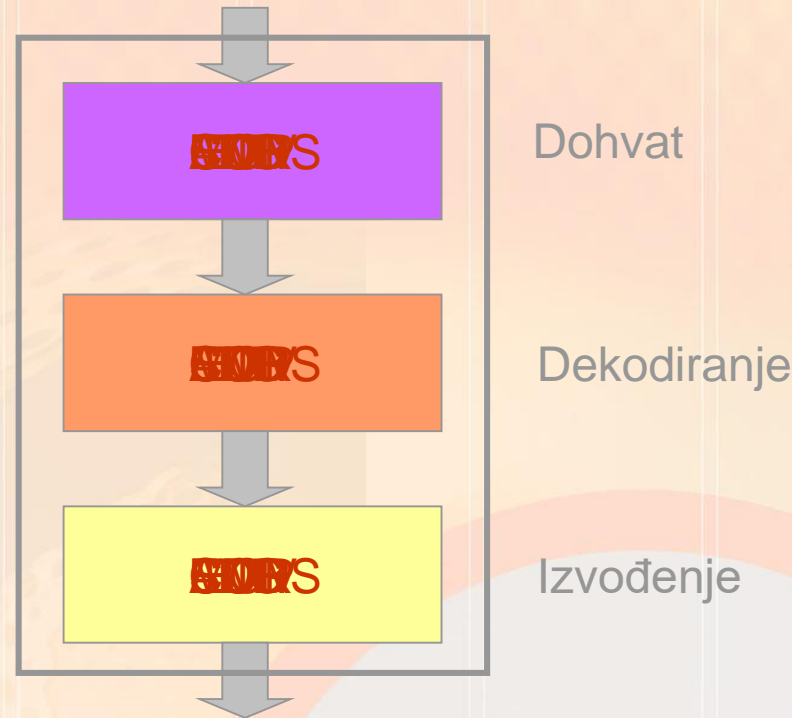
- Naredba dekodirana u prethodnom periodu se izvodi
- Vidimo da **IZVOĐENJE** ne utječe na dio procesora koji obavlja **DOHVAT** naredbe niti na dio procesora koji obavlja **DEKODIRANJE** naredbe te se dohvat, dekodiranje i izvođenje mogu paralelno izvoditi za tri naredbe

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: 0000



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

IZVOĐENJE: Naredbe za obradu podataka

- Na temelju prethodnih razmatranja vidjeli smo da naredbe za obradu podataka optimalno koriste protočnu arhitekturu
- 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



IZVOĐENJE: Naredbe za prijenos podataka

- Naredbe za prijenos podataka (LDR, STR, LDM, STM, ...) trebaju komunicirati s memorijom te time zauzimaju memorijsku sabirnicu
- Zbog Von Neumannove arhitekture, procesor ne može u istom trenutku čitati sljedeću naredbu (ciklus dohvata) i prenositi neki podatak u/iz memorije (ciklus izvođenja)
- **Strukturni hazard**
- **Posljedica ovog hazarda:**
 - 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

Izvođenje naredaba za prijenos podataka: STR, STM

- 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 + dohvat 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 + dohvat 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

Izvođenje naredaba za prijenos podataka: LDR, LDM

- 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.

IZVOĐENJE: Naredbe grananja

- Pojava naredbe za grananje može pri izvođenju uzrokovati dodatne periode tijekom kojih procesor ne koristi efikasno protočnu strukturu
 - Ako je grananje bezuvjetno ili ako je uvjet grananja zadovoljen, tada **nisu valjane** naredbe koje su unaprijed dohvaćene u protočnu strukturu procesora. Umjesto njih, procesor treba dohvatiti naredbu sa adrese skoka. Za ovu operaciju procesoru trebaju dodatni vremenski periodi.
 - Ako uvjet grananja nije zadovoljen, tada **su valjane** naredbe koje su unaprijed dohvaćene u protočnu strukturu procesora
- **Upravljački hazard**

IZVOĐENJE: naredbe grananja-uvjet nije 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	MOV	ADD	BEQ (Z=0)
t+4	...	MOV	ADD

} "Grananje"
traje 1 period

- 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	MOV	ADD	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*



Izbjegavanje upravljačkog hazarda - uvjetno izvođenje

- 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</i>	<i>R0, #0</i>	<i>1S</i>	}	Ukupno trajanje: F: 5S T: 1N + 3S
<i>BNE</i>	<i>DALJE</i>	<i>F:(1S) T:(1N+2S)</i>		
<i>MOV</i>	<i>R1, #1; uvjetni dio koda</i>	<i>1S</i>		
<i>MOV</i>	<i>R2, #2; uvjetni dio koda</i>	<i>1S</i>		
<i>MOV</i>	<i>R3, #3; uvjetni dio koda</i>	<i>1S</i>		

DALJE ...

Izvođenje istog odsječka korištenjem uvjetnog izvođenja naredaba:

<i>CMP</i>	<i>R0, #0</i>	<i>1S</i>	}	Ukupno trajanje: F: 4S T: 4S
<i>MOVEQ</i>	<i>R1, #1; uvjetni dio koda</i>	<i>1S</i>		
<i>MOVEQ</i>	<i>R2, #2; uvjetni dio koda</i>	<i>1S</i>		
<i>MOVEQ</i>	<i>R3, #3; uvjetni dio koda</i>	<i>1S</i>		

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

Izvođenje naredaba kod procesora ARM7

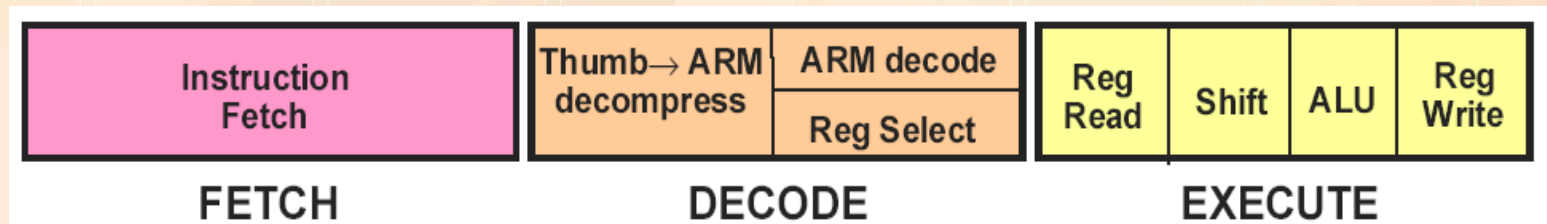
- Jednostavna protočna struktura s tri razine te Von Neumannova arhitektura dovode do određenih ograničenja u efikasnom iskorištenju procesora
- Analizom ranije navedenih problema arhitektura se može promijeniti kako bi se postigle veće performanse procesora
- Kao što ćemo vidjeti to dovodi do kompleksnijih (a time i skupljih) izvedbi protočnih struktura

Primjeri unaprjeđenja protočnih struktura ARM-a

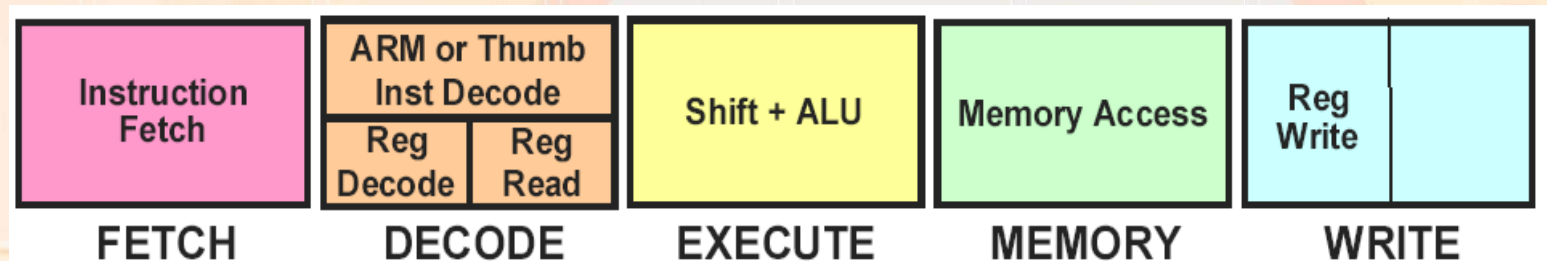
Primjeri protočnih arhitektura ARM-a

- S obzirom da čitanje/pisanje podataka u memoriju predstavlja čestu naredbu, strukturni hazard kod ARM7 znatno smanjuje efikasnost protočne strukture
- Zbog povećanja efikasnosti kod arhitekture ARM9 uvodi se:
 - Harvardska arhitektura memorijskog pristupa
 - razina izvođenja razdvaja se u 3 nove protočne razine

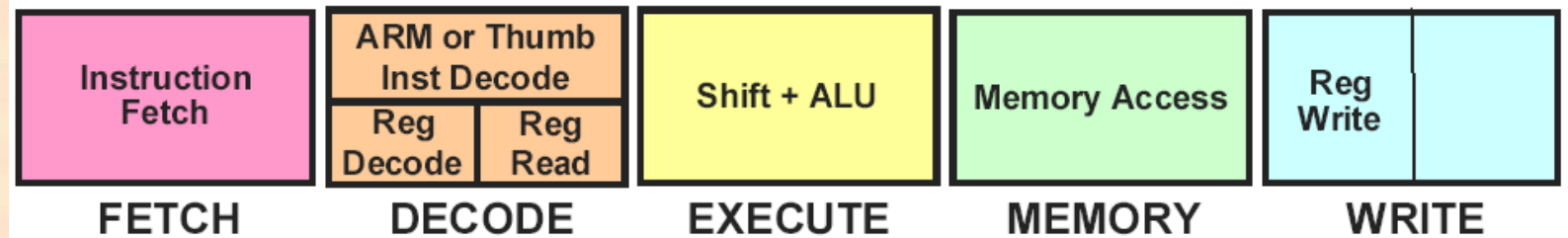
• ARM7



• ARM9

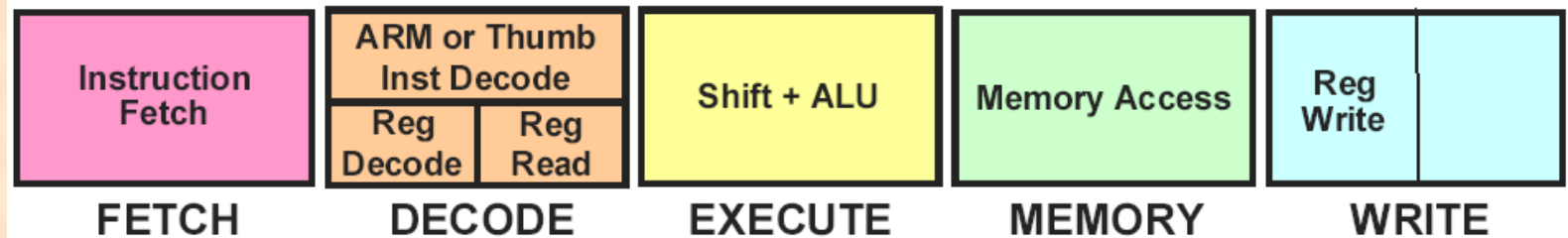


Protočne arhitekture: ARM9



- 1. razina je gotovo identična kao i u ARM7
- U 2. razini dekodira se naredba, ali se istodobno dohvaćaju operandi iz registara
- U 3. razini izvode se sve aritmetičko-logičke naredbe, ali se rezultat ne sprema u registar
- U 4. razini naredbe LD i ST čitaju ili spremaju podatak u memoriju
- U 5. razini se u registar sprema rezultat aritmetičko-logičke naredbe ili podatak pročitani iz memorije
- S obzirom da se dohvat naredbe i čitanje/pisanje podatka izvode na odvojenim sabirnicama, **nema strukturnog hazarda** te procesor tijekom faze izvođenja naredaba LD/ST normalno dohvaća naredbu iz programske memorije

ARM9 - Podatkovni hazard

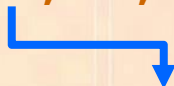


- Kako se rezultati ne spremaju u istom periodu u kojem se obavlja AL-operacija postavlja se pitanje: kako procesor može izvesti naredbe koje trebaju rezultat prethodne naredbe?
- Pri čitanju podatka iz memorije procesor također ne sprema podatak u registar u istom periodu te se opet može postaviti pitanje: kako procesor može izvesti naredbu ako ona treba podatak koji je iz memorije pročitala prethodna naredba?
- **PODATKOVNI HAZARD !!!**
- Analizirat ćemo oba slučaja...

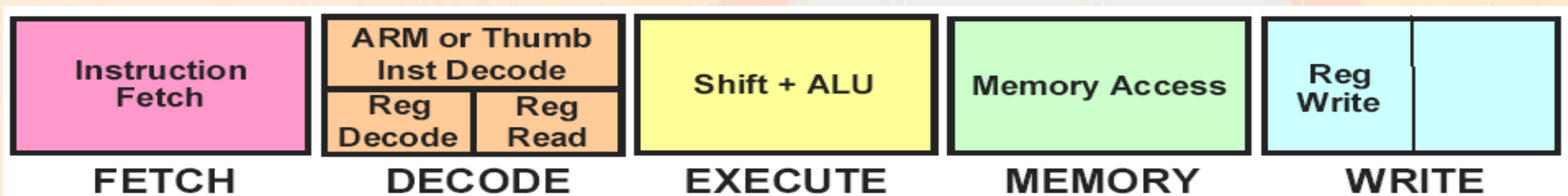
ARM9 - Podatkovni hazard

Razmotrimo **prvi slučaj** koji nastaje prilikom izvođenja naredbe koja ima **podatkovnu ovisnost o prethodnoj naredbi za obradu podataka**:

ADD R0, R1, R2



SUB R4, R6, R0 ; ova naredba ovisi o rezultatu prethodne naredbe



ARM9 - Podatkovni hazard

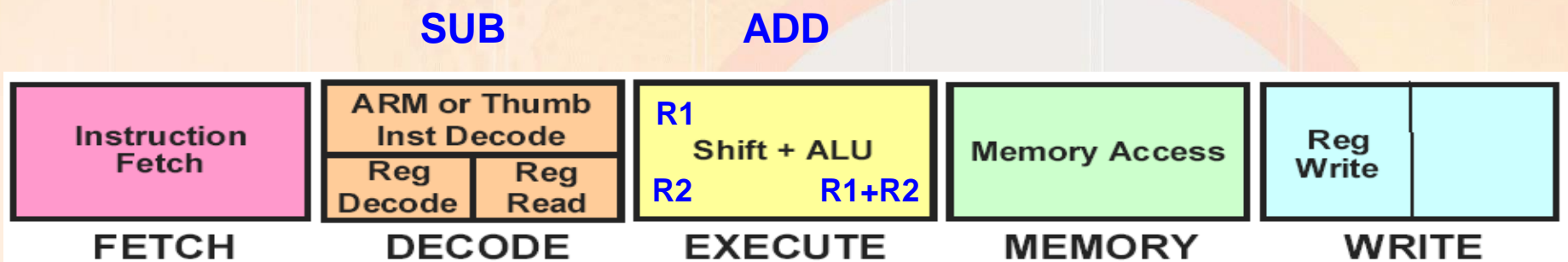
ADD R0, R1, R2

SUB R4, R6, R0

Počnimo analizu u trenutku t kad se naredba ADD nalazi u razini EXECUTE, a naredba SUB se nalazi u razini DECODE:

t : ADD - zbrajaju se podaci R1 i R2 i na kraju perioda rezultat se pamti u pomoćnom registru (koji programer ne vidi)

SUB - dekodira se i selektiraju se registri kao priprema za izvođenje



ARM9 - Podatkovni hazard

ADD R0, R1, R2

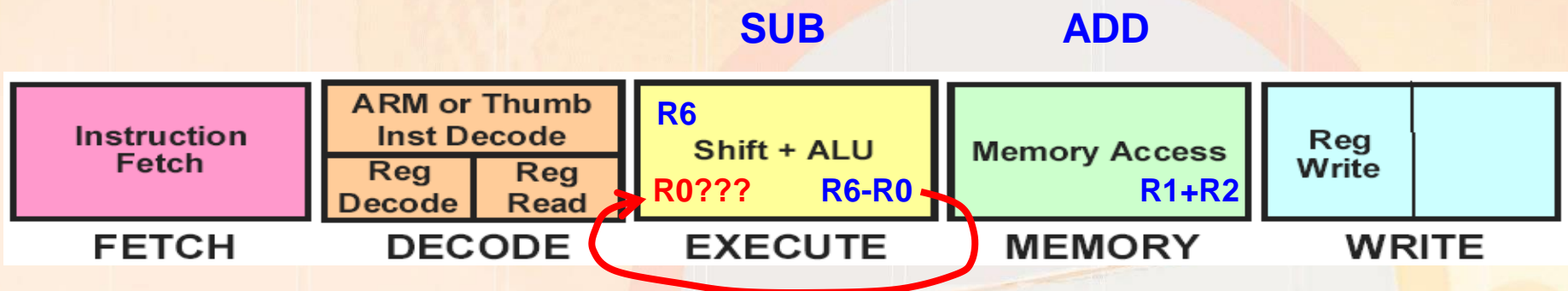
SUB R4, R6, R0

$t+1$: ADD - prolazi kroz razinu MEMORY, ne dešava se ništa

SUB - oduzimaju se podaci R6 i R0. **Kako kad R0 nije valjan ???**

Trebalo bi pričekati da se R1+R2 spremi u R0 (u razini WRITE)

Da ne bi došlo do podatkovnog hazarda (i dodatnih perioda čekanja), u arhitekturi je izvedeno prosljeđivanje rezultata (engl. **result forwarding**, prikazano strelicom na slici) tako da se rezultati iz ALU mogu koristiti odmah u sljedećoj naredbi



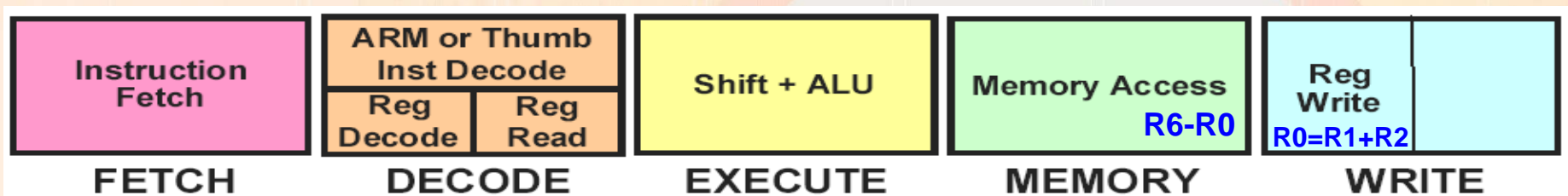
ARM9 - Podatkovni hazard

ADD R0, R1, R2

SUB R4, R6, R0 ; ova naredba ovisi o rezultatu prethodne naredbe

t+2: ADD - rezultat se sprema u registar R0

SUB - prolazi kroz razinu MEMORY, ne dešava se ništa



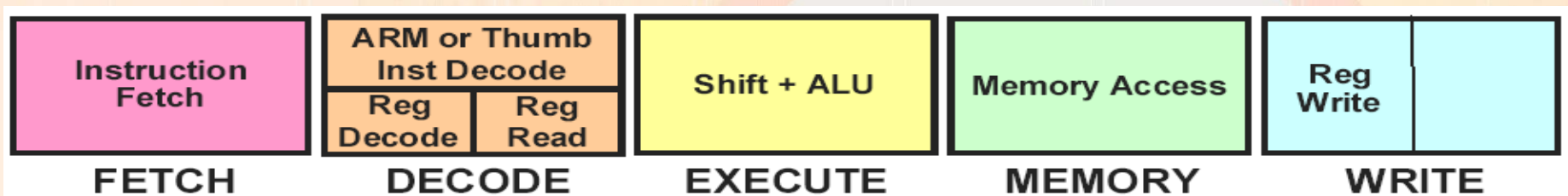
ARM9 - Podatkovni hazard

Razmotrimo **drugi slučaj** koji nastaje prilikom izvođenja naredbe koja **podatkovno ovisi o prethodnoj naredbi za prijenos podataka**:

LDR R0, [R1,#4]

ADD R2, R3, R0

; ova naredba ovisi o rezultatu prethodne naredbe



ARM9 - Podatkovni hazard

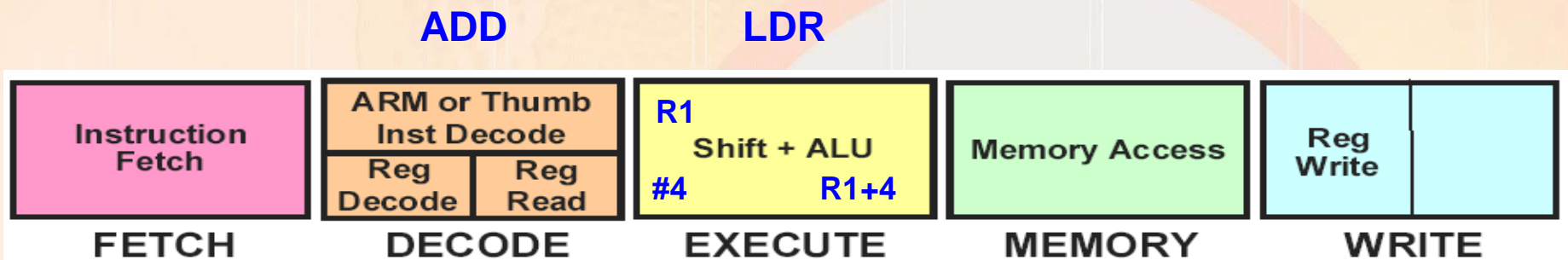
LDR R0, [R1,#4]

ADD R2, R3, R0

Počnimo analizu u trenutku t kad se naredba LDR nalazi u razini EXECUTE, a naredba ADD se nalazi u razini DECODE

t : LDR - u ovoj razini računa se adresa kojoj se želi pristupiti: $R1+4$

ADD - naredba se dekodira



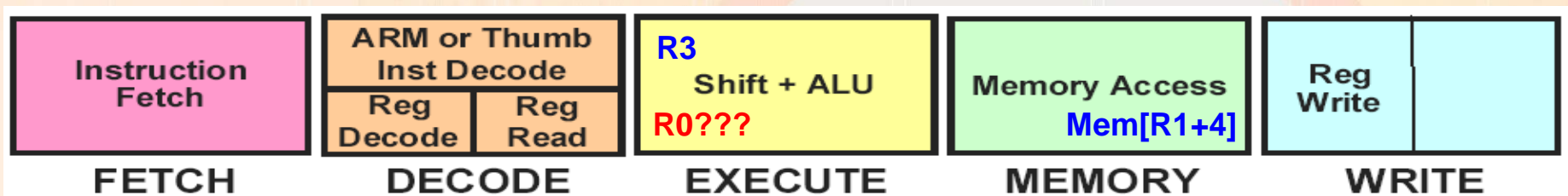
ARM9 - Podatkovni hazard

LDR R0, [R1,#4]

ADD R2, R3, R0

t+1 : LDR - prolazi kroz razinu MEMORY tijekom koje se učitava podatak s adrese [R1+4]. Tek se na kraju perioda podatak s adrese [R1+4] učitava u procesor.

ADD - trebalo bi zbrojiti podatke R3 i R0, no R0 na početku perioda nije valjan! ALU ne može obaviti operaciju. Ovakav podatkovni hazard u ARM-u zovu *interlock*. ADD mora čekati na operand pa procesor **ubacuje mjehurić** (vidi sliku na sljedećem slajdu).



ARM9 - Podatkovni hazard

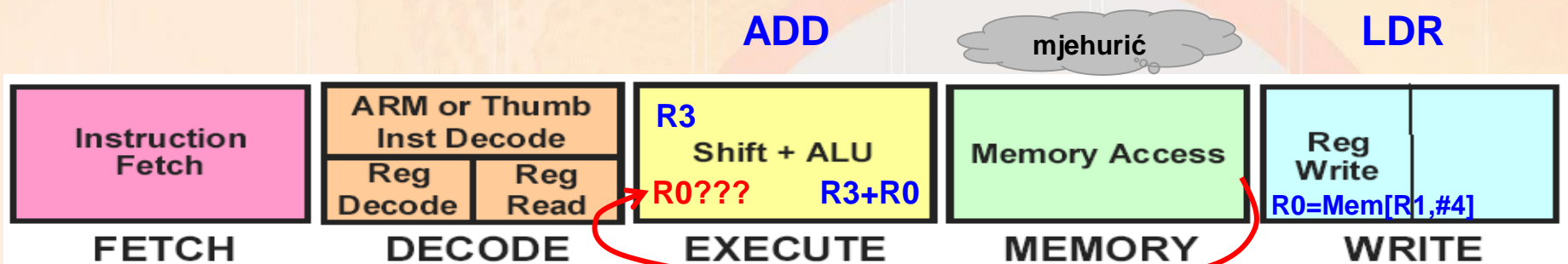
LDR R0, [R1,#4]

ADD R2, R3, R0

$t+2$: LDR - podatak pročitani iz memorije zapisuje se u registar R0

ADD - Da ne bi ponovno došlo do podatkovnog hazarda, jer na početku perioda podatak u R0 još uvijek nije valjan (pa bi došlo do još jednog perioda čekanja) u arhitekturi je izvedeno **prosljeđivanje rezultata** i s razine MEMORY: prikazano strelicom na slici) tako da se rezultati pročitani iz memorije mogu koristiti odmah nakon što su dovedeni u procesor.

Na taj način naredba ADD već u ovom periodu može izvesti zbrajanje.



$\text{Mem}[R1+4]$

Protočne arhitekture: ARM7 i ARM9

- Na sličan način kao ranije mogu se analizirati i druge situacije koje dovode do podatkovnog hazarda
- U sljedećoj tablici dana je kratka usporedba efikasnosti izvođenja kritičnih naredaba kod arhitektura ARM7 i ARM9:

Naredba	ARM7 Periodi izvođenja	ARM9 Periodi izvođenja + [mogući dodatni periodi zbog podatkovnog hazarda]
LDR	3	1 + [0 ili 1]
LDRH, LDRB, LDRSH, LDRSB	3	1 + [0 do 2] (mogući dodatni periodi: 1 zbog pod. hazarda, 1 zbog potrebe za rotacijom podatka)
STR, STRB, STRH	2	1

- Ubrzanja ARM9 u odnosu na ARM7:
 - frekvencija: ubrzanje 1.8 – 2.2 puta (manja kompleksnost razina)
 - broj perioda potrebnih programu: smanjenje za otprilike 30%
 - gornja dva faktora **kumulativno** doprinose ubrzanju arhitekture ARM9

Predviđanje grananja

- Problem koji smo spomenuli na ranijim predavanjima, nazvan upravljački hazard, javlja se npr. kod izvođenja naredbi grananja
- Ovaj primjer imali smo na arhitekturi ARM7...

Predviđanje grananja (podsjetnik)

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	MOV	ADD	BEQ
t+4	SUB		
t+5	MVN	SUB	
t+6	EOR	MVN	SUB

- Ako je uvjet za grananje zadovoljen, procesor mora zanemariti naredbe do tada učitane u protočnu strukturu i započeti dohvat nove naredbe. Zbog toga naredba grananja kod ARM7 efektivno traje dodatna 2 perioda (dok sljedeća naredba ne stigne u razinu Izvođenja)

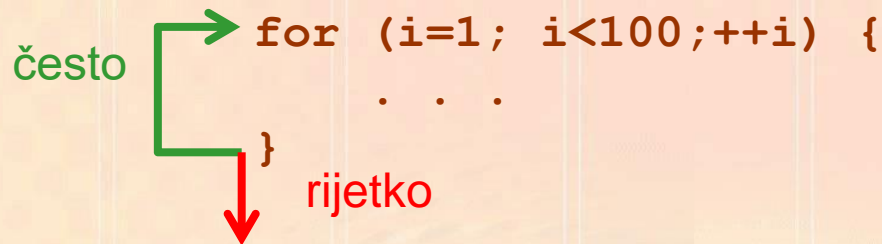
- Da li se to može izbjeći ???**

Predviđanje grananja

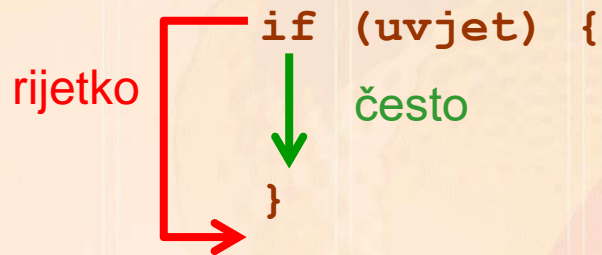
- Neke arhitekture procesora uvode **predviđanje grananja**
- Načelo predviđanja grananja:
 - Na neki način predvidjeti da li će uvjet grananja biti zadovoljen ili ne
 - Na temelju rezultata predviđanja:
 - ako je grananje vjerojatno: dohvaćati naredbe sa adrese za grananje
 - ako grananje nije vjerojatno: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

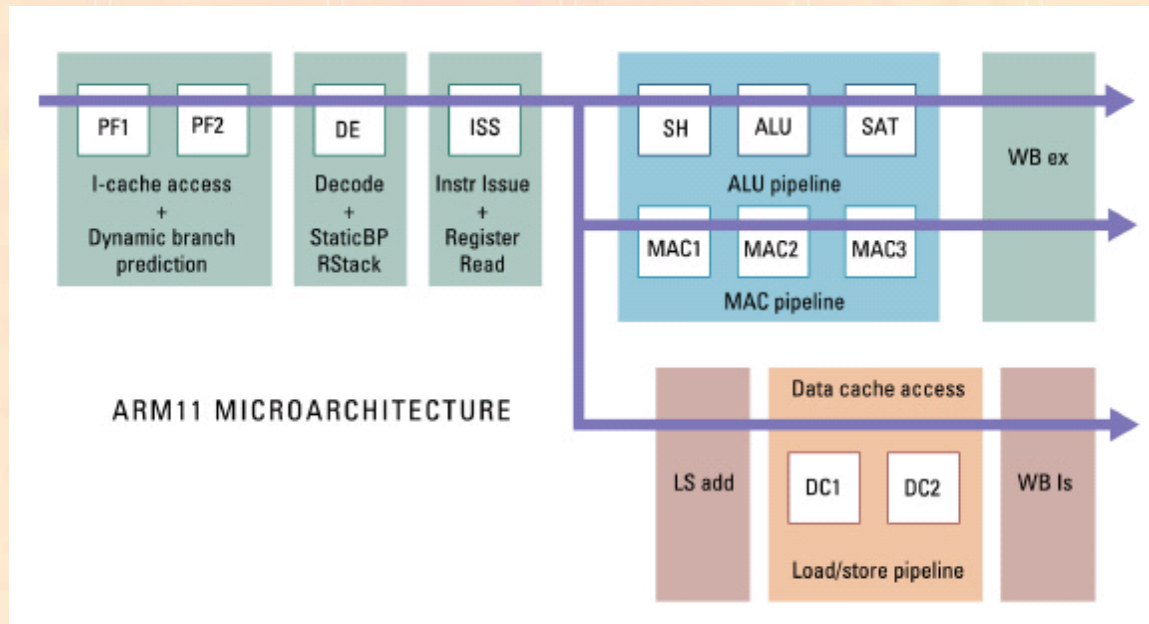


- Jednostavno za izvedbu, a poboljšava performanse

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

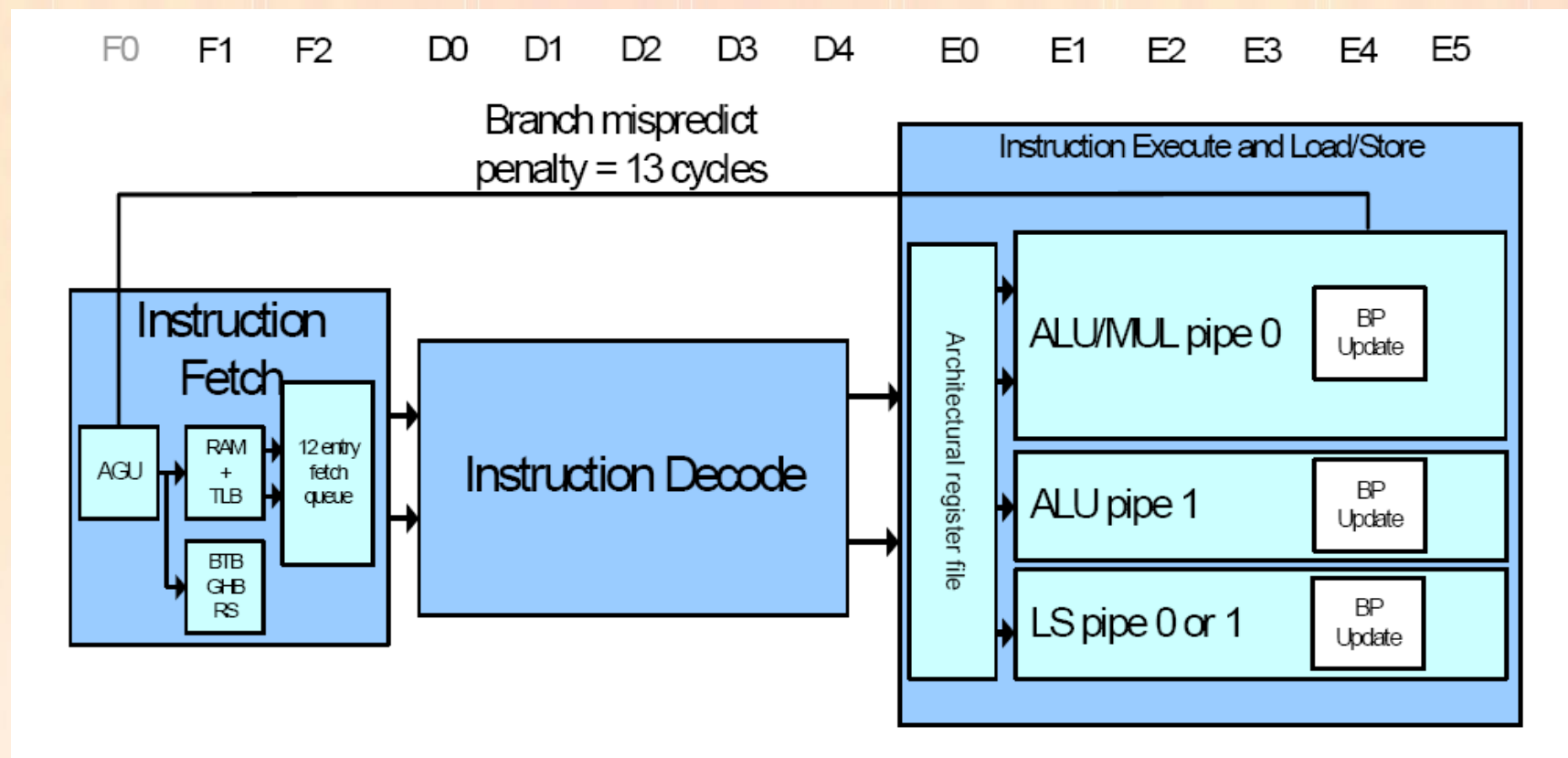
Primjer: ARM11



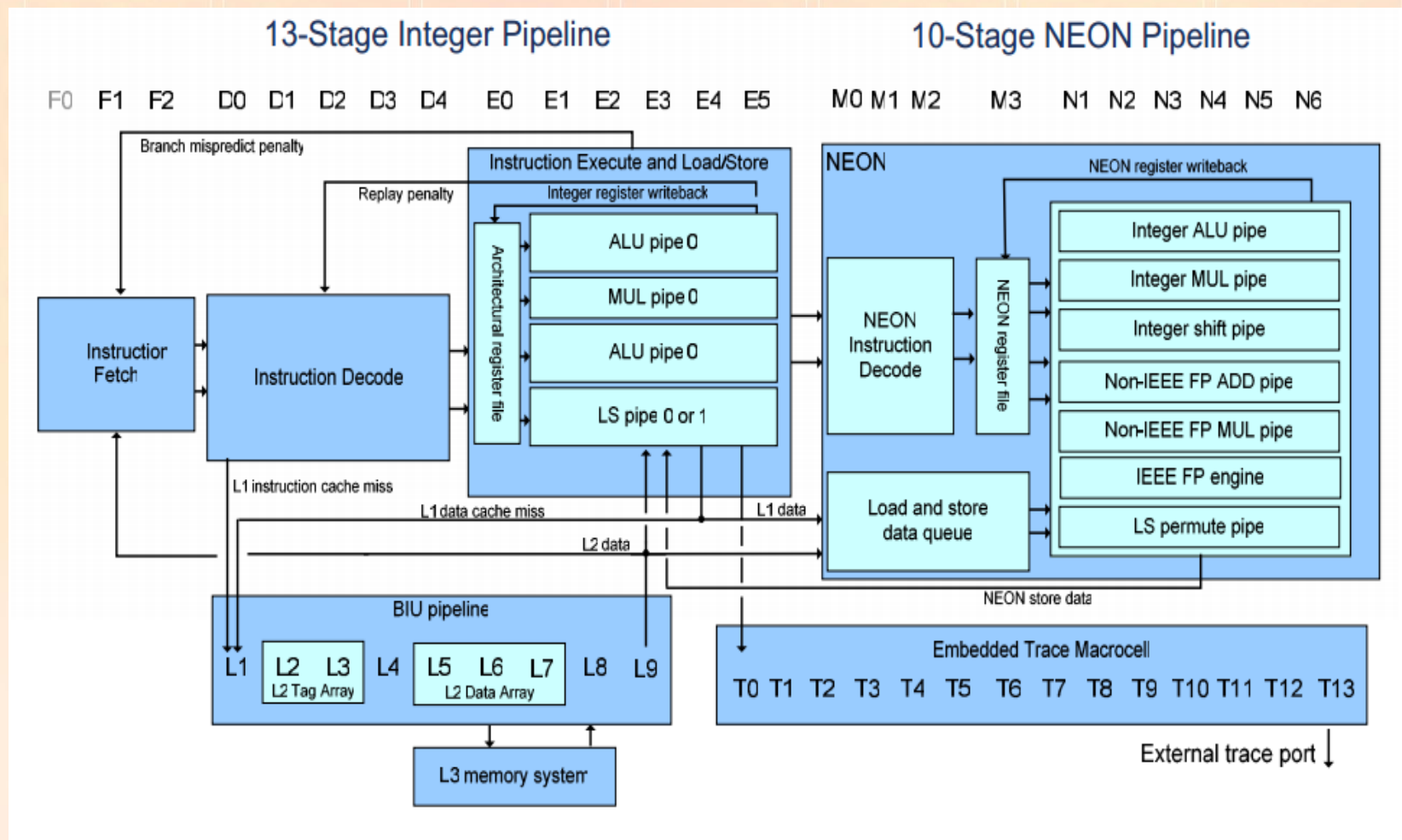
- Protočna struktura sa 8 razina
- Na razinama 1 i 2 obavlja se dinamičko predviđanje grananja
- Na razini 3 obavlja se statičko predviđanje
- ARM tvrdi da je točnost predviđanja 85% !
- DODATNO: Paralelizam Integer, MAC i LoadStore protočnih struktura

ARM Cortex-A8 Microarchitecture

- Protočna struktura od 13 razina



ARM Cortex-A8 Microarchitecture + NEON



Paralelizam protočnih segmenata

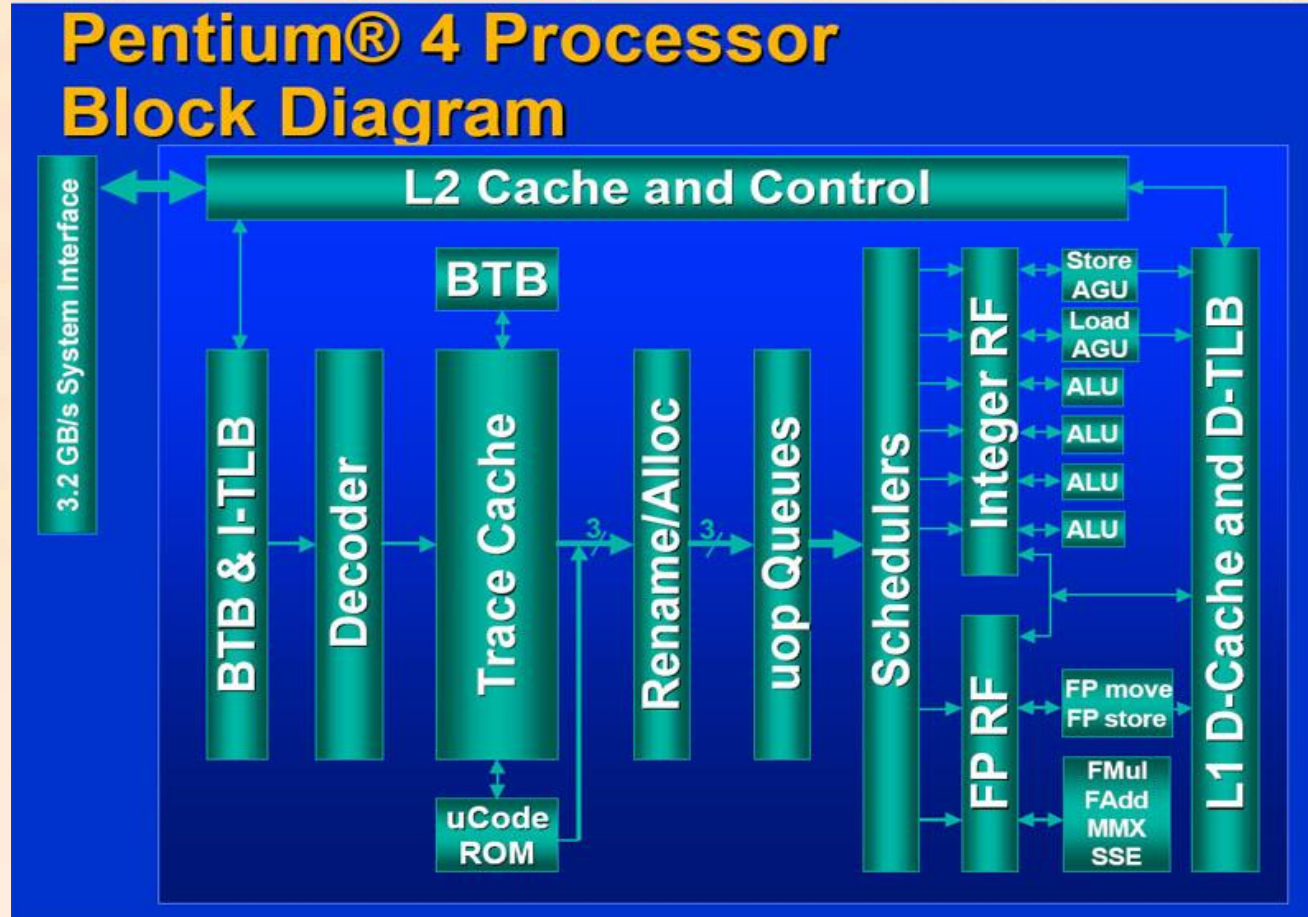
- Na primjeru prethodnih arhitektura može se vidjeti novo načelo:
 - Paralelizam izvođenja naredaba koje ne koriste iste resurse
 - Mogućnost izvođenja više naredaba u isto vrijeme
 - Veoma bitna arhitektura memorijskog sustava te podrška za predviđanje grananja
- Kod nekih arhitektura se dijelovi protočne strukture multipliciraju
 - Može se postići da se NEKOLIKO naredaba šalje na izvođenje u jednom trenutku čime se ostvaruje da procesor izvodi VIŠE OD JEDNE naredbe po vremenskom periodu
 - Takve arhitekture koriste načela više-dretvene obrade (multi threaded processing)
 - (npr. Intel: hyperthreading)

Protočne arhitekture drugih procesora



Intel Pentium 4

- P4 Willamette, Northwood (20 razina)



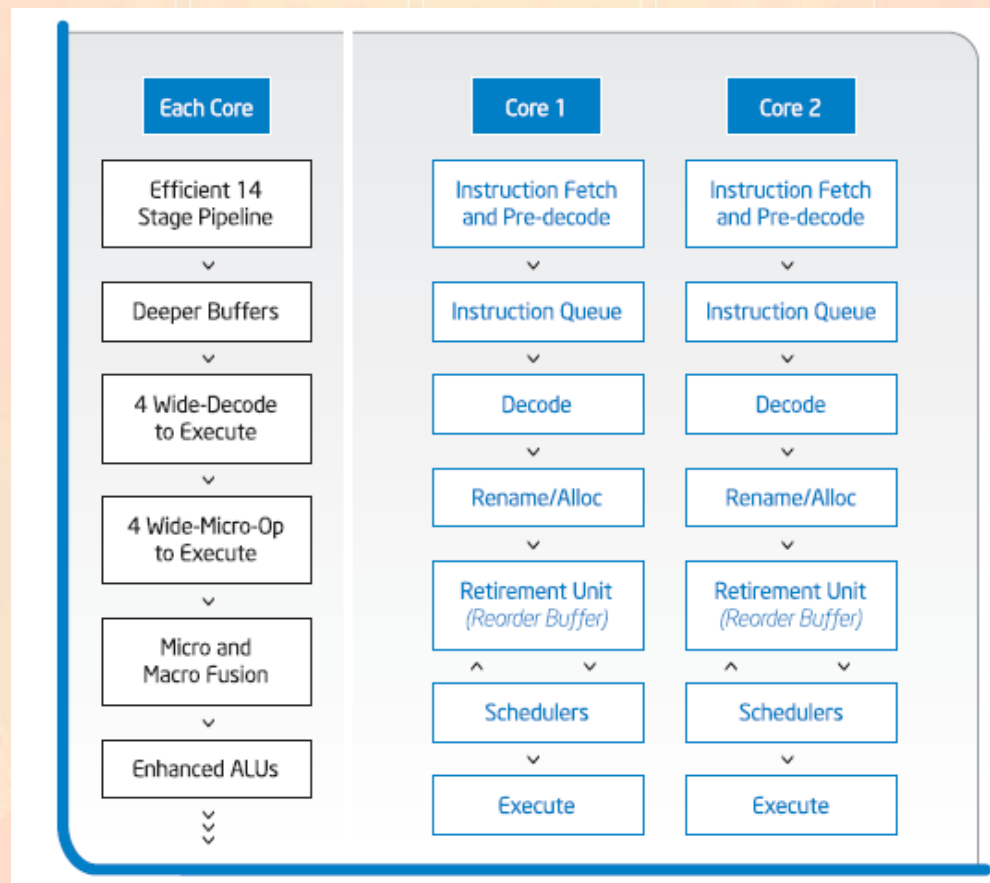
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC Nxt IP	TC Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Flgs	Br Ck	Drive			

Intel Prescott P4

- Protočna struktura proširena na 31 !!! razinu
- Procesor je doživio neslavnu sudbinu zbog brojnih problema sa zagrijavanjem te je na kraju ova arhitektura odbačena
- Arhitektura protočne strukture nije javno objavljena...

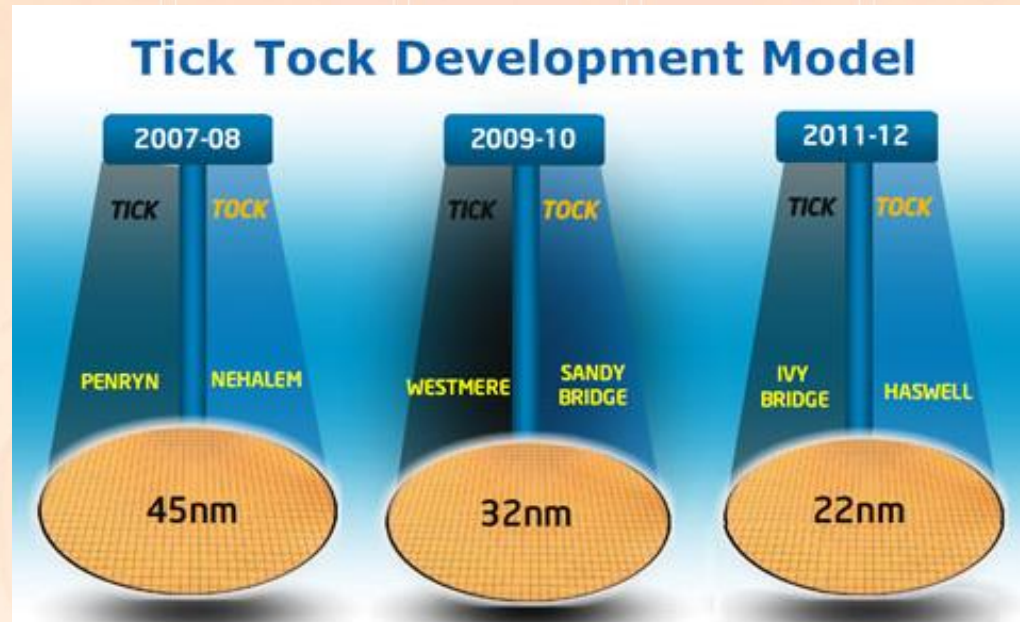
Intel Core Microarchitecture

- Nastavak razvoja krenuo je u sasvim drugom smjeru tako da nova arhitektura Intelovih procesora nazvana Intel Core Microarchitecture ima 14 razina
- Svaka jezgra može paralelno obraditi 4 naredbe (tako npr. procesor s 4 jezgre može obraditi u isto vrijeme 16 naredaba)



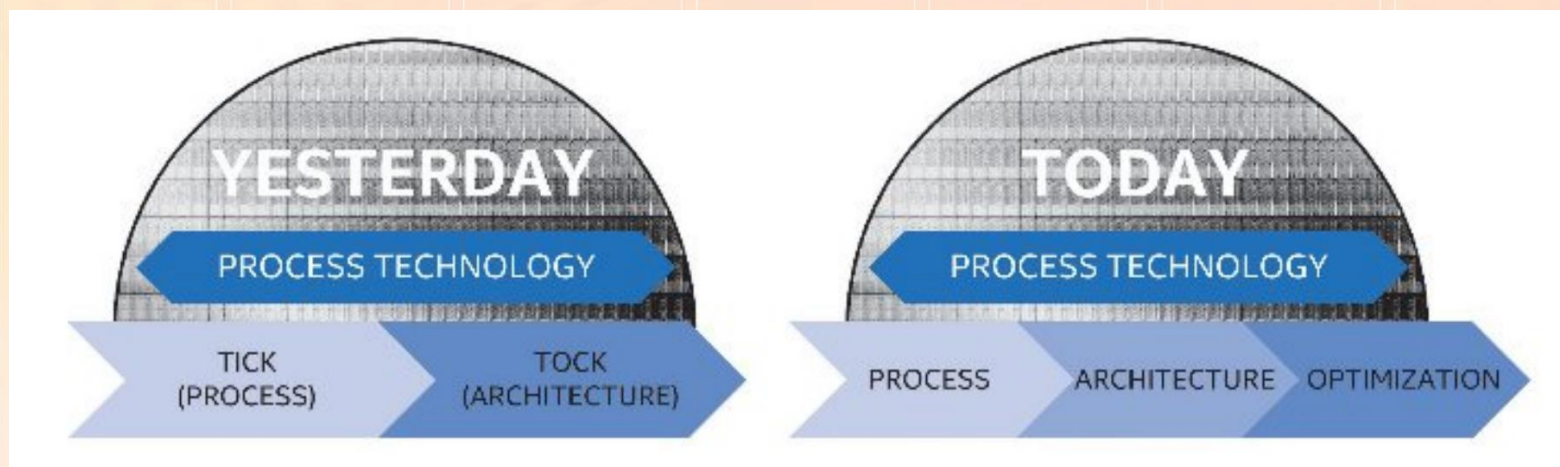
Intel Tick Tock ****

- Tick Tock model: Tick/Tock tijekom 2 godine
 - Tick: smanjenje tehnologije prijašnje mikroarhitekture
 - Tock: nova mikroarhitektura

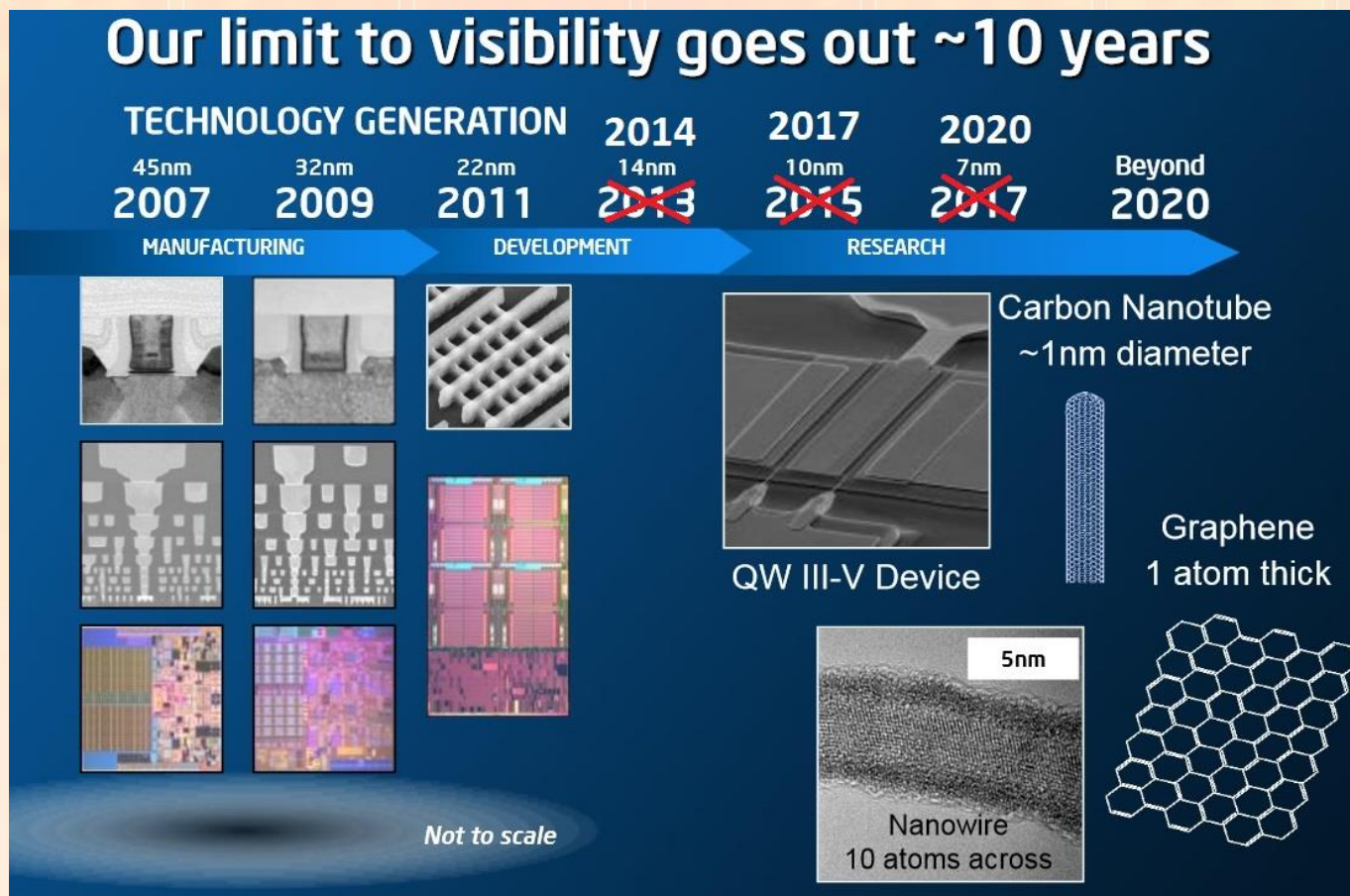


Novi model-Process-Architecture-Optimization

- Tick-Tock je službeno odbačen !!
- Novi model



3 godine za novu tehnologiju



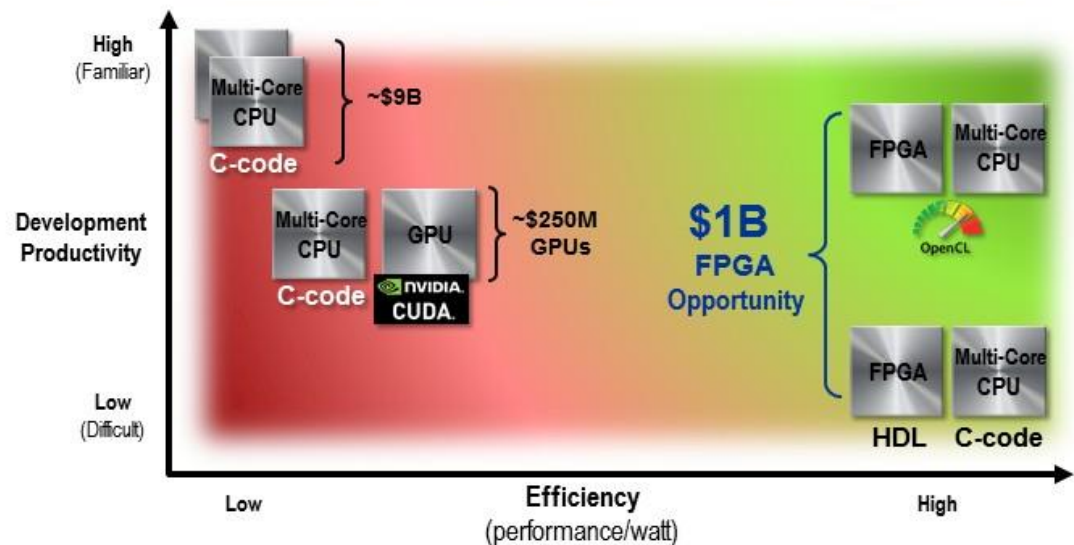
INTEL + INTEL FPGA

- Intel Completes \$16.7 Billion Buy of Altera (December 28, 2015)



FPGAs and OpenCL Drive Large Opportunity

PARALLEL COMPUTING IN THE DATA CENTER



AMD + XILINX

- AMD to Acquire Xilinx for \$35 billion, Creating the Industry's High Performance Computing Leader | AMD (Oct 27, 2020)



NVIDIA + ARM

- NVIDIA to Acquire Arm for \$40 Billion, Creating World's Premier Computing Company for the Age of AI (September 13, 2020)





DETALJI U IDUĆEM PREDAVANJU..

