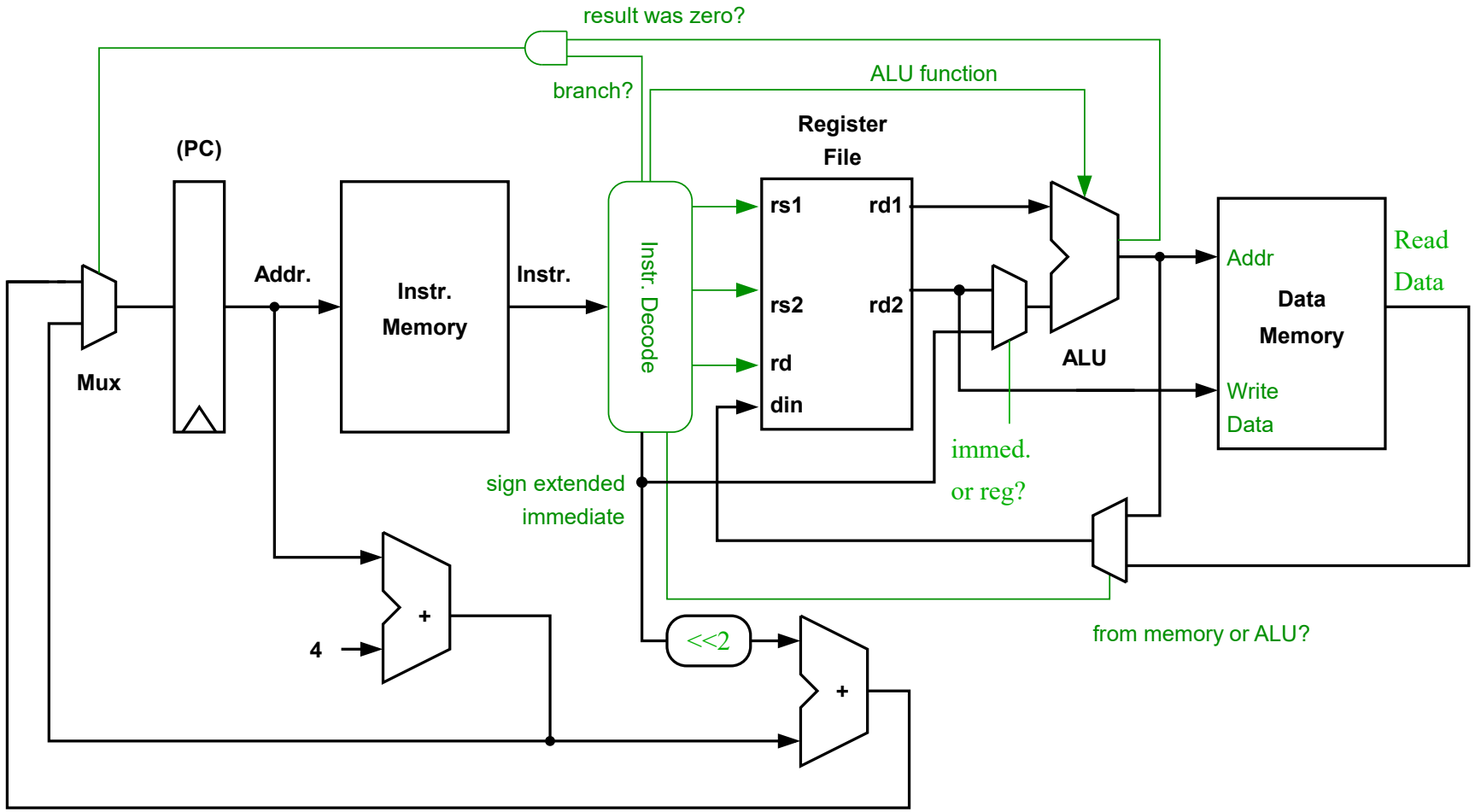
Computer architecture

Pipeline, rizici u pipeline strukturama i utjecaj na performanse

Sadržaj

- Implementacija pipelining struktura
- Rizici pipeline struktura i ovisnost podataka
- Performanse sa pipeline strukturama

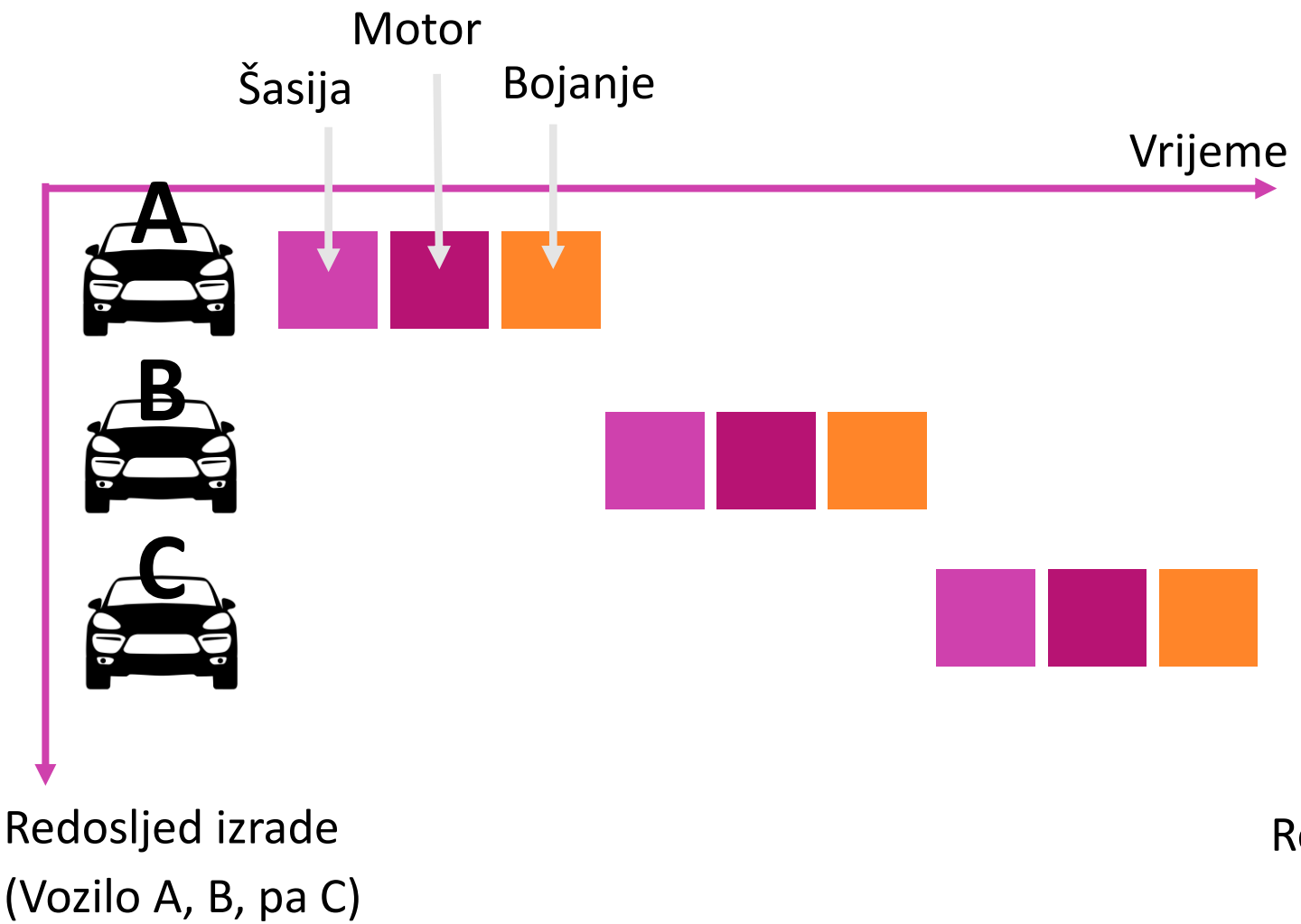
Pojednostavljeni procesor

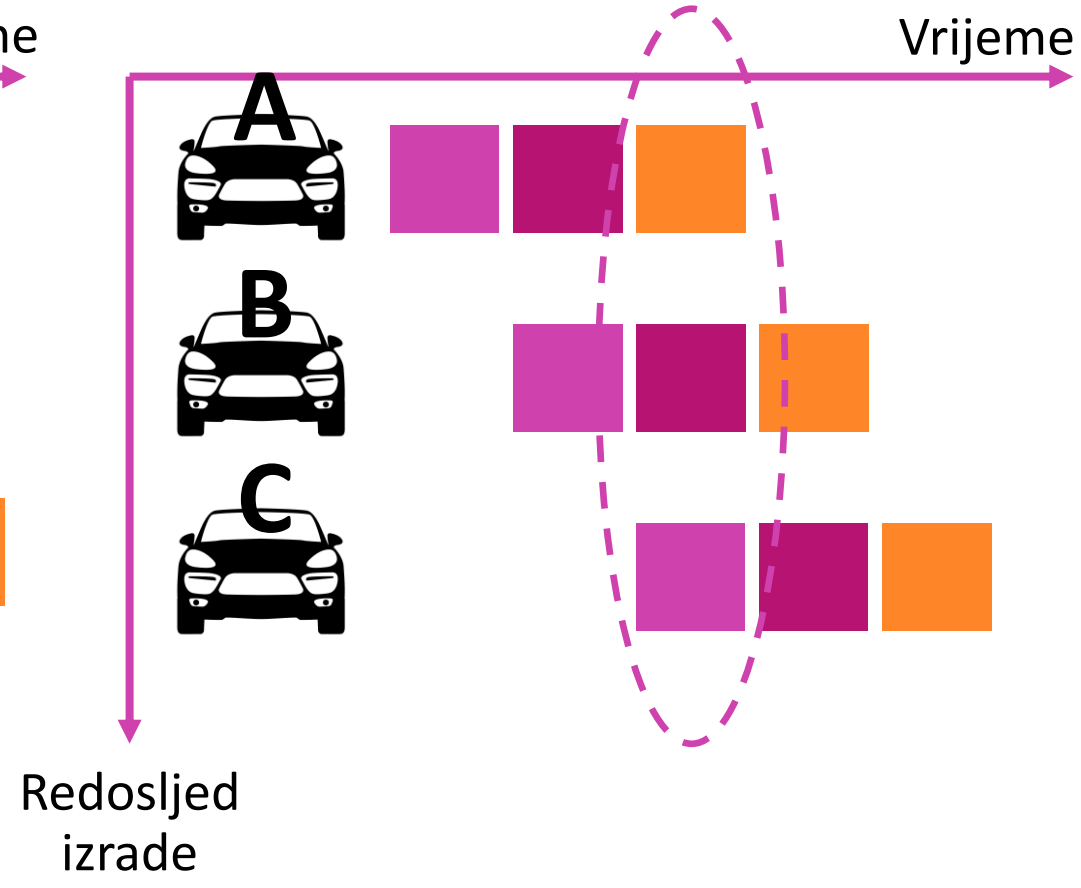


Pojednostavljeni procesor

- Naš procesor izvršava jednu instrukciju po ciklusu dakle ima Clocks Per Instruction (CPI) faktor točno 1.
- Minimalni ciklus definiran je kao najgori put kroz sve logičke sklopove i memoriju, na što su pribrojene oscilacije i mogući pomaci zbog utjecaja procesa, napona i temperature (Process, Voltage, and Temperature, poznat kao “PVT”)
- Kako da povećamo broj ciklusa u sekundi (clock) bez značajnog povećanja CPI faktora?

Što je Pipelining?





Što je Pipelining?

- Pokušavamo preklopiti različite faze rada kako bi iskoristili vremenski paralelizam u samom procesu
- Kako ovo utječe na latenciju i propusnost sustava?

Alden Jewell CC BY 2.0

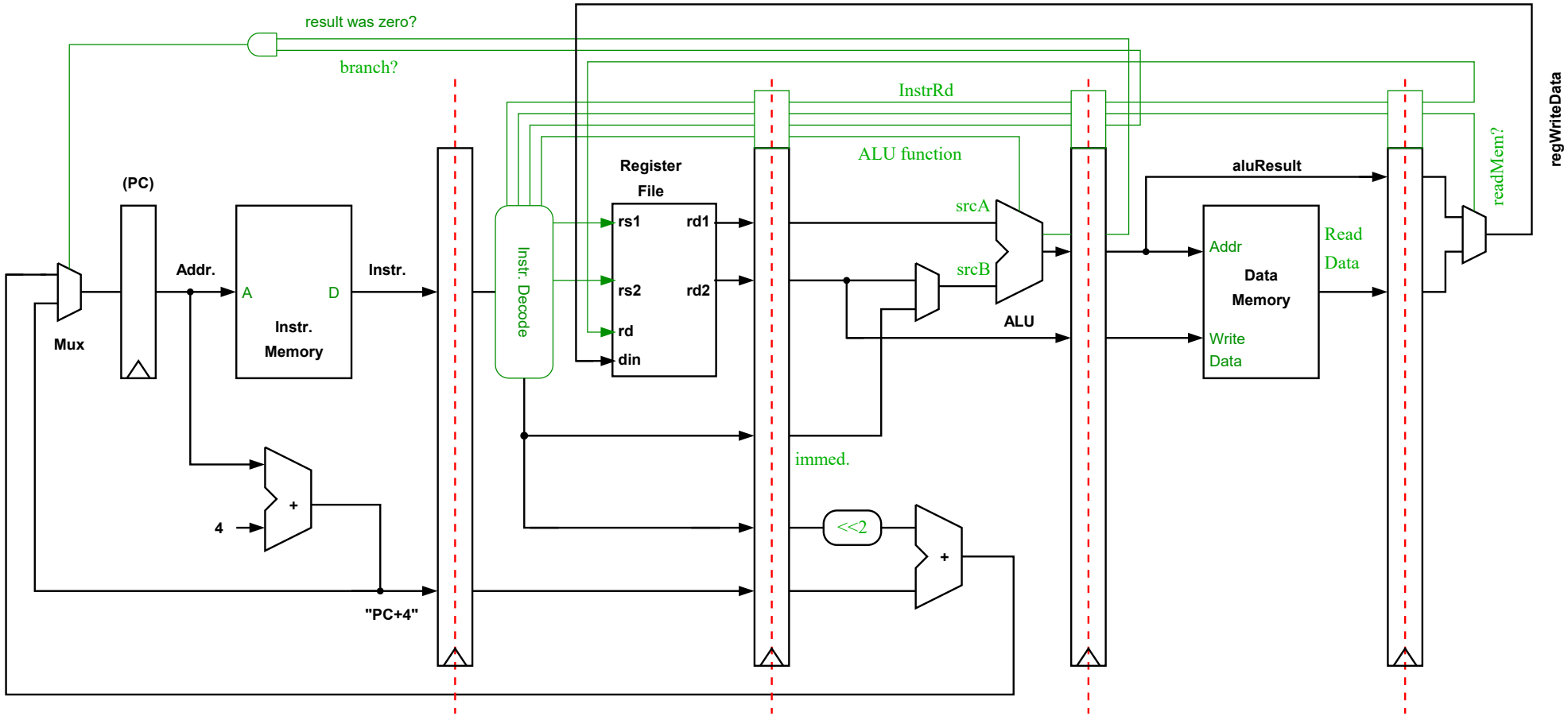
Pipelining i održavanje ispravnosti rada sustava

- Izvršavanje instrukcija možemo rastaviti u stupnjeve te preklopiti izvršavanje različitih instrukcija
- Moramo osigurati da su rezultati našeg novog procesora koji sadrži pipeline strukture isto kao i rezultat procesora koji ih nema
- Naše izmjene u načinu izvršavanja instrukcija ne smiju utjecati na krajnji rezultat obrade podataka.
- Koji bi mogli biti problemi koje stvaraju pipeline strukture i njihovo uvođenje u procesor?

Pipelining

- U naš procesor možemo ubaciti dodatne registre (pipelining registers) koji će omogućiti da se logika može podijeliti u stupnjeve pri izvršavanju
- Ako se ovakvi registri ispravno ubace mogu smanjiti kašnjenje uzrokovano najgorim slučajem prebacivanja podataka između registara
- Potrebno je dodati i dodatne kontrolne signale kako bi informacije o upravljanju podacima pratile naše instrukcije kroz sve stupnjeve izvršavanja

Pipelining u našem procesoru



Pipelining

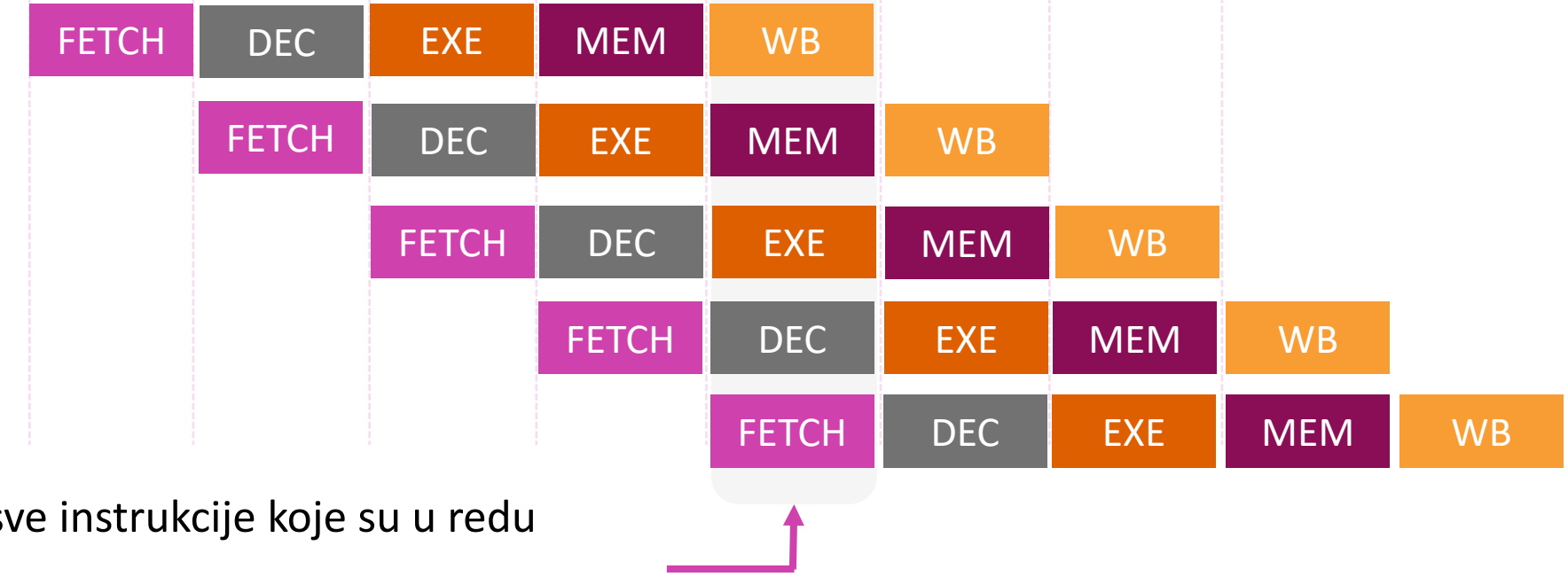
- Uzeli smo osnovni procesor i dodali u njega pipelining registre time mijenjajući tijek podataka. Ovo mijenja ponašanje procesora, više detalja malo kasnije
- Ovime stvaramo pipeline koji ima 5 stupnjeva:
  - **FETCH** – pristup našoj memoriji sa instrukcijama
  - **DECODE** – dekodiranje instrukcije i čitanje .
  - **EXECUTE** – izvršavanje instrukcija u ALU, izračun memorijskih adresa, po potrebi izračun odredišnih adresa
  - **MEMORY** – pristup podacima u memoriji.
  - **WRITEBACK** – zapisivanje podataka natrag u registre.

Kako funkcionira pipeline

- Clock
- Cycle

1 2 3 4 5 6 7 8

- Instrukcija
- 1. LDR X1, [X2]
- 2. ADD X2, X2, X3
- 3. STR X4, [X5], #4
- 4. SUB X0, X0, #1
- 5. LDW X5, [X3]



U ciklusu broj 5 pipeline sadrži sve instrukcije koje su u redu izvršavanja

Idealni Pipeline

U idealnom slučaju naš CPI je 1, dakle nikada se ne događaju usporavanja u pipelinu.

Bilo kakvo usporavanje pipelinea znači povećanje CPI:

Ako moramo čekati 1 ciklus za 20% instrukcija i 3 ciklusa za 5% instrukcija naš novi CPI se može izračunati kao:

Pipeline CPI = idealni pipeline CPI + čekanje po instrukciji

= 1 + 1*0.20 + 3*0.05 = 1.35

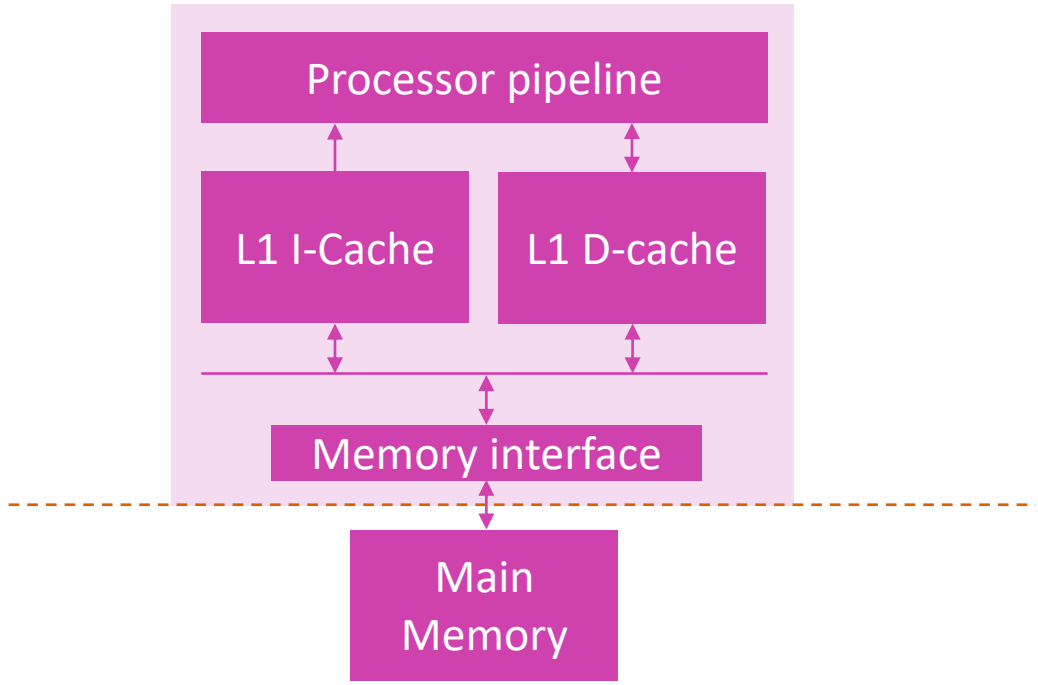
Napomena: Ako želimo maksimalno iskoristiti pipeline moramo izbjegavati bilo kakva čekanja, bez povećavanja trajanja jednog ciklusa. To je zato što je:

Vrijeme = broj izvršenih instrukcija × Clocks Per Instruction (CPI)× trajanje jednog ciklusa

Čekanje zbog pristupa memoriji

Latencija zbog pristupa memoriji koja nije u procesoru (primjerice DRAM) je 10-100 puta veća od trajanje jednog ciklusa našeg procesora.

Da bi izbjegli kašnjenje i čekanje moramo koristiti cache memoriju u procesoru.



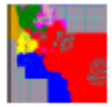
A single Arm Cortex A35 with 8K L1 instruction and data caches, no L2

For a 28 nm process:

Area: < 0.4 mm^2

Clock: 1 GHz

Power: ~90 mW



Rizici Pipeline struktura

Rizici Pipeline struktura

- Struktura pipelinea omogućava novim instrukcijama da se krenu izvršavati dok su prethodne instrukcije još u fazi izvršavanja, dakle izvršavanje se preklapa.
- Postoje situacije u kojima instrukcija mora čekati na rezultate neke od prethodnih instrukcija kako bi se osigurala ispravnost izvršavanja.
- Ovu situaciju nazivamo rizikom, te ih dijelimo na nekoliko slučajeva:
  - **Structural hazard** – strukturni rizici zbog pristupa dijeljenim resursima
  - **Data hazard** – podatkovni rizici zbog potrebe da se osigura međuovisnost podataka u različitim instrukcijama.
  - **Control hazard** – kontrolni rizici koje uzrokuju instrukcije koje mijenjaju PC registar, dakle grananja i skokovi u programu

Strukturni rizici

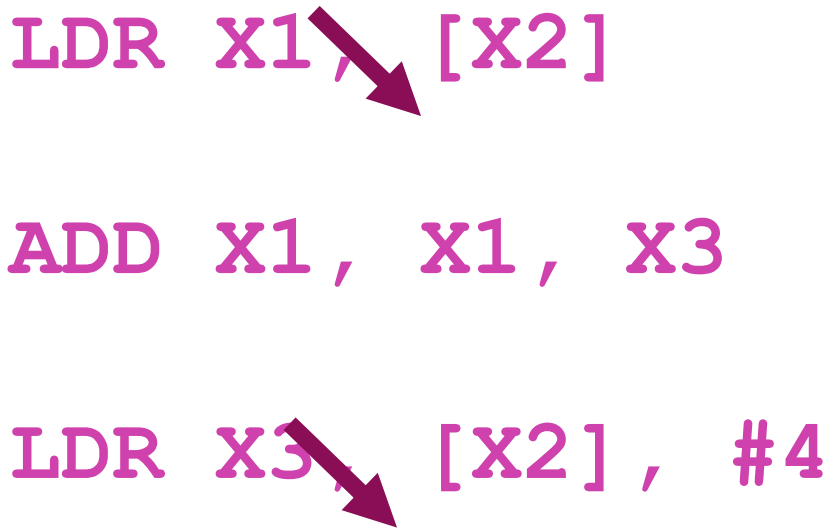
- Instrukcija mora čekati pristup nekom dijeljenom resursu. Primjerice: :
  - Nekoj funkcionalnoj jedinici koja nema pipeline strukture
  - Pristup zajedničkom registru za čitanje i pisanje
  - Memoriji
- Zašto dozvoljavamo strukturne rizike?
  - Dizajn koji pokušava riješiti najgori mogući slučaj izvršavanja može produžiti izvršavanje prosječnog slučaja (instrukcije). Prevedeno, dodavanje kompleksnosti u sustav možda održi CPI ali će povećati ukupno vrijeme potrebno za jedan ciklus.
  - Dodavanje podrške za najgori mogući slučaj može biti preskupo (kada govorimo o potrošnji i površini na procesoru). Možda postoje striktna ograničenja potrošnje ili površine pa ono što nam je na raspolaganju želimo iskoristiti na nekom drugom mjestu.

Podatkovna ovisnost, prava međuovisnost podataka

Prava podatkovna ovisnost – poznata kao i Read-After-Write (RAW) ovisnost.

Zamislimo dvije instrukcije, instrukciju i nakon koje slijedi instrukcija j.

Ako j koristi rezultat instrukcije i onda kažemo da je j **podatkovno ovisna** o i.

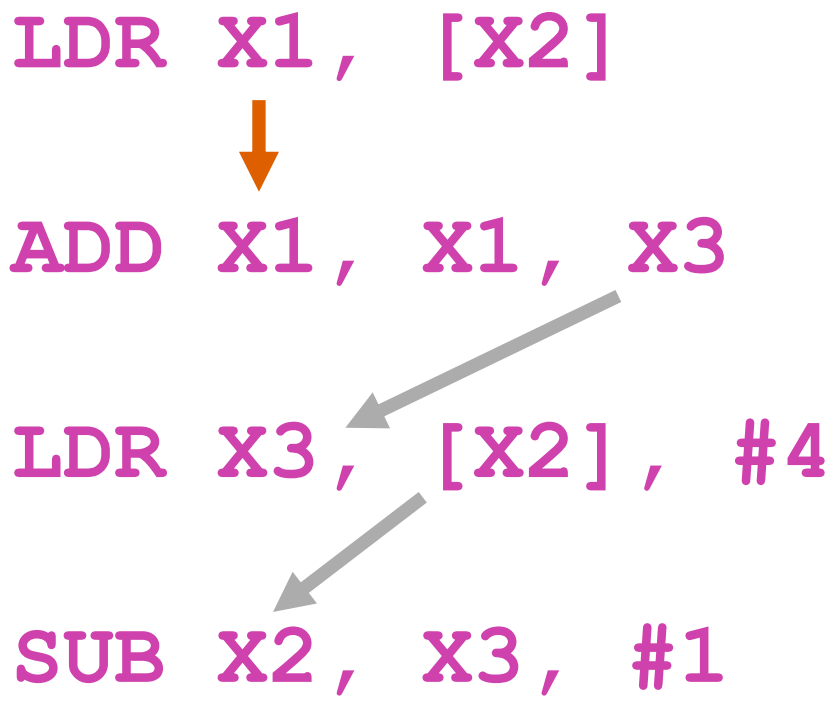


Podatkovna ovisnost – ovisnost o imenu

Imenovana ovisnost ili ovisnost o imenu može postojati i ako se dvije upute odnose na isti registar. Za razliku od stvarnih ovisnosti o podacima, podaci se ne prenose između instrukcija:

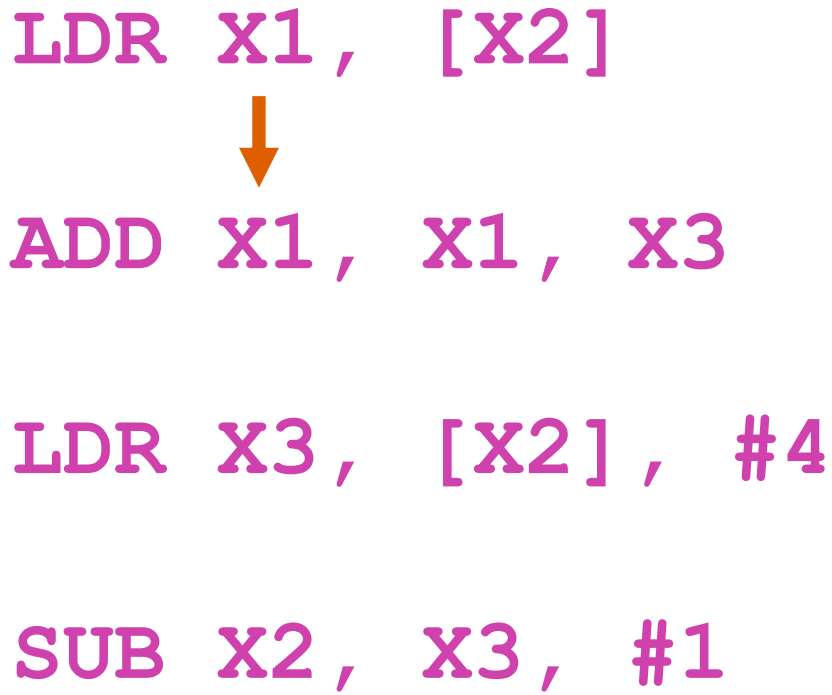
Izlazna ovisnost (crvena strelica)

- Anti-ovisnost (zlatna strelica)



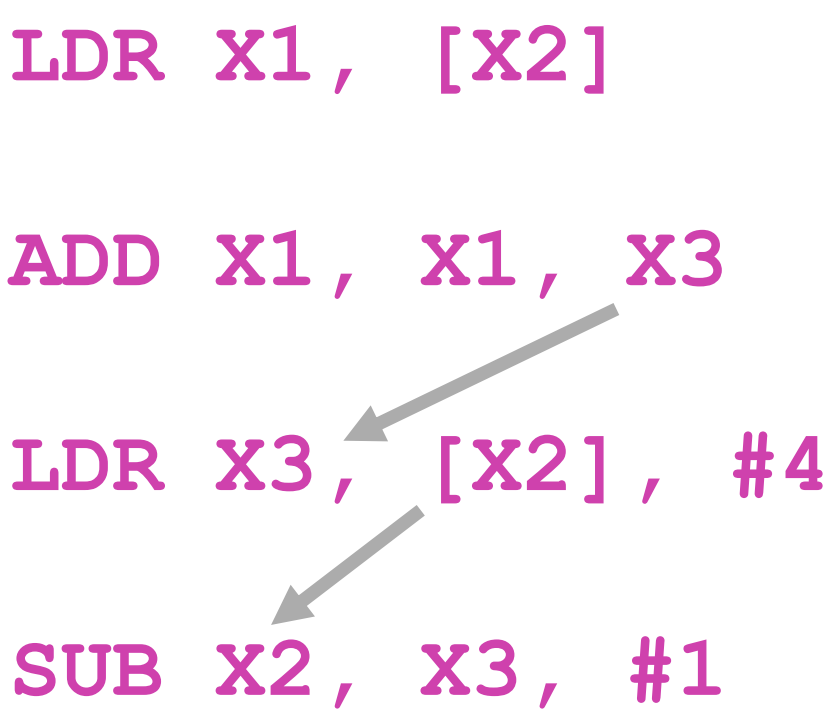
Podatkovna ovisnost – ovisnost o imenu

- **Izlazna ovisnost** (crvena strelica). Poznata i kao **Write-After-Write (WAW)** ovisnost
- Moramo se pobrinuti da ne presložimo pisanje u isti registar odnosno da ne mijenjamo red kojim se sadržaj registra mijenja. To bi značilo da bi naknadne upute mogle dobiti pogrešnu vrijednost podataka.



Podatkovna ovisnost – ovisnost o imenu

- **Anti-ovisnost (zlatne strelice). Poznata i kao ovisnost o pisanju nakon čitanja (WAR)** Opet, moramo biti oprezni da ne prebrišemo registar čija je trenutna vrijednost još uvijek potrebna ranijim instrukcijama.
- *Npr. ne možemo zakazati STR instrukciju prije instrukcije ADD.*
-



Podatkovni rizici

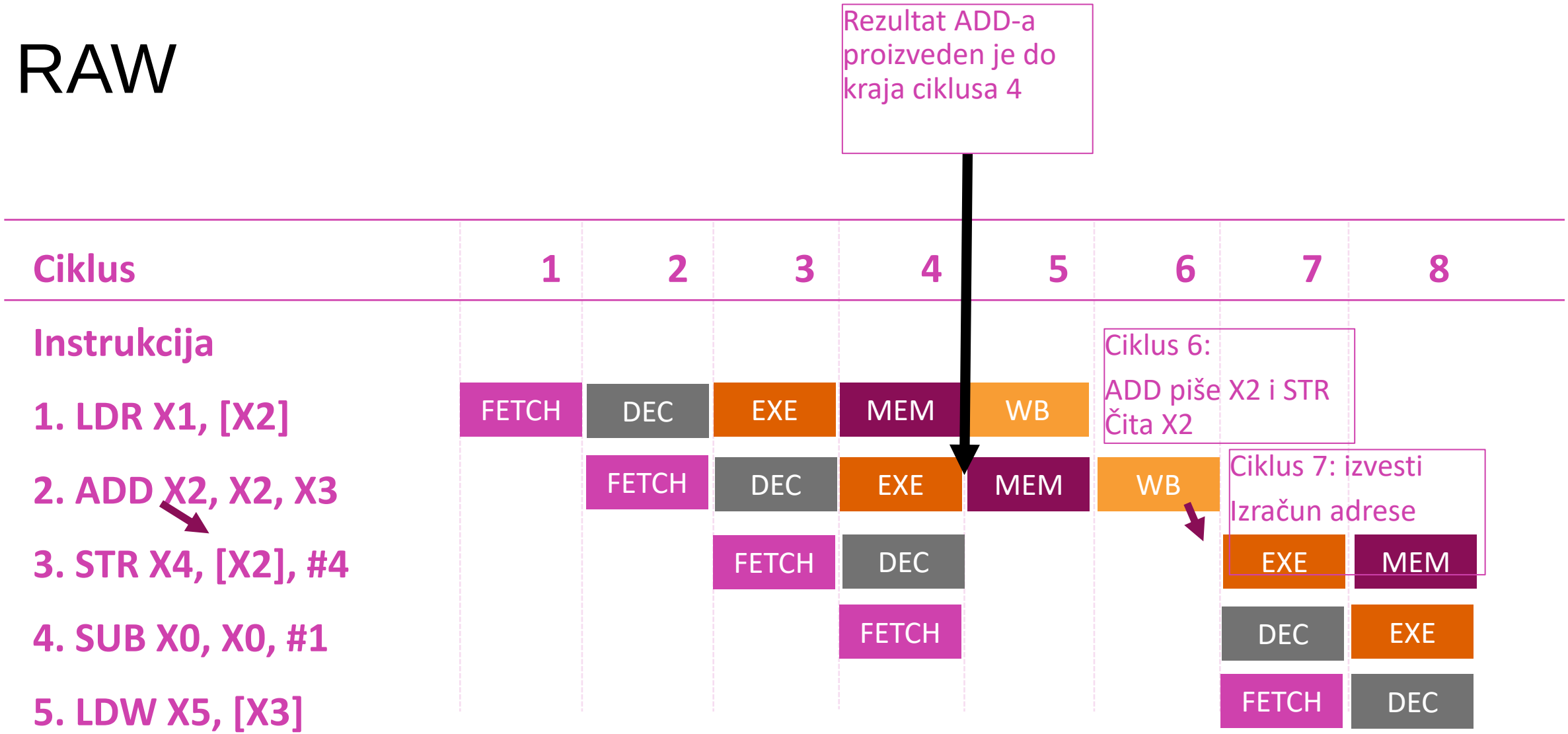
Podatkovni rizik stvara se kad god paralelno izvršavanje instrukcija omogućuje kršenje međuovisnosti pojedinih instrukcija.

Read After Write (RAW) – nastala istinskim ovisnostima o podacima, tj. pokušava pročitati izvorni registar nakon što mu je vrijednost već promijenjena idućom instrukcijom.

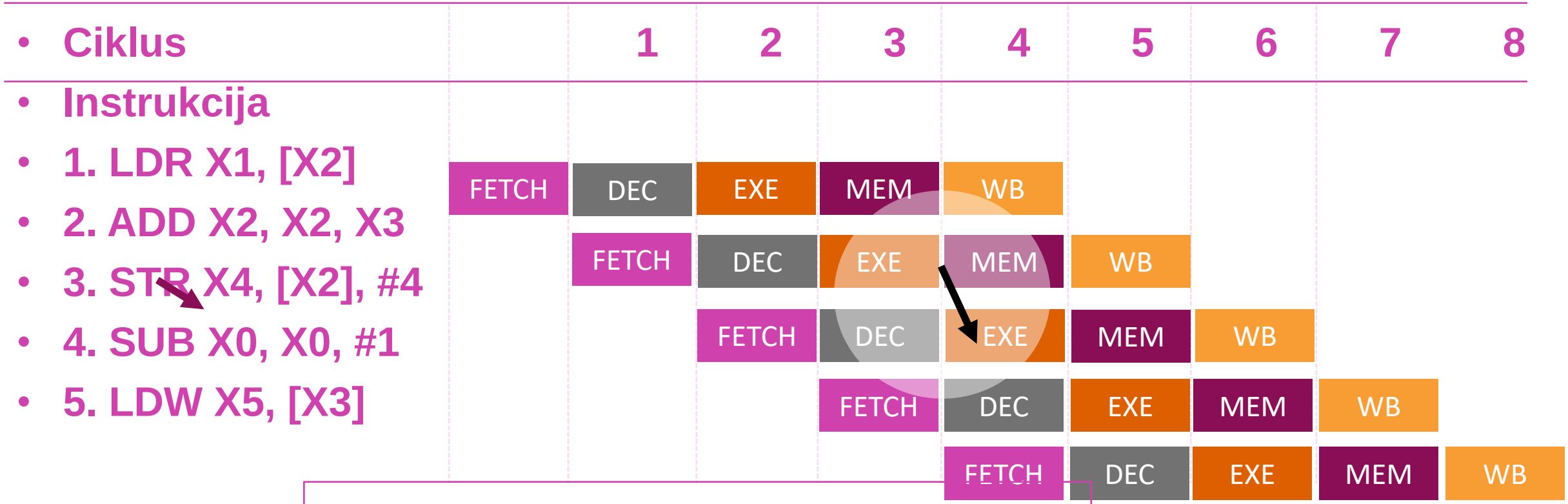
Write After Write (WAW) – proizveden izlaznim ovisnostima, odnosno j pokušava pisati u odredišni registar prije nego što ga zapiše i.

Write After Read (WAR) – nastao kao posljedica anti-ovisnosti, i.e., *j* piše po odredišnom registru prije nego ga je *i* pročitao.

RAW

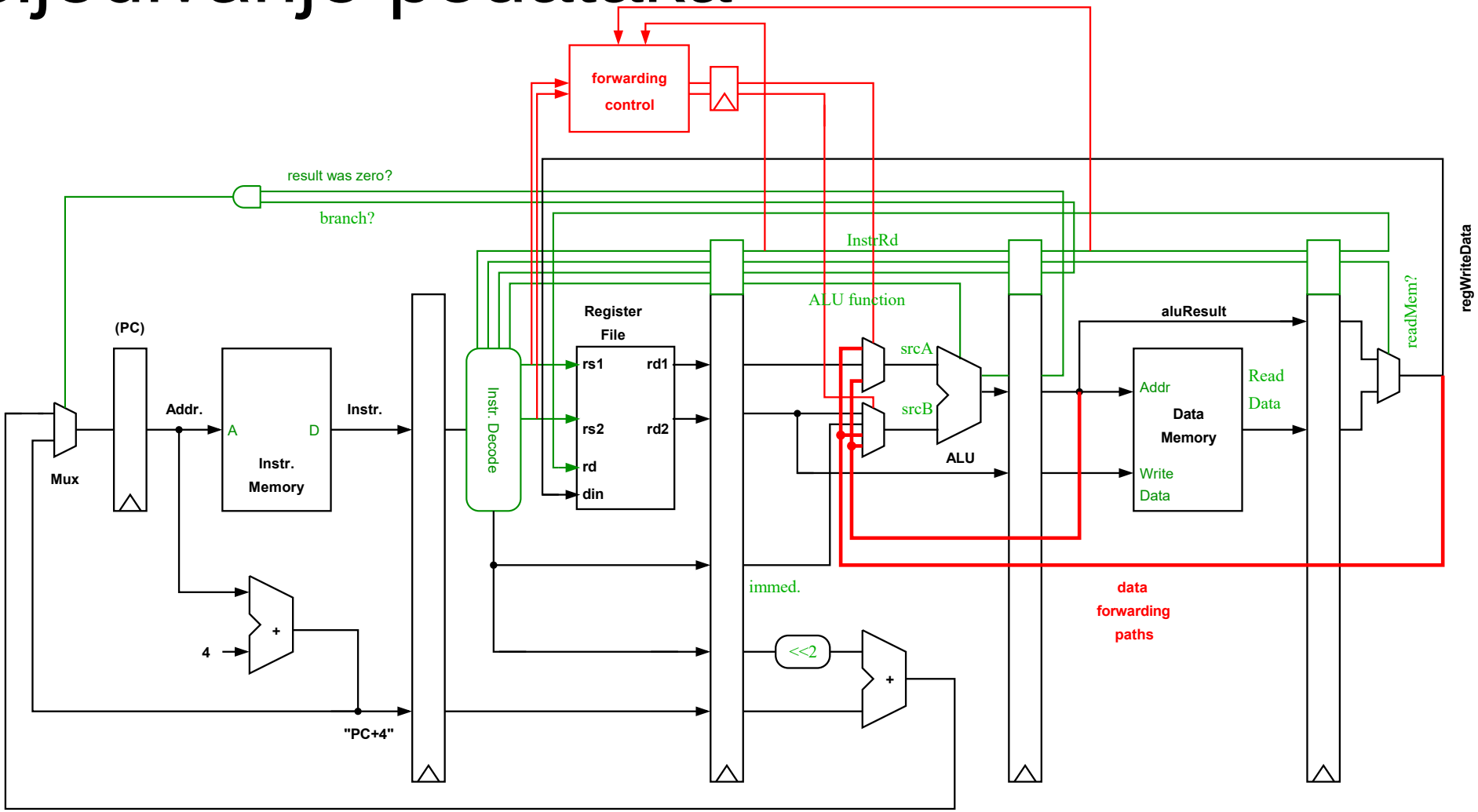

Rezultat ADD-a proizveden je do kraja ciklusa 4
Ciklus
1
2
3
4
5
6
7
8
Instrukcija
1. LDR X1, [X2]
2. ADD X2, X2, X3
3. STR X4, [X2], #4
4. SUB X0, X0, #1
5. LDW X5, [X3]
FETCH
DEC
EXE
MEM
WB
Ciklus 6:
ADD piše X2 i STR Čita X2
Ciklus 7: izvesti
Izračun adrese

Prosljeđivanje podataka

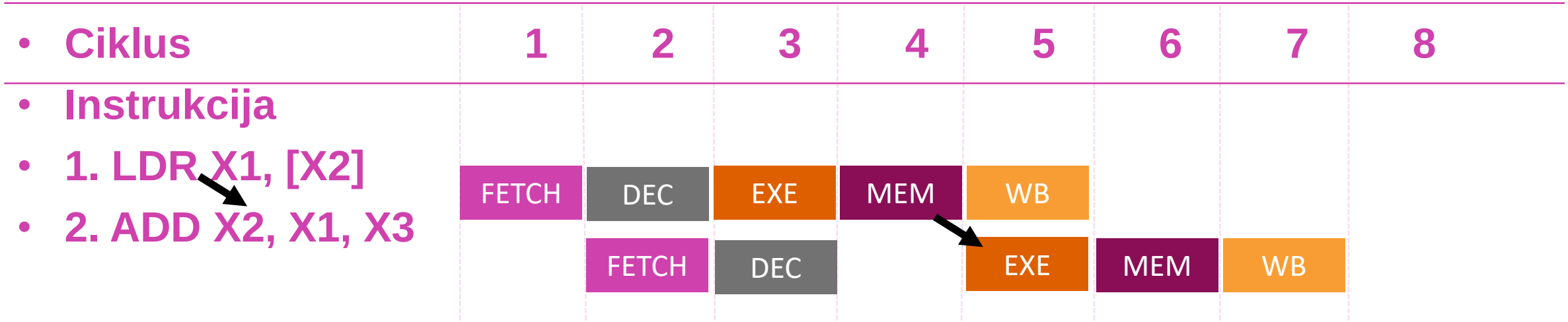


Kako bismo izbjegli usporavanje zbog RAW rizika, moramo "proslijediti" rezultat iz registra faze izvršenja na registar za unos ALU-a (umjesto komunikacije putem registra).

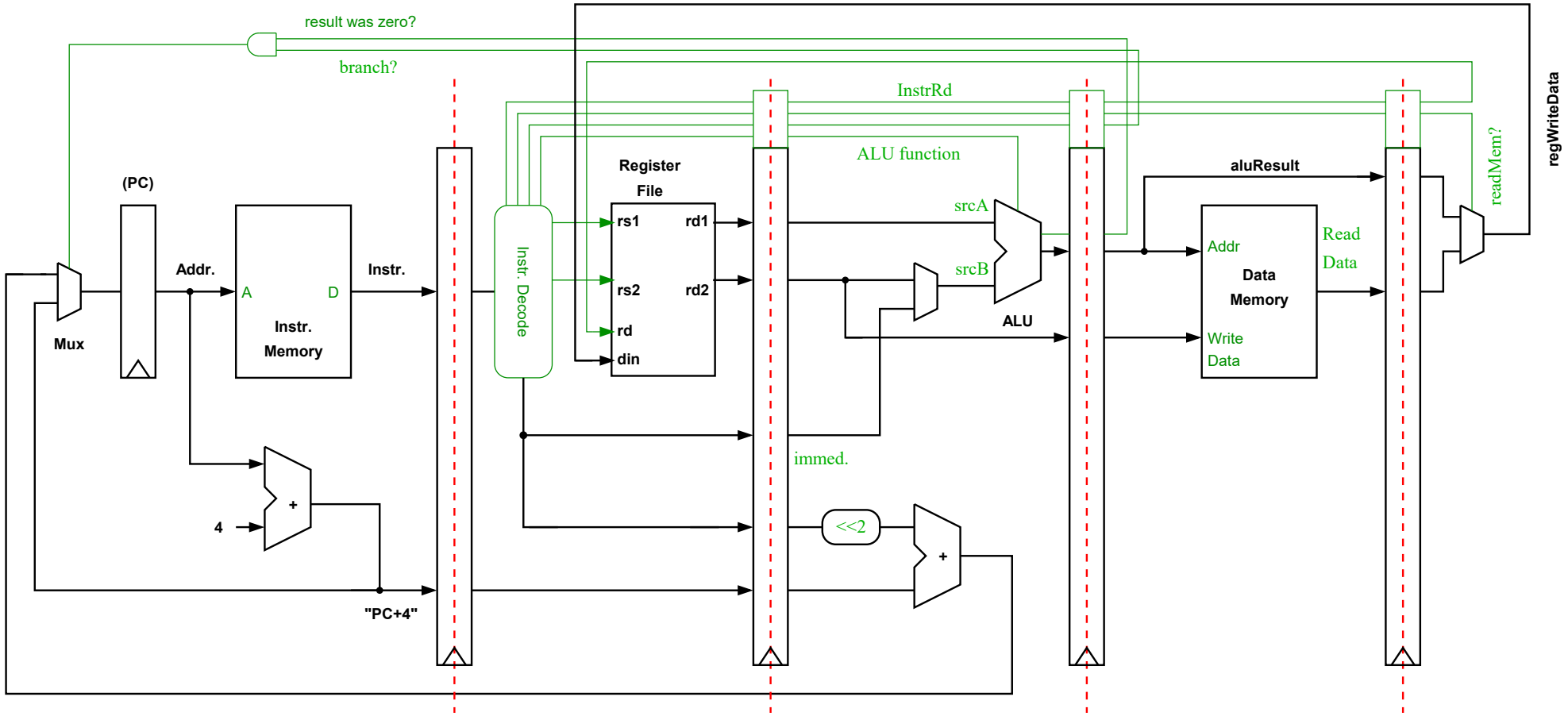
Prosljeđivanje podataka



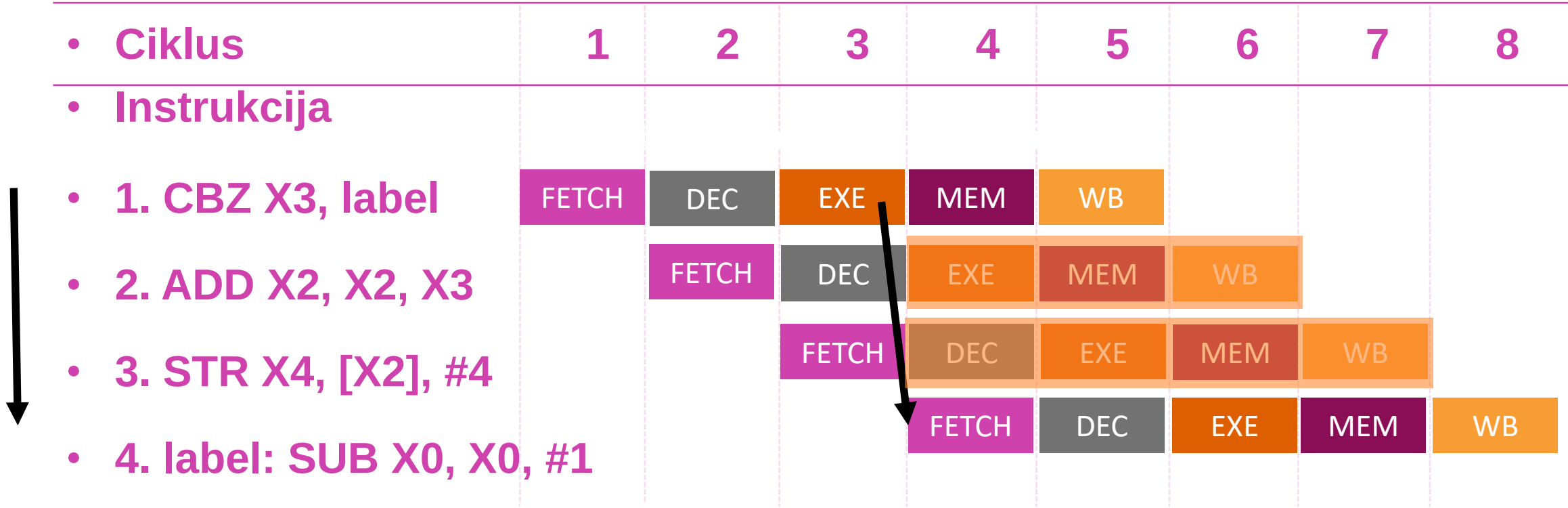
Kašnjenje učitavanja i korištenja



Kontrolni rizici



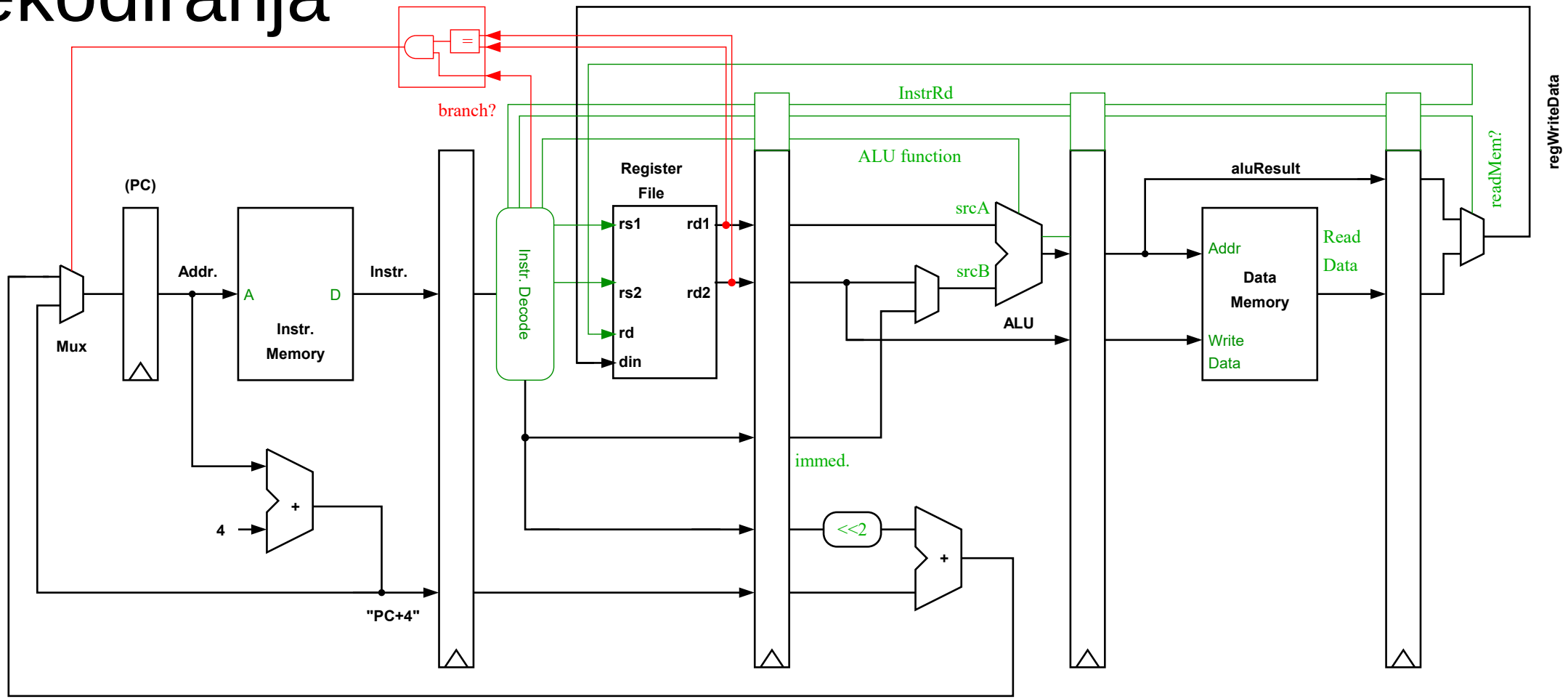
Kontrolni rizici



Kontrolni rizici – Procjena grananja u fazi dekodiranja

- Da bismo smanjili troškove grananja, mogli bismo procijeniti grananje u fazi dekodiranja.
- Neizvršeno grananje uključuje samo jedan "mrtvi" ciklus.
- Potencijalni rizici za podatke
  - Ako instrukcija neposredno prije grananja piše u registar koji se testira pri grananju, moramo pričekati jedan ciklus (tj. dok ova instrukcija ne generira svoj rezultat).
  - Također će nam trebati staze za prosljeđivanje od faza EXE i MEM do faze dekodiranja.

Kontrolni rizici – Procjena grananja u fazi dekodiranja



Pipeline i performanse

Pipeline CPI

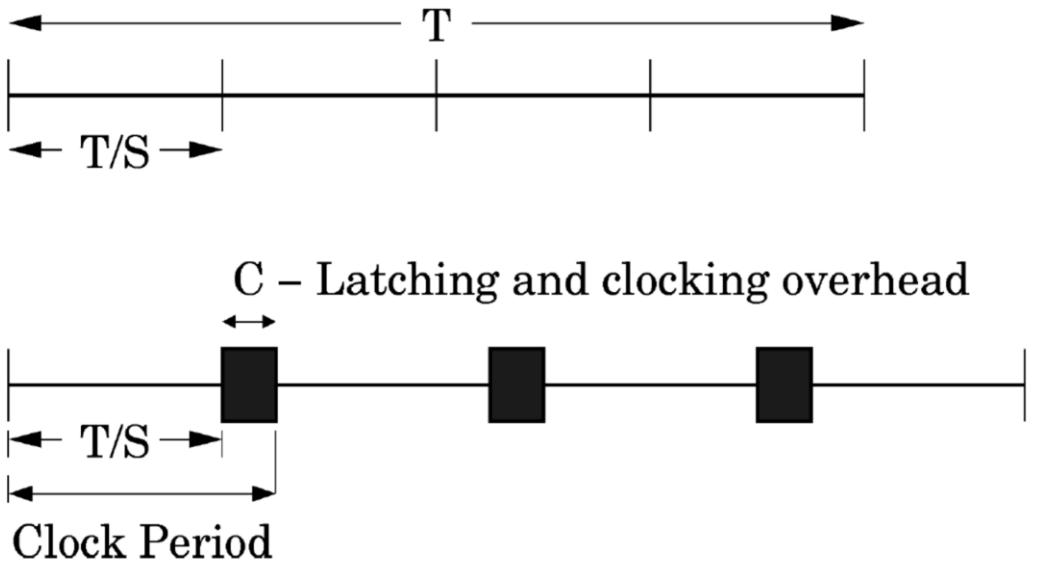
Pipeline CPI = idealni CPI + strukturna čekanja + čekanja zbog podatkovnih rizika + kontrolna čekanja

Čekanja se mogu smanjiti kombinacijom kompajlera (npr. Optimizacije) i hardverskih tehnika.

Hardverske tehnike obično povećavaju površinu i složenost našeg procesora. Slijedom toga, potrošnja energije obično brzo raste (ne samo linearno) dok pokušavamo poboljšati performanse našeg procesora.

Analitički model izvedbe

- Konstruirajmo jednostavan analitički model performansi piplinea.
- Počinjemo kritičnim putem kašnjenja T.
- Podijelite ga u S faza kašnjenja T / S.
- Onda dodajemo vremensko kašnjenje ciklusa C koje nastaje zbog upravljanja. Time dobivamo T/S+C.



Analitički model izvedbe

Pipeline CPI = idealni CPI + čekanja u pipelineu zbog instrukcija

Freq = 1 / (trajanje ciklusa) = 1 / (T/S + C)

Propusnost = Freq / CPI

Pretpostavimo da se zastoji javljaju frekvencijom B, a njihov trošak je proporcionalan dubini pipelinea, recimo (S-1):

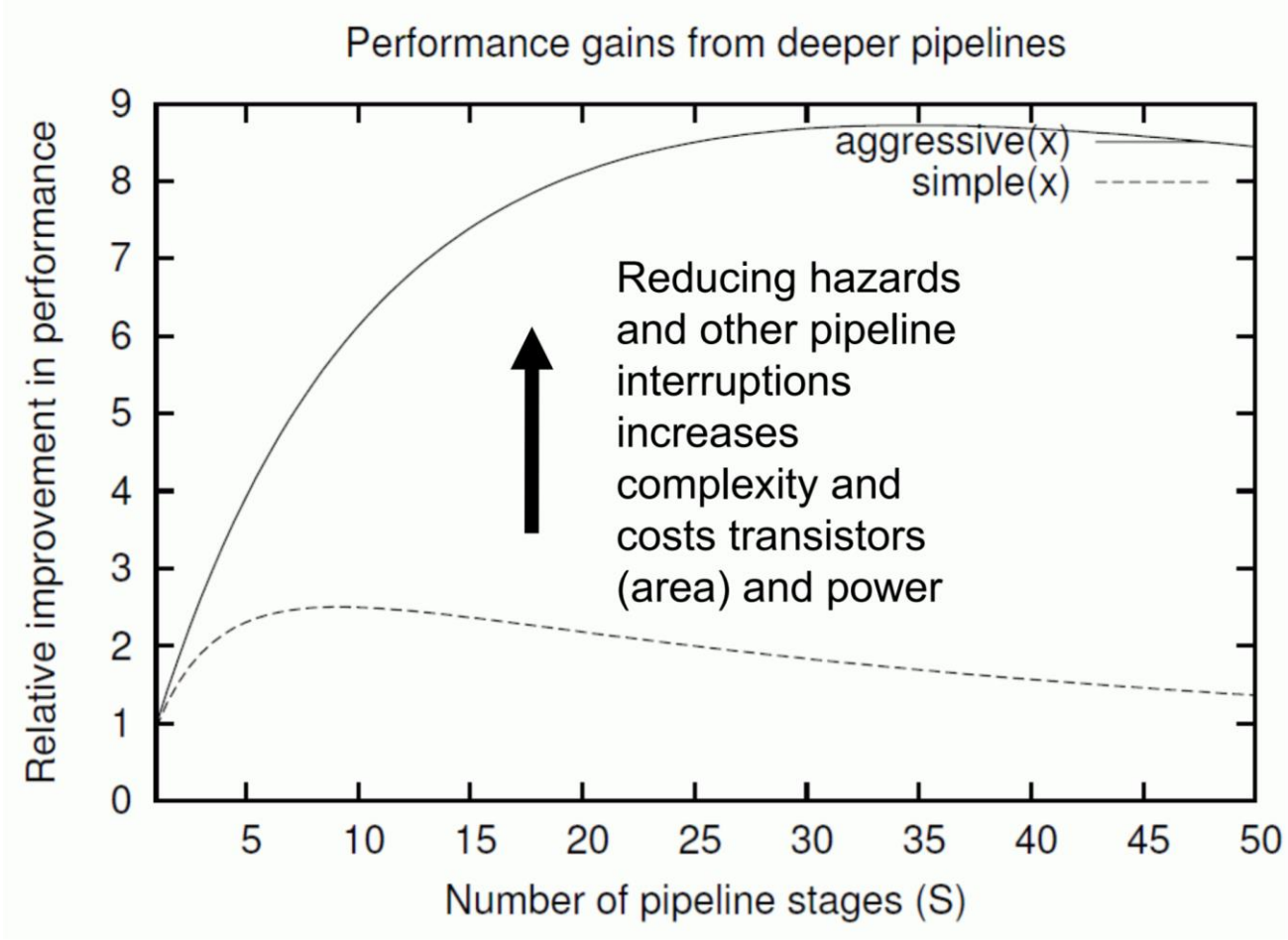
Propusnost = 1 / (1+(S-1)b) x 1 / (T/S+C)

Optimalna dubina pipelinea

T = 5 ns, kašnjenje zbog prekida (S-1)

Jednostavni pipeline dizajn
C= 300 ps
Pipeline prekid svakih 6 instrukcija

Aggressivni dizajn pipelinea
C = 100 ps
Pipeline kasni svakih 25 instrukcija



Tipične duljine pipelinea

- Jezgre optimizirane za manju površinu mogu imati 2-3 faze.
- Jednostavni, učinkoviti skalarni pipelineovi obično se provode s 5-7 faza.
- Jezgre s višim performansama, koje postižu više instrukcija u jednom ciklusu, mogu imati 8-16 faza.
- Duljine pipelineova za procesore opće namjene dosegle su vrhunac od 31 faze s Intelovim Pentiumom 4 2004. godine. Zašto su se smanjili od 2004. umjesto da se povećavaju?
-

Sažetak

Pipelining je često učinkovit i efikasan način za poboljšanje performansi. Naravno, moramo paziti da se dobici od bržeg ciklusa ne izgube zbog troškova kašnjenja u pipelineu.

Moramo se pobrinuti da:

1. Instructions stižu bez kašnjenja:
   - Osigurati učinkovito rukovanje grananjem
   - Pokušajte smanjiti stopu promašaja predmemorije instrukcija.
   - Dostava podataka – minimiziranje broja promašaja u predmemoriji podataka.

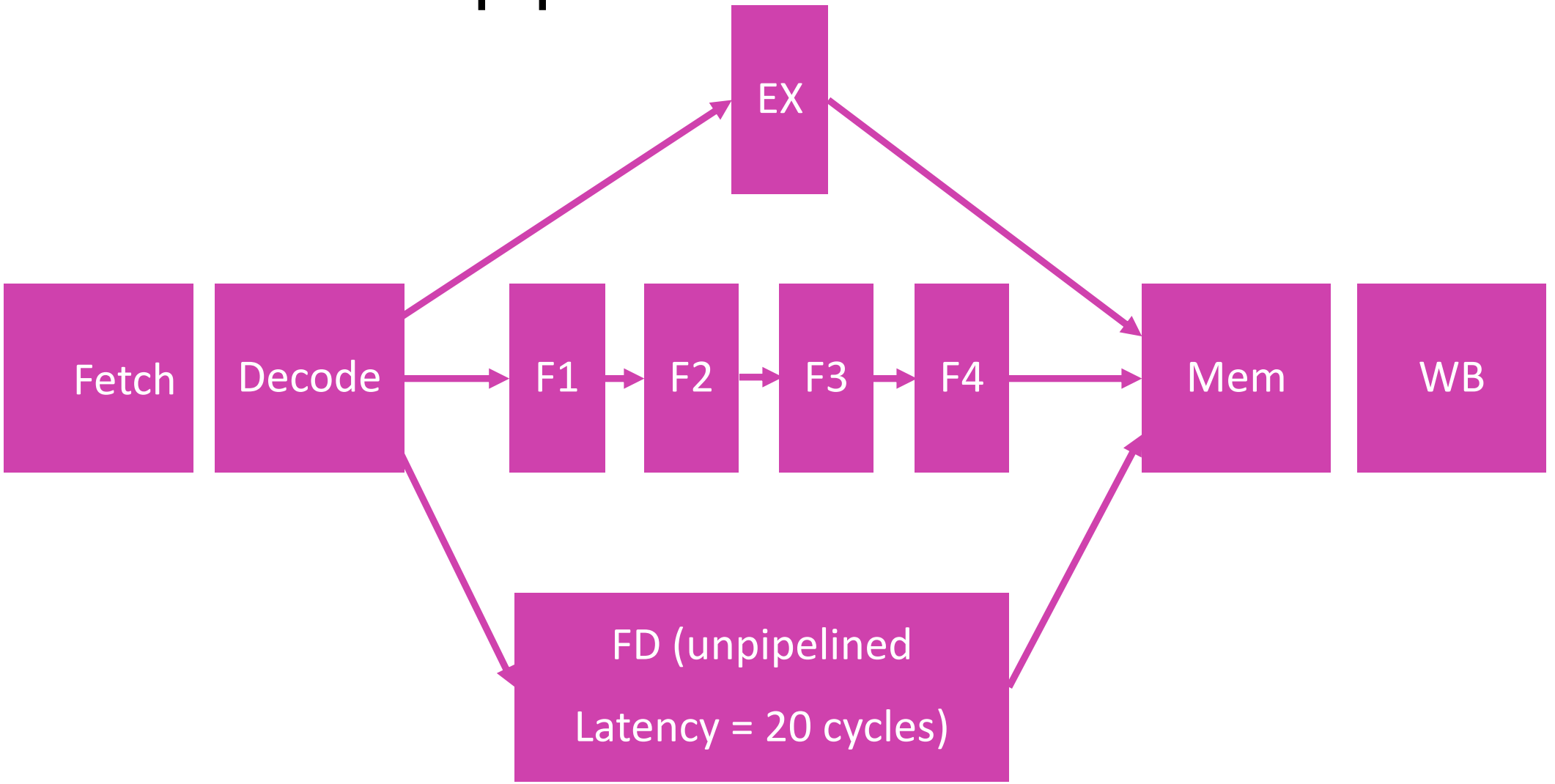
3. Smanjiti zastoje u pipelineu zbog strukturnih i podatkovnih rizika.

Raznoliki pipelineovi i Studija slučaja za Arm10

Diverzificirane pipeline strukture

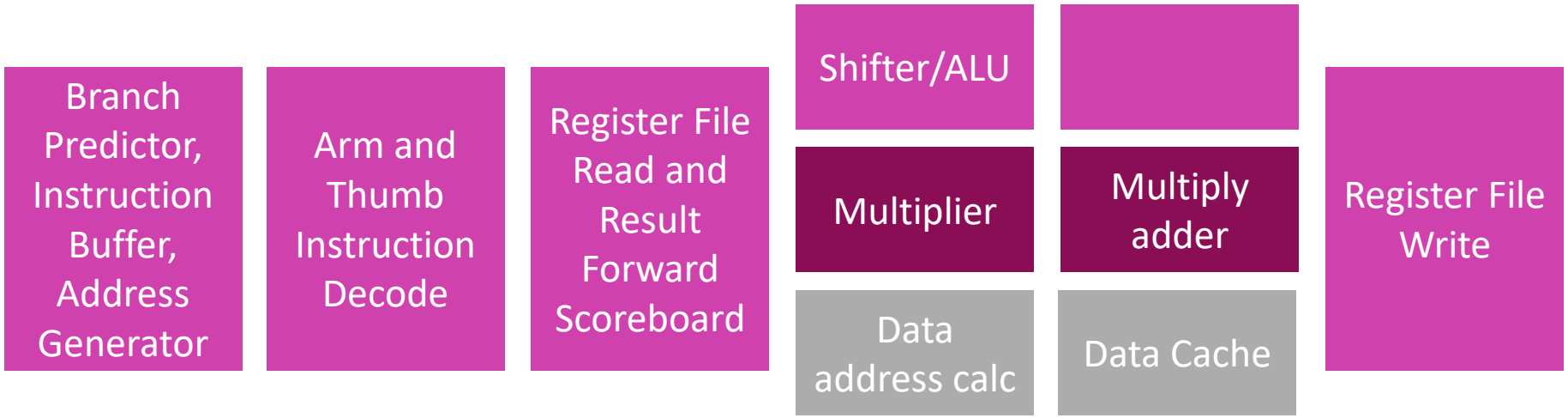
- Nepraktično je zahtijevati da se sve instrukcije izvršavaju u jednom ciklusu.
- Također želimo izbjeći slanje svih instrukcija niz pipeline strukturu.
- Možemo kreirati mnogo (“diverzificirati”) izvršnih pipeline struktura.
- Može li to uvesti nove rizike?

Diverzificirane pipeline strukture



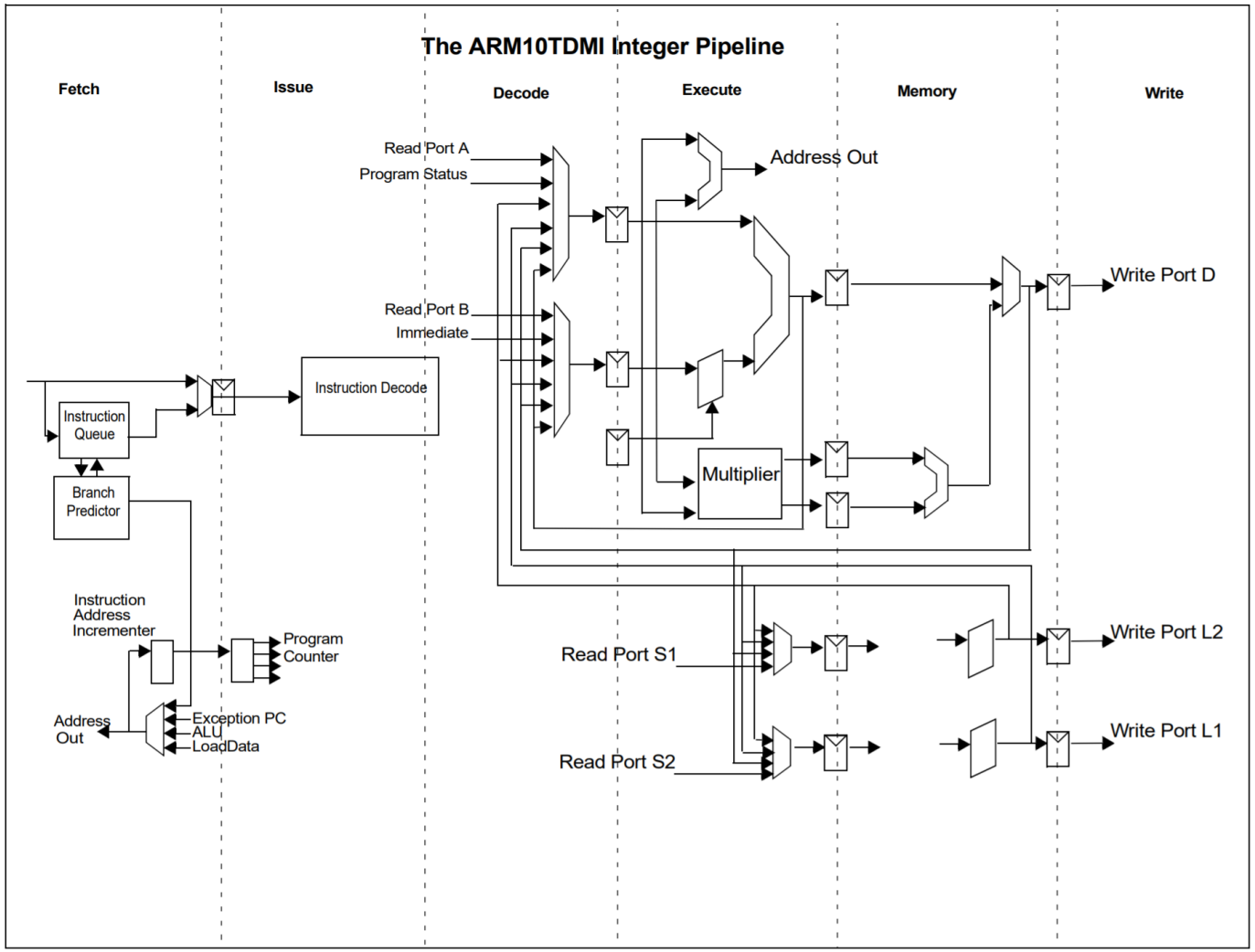
divider

Primjer: Arm10 Pipeline (1999)



parallel in these stages

Arm10 Pipeline



Arm10 Pipeline

- Upute ALU-a ne moraju nužno čekati da se dovrši izvršavanje Load/store u pipelineu load store instrukcija, ako je to moguće napraviti nezavisno, tj.
- Podatkovni rizici
  - Ne možemo dopustiti svim ALU uputama za zaobilaženje instrukcija koje pristupaju memoriji. Moramo poštovati ovisnosti.
  - RAW (i.e., ALU čeka rezultat load instrukcije) ili WAW rizik
- Strukturalni rizici
  - Load/store registri ili pipeline su zauzeti
- Provjeravajte rizike kasnije u sustavu.
  - Korištenje blokada minimizira se provjerom rizika kasno, a ne samo u fazi dekodiranja.

Hvala vam na pažnji!