Protočnost

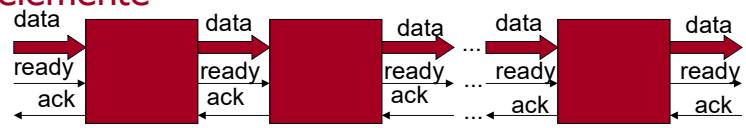
(Pipelining)

Protočnost (pipelining)

- Protočnost je tehnika projektovanja hardvera kojom se uvodi konkurentnost u računarski sistem tako što se neke osnovne funkcije (f) čije se izvršenje često zahteva dele na niz podfunkcija f1, f2, ..., fk, tako da budu zadovoljeni sledeći kriterijumi:
  - Izračunavanje osnovne funkcije f je ekvivalentno sekvencijalnom izračunavanju podfunkcija f1, f2, ..., fk.
  - izlazi prethodne podfunkcije predstavaljaju ulaze za sledeću podfunkciju u nizu podfunkcija koje se izvršavaju
  - Osim razmene podataka izmedju podfunkcija ne postoji nikakva druga zavisnost
  - Može se projektovati hardver za izračunavanje svake podfunkcije
  - Vremena potrebna ovim hardverskim jedinicama da obave individalana izračunavanja su približno jednaka

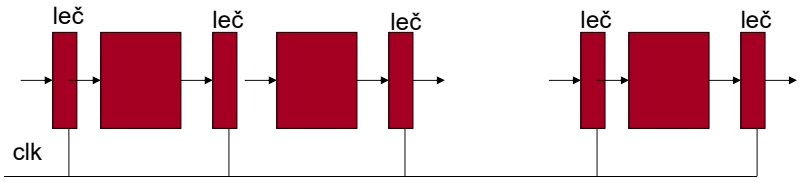
Protočnost (nast.)

- Hardver za izračunavanje bilo koje podfunkcije zove se stepen protočnog sistema (pipeline stage)
- U zavisnosti od načina upravljanja tokom podataka kroz protočni sistem mogu se razlikovati
  - asinhroni protočni sistemi
  - sinhroni protočni sistemi
- Asinhroni model – razmenom podataka izmedju dva susedna stepena upravlja se nekom handshake procedurom. Hardverski stepeni sadrže memorijske elemente



Protočnost (nast.)

- Sinhroni model – razmenom podataka upravlja se pomoću globalnog clk. Hardverski stepeni ne sadrže memorijske elemente. Zato se izmedju stepena ubacuju lečevi.



- svi stepeni su aktivni u svakom klok ciklusu.

Stepen i unosi kašnjenje Ti. Klok perioda protočnog sistema iznosi

•$T=\max\{T1, T2,\ldots, Tk\}+T_L$, gde je T_L kašnjenje koje unosi leč

Protočnost – ilustrativni primer

Stanari zgrade koriste zajedničku vešarnicu za pranje, sušenje i peglanje veša

	30	40	20	30	40	20	30	40	30		
Ana	Pranje	sušenje	peglanje								
Pera				Pranje	sušenje	peglanje					
Mika							Pranje	sušenje	peglanje		
Laza										Pranje	sušenje

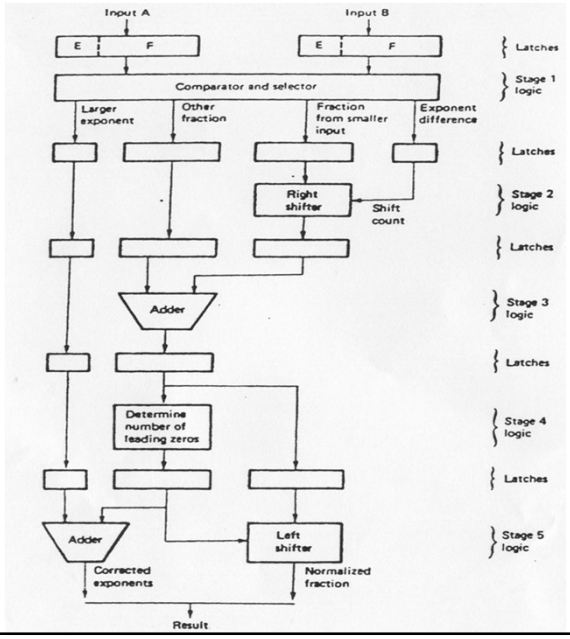
Korišćenjem protočnosti ukupno vreme se može smanjiti

	40	40	40	40	40	40	40	40	40		
Ana	Pranje	sušenje	peglanje								
Pera		Pranje	sušenje	peglanje							
Mika			Pranje	sušenje	peglanje						
Laza				Pranje	sušenje	peglanje					

Primer: projektovanje protočnog FP sabiarača

- $A=a*2^p$, $B=b*2^q$
- korak1: Poredjenje eksponenata p i q da bi se pronašao veći, r=max(p,q) i razlika t=|p-q|.
- korak2: Pomeriti za t mesta u desno mantisu manjeg broja da bi se izjednačili eksponenti pre sabiranja
- korak3: Sabiranje mantisa i dobijanje medjurezultata
- korak4: Odredjivanje broja vodećih nula u sumi, recimo *u*.
- korak5: Pomeranje dobijene sume za *u* mesta u levo da bise dobila normalizovana mantisa i ažuriranje većeg eksponenta: r+u

Protočni sabirač



Protočni sabirač

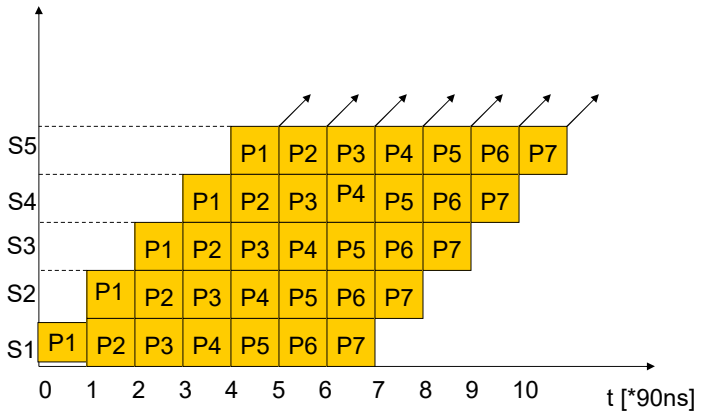
- Neaka su kašnjenja koja unose pojedini stepeni
  - T1=60 ns
  - T2=50 ns
  - T3=80 ns
  - T4=50 ns
  - T5=80 ns
  - T_L=10 ns
- Klok perioda protočnog sistema je T=max{60, 50, 80, 50, 50}+10=90ns
- Vreme potrbno neprotočnom sabiraču da sabere dva FP broja iznosi Tnp=60+50+80+50+80=320ns
- Vreme potrebno protočnom sabiraču da sabere dva FP broja iznosi Tpr=5*90=450ns

Gde je dobit od uvodjenja protočnosti?

- Ako je potrebno sabrati n parova brojeva neprotočnom sabiraču će biti potrebno
  - Tnp=n*320 ns
- a protočnom
  - Tpr=450+(n-1)*90 ns
- Za n=10
  - Tnp=10*320=3200 ns
  - Tpr=450+9*90=1360 ns
- Što je veće n performanse protočnog sistema su bolje. Za dovoljno veliko n ubrzanje protočnog sistema jednako je broju stepena, k.

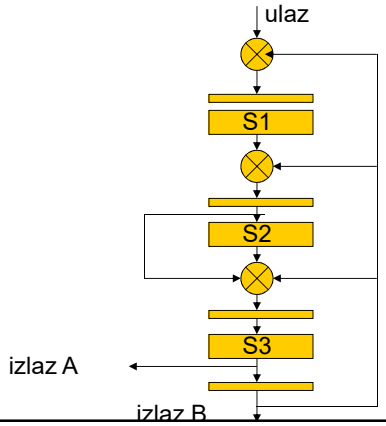
Gantov dijagram

- Prikazuje zauzetost pojedinih stepena u vremenu



Klasifikacija protočnih sistema

- U odnosu na način povezivanja hardverskih stepena:
  - Linearni (kaskadna veza izmedju stepena; sabirač iz prethodnog primera)
  - Nelinearni – pored kaskadnih veza postoje veze izvedene u napred i povratne (u nazad)



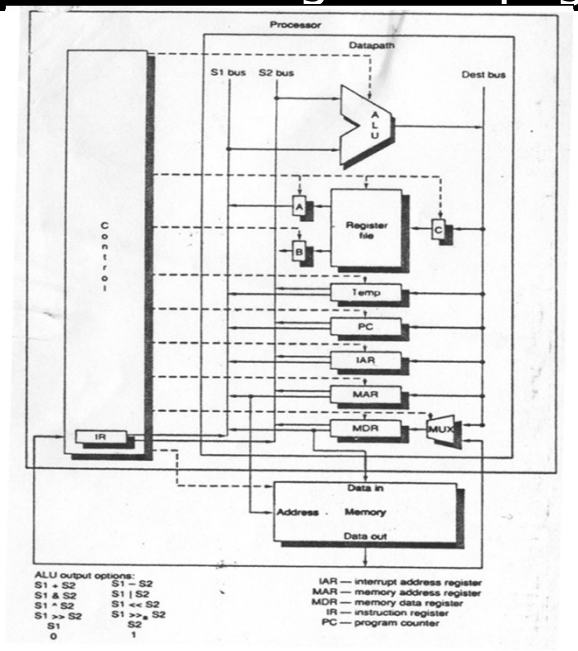
Klasifikacija protočnih sistema

- U odnosu na mogućnosti obrade
  - Jednofunkcijski – protočni sistemi sa fiksno dodeljenom funkcijom (sabirač iz prethodnog primera)
  - Višefunkcijski – mogu obavljati više funkcija u isto ili različitim vremenskim trenucima. Mogu biti
    - Statički
    - Dinamički

- Protočnost se kod savremenih računara koristi na nivou:
  - izvršenja instrukcija
  - izvršenja ALU operacija
  - kod pristupa memoriji

Organizacija protočne staze podataka

RISC arhitektura – globalni pogled



Faze izvršenja

- Svaka integer instrukcija RISC procesora se može obaviti za najviše pet klok ciklusa:

1 Instruction fetch cycle (IF):

```
IR ← Mem[PC]
NPC ← PC + 4
```

2 Dekodiranje instrukcije i pribavljanje operanda (ID):

```
A ← Regs[Rs1];
B ← Regs[Rs2];
Imm ← ((IR16)16##IR 16..31)
```

sign-extended immediate field of IR

Napomena: IR (instrukcioni registar), NPC (next sequential program counter register) A, B, Imm su privremeni registri

Faze izvršenja- nastavak

3 Execution/Effective address cycle (EX):

- **Memory reference:**

```
ALUOutput ← A + Imm;
```

- **Register-Register ALU instruction:**

```
ALUOutput ← A func B;
```

- **Register-Immediate ALU instruction:**

```
ALUOutput ← A op Imm;
```

- **Branch:**

```
ALUOutput ← NPC + Imm;
Cond ← A op 0 (op je relacioni operator definisan kodm instrukcije)
```

Faze izvršenja- nastavak

4 Obraćanje memoriji /okončanje grananja (MEM):

- **Obraćanje memoriji:**

```
LMD ← Mem[ALUOutput]   ili
Mem[ALUOutput] ← B;
```

- **Branch:**

```
if (cond) PC ← ALUOutput else PC ← NPC
```

napomena: LMD (load memory data) register

Faze izvršenja- nastavak

5 Write-back cycle (WB) – upis u registarski fajl:

- **Register-Register ALU instruction:**

$Reg[IR_{16..20}] \leftarrow ALUOutput;$

- **Register-Immediate ALU instruction:**

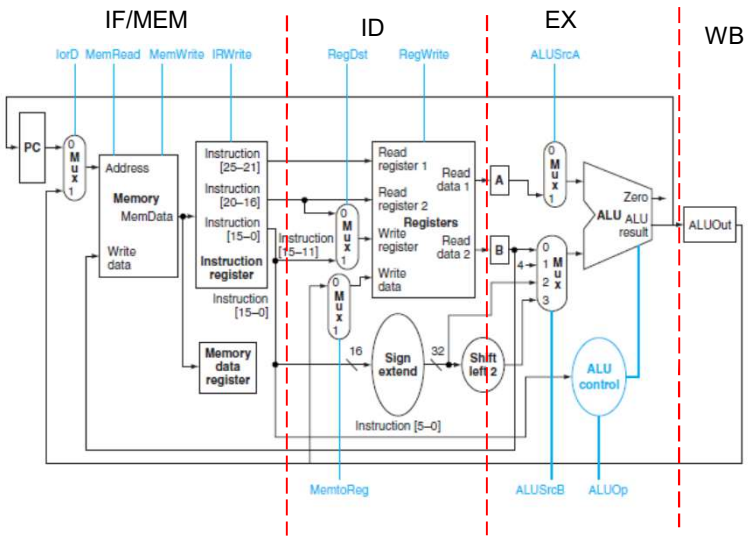
$Reg[IR_{11..15}] \leftarrow ALUOutput;$

- **Load instruction:**

$Reg[IR_{11..15}] \leftarrow LMD;$

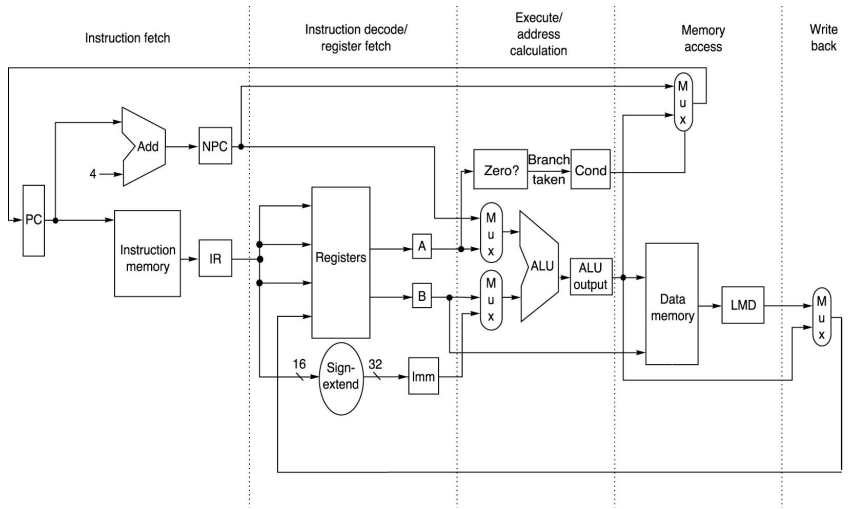
Napomena: LMD (load memory data) register

Struktura staze podataka – bez protočnosti



- Prelazak sa višetaktnog sekvencijalnog procesora na protočni procesor zahteva određene izmene u implementaciji procesora.
  - Umesto jedinstvene memorije za programe i podatke, protočni procesor koristi odvojene memorije za programe i podatke.
  - Ovim se izbegavaju zastoji u radu procesora zbog jedinstvene memorije, koja može biti kritičan resurs sistema.

Struktura staze podataka i tok instrukcije



Protočnost se ostvariti tako što bi se u svakom klok ciklusu pribavila nova instrukcija

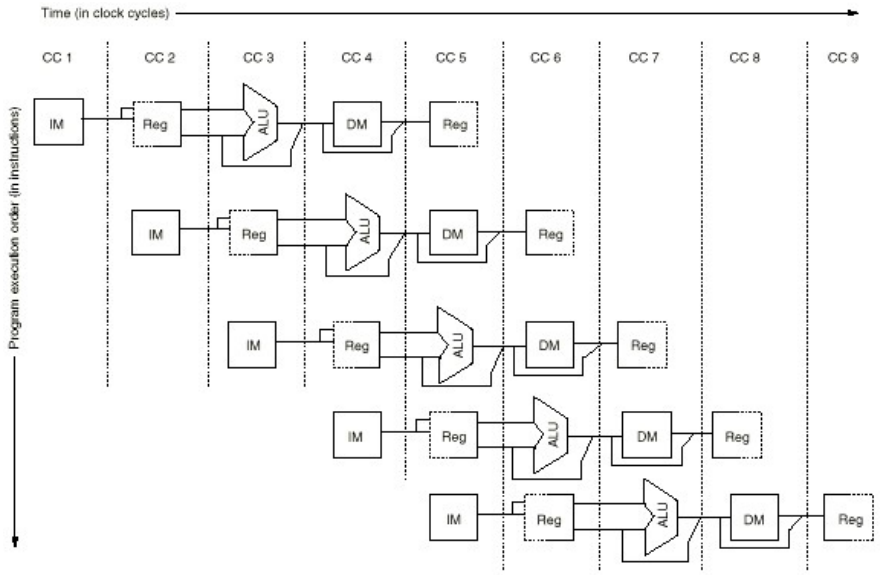


FIGURE 3.3 The pipeline can be thought of as a series of datapaths shifted in time.

Zašto protočnost?

- protočnost ne smanjuje vreme izvršenja pojedinačne instrukcije
  - može dovesti do neznatno dužeg vremenna izvršenja
- Uvođenjem protočnosti se povećava propusnost
  - smanjuje se srednji broj taktova po instrukciji
    - Idealno 1 instr po clk. ciklusu

	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

FIGURE 3.2 Simple DLX pipeline. On each clock cycle, another instruction is fetched and begins its five-cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of a machine that is not pipelined.

Problemi:

- PC se mora inkrementirati u svakom klok ciklusu da bi se pribavila nova instrukcija. Ovo se mora obaviti u IF fazi. U neprotočnoj verziji to se obavlja u MEM fazi.
  - Problem nastupa kod instrukcija grananja koje mogu promeniti sadržaj PC
  - Da li dolazi do grananja ili ne zna se tek na kraju MEM faze
- Nova instrukcija se mora pribaviti u svakom klok ciklusu (u IF fazi).
  - To zahteva da se memoriji pristupa u svakom klok ciklusu.
    - Ni jedna memorija ne može podržati takve zahteve jer jedan memorijski ciklus traje 4 do 20 procesorskih ciklusa
    - Zbog toga se uvode keš memorije
  - različiti stepeni protočnog sistema mogu jednovremeno zahtevati pristup memoriji (pribavljanje instrukcije u IF i pribavljanje operanda u MEM, a ove faze se u vremenu mogu poklapati
  - rešenje je u korišćenju odvojenih keševa za instrukcije i podatke

Problemi (nastavak)

- Registarski fajl se koristi u dva stepena:
  - za čitanje u ID fazi i za upis u WB fazi.
  - To znači da je svakom klok ciklusu potrebno obaviti dva čitanja i jedan upis.
    - Šta ako se čitanje i upis vrše u isti registar?
- Uvodjenje protočnosti u stazu podataka zahteva da vrednosti koje se prosledjuju iz jednog stepena u drugi budu zapamćene u posebnim registrima (lečevima)
  - bilo koja vrednost koja može biti potrebna u kasnijim protočnim stepenima mora biti zapamćena u protočnim registrima koji se umeću izmedju pojedinih stepena i kopirana iz jednog regista u drugi sve dok je to potrebno

Modifikovana staza podtaka

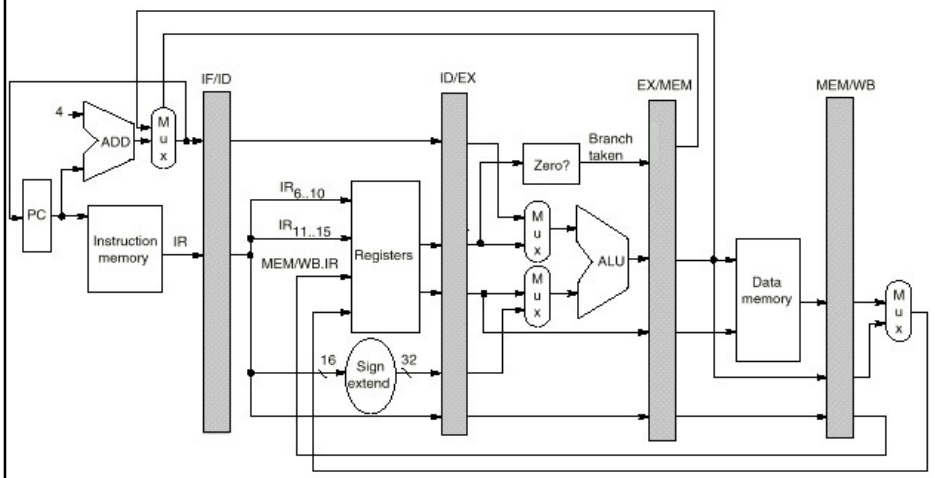


FIGURE 3.4 The datapath is pipelined by adding a set of registers, one between each pair of pipe stages.

Aktivnosti u pojedinim fazama protočne organizacije

Stage	Any instruction		
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC,PC ← (if EX/MEM.cond {EX/MEM.NPC} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.IR$_{6..10}$]; ID/EX.B ← Regs[IF/ID.IR$_{11..15}$]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← (IR$_{16}$)^{16}##IR$_{16..31}$;		
	ALU instruction	**Load or store instruction**	**Branch instruction**
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput← ID/EX.A *op* ID/EX.B; or EX/MEM.ALUOutput ← ID/EX.A *op* ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR← ID/EX.IR EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0; EX/MEM.B← ID/EX.B;	 EX/MEM.ALUOutput ← ID/EX.NPC+ID.EX.Imm; EX/MEM.cond ← (ID/EX.A *op* 0);
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]; or Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs[MEM/WB.IR$_{16..20}$] ← MEM/WB.ALUOutput; or Regs[MEM/WB.IR$_{11..15}$] ← MEM/WB.ALUOutput;	Regs[MEM/WB.IR$_{11..15}$] ← MEM/WB.LMD;	

FIGURE 3.5 Events on every pipe stage of the DLX pipeline.

Hazardi protočnih sistema

- Hazardi su situacije koje sprečavaju da izvršenje instrukcije otpočne u predvidjenom clok ciklusu.
- Hazardi redukuju idealne performanse protočnog sistema (izvršenje jedne instrukcije po klok ciklusu).
- Hazardi se mogu klasifikovati u tri grupe:
  - Strukturni hazardi
  - Hazardi podataka
  - Kontrolni hazardi

Hazardi protočnih sistema (nast.)

- Strukturni hazardi – nastaju zbog jednovremenih zahtva za korišćenje istog hardverskog resursa.
  - Da bi se ovi hazardi izbegli neophodno je duplirati resurse, ako je potrebno.
    - Npr. ALU se u EXE fazi koristi za izračunavanje ALU operacije, a u IF fazi je potreban za inkrementiranje sadržaja PC (programski brojač)
      - Rešenje je dodati poseban sabirač koji će u IF fazi obavljati inkrementiranje PC
    - pristup memoriji se zahteva u IF i MEM fazi
      - zbog toga se koriste odvojene keš memorije za instrukcije i podatke
  - izbegavanje hazarda može se postići zaustavljanjem protočnog sistema, pri čemu se nekim instrukcijama dozvoljava da nastave sa izvršenjem, a neke se zaustavljaju dok se hazard ne otkloni

Hazardi protočnih sistema (nast.)

- Hazardi po podacima –
  - nastupaju zato što je redosled pristupa operandima izmenjen uvodjenjem protočnosti u odnosu na sekvencijalno izvršenje instrukcija
- Kontrolni hazardi –
  - nastupaju zbog zavisnosti u redosledu izvršenja instrukcija (izazivaju ih instrukcije koje mogu promeniti sadržaj PC)
- hazardi mogu izazvati zastoje u protočnom sistemu
  - neke instrukcije se zaustavljaju, a nekima se dozvoljava da nastave sa izvršenjem

Hazardi po podacima

Primer:

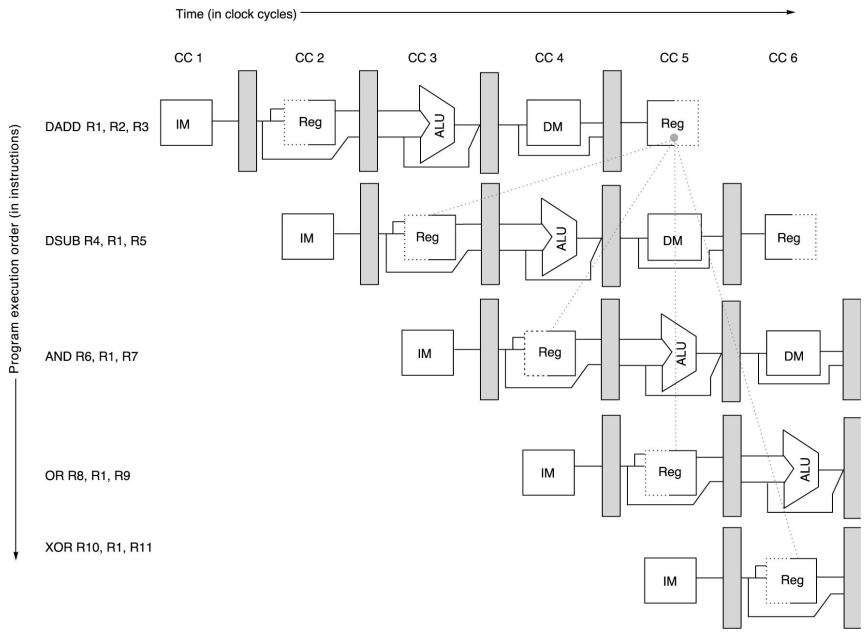
```
ADD   R1, R2, R3
SUB   R4, R1, R5
AND   R6, R1, R7
OR    R8,R1,R9
XOR   R10, R1, R11
```

- Sve instrukcije nakon ADD imaju kao izvorni operand R1 koji je odredišni za ADD

vrednost upisana u R1

ADD	IF	ID	EXE	MEM	WB
SUB		IF	ID		

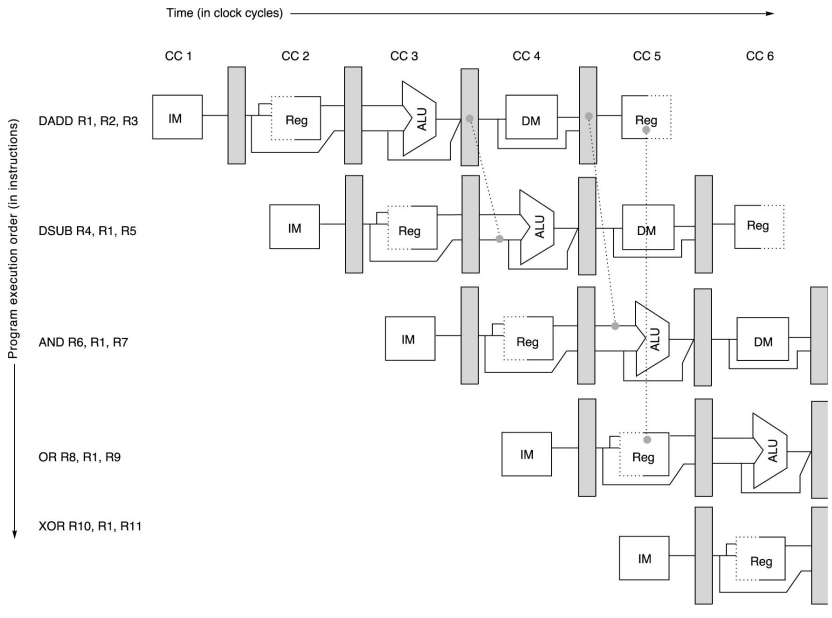
ovde se čita R1



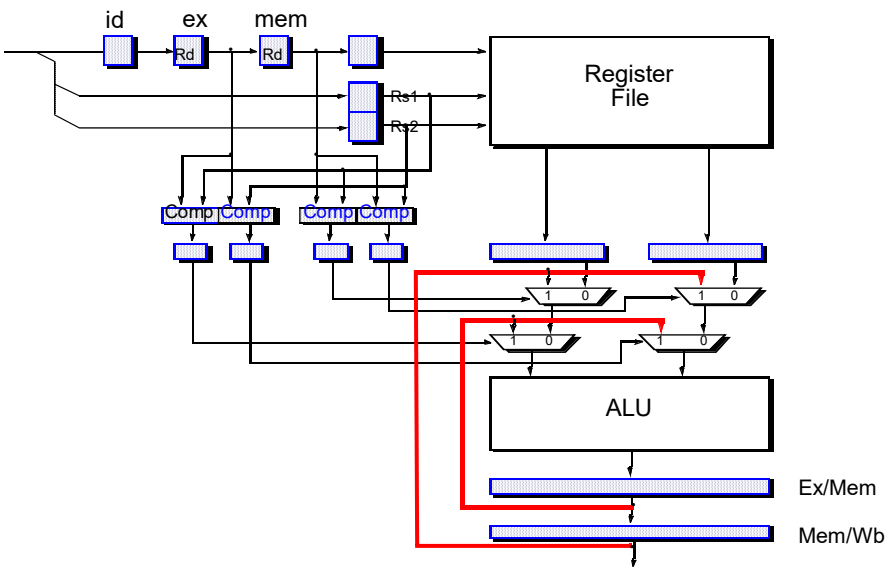
Rešenje – pribavljanje unapred

- Svi hzardi podataka se detektuju u ID fazi
  - Zaustavljanje protpčnog sistema se može izbeći jednostavnom hardverskom tehnikom – pribavljanjem unapred (forwarding, bypassing)
    - Rezultat ALU operacije se uvek vraća na ALU ulazne lečeve
  - Ako hardver za detekciju zavisnosti otkrije da dve susedne instrukcije dele izvor i odredište, upravljačka logika selektuje premošćeni rezultat a ne vrednost učitanu iz registarskig fajla.
  - U razmatranom primeru, premošćeni rezultat potrebno je proslediti i instrukciji AND
  - WB faza ADD instrukcije i ID faza OR se poklapaju.
    - Problem za ovu instrukciju se rešava tako što se upis u registarski fajl vrši u prvoj polovini klok ciklusa a čitanje u drugoj polovini

Premošćavanje



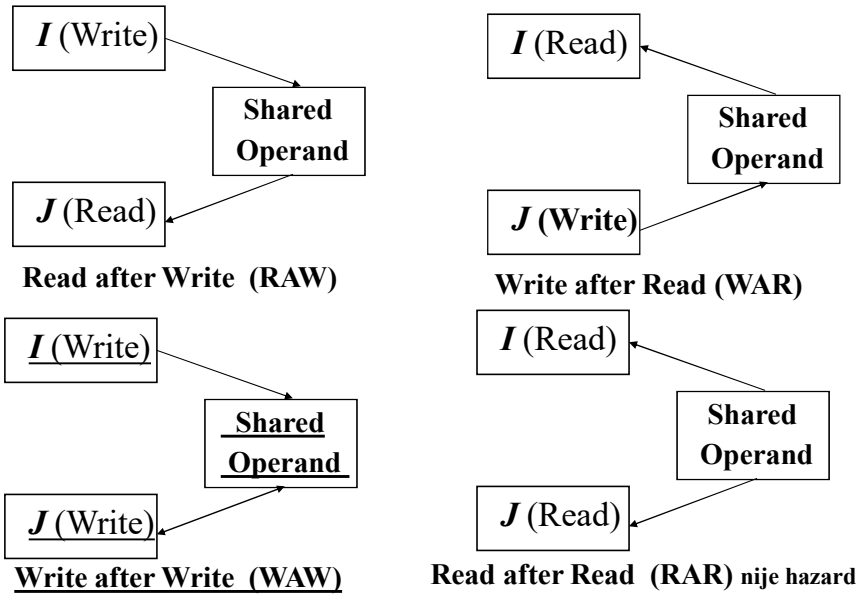
Implementacija prosledjivanja za ALU instrukciju



Klasifikacija hazarda po podacima

- Ako su I i J dve instrukcije, pri čemu I prethodi J, hazardi po podacima se u zavisnosti od redosleda upisa i čitanja mogu klasifikovati u tri grupe:
  - **RAW** (Read After Write) – nastupa kada instrukcija J pokušava da pročita operand pre nego što je instrukcija I obavila upis (najčešći tip hazarda)
  - **WAR** (Write After Read) – nastupa kada instrukcija J pokušava da upiše novu vrednost pre nego šro je instrukcija I obavila čitanje
  - **WAW** (Write After Write) – nastupa kada instrukcija J pokušava da upiše vrednost pre instrukcije I
  - **RAR** nije hazard.
- WAR i WAW su hazardi imenovanja (name dependencies) a RAW su pravi hazardi (true dependencies)

Klasifikacija hazarda



Primer

- Za sledeći niz instrukcija odrediti sve zavisnosti po podacima tipa RAW, WAR i WAW.
  - I0: A = B + C ;
  - I1: C = A – B ;
  - I2: D = A + C ;
  - I3: A = B * C * D ;
  - I4: C = F / D ;
  - I5: F = A - G ;
  - I6: G = F + D

RAW hazard uzrokovan load instrukcijom

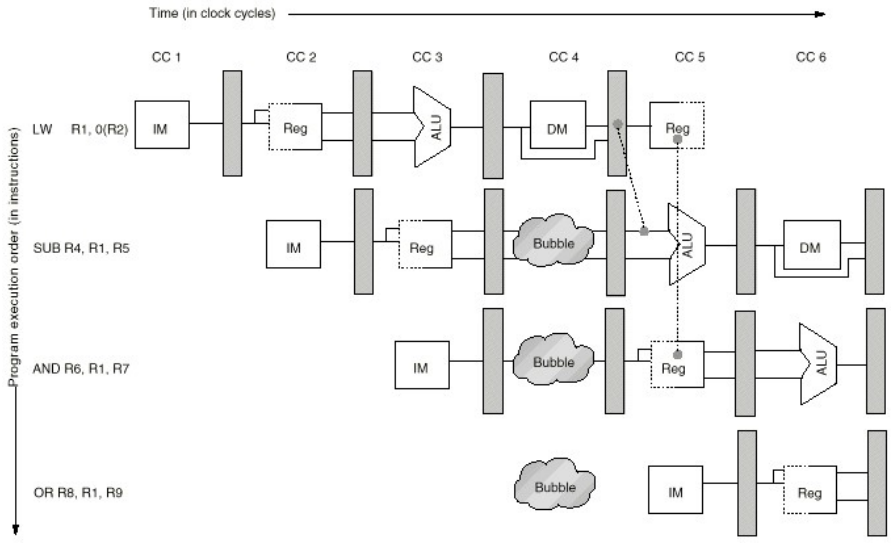
Primer:

```
LW    R1, 0(R2)
SUB   R4, R1, R5
AND   R6, R1, R7
OR    R8, R1, R9
```

podatak u LMD

LW	IF	ID	EXE	MEM	WB
SUB		IF	ID	—	EXE

• Nema načina da se izbegne zaustavljanje protočnog sistema pribavljanjem unapred.
•Svi hazardi se detektuju u ID fazi. Ako hazard postoji protočni sistem se zaustavlja.



Detekcija hazarda (svi hazardi se detektuju u ID fazi)

Situation	Example code sequence	Action

No dependence	`LW R1,45(R2)`
	`ADD R5,R6,R7`
	`SUB R8,R6,R7`
	`OR R9,R6,R7`
		No hazard possible because no dependence exists on R1 in the immediately following three instructions.

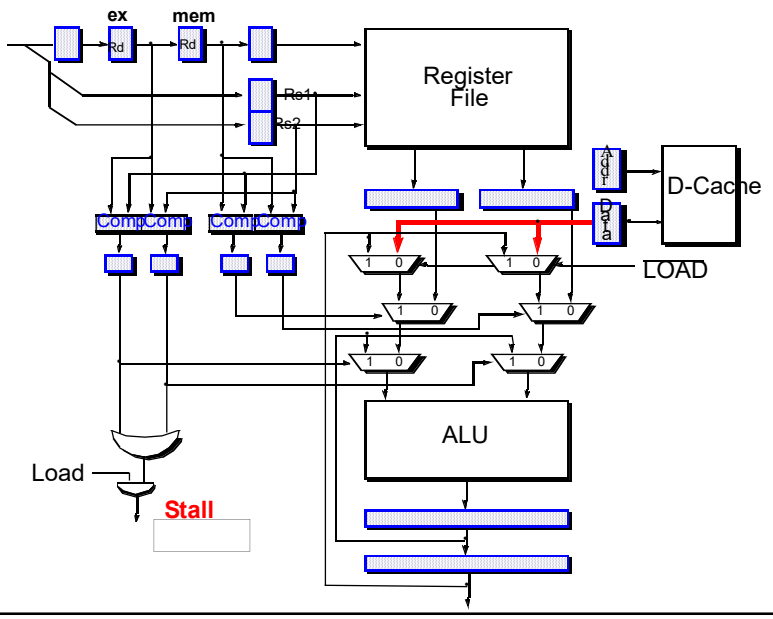
Dependence requiring stall	`LW R1,45(R2)`
	`ADD R5,R1,R7`
	`SUB R8,R6,R7`
	`OR R9,R6,R7`
		Comparators detect the use of R1 in the ADD and stall the ADD (and SUB and OR) before the ADD begins EX.

Dependence overcome by forwarding	`LW R1,45(R2)`
	`ADD R5,R6,R7`
	`SUB R8,R1,R7`
	`OR R9,R6,R7`
		Comparators detect use of R1 in SUB and forward result of load to ALU in time for SUB to begin EX.

Dependence with accesses in order	`LW R1,45(R2)`
	`ADD R5,R6,R7`
	`SUB R8,R6,R7`
	`OR R9,R1,R7`
		No action required because the read of R1 by OR occurs in the second half of the ID phase, while the write of the loaded data occurred in the first half.

Situacije koje hw za detekciju zavisnosti može videti poredeći izvorne i odredišne registre susednih instrukcija

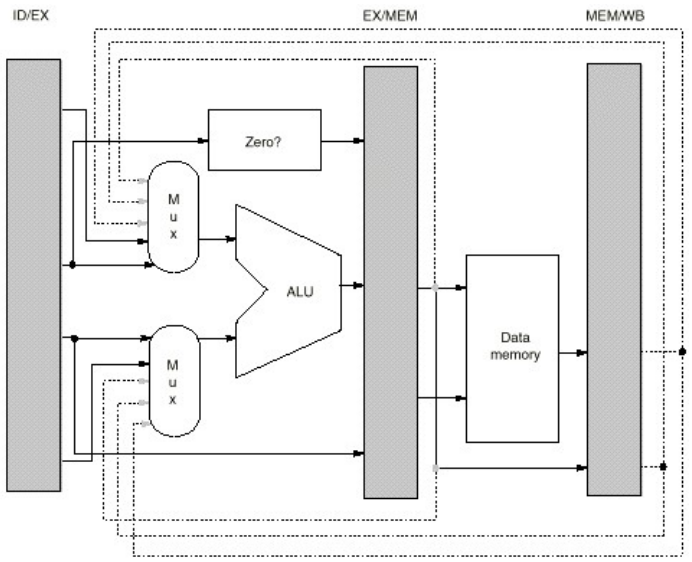
Implementacija prosledjivanja za LOAD instrukciju



svi hazardi se detektuju u ID fazi!

Pipeline register containing source instruction	Opcode of source instruction	Pipeline register containing destination instruction	Opcode of destination instruction	Destination of the forwarded result	Comparison (if equal then forward)
EX/MEM	Register-register ALU	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	$EX/MEM.IR_{16..20} = ID/EX.IR_{6..10}$
EX/MEM	Register-register ALU	ID/EX	Register-register ALU	Bottom ALU input	$EX/MEM.IR_{16..20} = ID/EX.IR_{11..15}$
MEM/WB	Register-register ALU	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	$MEM/WB.IR_{16..20} = ID/EX.IR_{6..10}$
MEM/WB	Register-register ALU	ID/EX	Register-register ALU	Bottom ALU input	$MEM/WB.IR_{16..20} = ID/EX.IR_{11..15}$
EX/MEM	ALU immediate	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	$EX/MEM.IR_{11..15} = ID/EX.IR_{6..10}$
EX/MEM	ALU immediate	ID/EX	Register-register ALU	Bottom ALU input	$EX/MEM.IR_{11..15} = ID/EX.IR_{11..15}$
MEM/WB	ALU immediate	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	$MEM/WB.IR_{11..15} = ID/EX.IR_{6..10}$
MEM/WB	ALU immediate	ID/EX	Register-register ALU	Bottom ALU input	$MEM/WB.IR_{11..15} = ID/EX.IR_{11..15}$
MEM/WB	Load	ID/EX	Register-register ALU, ALU immediate, load, store, branch	Top ALU input	$MEM/WB.IR_{11..15} = ID/EX.IR_{6..10}$
MEM/WB	Load	ID/EX	Register-register ALU	Bottom ALU input	$MEM/WB.IR_{11..15} = ID/EX.IR_{11..15}$

FIGURE 3.19 Forwarding of data to the two ALU inputs (for the instruction in EX) can occur from the ALU result (in EX/MEM or in MEM/WB) or from the load result in MEM/WB.



Prosledjivanje rezultata na ALU ulaze zahteva tri dodatna ulaza na svakom ALU MUX i tri dodtna puta za ove ulaze

Kompajlerska tehnika –zakašnjeni LOAD

- Umesto zaustavljanja protočnog sistema, kompajler može preurediti kod i izbeći zaustavljanje
- Preuredjenjem koda u fazi kompilacije moguće je izbeći zaustavljanje protočnog sistema zbog RAW hazarda uzrokovanog LOAD instrukcijom.
- Kompajler će izbeći generisanje koda sa LOAD instrukcijom iza koje odma sledi instrukcija koja kao izvorni operand ima ono što je odredišni za LOAD.
- Tehnika je poznata pod nazivom zakašnjeni load (delayed load)
- Prvi put predložena 60-tih a široko prihvaćena 80-ih

Primer

a = b + c

d = e - f

Originalni kod sa zaustavljanjem:

```
        LW    Rb,b
        LW    Rc,c
Stall → ADD   Ra,Rb,Rc
        SW    a Ra
        LW    Re,e
        LW    Rf,f
Stall → SUB   Rd,Re,Rf
        SW    d, Rd
```

Preuredjenni kod bez zaustavljanja

```
LW    Rb,b
LW    Rc,c
LW    Re,e
ADD   Ra,Rb,Rc
LW    Rf,f
SW    Ra,a
SUB   Rd,Re,Rf
SW    Rd,d
```