Protočnost

(Pipelining)

Protočnost (pipelining)

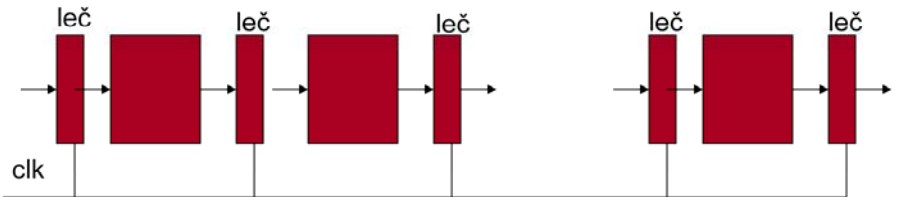
- Protočnost je tehnika projektovanja hardvera kojom se uvodi konkurentnost u računarski sistem tako što se neke osnovne funkcije (f) čije se izvršenje često zahteva dele na niz podfunkcija f1, f2, ..., fk, tako da budu zadovoljeni sledeći kriterijumi:
  - Izračunavanje osnovne funkcije f je ekvivalentno 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					

Aktivnost koja najduže traje određuje kolika će biti perioda sistema. U ovom primeru aktivnost koja najduže traje je sušenje, 40 min, pa će zato za svaku aktivnost biti rezervisano 40 min.

Potrebno je uočiti da se uvođenjem protočnosti vreme obsluživanja pojedinačnog korisnika nije skratilo.

Naprotiv, pojedinačno vreme okončanja posla se produžilo

Uvođenjem protočnosti povećava se propusnost, tj. obim obavljenog posla u jedinici vremena.

Protočnost

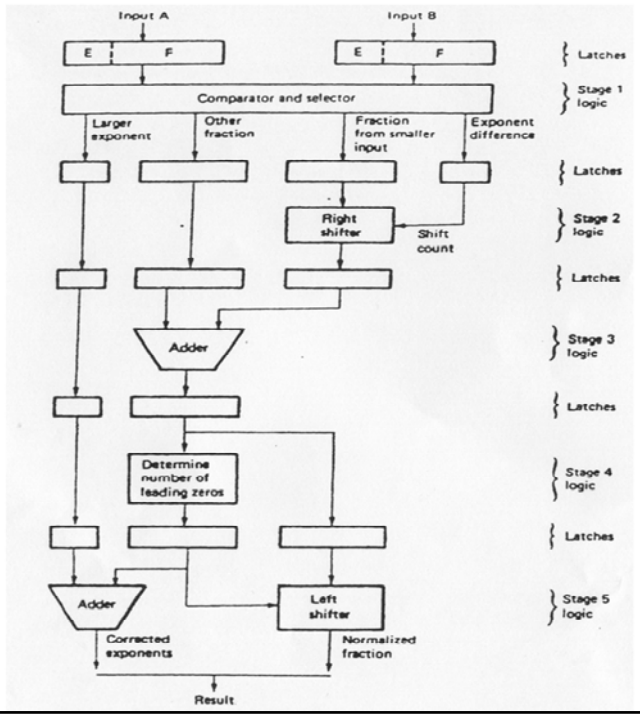
- Potrebno je uočiti da se uvođenjem protočnosti vreme obsluživanja pojedinačnog korisnika nije skratilo.
- Naprotiv, pojedinačno vreme okončanja posla se produžilo
- Uvođenjem protočnosti povećava se propusnost, tj. obim obavljenog posla u jedinici vremena.

Primer2: projektovanje protočnog FP sabiarača

- $A=a*2^p$, $B=b*2^q$, a i b su normalizovane mantise brojeva; p i q su eksponenti
- 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

Kako bismo "ručno" sabrali brojeve $A=0.25*10^3$ i $B=0.999*10^5$? Očigledno je da ne možemo sabrati mantise brojeva koji imaju različite eksponente. Zbog toga moramo prvo da izjednačimo eksponente brojeva. U računarskoj obradi se uvek vrši izjednačavanje u odnosu na veći eksponent. Dakle oba broja ćemo svesti na eksponent 5. To zahteva da mantisu manjeg broja pomerimo u desno za dve pozicije, tj. broj A će sada biti $A=0.0025*10^5$. Sada se mogu sabrati mantise 0.0025 i 0.998. Rezultat je $1.0015*10^5$. Mantisa rezultata nije normalizovana, pa je moramo pomeriti u levo za jedno mesto i korigovati eksponent rezultata: $0.10015*10^6$.

Protočni sabirač



Treba primetiti da se između svakog stepena nalaze lečevi koji moraju da zapamte podatke koji su neophodni za sledeće stepene protočnog sistema

Pošto se projektuje sinhroni protočni sabirač, između svakog stepena protočnog sistema (stge logic na slici) se moraju ubaciti lečevi koji pamte sve podatke koji su neophodni za izvršenje instrukcije u narednom klok ciklusu. Tako, vidimo da se na izlazu iz stepena 1 (Stage 1 logic) pamte vrednosti većeg eksponenta (Larger exponent), mantisa većeg broja (other fraction), mantisa manjeg broja (fraction from smaller input) i razloka eksponenata (exponent difference) koja određuje za koliko pozicija treba pomeriti mantisu manjeg broja u desno.

Nakon što se obavi pomeranje mantise manjeg broja u stepenu 2, pomoću kola koje je označeno sa right shifter, rezulat se upisuje u odgovarajući leč, a vrednosti koje su bile zapamćene u lečevima koji se nalaze između stepena 1 i 2 se sada upisuju u lečeve koji se nalaze između stepena 2 i 3 (vrednosti koje se prenose su očigledno veći eksponent i mantisa većeg broja).

U stepenu 3 se obavlja sabiranje mantisa pomoću kola označenog sa Adder. Rezultat sabiranja se upisuje u odgovarajući leč na izlazu sabirača, a u drugi leč se prenosi vrednost većeg eksponenta.

Kao rezultat sabiranja mantsa može se dobiti vrednost koja nije u normalizovanom obliku (tj. u obliku 0.XYZ već u obliku X.YZ) pa se u stepenu 4 određuje za koliko mesta treba pomeriti dobijenu sumu da bi se obavila normalizacija.

U stepenu 5 se obavlja normalizacija rezultata (kolo Left shifter) i korigovanje eksponenta (kolo Adder).

U stepenu 5 se obavlja normalizacija mantise rezultata korišćenjem kola left shifter

Protočni sabirač

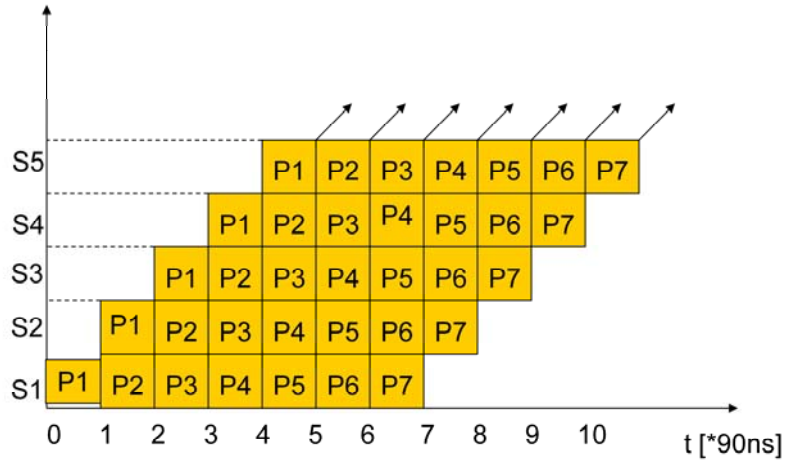
- Neaka su kašnjenja koja unose pojedini stepeni
  - T1=60 ns
  - T2=50 ns
  - T3=80 ns
  - T4=50 ns
  - T5=80 ns
  - T_L=10 ns
- Klok perioda protočnog sistema je T=max{60, 50, 80, 50, 50}+10=90ns
- Vreme potrbno neprotočnom sabiraču da sabere dva FP broja iznosi Tnp=60+50+80+50+80=320ns
- Vreme potrebno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



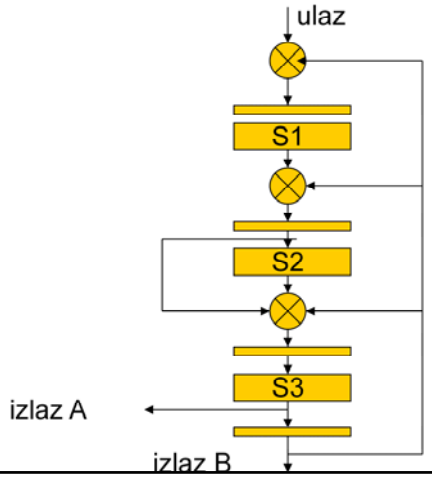
U trenutku 0 se aktivira prvi stepen, S1 protočnog sistema. Nakon 1*90 ns se okončava obrada u stepenu S1 i podaci napreduju u stepen S2, a stepen S1 sada prihvata novi par brojeva koje treba sabrati.

U trenutku 2*90ns prvi par brojeva ulazi u treći stepen S3, drugi paru u stepen S2, a stepen S1 prihvata treći par brojeva, itd. Očigleno je da se različite faze mogu preklapati u izvršenju, pa se protočna obrada naziva i obrada sa preklapanjem.

Nakon 5*90ns na izlazu iz stepena S5 će se pojaviti prvi rezultat. Nadalje će se svaki novi rezultat pojavljivati u narednom clk cilusu (tj. nakon svakih 90ns)

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)



Protčni sabirač koji smo razmatrali je primer linearnog protočnog sistema. Kod ovih sistema postoji samo jedan ulaz i jedan izlaz.

Kod nelinearnih protočnih sistema pored kaskadnih veza, postoje i veze izvedene u napred i povratne veze. Može postojati i više izlaza

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

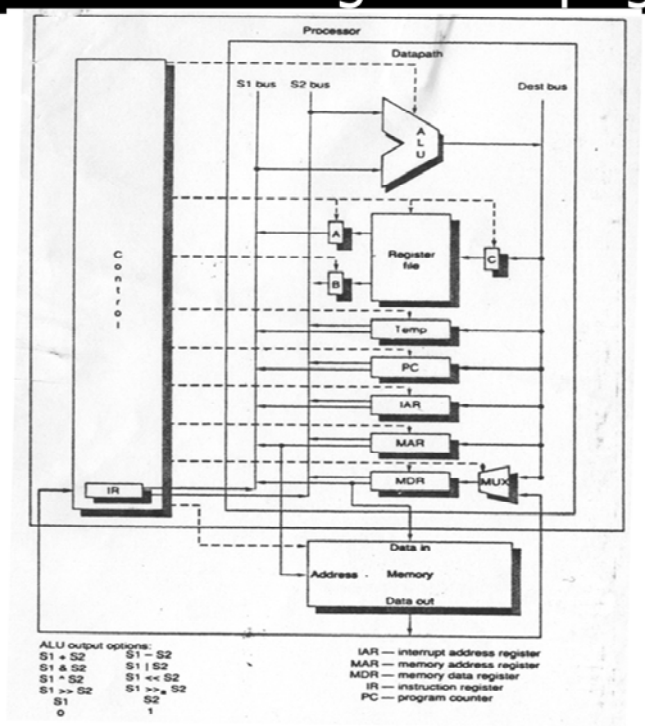
Jednofunkcijski protočni sistem može obavljati samo jednu funkciju (onu za koju je projektovan). To su po pravilu linearni sistemi, tj. između protočnih stepena postoje samo kaskadne veze.

Višefunkcijski protočni sistemi mogu obavljati više funkcija. Kod statičkih višefunkcijskih sistema se tip obrade ne menja za svaki ulazni podatak, nego se sistem prvo konfiguriše za obradu koju treba da obavi. Ako je potrebno obaviti drugi tip obrade, sistem se mora rekonfigurisati. Između dve rekonfiguracije ovi sistemi se ponašaju kako jednofunkcijski. Ako bi se tip obrade često menjao performanse sistema bi bile loše (zbog čestih rekonfiguracija)

Kod dinamičkih višefunkcijskih protočnih sistema se tip obrade može menjati za svaki ulazni podatak. Očigledno je da je u ovom slučaju na ulaz protočnog sistema sem podataka neophodno dovesti i informaciju o tipu obrade koja se zahteva.

Organizacija protočne staze podataka

RISC arhitektura – globalni pogled



Kod RISC procesora se tehnika protočnosti koristi kod izvršenja instrukcija.

Slika prikazuje globalnu strukturu RISC procesora.

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 ← ((IR15)^16##IR 15..0)
```

sign-extended immediate field of IR

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

Kod integer instrukcija (instrukcija koje rade sa celobrojnim podacima) svaka od pet faza izvršenja instrukcija se obavlja u jednom klok ciklusu. Videćemo da EXE faza kod instrukcija koje rade sa podacima u pokretnom zarezu može trajati više klok ciklusa. Za sada razmatramo samo izvršenje integer instrukcija.

U faci prbavljanja instrukcije (IF faza) se na osnovu sadržaja programskog brojača, PC, pristupa memoriji, čita se instrukcija i smešta u poseban registar procesora, instrukcioni reginstar (IR). Zatim se inkrementira sadržaj programskog brojača i pamti u registar NPC (new program counter). Pravi sadržaj programskog brojača je poznat tek na kraju MEM faze kada se okončavaju instrukcije grananja. Tada će registar PC biti postavljen na vrednost NPC ili na adresu skoka ako je bila u pitanju naredba grananja kod koje je uslov grananja zadovoljen, ili na adresu skoka za slučaj da je bila u pitanju naredba bezuslovnog skoka.

U fazi dekodiranja (ID) se pored samog procesa dekodiranja tokom kojeg se ustanovljava koja instrukcija je pročitana iz memorije, vrši i pribavljanje potencijalnih operanada za pribavljenu instrukciju. Čitaju se sardžaji registara na koje ukazuju bitovi 25..21 (RS1), 20..16 (RS2) u pribavljenoj instrukciji koja se nalazi u IR i smeštaju u registre A i B, izdvaja se neposredni operand znakovnim proširivanjem bitova 15..0, dobija se 32-bitna vrednost i smešta u registar Imm.

Na ovaj način su podaci pripremljeni za fazu izvršenja. Podatak koji je pripremljen a nije potreban se jednostavno neće koristiti.

Prve dve faze su iste za sve instrukcije jer dekodiranje još nije obavljeno.

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)

Nakon okončanog dekodiranja, zna se koja je instrukcija pribavljena iz memorije. U zavisnosti od pribavljene instrukcije razlikuju se aktivnost.:

• ako je u pitanju Load ili Store instrukcija, u ovoj fazi se izračunava efektivna adresa podatka koji treba da bude pročitan ilu upisan iz/u memoriju. Efektivna adresa se dobija sabiranjem vrednosti koje se nalaze u registrima A i Imm i smešta se u registar ALUoutput.

• ako je pribavljena ALU isntrukcija R tipa, operandi se nalaze u registrima A i B i obavlja se operacija func koja je definisana kodom operacije i poljem func u formatu instrukcije i smešta u registar ALUoutput.

• ako je pribavljena ALU instrukcija I tipa, tada je drugi operand neposredni operand i nalazi se u registru Imm. Tip obrade je određen sadržajem polja kod operacije u formatu instrukcije. Rezultat se i smešta u registar ALUoutput.

• ako je u pitanju naredba grananja vrši se testiranje uslova, tj. sadržaj regitra A se poredi na nulu (=, <, >, ≤, ≠, ≥ u zavisnosti od koda operacije naredbe grananja) i izračunava se potencijalna adresa skoka sabiranjem sadržaja registra NPC i Imm koja se pamti u registar ALUoutput. Rezultat poređenja se pamti registru Cond (to je jednobitni indikator koji se postavlja na 1 ako je uslov zadovoljen).

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

U MEM fazi su aktivne samo instrukcije koje se obraćaju memoriji (Load i Store) i definitivno se doznaje novi sadržaj programskog brojača.

•U slučaju Load instrukcije, pristupa se memoriji po adresi koja je izračunata u prethodnoj fazi i nalazi se u registru ALUoutput. Pročitani podatak se upisuje u registar LMD (Load Memory Data)

• U slučaju Store instrukcije, vrednost koju treba upisati u memoriju nalazi se u registru B. Upis se obavlja po adresi koja se nalazi u registru ALUoutput.

• Ako je u pitanju naredba grananja, tada: ako je sadržaj registra Cond=1, PC se postavlja na vrednost adrese skoka koja se nalazi u registru ALUoutput. U protivnom, novi sadržaj PC je jednak NPC.

Faze izvršenja- nastavak

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

- **Register-Register ALU instruction:**

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

- **Register-Immediate ALU instruction:**

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

- **Load instruction:**

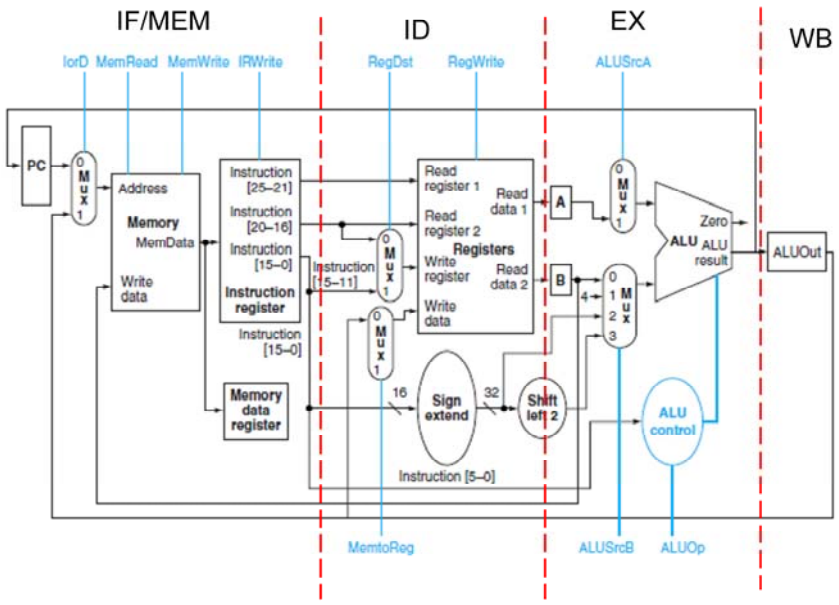
  $Reg[IR_{20..16}] \leftarrow LMD;$

Napomena: LMD (load memory data) register

Poslednja faza u izvršenju je faza upisa u registarski fajl (WB). U ovoj fazi se rezultat izvršenja ALU operacije upisuje u registarski fajl. Ako je u pitanju Load instrukcija vrednost iz LMD registra se upisuje u registarski fajl.

Pozicija odredišnog registra je poznata nakon dekodiranja.

Struktura staze podataka – bez protočnosti



Slika prikazuje koji resursi staze podataka se koriste u pojedinim fazama izvršenja instrukcije. Protočnost još uvek nije uvedena.

Očigledno je da se isti resursi koriste u IF i MEM fazi, i to je memorija. U IF fazi se pristupa memoriji radi čitanja instrukcije, a u MEM fazi radi čitanja ili upisa podatka.

Da bi se uvela protočnost u stazu podataka neophodno je da se resursi koji se koriste u više različitih faza izvršenja instrukcije multipliciraju.

U našem slučaju neophodno je razdvojiti memoriju za instrukcije i memoriju za podatke. Zaista, kod savremenih procesora postoje odvojene keš memorije za instrukcije i podatke.

Takođe, ALU se u IF fazi koristi za inkrementiranje sadržaja programskog brojača a u EXE fazi za izračunavanje ALU operacije ili efektivne adrese. Zbog toga je neophodno dodati poseban sabirač koji će se u IF fazi koristiti da inkrementira sadržaj programskog brojača.

Registarskom fajlu se pristupa u ID fazi i WB fazi. Koriste se dva porta za čitanje i jedan za upis u registarski fajl, pri čemu se upis obavlja u prvoj polovini clk ciklusa a čitanje u drugoj.

Prelazak na protočnu stazu podataka

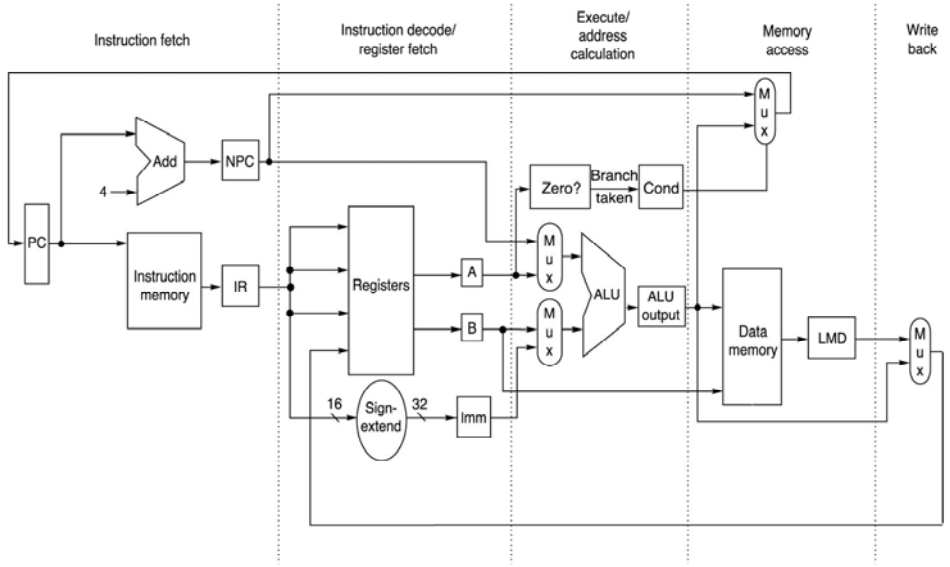
- Prelazak sa višetaktnog sekvencijalnog procesora na protočni procesor zahteva određene izmene u implementaciji procesora.
- Da bi se uvela protočnost u stazu podataka neophodno je da se resursi koji se koriste u više različitih faza izvršenja instrukcije multipliciraju.
  - U našem primeru, očigledno je da se isti resursi koriste u IF i MEM fazi, i to je memorija.
    - U IF fazi se pristupa memoriji radi čitanja instrukcije, a u MEM fazi radi čitanja ili upisa podatka.
    - Zbog toga je neophodno je razdvojiti memoriju za instrukcije i memoriju za podatke.
      - Kod savremenih procesora postoje odvojene keš memorije za instrukcije i podatke.
  - Takođe, ALU se u IF fazi koristi za inkrementiranje sadržaja programskog brojača a u EXE fazi za izračunavanje ALU operacije ili efektivne adrese.
    - Zbog toga je neophodno dodati poseban sabirač koji će se u IF fazi koristiti da inkrementira sadržaj programskog brojača.

Da bi se uvela protočnost u stazu podataka neophodno je da se resursi koji se koriste u više različitih faza izvršenja instrukcije multipliciraju zato što se različite faze izvršenja različitih instrukcija mogu preklapati (pogledati Gantov dijagram na slajdu 11)

U našem primeru, očigledno je da se isti resursi koriste u IF i MEM fazi, i to je memorija. Kada se uvede protočnost, dok je jedna instrukcija u, recimo, MEM fazi, neka kasnija instrukcija može biti u svojoj IF fazi, a ove faze zahtevaju korišćenje istog resursa jednovremeno, što nije moguće. Problem može da se reši tako što će se jednoj od instrukcija zabraniti pristup memoriji dok druga ne okonča pristup, ili tako što će se duplirati rsursi i to je pristup koji se koristi kod savremenih RISC procesora: Kod savremenih procesora postoje odvojene keš memorije za instrukcije i podatke.

Struktura staze podataka i tok instrukcije

Modifikovana staza podataka sa razdvojenim memorijama za instrukcije i podatke i dodatim sabiračem u IF fazi



Na slici je prikazana modifikovana staza podataka u kojoj su odvojene memorije za instrukcije i podatke i dodat sabirač koji obavlja inkrementiranje sadržja PC u IF fazi. Sa ovako modifikovanom stazom podataka smo se pripremili za uvođenje protočnosti kod izvršenja instrukcija. (To nije sve, videćemo da je potrebno još puno stvari rešiti) Resursi koji se koriste u pojedinim fazama izvršenja se stepeni protočnog sistema.

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

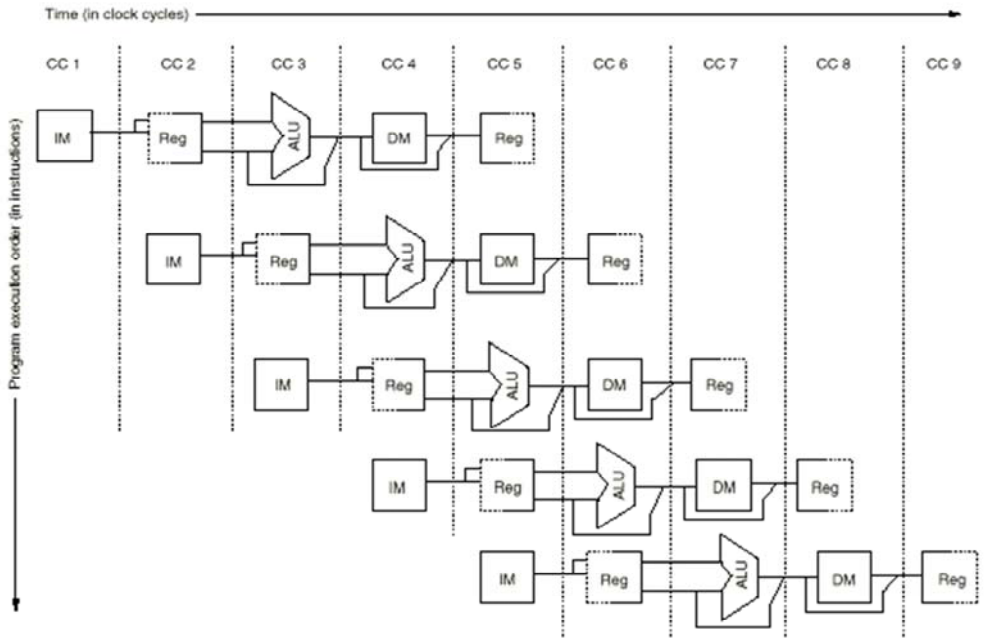


FIGURE 3.3 The pipeline can be thought of as a series of datapaths shifted in time.

Koncepcijski, protočnost u izvršenje instrukcija bismo mogli uvesti tako što bi u svakom clk ciklusu pribavili novu instrukciju: u CC1 pribavljamo prvu instrukciju i to je njena IF faza, u CC2 prva instrukcija prelazi u sledeću fazu izvršenja (ID fazu u kojoj se pristupa registarskom fajlu Reg), a oslobađa se prvi stepen koji sada može da zauzme sledeća instrukcija. U CC3 prva instrukcija napreduje u EXE fazu, druga u ID fazu, a treća instrukcija se pribavlja, itd.

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.

Uvođenje protočnosti ne smanjuje vreme izvršenja pojedinačne instrukcije, već smanjuje vreme izvršenja niza instrukcija, tj. povećava se propusnost sistema. Npr. ako se ne korsiti protočnost i potrebno je izvršiti 10 instrukcija, to će zahtevati 50 clk ciklusa (po 5 za svaku instrukciju)

Ako koristimo protočnost, onda će za izvršenje 10 instrukcija biti potrebno 5+(10-1)=14 clk ciklusa (5 clk ciklusa dok se prva instrukcija ne izvrši, a nakon toga će se u svakom narednom klok ciklusu okončati po jedna instrukcija)

Primetimo da su u petom klok ciklusu svi stepeni protočnog sistema aktivni i da se jednovremeno izvršava 5 instrukcija koje se nalaze u različitim fazama izvršenja.

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

Uvođenje protočnosti u izvršenje instrukcija dovodi do problema kojih nema kod sekvencijalnog izvršenja.

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

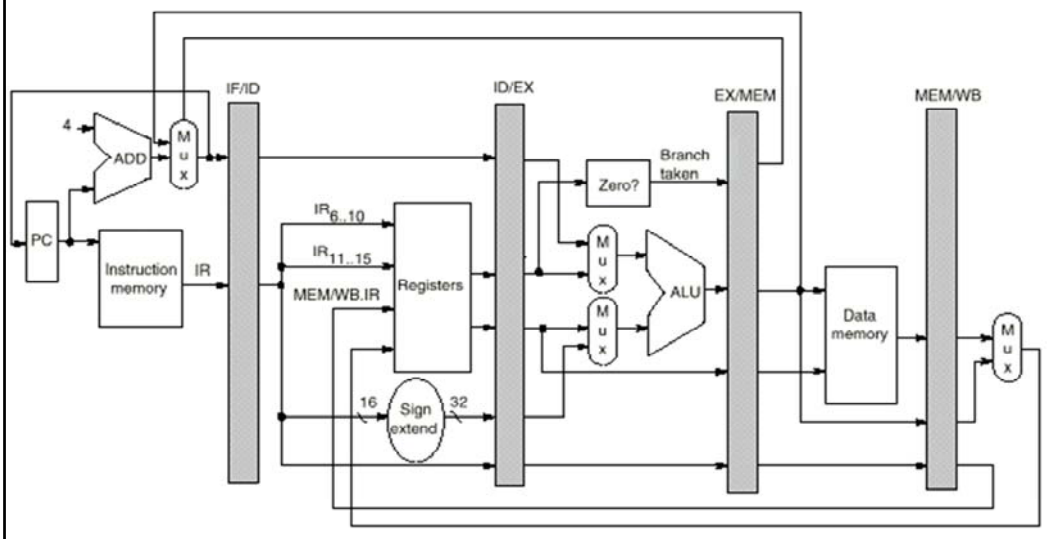


FIGURE 3.4 The datapath is pipelined by adding a set of registers, one between each pair of pipe stages.

Slika prikazuje modifikovanu stazu podataka kojoj su pridodati registri između susednih stepena. Registri su označeni imenom stepena između kojih se nalaze: tako, IF/ID označava regisar koji se nalazi između stepena IF i ID, registar ID/EX se nalazi između stepena ID i EX, itd. Kako instrukcija u svom izvršenju napreduje kroz protočni sistem, tako se svi podaci potrebni za njeno izvrenje premeštaju iz jednog protočnog registra u drugi. Tako, npr. podaci koji se nalaze u registru IF/ID i ono što je obavljeno u ID fazi se pamti u sledećem klok ciklusu u registar ID/EX. Ti podaci će se u sledećem klok ciklusu naći u registru EX/MEM, itd.

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←Regs{IF/ID.IR$_{25..21}$]; ID/EX.B ←Regs[IF/ID.IR$_{20..16}$]; ID/EX.NPC←IF/ID.NPC; ID/EX.IR ←IF/ID.IR; ID/EX.Imm←(IR$_{15}$)^{16} ##IR$_{15..0}$;		
	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$_{15..11}$]← MEM/WB.ALUOutput; or Regs[MEM/WB.IR$_{20..16}$]← MEM/WB.ALUOutput;	Regs[MEM/WB.IR$_{20..16}$] ← MEM/WB.LMD;	

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

Aktivnosti po pojedinim fazama su veoma slične kao kod neprotočne verzije staze podataka. Razlika je što se koriste lečevi između svakog stepena.
Lečevi su označeni imenima stepena između kojih se nalaze: IF/ID, ID/EX, EX/MEM i MEM/WB. Svaki protočni registar se sastoji iz više polja (komponenti). Ekstenzija iza naziva protočnog registra je polje koje odgovara registru iz neprotočne organizacije procesora. Tako, npr IF/ID.IR ← Mem[PC] označava se se u IF/ID registar u polje IR smešta instrukcija iz memorije sa adrese na koju ukazuje programski brojač (PC).