Paralelni sistemi

Paralelizam na nivou instrukcija

Da se podsetimo....

- Protočno izvršenje instrukcija:
  1. Instruction fetch cycle (IF) – pribavljanje istrukcije
  2. Dekodiranje instrukcije i pribavljanje operanada (ID)
  3. Izvršenje / izračunavanje efektivne adrese (EXE)
  4. Obraćanje memoriji /okončanje grananja (MEM)
  5. Upis rezultata u registarski fajl (WB)

- Protočno izvršenje instrukcija

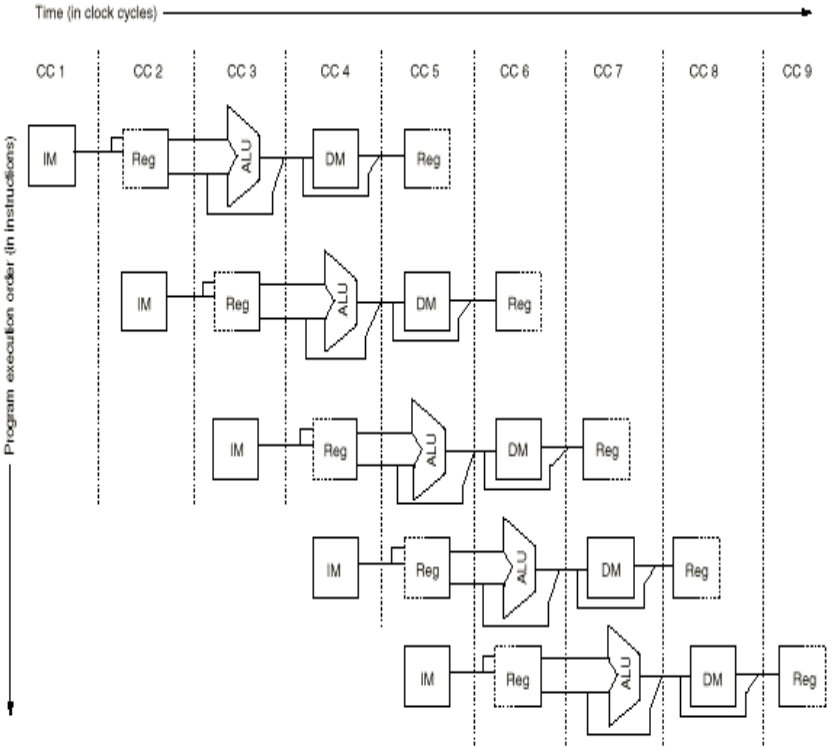
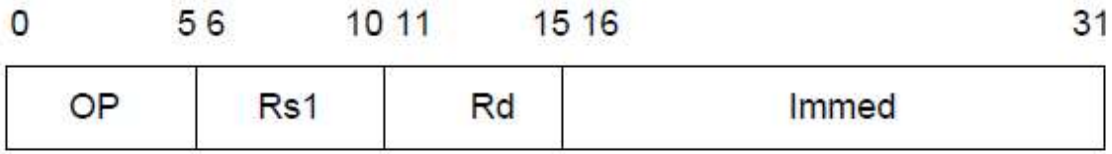


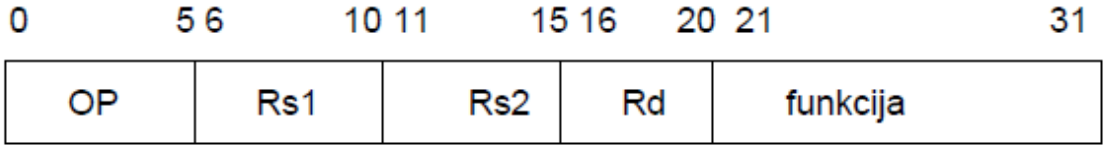
FIGURE 3.3 The pipeline can be thought of as a series of datapaths shifted in time.

Formati instrukcija

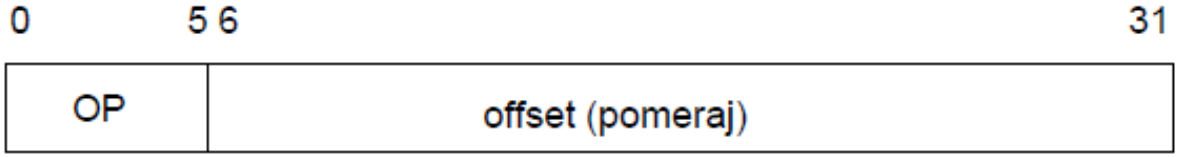
I format (load, store, branch, ALU operacije sa neposrednim operandom)



R format (ALU operacije tipa registar-u registar)



J format – jump instrukcija



Staza podataka

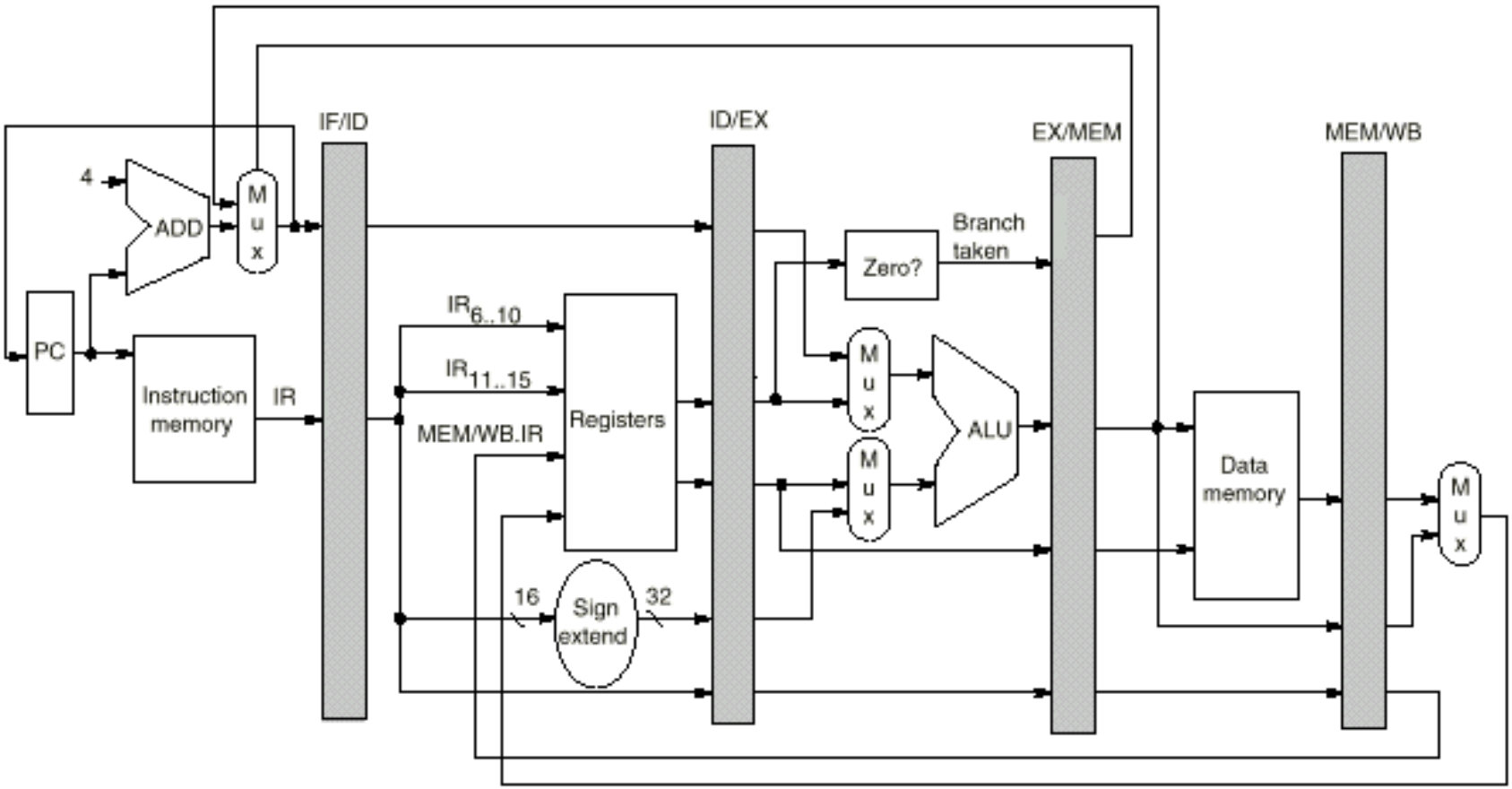


FIGURE 3.4 The datapath is pipelined by adding a set of registers, one between each pair of pipe stages.

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.

Primeri instrukcija

Rd Imm Rs1

- LW R1, 30 (R2) dejstvo R1← Mem[30 +[R2]]
- SW 500(R4), R3 dejstvo Mem[500+[R4]]←[R3]
- ADDI R1,R2,#3 dejstvo R1← [R2]+3
- BEQZ R1, ime dejstvo if R1=0 then PC←PC+ime

Rd Rs1 Rs2

- ADD R1,R2,R3 dejstvo R3 ← [R1]+[R2]

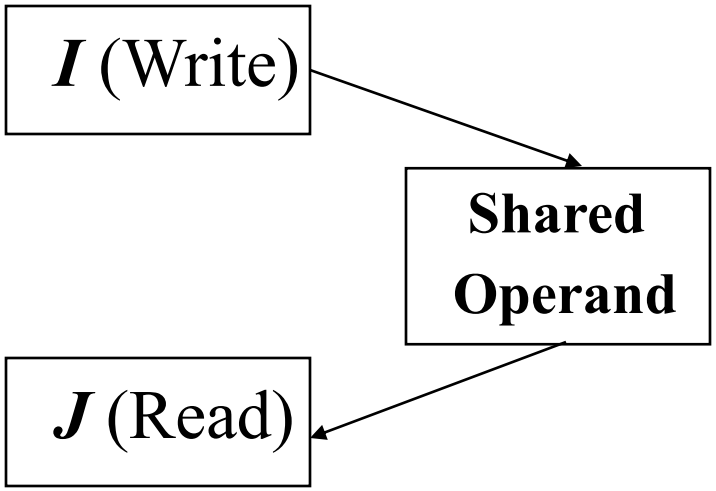
Problemi – hazardi protočnih sistema

- 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 – nastaju zbog jednovremenih zahtva za korišćenjem istog hardverskog resursa
  - 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 zaustavljanje protočnog sistema (nekim instrukcijama se dozvoljava da produže sa izvršenjem)

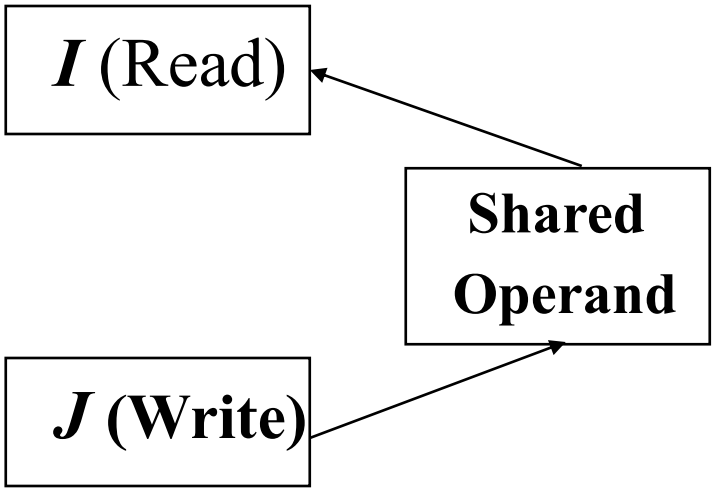
Klasifikacija hazarda po podacima

- 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 što 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)

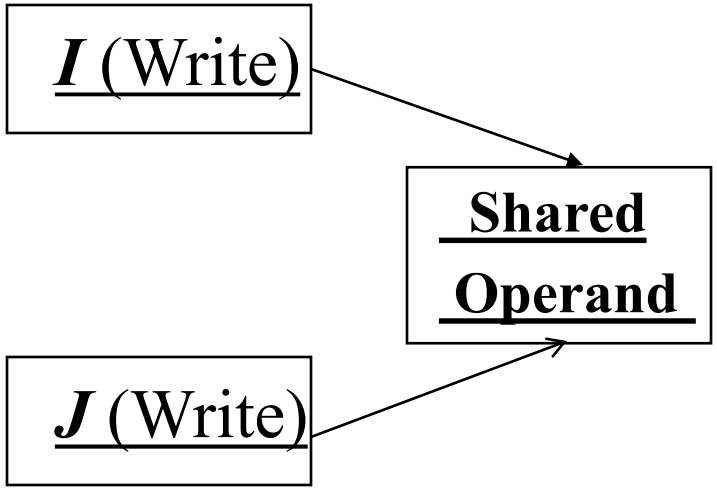
Klasifikacija hazarda



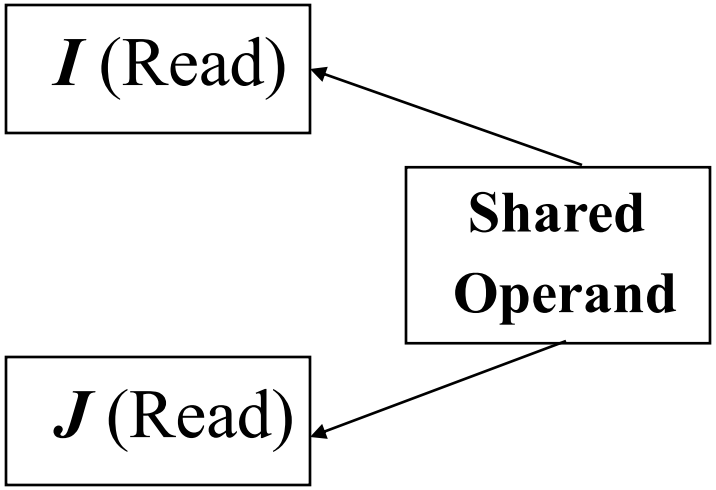
Read after Write (RAW)



Write after Read (WAR)



Write after Write (WAW)



Read after Read (RAR) nije hazard

RAW Hazardi

ADD R1, R2, R3

SUB R4, R1, R5

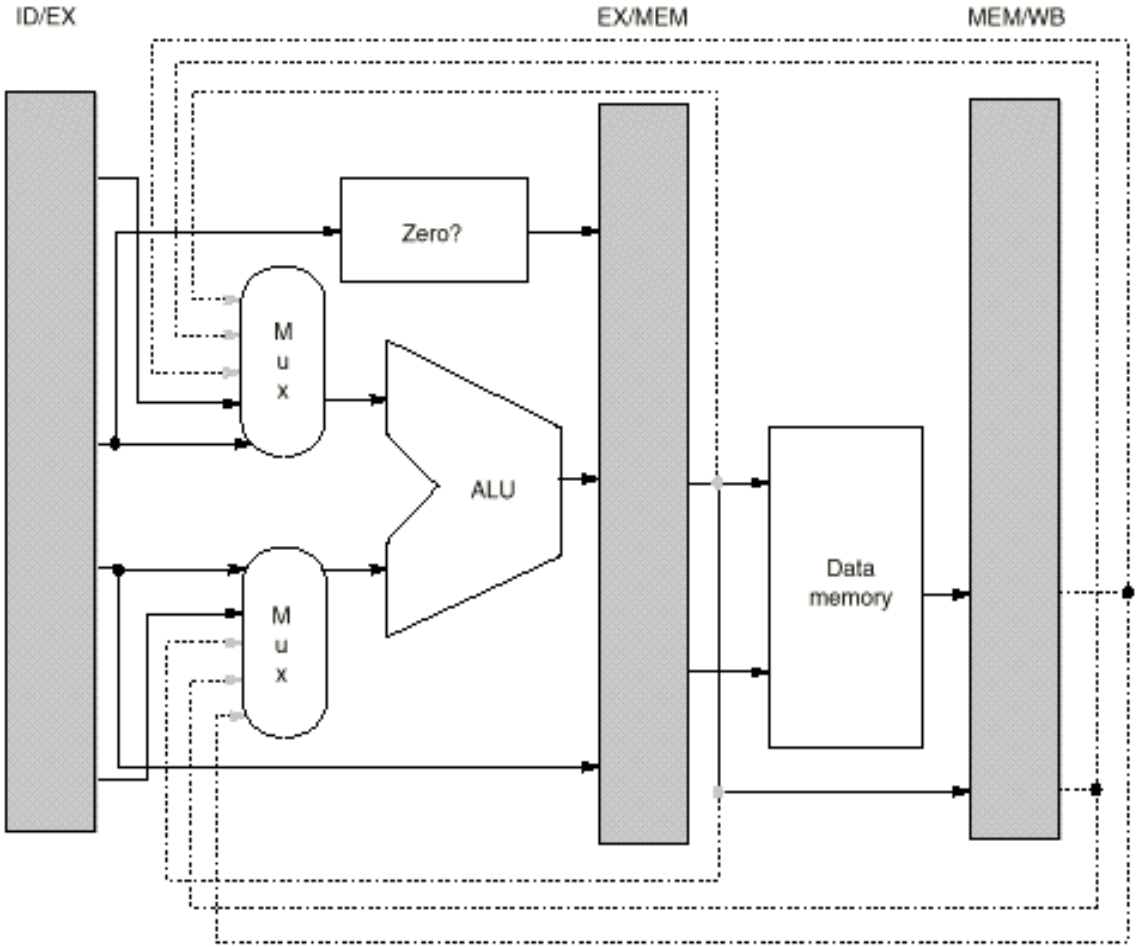
vrednost upisana u R1

ADD	IF	ID	EXE	MEM	WB
SUB		IF	ID		

ovde se čita R1

Zaustavljanje protočnog sistema uzrokovano RAW hazardima se može eliminisati pribavljanjem unapred (bypasing, forwarding)

RAW Hazardi



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

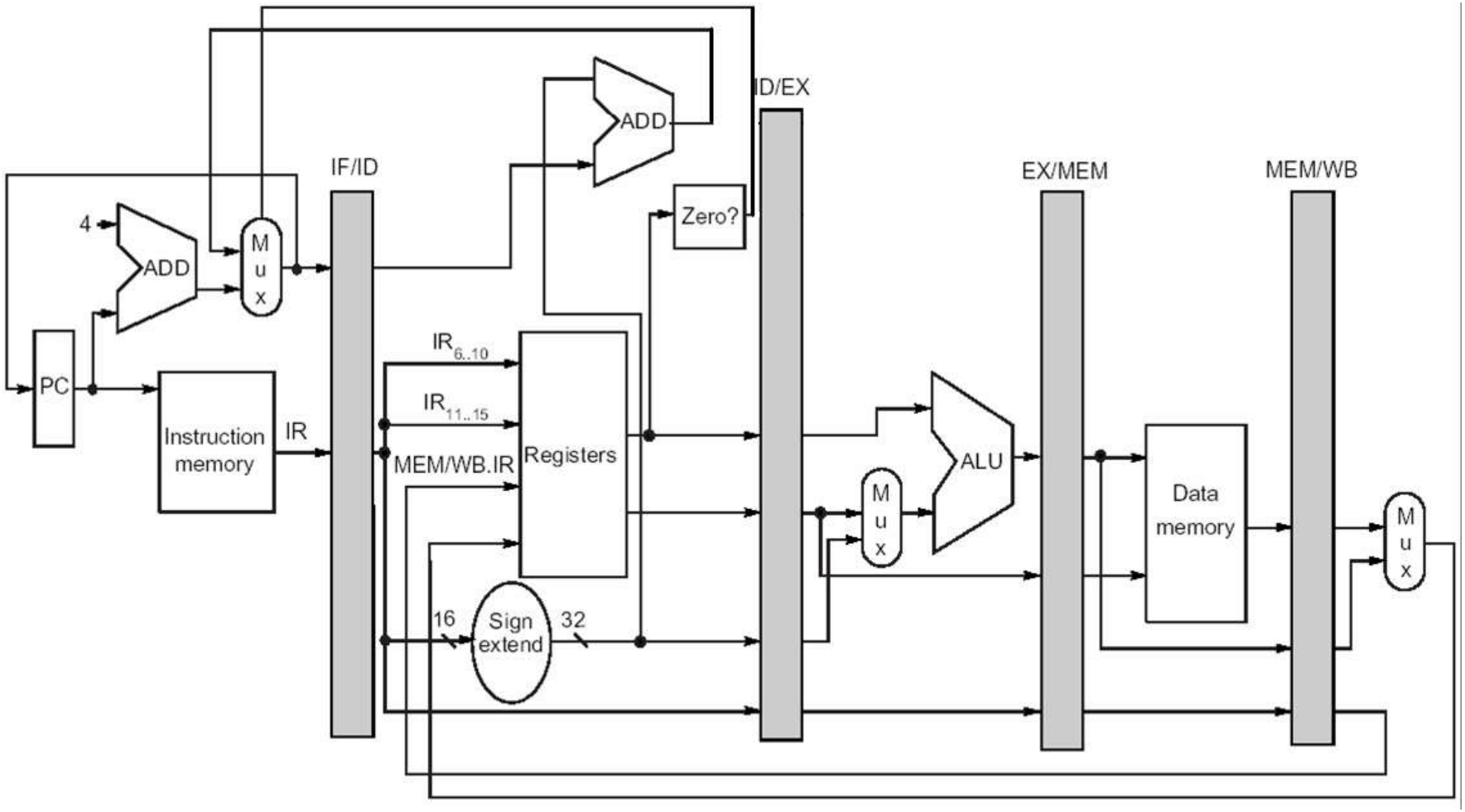
kontrolni hazardi

- Mogu uzrokovati veći gubitak performansi nego hazardi po podacima.
- Nastupaju zbog instrukcija koje mogu promeniti sadržaj PC (branch, jump, call, return).
- Primer branch instrukcije: novi sadržaj PC poznat tek u MEM fazi, posle izračunavanja adrese i testiranja uslova.
- Neophodno zaustaviti protočni sistem dok se ne dozna novi sadržaj PC.

branch	IF	ID	EX	MEM	WB
i+1		IF	--	--	IF

zaustavljanje protočnog sistema nije moguće odmah nakon pribavljanja branch jer nije završeno dekodiranje. Pribavljena instr. se briše (IF/ID registar) Nakon Mem faze vrši se novo pribavljanje

kontrolni hazardi – redukovanje gubitaka



- Redukovanje kašnjenja zbog branch hazarda pomeranjem testiranja na 0 i izračunavanjem adrese u ID fazi.

Zakašnjeno grananje

- Ideja je da se iza naredbe grananja postavi instrukcija koja će se izvršiti bez obzira da li će se grananje obaviti ili ne (dok se ne odredi uslov i novi sadržaj PC).
- Naziv potiče od činjenice da se efekat naredbe grananja odlaže.

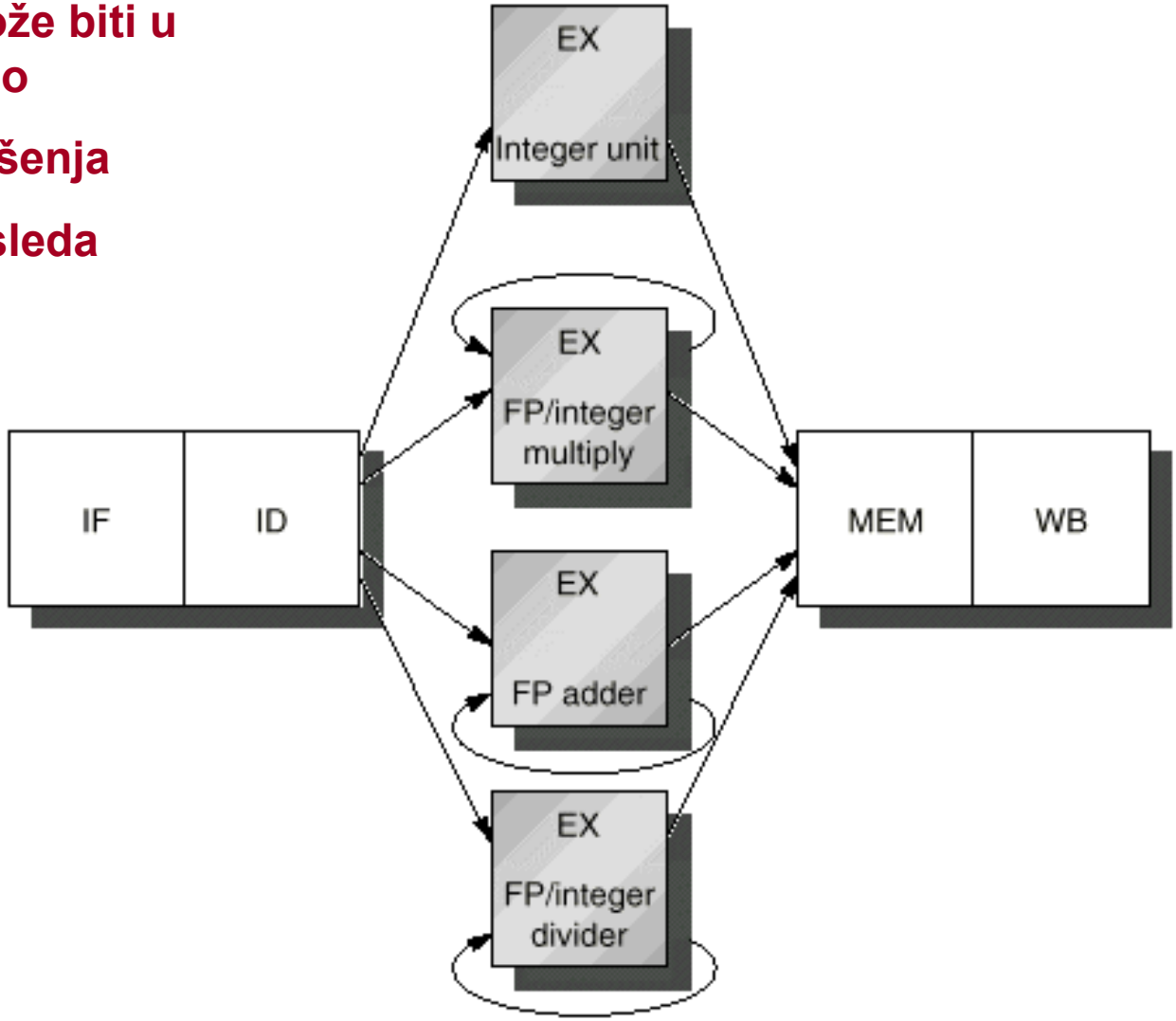
conditional branch instruction
sequential successor$_1$
sequential successor$_2$
.....
sequential successor$_n$
branch target if taken

slot zakašnjenog grananja

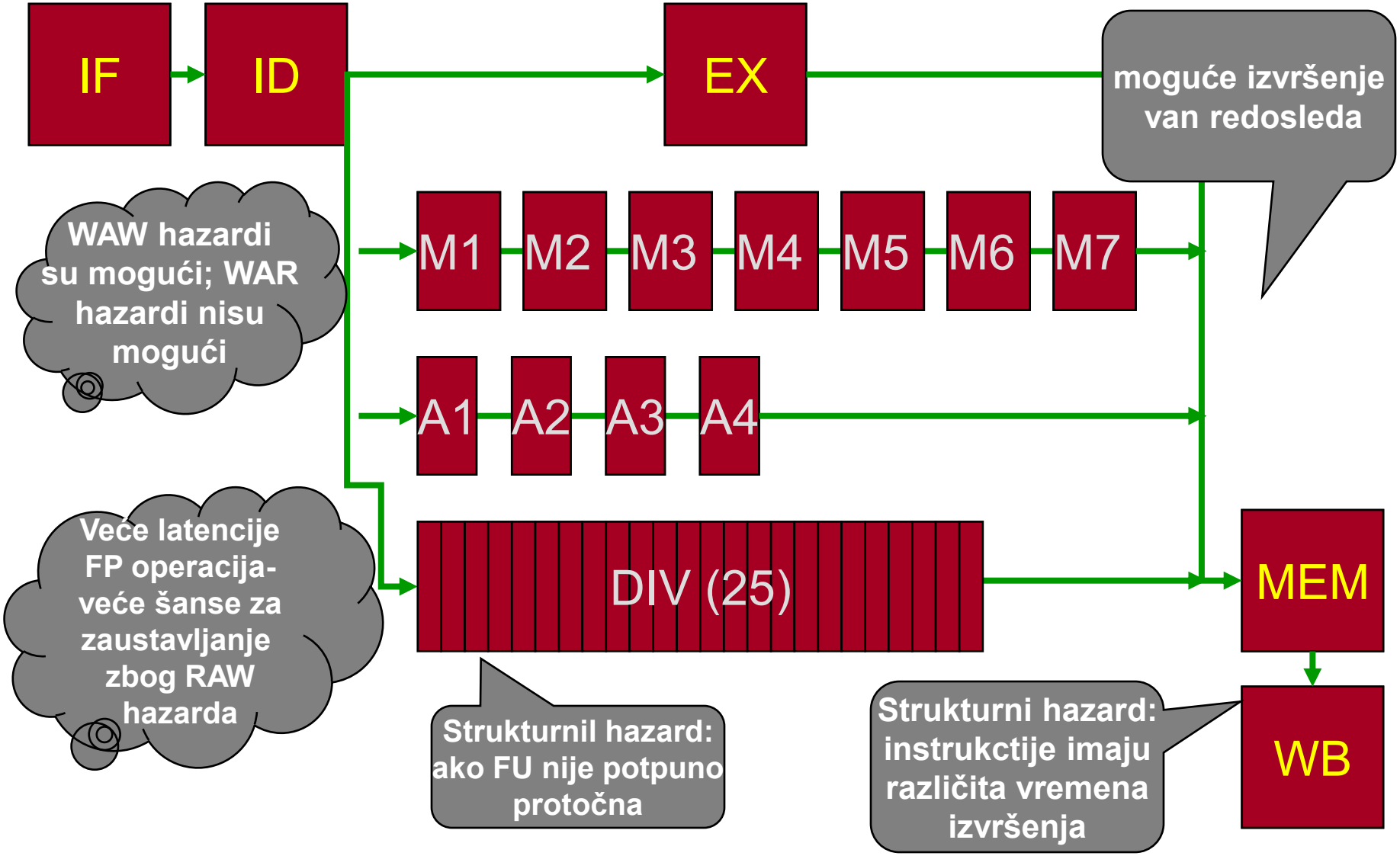
- Za instrukcije koje slede iza naredbe grananja se kaže da se nalaze u slotu (prozoru) zakašnjenog grananja. Ove instrukcije se izvršavaju bez obzira da li dolazi do grananja ili ne.
- U praksi je veličina prozora najčešće 1.
- Zadatak kompajlera je da u ovaj prozor postavi važeće i korisne instrukcije

Protočni sistem sa FP funkcionalnim jedinicama

- **više instrukcija može biti u EX fazi jednovremeno**
- **različito vreme izvršenja**
- **izvršenje van redosleda pribavljanja**



FP Operacije – posledice



- Paralelizam na nivou instrukcija
- Odmotavanje petlje
- Dinamičko preuredjenje instrukcija
  - Scoreboard
  - Tomasulov algoritam

Performanse protočnog sistema u prisustvu hazarda

- Osnovni cilj uvodjenja protočnosti u izvršenju instrukcija je postizanje CPI (Clocks per Instruction) od 1instr/clk
- Zbog postojanja hazarda, realni CPI je veći

Pipeline CPI = Ideal Pipeline CPI + Structural Stalls + RAW Stalls + WAR Stalls + WAW Stalls + Control Stalls

- Da bi se postigle bolje performanse potrebno je iskoristiti paralelizam koji postoji na nivou instrukcija (ILP – Instruction Level Parallelism)
  - LW R1, 30(R2)
  - ADDI R3,R3,#1
  - ADD R4, R4, R5
    - ILP=3

ADDI R3, R3, #1
ADD R4, R3, R2
MUL R5, R3, R4
ILP=1

Kako povećati ILP?

- Količina raspoloživog ILP u okviru tvz. osnovnog bloka (pravolinijski kod koji ne sadrži naredbe grananja izuzev na početku i na kraju) je veoma mala
- Dinamika pojavljivanja naredbi grananja u programima je oko 15%, što znači da se 6-7 instrukcija izvršava izmedju dve naredbe grananja
  - instrukcije unutar osnovnog bloka su verovatno zavisne jedna od druge, pa je ILP manji od 6.
- Da bi se postiglo značajnije poboljšanje performansi, mora se eksploatisati ILP van osnovnog bloka.
- Najjednostavniji način za povećanje ILP je da se iskoristi paralelizam izmedju različitih iteracija petlje
  - paralelizam koji postoji izmedju različitih iteracija petlje se zove LLP – Loop Level Paralleism
  - Primer:
    - for i:=1 to 100
    - x(i):=x(i)+y(i)

Ova petlja je potpuno paralelna.
Svaka iteracija petlje može se preklapati u izvršenju sa bilo kojom drugom iteracijom

kako povećati ILP?

- Postoji puno tehnika za konverziju LLP u ILP. Tehnike rade tako što vrše odmotavanje petlje, statički uz pomoć kompajlera, ili dinamički pomoću hw, čime se povećava veličina osnovnog bloka.
- Važan alternativni metod za eksploataciju LLP je korišćenje vektorskih instrukcija.
- Da bi se protočni sistem održao punim, moraju se pronaći instrukcije koje ne zavise jedna od druge i koje se mogu bezbedno izvršavati sa preklapanjem bez mogućnosti za nastupanje hazarda.
- Da bi se izbeglo zaustavljanje protočnog sistema zbog nastupanja hazarda, instrukcije koje zavise jedna od druge moraju se razdvojiti drugim instrukcijama.
  - broj instrukcija kojima se moraju razdvojiti medjusobno zavisne instrukcije tako da se izbegne zaustavljanje protočnog sistema zavisi od latentnosti funkcionalne jedinice koja generiše rezultat.
  - da li će kompajler moći da izvrši preuredjenje koda tako da ne nastupi zastoj, zavisi od količine raspoloživog ILP.

Usvojene latencije funkcionalnih jedinica

- Sve funkcionalne jedinice su potpuno protočne (period inicijacije =1)
  - latencije FP funkcionalnih jedinica:

Instrukcija koja generiše rezultat	Instrukcija koja koristi rezultat	Latentnost u Clock Cycles
FP ALU Op	druga FP ALU Op	3
FP ALU Op	Store Double	2
Load Double	FP ALU Op	1
Load Double	Store Double	0

- latentnost ALU integer operacija je 0, latentnost za LOAD i Branch 1 clk

Odmotavanje petlje – Primer

```
for (i=1000; i>0; i=i-1)
    x[i] = x[i] + s;
```

- Petlja je potpuno paralelna, jer je svaka iteracija petlje nezavisna.
- Prvi korak u odmotavanju je prevodjenje na asembler.
  - usvajamo:
    - registar R1 inicijalno sadrži adresu poslednjeg elementa polja
    - Registar F2 sadrži skalarnu vrednost S
    - element x[1] se nalazi na adresi 0.

```
Loop: LD   F0,0(R1)  ;F0= element vektora
      ADDD F4,F0,F2  ;saberi sa skalarom iz F2
      SD   0(R1),F4  ;zapamti rezultat
      SUBI R1,R1,8   ;dekrementirati pointer za 8B (DW)
      BNEZ R1,Loop   ;skok na Loop ako je R1 ≠ 0
```

Kako izgleda protočno izvršenje instrukcija sa usvojenim latencijama:

```
1 Loop: LD    F0,0(R1)   ;F0=vector element
2       zastoj
3       ADDD  F4,F0,F2   ;add scalar in F2
4        zastoj
5        zastoj
6       SD    0(R1),F4   ;store result
7       SUBI  R1,R1,8    ;decrement pointer 8B (DW)
8       BNEZ  R1,Loop    ;branch ako je R1≠ 0
9        zastoj          ;delayed branch slot
```

9 clk/iteraciji: može li se kod preurediti da se minimiziraju zastoji?

Preuredjeni kod sa minimiziranim zastojima

```
1  Loop:  LD     F0,0(R1)
2         zastoj
3         ADDD   F4,F0,F2
4         SUBI   R1,R1,8
5         BNEZ   R1,Loop     ;delayed branch
6         SD     8(R1),F4    ;instrukcija postavljena u slot zak. grananja
```

Preuredjenjem instrukcija vreme izvršenja je smanjeno sa 9 na 6 clk/iteraciji.

Napomena: **Da bi primenio zakašnjeno grananje, kompajler mora da utvrdi da može da izmeni redosled SUBI i SD tako što će promeniti adresu u koju SD pamti: adresa je bila 0(R1), a sadaje 8(R1)**

- **Ovo zapažanje nije trivijalno, jer bi većina kompajlera utvrdila da SD zavisi od SUBI i ne bi odlučila da promeni redosled ovih instrukcija**
- **Jedna iteracija petlje se obavi za 6 clk, ali korisni posao nad elementima polja zahteva samo 3 clk ciklusa (LD, ADD i SD). Preostala 3 clk ciklusa potiču od instrukcija za obradu petlje (SUBI i BNEZ) i zastoja zbog RAW hazarda uzrokovanog LD instrukcijom (50% gubitaka)**

Kako dalje smanjiti broj clk/iteraciji?

- Potrebno je da postoji više “korisnih” instrukcija unutar petlje u odnosu na ukupan broj instrukcija.
- Jednostavan način za povećanje broja “korisnih” instrukcija je odmotavanje petlje.
- Odmotavanje se obavlja repliciranjem tela petlje više puta i podešavanjem dela koda za okončanje petlje
- Da bi se izbegli zastoji, različite iteracije tela petlje koriste različite registre

Petlja odmotana četiri puta

```
1 Loop: LD      F0,0(R1)
2       ADDD    F4,F0,F2
3       SD      0(R1),F4        ;uklonjene SUBI & BNEZ
4       LD      F6,-8(R1)
5       ADDD    F8,F6,F2
6       SD      -8(R1),F8       ; uklonjene SUBI & BNEZ
7       LD      F10,-16(R1)
8       ADDD    F12,F10,F2
9       SD      -16(R1),F12     ; uklonjene SUBI & BNEZ
10      LD      F14,-24(R1)
11      ADDD    F16,F14,F2
12      SD      -24(R1),F16
13      SUBI    R1,R1,#32       ;sve SUBI su objedinjene u jednu (4*8=32)
14      BNEZ    R1,LOOP
```

pretpostavka je da je R1 umnožak od 32

Protočno izvršenje odmotane petlje

```
1  Loop:  LD      F0,0(R1)
2         zastoj
3         ADDD    F4,F0,F2
4,5       zastoj, zastoj
6         SD      0(R1),F4
7         LD      F6,-8(R1)
8         zastoj
9         ADDD    F8,F6,F2
10,11     zastoj, zastoj
12        SD      -8(R1),F8
13        LD      F10,-16(R1)
14        zastoj
15        ADDD    F12,F10,F2
16,17     zastoj, zastoj
18        SD      -16(R1),F12
19        LD      F14,-24(R1)
20        zastoj
21        ADDD    F16,F14,F2
22,23     zastoj, zastoj
24        SD      -24(R1),F16
25        SUBI    R1,R1,#32
26        BNEZ    R1,LOOP
27        zastoj
```

27/4= 6.8 clk/iteraciji

```
1  Loop:  LD      F0,0(R1)
2         LD      F6,-8(R1)
3         LD      F10,-16(R1)
4         LD      F14,-24(R1)
5         ADDD    F4,F0,F2
6         ADDD    F8,F6,F2
7         ADDD    F12,F10,F2
8         ADDD    F16,F14,F2
9         SD      0(R1),F4
10        SD      -8(R1),F8
11        SD      -16(R1),F12
12        SUBI    R1,R1,#32
13        BNEZ    R1,LOOP
14        SD      8(R1),F16;8-32 = -24
```

14 clk ciklusa, ili 3.5 clk po iteraciji

Rezime-odmotavanja petlje

- U prethodnom primeru ključna su sledeća zapažanja:
  - Otkriti da je legalno premestiti SD posle SUBI i BNEZ; pronaći offset za SD.
  - Utvrditi da su iteracije petlje nezavisne i da će odmotavanje dovesti do poboljšanja.
  - Korišćenje različitih registara za različite iteracije da bi se izbegli WAR i WAW hazardi.
  - Eliminisanje ekstra testiranja i grananja i podešavanjem koda za okončanje petlje.
  - Load i store iz različitih iteracija mogu zameniti mesta
  - Preuredjenje koda tako da se sačuvaju sve zavisnosti koje obezbedjuju korektno izvršenje programa.

Zavisnosti izmedju instrukcija

- Ključni momenat kod svih transformacija je razumevanje zavisnosti izmedju instrkcija i kako se instrukcije mogu preurediti kada postoje takve zavisnosti
- Ako su dve instrukcije nezavisne, one se mogu izvršavati simultano u protočnom sistemu bez izazivanja zastoja (pod pretpostavkom da ima dovoljno resursa, tj. nema strukturnih hazarda.)
- Instrukcije koje su zavisne jedna od druge se ne mogu preurediti.
- postoje tri tipa zavisnosti izmedju instrukcija :
  - **Prave zavisnosti** (odgovaraju RAW)
  - **Zavisnosti imenovanja**
  - **kontrolne zavisnosti**

Prave zavisnosti

- Prava zavisnosti izmedju instrukcije *i* i instrukcije *j* postoji ako važi jedno od sledećeg:
  - instrukcija *i* proizvodi rezultat koji koristi *j*
  - instrukcija *j* zavisi od instrukcije *k*, a instrukcija *k* od instrukcije *i*, što ukazuje na postojanje lanca zavisnosti izmedju instrukcija

Primer: Strelice ukazuju na zavisnost koje postoje. Ove zavisnosti ukazuju na redosled izvršenja instrukcija koji mora biti ispoštovan da bi izvršenje programa bilo korektno.

```
Loop:   LD      F0, 0 (R1)   ; F0=array element
        ADDD    F4, F0, F2   ; add scalar in F2
        SD      0(R1), F4    ; store result
i
        SUBI    R1,R1, 8
        BNEZ    R1, Loop
```

Zavisnosti izmedju instr. –nast.

- Zavisnosti su svojstvo programa. Da li data zavisnost dovodi do hazarda koji se detektuje i da li taj hazard uzrokuje zaustavljanje su osobine protočnog sistema.
- prisustvo zavisnosti ukazije na potencijalni hazard, ali stvarni hazard i dužina zastoja zavise od konkretnog protočnog sistema
  - u prethodnom primeru postoji zavisnost izmedju SUBI i BNEZ, ali ova zavisnost ne dovodi do zastoja u našem protočnom sistemu, jer se pribavljanjem unapred eliminiše
  - zavisnost ADDD od LD dovodi do hazarda i zastoja u protočnom sistemu

Zavisnosti imenovanja

- Zavisnosti imenovanja nastupaju kada dve instrukcije koriste isti registar.
- Izmedju instrukcija ne postoji nikakva razmena podataka.
- Ako instrujcija *i* prethodi instrukciji *j*, tada dva tipa zavisnosti imenovanja mogu da nastupe:
  - **Antizavisnost nastupa kada *j* upisuje u registar, a *i* čita isti registar i *i* se prva izvršava (WAR hazard)**
  - **Izlazna zavisnost nastupa kada instrukcije *i* i *j* vrše upis u isti registar, dovodeći do WAW hazarda koji zahteva da redosled izvršenja instrukcija mora da se ispoštuje.**

Primer zavisnosti imenovanja

Odmotana petlja koja koristi iste registre za različite iteracije

antizavisnosti

izlazne zavisnosti

```
1 Loop: LD      F0,0(R1)
2       ADDD    F4,F0,F2
3       SD      0(R1),F4        ;uklonjeno SUBI & BNEZ
4       LD      F0,-8(R1)
2       ADDD    F4,F0,F2
3       SD      -8(R1),F4       ; uklonjeno SUBI & BNEZ
7       LD      F0,-16(R1)
8       ADDD    F4,F0,F2
9       SD      -16(R1),F4      ; uklonjeno SUBI & BNEZ
10      LD      F0,-24(R1)
11      ADDD    F4,F0,F2
12      SD      -24(R1),F4
13      SUBI    R1,R1,#32       ;promenjeno na 4*8
14      BNEZ    R1,LOOP
15      NOP
```

zavisnosti se mogu otkloniti preimenovanjem registara!

Zašto je važno poznavati zavisnosti?

- ukazuju na mogućnost hazarda
- Odredjuju redosled po kome se rezultati moraju izračunavati
- Uspostavljaju gornju granicu potencijalnog ILP koji se može eksploatisati
- Pošto zavisnosti izmedju instrukcija mogu ograničiti količinu ILP, cilj je ukloniti ovo ograničenje. To se može postići na dva načina:
  - Održavanjem zavisnosti, ali izbegavanjem hazarda (preuredjenjem instrukcija u fazi prevodjenja, ili dinamički pomoću hw)
  - Eliminisanjem zavisnosti transformacijom koda (preimenovanjem registara)

Dinamičko planiranje izvršenja instrukcija

- U dosadašnjoj analizi smo pretpostavljali da protočni sistem pribavlja i izdaje instrukcije sve dok ne detektuje zavisnost po podacima koja ne može biti prevazidjena pribavljanjem unapred.
  - Logika za pribavljanje u napred redukije zastoje tako što izvesne zavisnosti ne dovode do hazarda.
  - Ako postoji zavisnost koja ne može biti prevazidjena, tada hw za detekciju zavisnosti zaustavlja protočni sistem, počev od instrukcije koja koristi rezultat.
  - Nove instrukcije se ne pribavljaju i ne izdaju dok se hazard ne obriše.
- Kompajlerskim tehnikama (zakašnjeni load i zakašnjeno grananje) su se razdvajale instrukcije koje su medjusobno zavisne i na taj način se minimizirao broj hazarda i zastoji koji iz toga proističu.
- Dinamičkim planiranjem izvršenja instrukcija se ne mogu ukloniti zavisnosti izmedju instrukcija, ali se mogu izbeći zastoji kada su zavisnosti prisutne

Dinamičko planiranje – ideja

- Osnovno ograničenje u protočnom sistemu koji smo do sada analizirali, je da se zahteva izdavanje instrukcija po redosledu pribavljanja:
  - ako je neka instrukcija zaustavljena u protočnom sistemu, ni jedna koja je pribavljena iza nje ne može da produži sa izvršenjem.
  - ako postoji zavisnost izmedju dve prostorno bliske instrukcije, doći će do zaustavljanja protočnog sistema.
  - Ako postoji više funkcionalnih jedinica, ostaće neupošljene.
  - primer:
    - DIVD F0, F2, F4
    - ADDD F10, F0, F8
    - SUBD F12, F8, F14

•instrukcija SUBD se ne može izvršiti zato što zavisnost ADDD od DIVD uzrokuje zaustavljanje protočnog sistema.

•SUBD ne zavisi ni od jedne druge instrukcije.

•Zaustavljanje protočnog sistema se može eliminisati tako što se neće zahtevati da se instrukcije izvršavaju po redosledu pribavljanja

Dinamičko planiranje – ideja

- U protočnom sistemu se svi hazardi (strukturni, po podacima, kontrolni) proveravaju u ID fazi (u toku dekodiranja)
- Ako instrukcija može da se izvrši korektno (bez izazivanja hazarda), ona prelazi iz ID u EX fazu (izdaje se)
- Da bi omogućili da izvršenje SUBD otpočne što ranije (jer ne zavisi od DIVD i ADDD) potrebno je ID fazu podeliti na dva dela:
  - proveru strukturnih hazarda
  - proveru hazarda po podacima
    - Ako nema strukturnih hazarda, cilj je da izvršenje instrkcije otpočne čim su dostupni njeni operandi.
    - na taj način može doći do izvršenje instrukcija van redosleda, što će dovesti do završetka-van-redosleda.

Dinamičko planiranje – ideja

- ID faza se deli na dva dela:
  - ISSUE(odluka, izdavanje) – dekodiranje instrukcije i provera strukturnih hazarda
  - Read operands (čitanje operanada)- čekanje na brisanje data hazarda, zatim čitanje operanada
    - Sve instrukcije prolaze kroz Issue stepen u redosledu pribavljanja, ali u read operands stepenu mogu biti zaustavljene ili propuštene u izvršenje-van-redosleda
- Pribavljanje instrukcije (IF) prethodi Issue stepenu. Instrukcije se pribavljaju u instrukcioni bafer veličine ≥1.
- EX stepen sledi iza stepena u kome se čitaju operandi.
- Kao i kod standardnog protočnog sistema izvršenje FP operacije može trajati više klok ciklusa i više instrukcija može biti u EX fazi jednovremeno.
- dve hardverske tehnike za dinamičko planiranje izvršenja instrukcija postoje:
  - Dinamičko planiranje sa Scoreboard tehnikom (CDC6600)
  - Tomasulov algoritam (IBM 360/91)

Scoreboard tehnika

- Prvi put primenjena kod CDC6600. (1963)
- Scoreboard tehnika omogućava izvršenje instrukcija van redosleda kada ima dovoljno resursa i nema zavisnosti po podacima.
- Cilj tehnike je obezbediti brzinu izvršenja od 1 clk po instrukciji, tako što će se instrukcije izvršavati što je ranije moguće
- Ako se dozvoli izvršenje instrukcija van redosleda može doći i do WAR i WAW hazarda
  - Primer:
    - DIVD F0, F2, F4
    - ADDD F10, F0, F8
    - SUBD F8, F8, F14

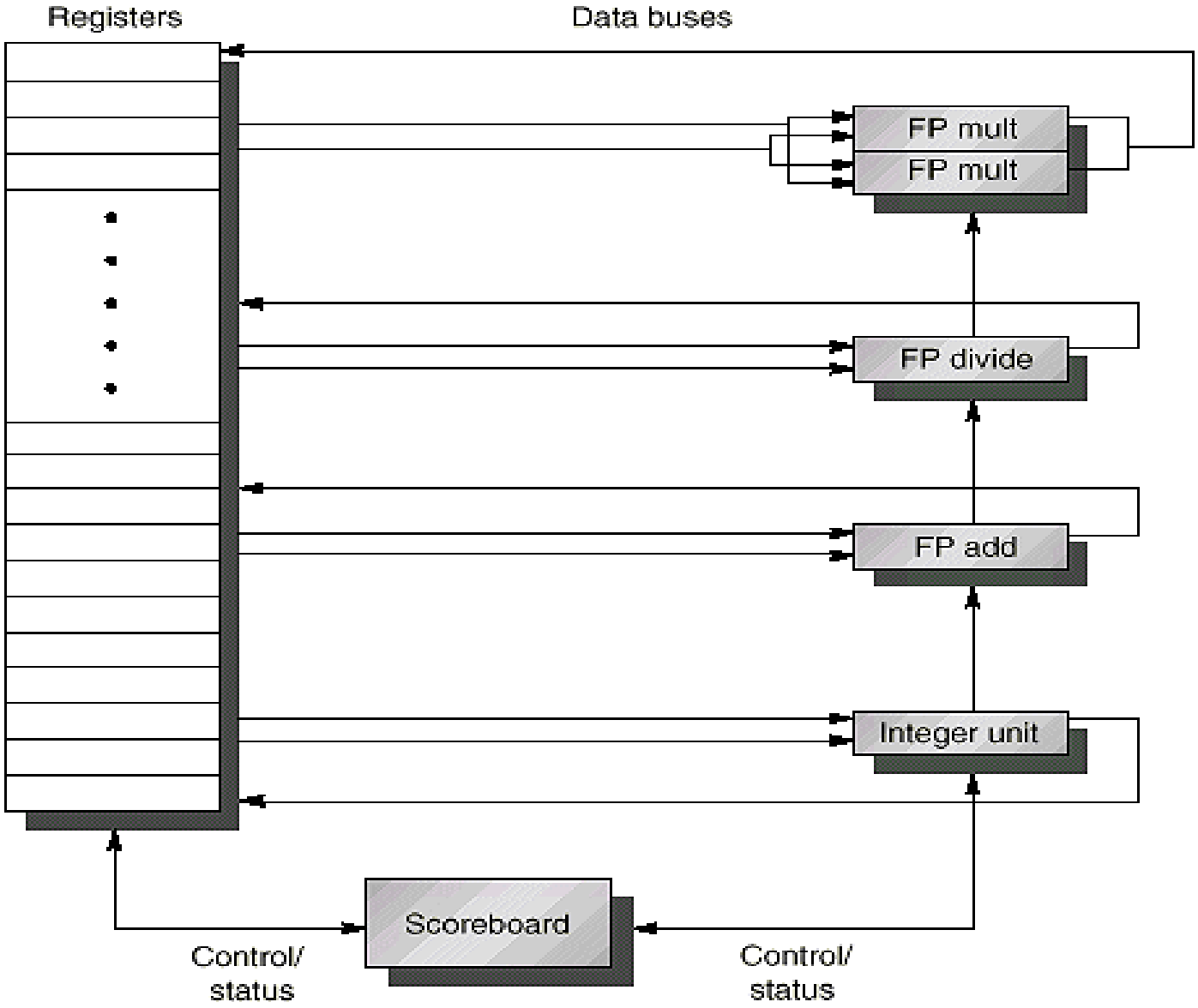
Postoji antizavisnost izmedju ADDD i SUBD. Ako se SUBD izvrši pre ADDD, narušiće se antizavisnost i izvršenje će biti nekorektno

Scoreboard (nast.)

- Svaka instrukcija prolazi kroz Scoreboard gde se beleži zapis o zavisnostima po podacima.
- Scoreboard sistem ima više funkcionalnih jedinica. Status svake funkcionalne jedinice se beleži u Scoreboard.
- Ako scoreboard utvrdi da se instrukcija ne može odma izvršiti, on izvršava sledeću instrukciju koja čeka na izvršenje i nastavlja da nadgleda status funkcionalne jedinice da bi utvrdio kada instrukcija može da krene u izvršenje.
- Scoreboard odlučuje kada instrukcija može izvršiti upis u registarski fajl (detekcija i razrešenje hazarda je centralizovano u scoreboard).

- Pretpostavke za analizu:
  - Sistem ima 2 FP množača
  - jedan FP sabirač,
  - jednu jedinicu za FP deljenje
  - jednu integer jedinicu za obraćanje memoriji, grananje i ALU operacije

Primer sistema sa Scoreboard



Faze izvršenja instrukcija kod Scoreboard

- Četiri koraka koji zamenjuju ID, Ex i WB su (ne razmatramo stepen za obraćanje memoriji):
  - Issue – ako je funkcionalna jedinica slobodna i ni jedna druga aktivna instrukcija nema isti odredišni registar, Sc. propušta instrukciju u funkcionalnu jedinicu i ažurira internu strukturu podataka. Ovde se detektuju strukurni i WAW hazardi. Ako strukturni ili WAW hazard postoji, onda se izdavanje svih instrukcija zaustavlja dok se hazard ne obriše.
    - Kada se Issue stepen zaustavi, bafer izmedju IF i Issue stepena će se napuniti i prestaće da se pribavljaju nove instrukcije.
    - ključno je uočiti da instrukcije kroz Issue stepen prolaze po redosledu pribavljanja
  - Read operands – Sc. nadgleda raspoloživost izvornih operanada. Izvorni operand je raspoloživ ako ni jedna druga aktivna instrukcija koja je prethodno izdata neće da ga generiše (upiše rezultat).
    - kada su operandi dostupni, Sc. kaže funkcionalnoj jedinici da pročita operande iz RF i otpočne izvršenje.
    - Sc. rešava RAW hazarde dinamički u ovom koraku i instrukcija može biti poslata u izvršenje van redosleda.

Faze izvršenja instrukcija kod Scoreboard- nast.

- Execution (izvršenje) – Funkcionalna jedinica (FU) otpočinje izvršenje nakon pribavljanja operanada. Kada je rezultat spreman FU obaveštava Sc.
- Write results (upis rezultata)- kada je Sc. obavešten da je FU okončala sa izvršenjem, proverava WAR hazarde i zaustavlja instrukciju ako je potrebno.
  - ako WAR ne postoji, ili se obriše, SC. kaže FU da upiše rezultat u odredišni registar.

* Pošto se operandi instrukcije čitaju tek kada su oba dostupna Sc. ne koristi prednosti pribavljanja unapred.
  - Regisri se čitaju tek kada su oba operanda dostupna.
* Na osnovu evidencije koju vodi, Sc kontroliše napredovanje instrukcije kroz protočni sistem.

Kako Sc. vodi evidenciju

- Postoje tri tabele preko koji Sc. vodi evidenciju o statusu instrukcije, statusu FU, raspoloživosti operanada:
  - **Status instrukcije (instruction status)-** govori u kojoj od 4 faze izvršenja se instrukcija nalazi.
  - **Status funkcionalne jedinice** –ukazuje na stanje FU. Za svaku FU postoji 9 polja u tabeli statusa
    - **Busy** — **govori da li je FU slobodna ili ne**
    - **Op** — **Operacija koju obavlja FU (npr., + ili –)**
    - **Fi** — **Odredišni registar**
    - **Fj, Fk** — **Brojevi izvornih registara**
    - **Qj, Qk** — **FU koja proizvodi podatke za izvorne registre Fj, Fk**
    - **Rj, Rk** — **Flegovi koji ukazuju kada su Fj, Fk spremni**

      (postavljaju se na Yes kada su operandi dostupni, brišu se – postavljaju na No kada se pročitaju operandi)

Kako Sc. vodi evidenciju- nast.

- **Tabela statusa registra rezultata (Register result status)-** ukazuje koja će FU izvršiti upis u koji regstar, ako aktivna instrukcija ima registar kao odredište.
  - Ako ne postoji FU koja vrši upis u dati registar, odgovrajuće polje je prazno.

Primer

Functional Unit (FU)	broj FUs	latentnost EX
Integer	1	0
Floating Point Multiply	2	10
Floating Point add	1	2
Floating point Divide	1	40

Fu nisu protočne

```
LD      F6, 34(R2)
LD      F2, 45(R3)
MULD    F0, F2, F4
SUBD    F8, F6, F2
DIVD    F10, F0, F6
ADDD    F6, F8, F2
```

Real Data Dependence (RAW)

Anti-dependence (WAR)

Output Dependence (WAW)

(In Appendix A.8)

Scoreboard Example Cycle 0

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2				
LD	F2	45+	R3				
MULD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
0	FU									

Scoreboard Example Cycle 1

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F6		R2				Yes
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
1	FU				Int					

Prva LD izdata (issue)

Scoreboard Example Cycle 2

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2		
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F6		R2				No
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
2	FU				Int					

Strukturni hazard po Integer jedinici; druga LD zaustavljena u IF stepenu

Scoreboard Example Cycle 3

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F6		R2				No
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
3	FU				Int					

Druga LD još uvek zaustavljena

Scoreboard Example Cycle 4

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3				
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F6		R2				No
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
4	FU									

Druga LD još zaustavljena; prva LD okončana

Scoreboard Example Cycle 5

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5			
MULT	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2		R3				Yes
Mult1	No								
Mult2	No								
Add	No								
Divide	No								

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
5	FU		Int							

Druga LD se izdaje jer se strukturni hazard po Integer jedinici obrisao

Scoreboard Example Cycle 6

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6		
MULT	F0	F2	F4	6			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2		R3				No
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	No								
Divide	No								

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
6	FU	Mul1	Int							

MULT se izdaje

Scoreboard Example Cycle 7

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULT	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2		R3				No
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Int	Yes	No
Divide	No								

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
7	FU	Mul1	Int			Add				

SUBD se izdaje; MULT zaustavljena zbog zavisnosti po LD

Scoreboard Example Cycle 8a

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	
MULT	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	Yes	Load	F2		R3				No
Mult1	Yes	Mult	F0	F2	F4	Integer		No	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2		Int	Yes	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
8	FU	Mul1	Int			Add	Div			

DIVD seje izdaje; SUBD zaustavljena zbog LD

Scoreboard Example Cycle 8b

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6			
SUBD	F8	F6	F2	7			
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
Mult2	No								
Add	Yes	Sub	F8	F6	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
8	FU	Mul1				Add	Div			

LD upisuje F2; MULT i SUBD mogu da nastave

Scoreboard Example Cycle 9

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Sub	F8	F6	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
9	FU	Mul1				Add	Div			

MULT i SUBD čitaju operande i ulaze u izvršenje

Scoreboard Example Cycle 10

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9		
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Sub	F8	F6	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
9	FU	Mul1				Add	Div			

Strukturni hazard po Add jedinici zaustavlja poslednju ADDD

Scoreboard Example Cycle 11

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Sub	F8	F6	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
11	FU	Mul1				Add	Div			

SUBD i MULT se još izvršavaju

Scoreboard Example Cycle 12

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2				

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
12	FU	Mul1					Div			

SUBD upisuje rezultat; Add jedinica slobodna; strukturni hazard otklonjen

Scoreboard Example Cycle 13

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13			

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			Yes	Yes
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
13	FU	Mul1			Add		Div			

postoji WAR hazard izmedju DIVD i ADDD

Scoreboard Example Cycle 14

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
14	FU	Mul1			Add		Div			

MULT se još izvršava; DIVD zaustavljena zbog F0 (RAW hazard)

Scoreboard Example Cycle 15

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14		

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
15	FU	Mul1			Add		Div			

MULT se još uvek izvršava

Scoreboard Example Cycle 16

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
16	FU	Mul1			Add		Div			

ADDD okončava izvršenje, spremna da upiše rezultat u F6

Scoreboard Example Cycle 17

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
17	FU	Mul1			Add		Div			

WAR hazard : ADDD zaustavljena u Write Result stepenu

Scoreboard Example Cycle 18

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9		
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
18	FU	Mul1			Add		Div			

DIVD zaustavljena (RAW hazard po F0), ADDD zaustavljena (WAR hazard po F6)

Scoreboard Example Cycle 19

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	Yes	Mult	F0	F2	F4			No	No
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		No	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
19	FU	Mul1			Add		Div			

MULT okončava izvršenje

Scoreboard Example Cycle 20

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8			
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6	Mult1		Yes	Yes

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
20	FU				Add		Div			

MULT upisuje rezultat; DIVD može da produži i pročita operande u sledećem ciklusu

Scoreboard Example Cycle 21

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	Yes	Add	F6	F8	F2			No	No
Divide	Yes	Div	F10	F0	F6			No	No

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
21	FU				Add		Div			

DIVD čita operande; WAR hazard po F6 je razrešen

Scoreboard Example Cycle 22

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21		
ADDD	F6	F8	F2	13	14	16	22

40 cycle Divide!

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6			No	No

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
22	FU						Div			

ADDD okončava upis rezultata

Scoreboard Example Cycle 61

Instruction Status

Instruction		j	k	Issue	Read Operand	Execution Complete	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULT	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	
ADDD	F6	F8	F2	13	14	16	22

Functional Unit Status

Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	No								
Mult1	No								
Mult2	No								
Add	No								
Divide	Yes	Div	F10	F0	F6			No	No

Register Result Status

CLOCK		F0	F2	F4	F6	F8	F10	F12	…	F31
61	FU						Div			

DIVD okončavaizvršenje; spremna da upiše rezultat

Scoreboard Example: Cycle 62

Instruction status

Instruction		j	k	Issue	Read operands	Execution complete	Write Result
L.D	F6	34+	R2	1	2	3	4
L.D	F2	45+	R3	5	6	7	8
MUL.D	F0	F2	F4	6	9	19	20
SUB.D	F8	F6	F2	7	9	11	12
DIV.D	F10	F0	F6	8	21	61	62
ADD.D	F6	F8	F2	13	14	16	22

Blok instrukcija izvršen

Functional unit status

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU for j Qj	FU for k Qk	Fj? Rj	Fk? Rk
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	No								

Register result status

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
62	FU									

- **Imamo:**
  - **In-oder issue,**
  - **Out-of-order execute and commit**

Scoreboard - zaključak

- Projektanti CDC su zabeležili poboljšanje performansi od 1.7 puta za FORTRAN code, i 2.5 za programe pisane na asembleru
- Ograničenja 6600 scoreboard
  - Ne koristi prednosti pribavljanja u napred
  - Ograničen na instrukcije u osnovnom bloku (mali issue prozor)
  - Broj funkcionalnih jedinica (strukturni hazardi)
  - Čekanje zbog WAR hazarda
  - Sprečavanje WAW hazarda

Scoreboard upravljanje

Instruction status	Wait until	Bookkeeping
Issue	Not busy (FU) and not result(D)	Busy(FU)← yes; Op(FU)← op; Fi(FU)← `D'; Fj(FU)← `S1'; Fk(FU)← `S2'; Qj← Result('S1'); Qk← Result(`S2'); Rj← not Qj; Rk← not Qk; Result('D')← FU;
Read operands	Rj and Rk	Rj← No; Rk← No
Execution complete	Functional unit done	
Write result	∀f((Fj(f)≠Fi(FU) or Rj(f)=No) & (Fk(f) ≠Fi(FU) or Rk(f)=No))	∀f(if Qj(f)=FU then Rj(f)← Yes); ∀f(if Qk(f)=FU then Rj(f)← Yes); Result(Fi(FU))← 0; Busy(FU)← No

format instrukcije

OP	D	S1	S2