

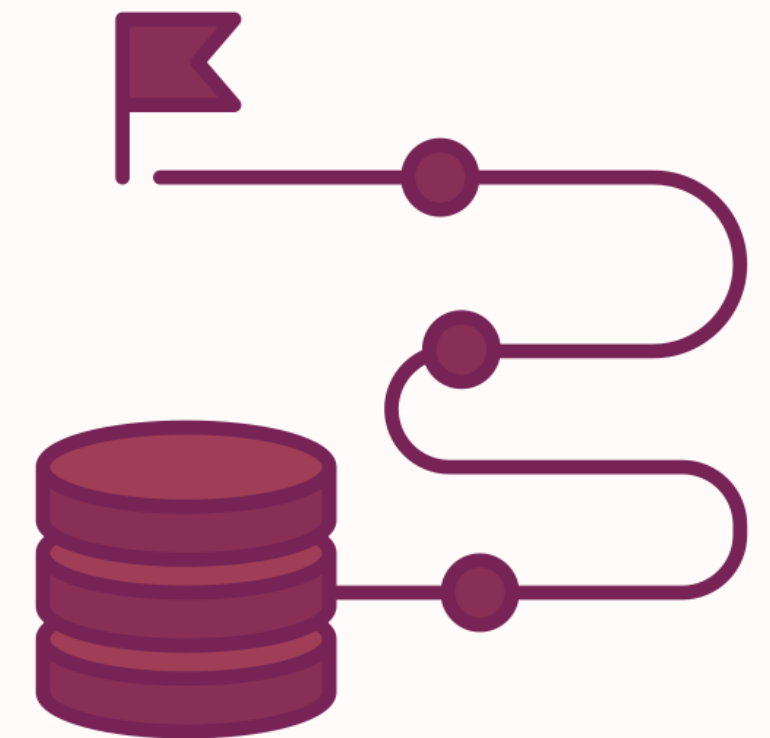
ELEKTROTEHNIČKI FAKULTET, UNIVERZITET U BEOGRADU
PARALELIZAM U ARHITEKTURI RAČUNARA I PROGRAMSKIM PREVODIOCIMA

PROTOČNOST I ZAVISNOST PO PODACIMA U PETLJAMA

doc. dr Maja Vukasović
as. ms Nikola Stanković

Protočnost (1)

- Za svaku fazu izvršavanja operacije postoji kombinaciona logika izmedju registara, tako da se ta faza izvršava u jednom ciklusu.
- **Protočni stepen** = kombinaciona logika + registri
- **Protočni niz** - niz protočnih stepeni gde je izlazni registar jednog protočnog stepena istovremeno ulazni registar drugog stepena.
 - Ulaz-izlaz kombinacione logike protočnih stepeni kontroliše se kontrolnim signalima.

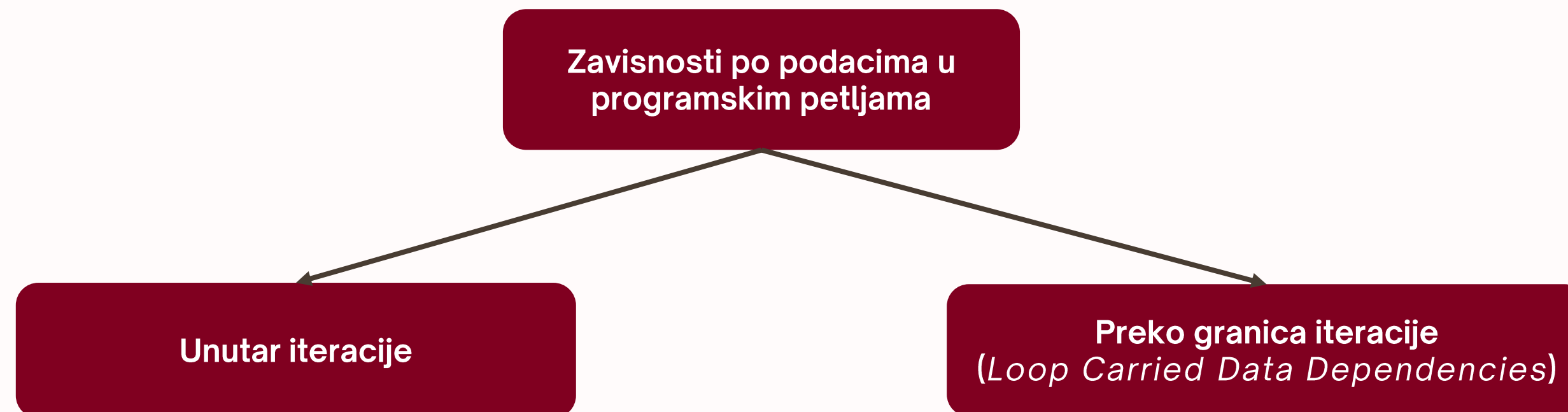


Protočnost (2)

- Primer: List Scheduling algoritam.
- Operacija traje onoliko ciklusa koliko ima protočnih stepeni u protočnom nizu.
- U svakom ciklusu operacija zauzima samo jedan protočni stepen u nizu.
 - Niz protočnih stepeni – značajan paralelizam.
- U svakom ciklusu može da se startuje izvršavanje nove operacije.
 - Paralelizam protočnog niza zavisi od broja protočnih stepeni.
 - Sve operacije mogu biti u različitoj fazi izvršavanja.
- Uslovno grananje u protočnom nizu?
 - Spekulativno se pretpostavlja ishod grananja.
 - Protočni stepeni se popunjavaju ciklusima operacija iz pretpostavljenih grana.

Zavisnosti po podacima u programskim petljama (1)

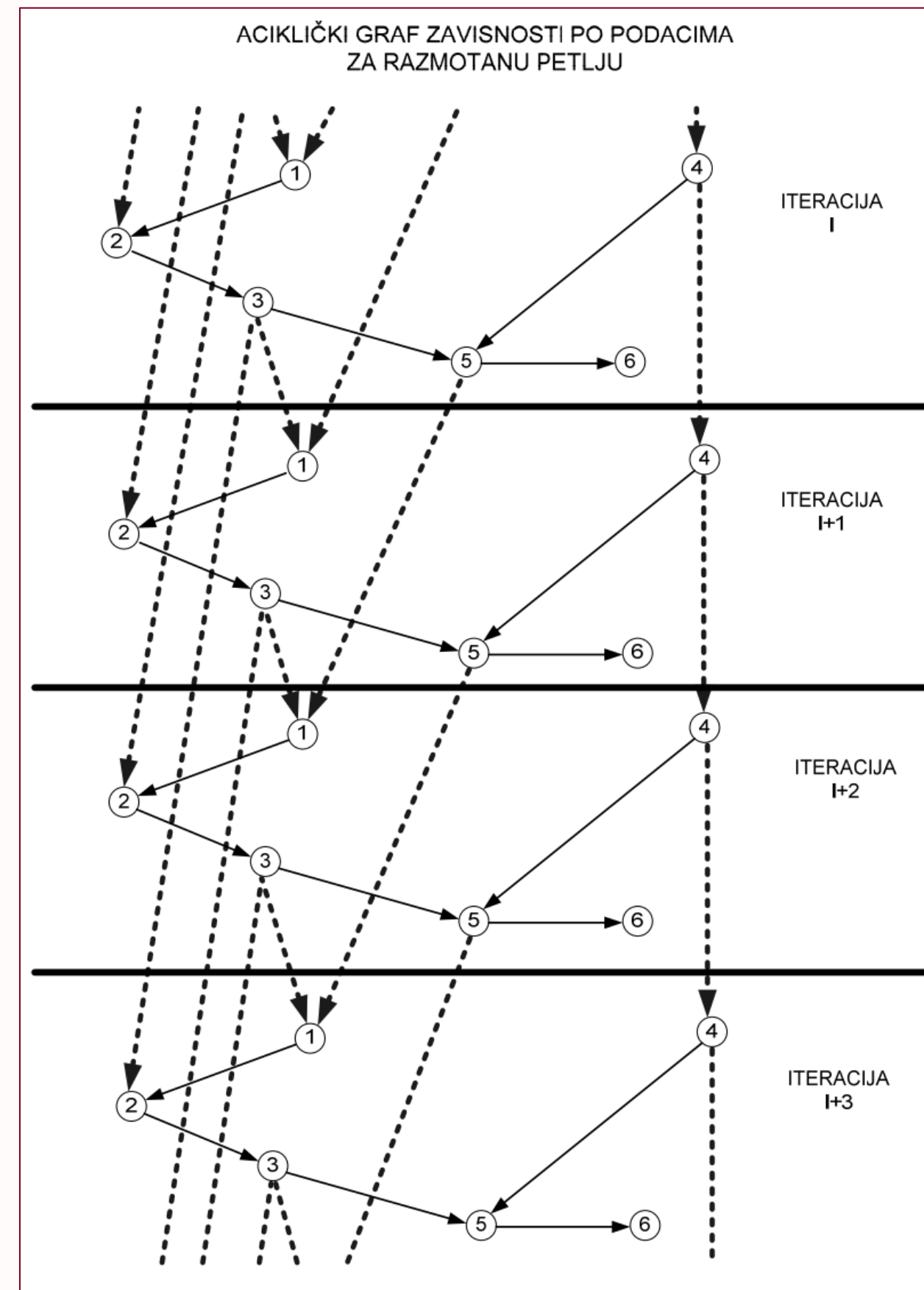
- Struktura petlje može uticati na primenu optimizacija.
- Primer: Trace Scheduling.



Zavisnosti po podacima u programskim petljama (2)

- Graf beskonačno razmotane petlje.
- Primer:

```
DO I = 3,100  
Op1:      A(I) := C(I-1) + E(I-2)  
Op2:      B(I) := A(I) * C(I-3)  
Op3:      C(I) := B(I) * K1  
Op4:      D(I) := D(I-1) + K2  
Op5:      E(I) := C(I) + D(I)  
Op6:      F(I) := C(I-1) + E(I)  
END
```

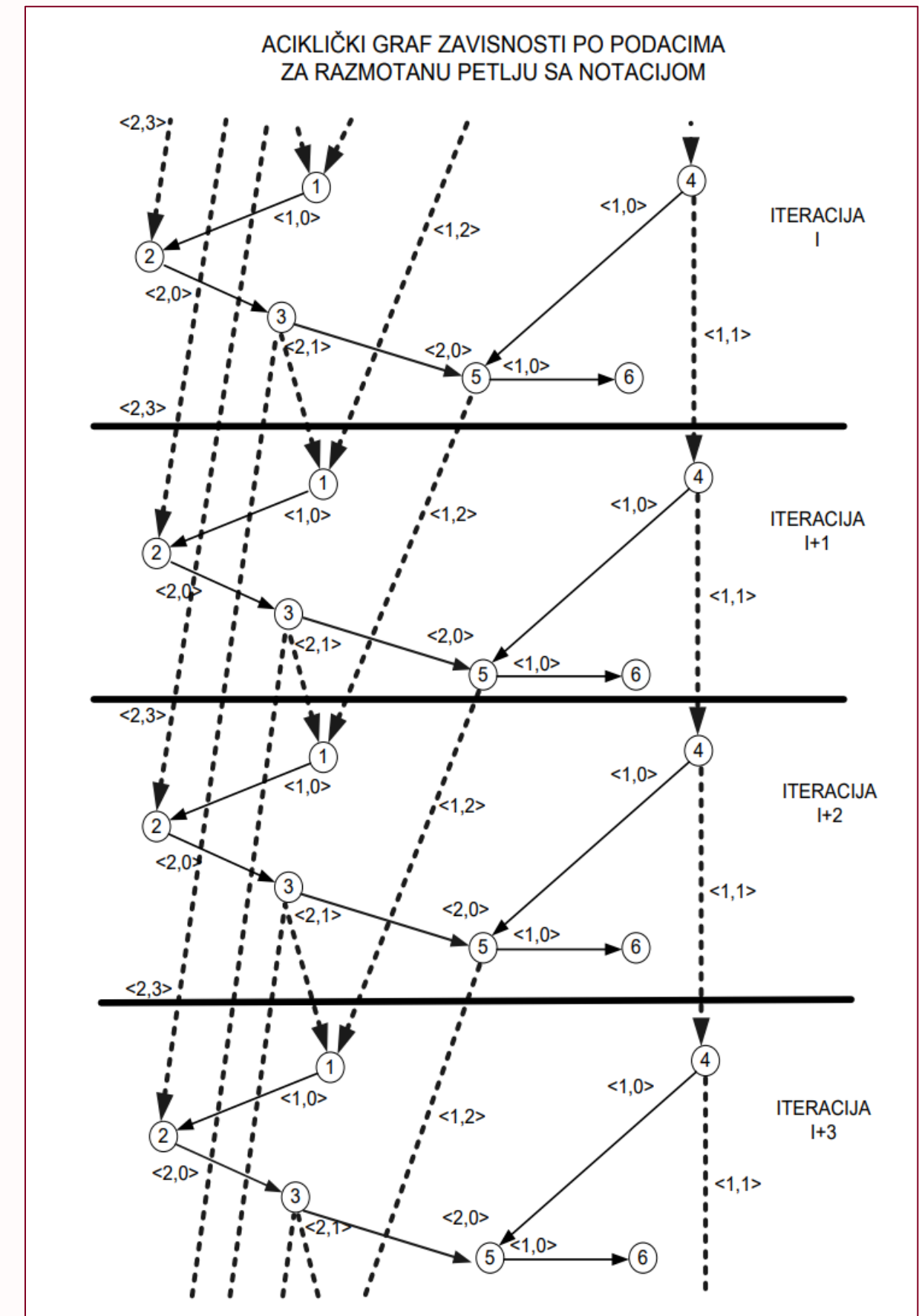


Zavisnosti po podacima u programskim petljama (3)

- Odredjivanje zavisnosti po podacima za operacije u iteracijama petlje može biti zahtevno ili nemoguće sprovesti statički.
 - Transformacije olakšavaju otkrivanje zavisnosti.
- Testovi za utvrđivanje zavisnosti po podacima:
 - GCD test
 - *Banerjee* test
 - Totalni test
 - Redukovani test

Zavisnosti po podacima u programskim petljama (4)

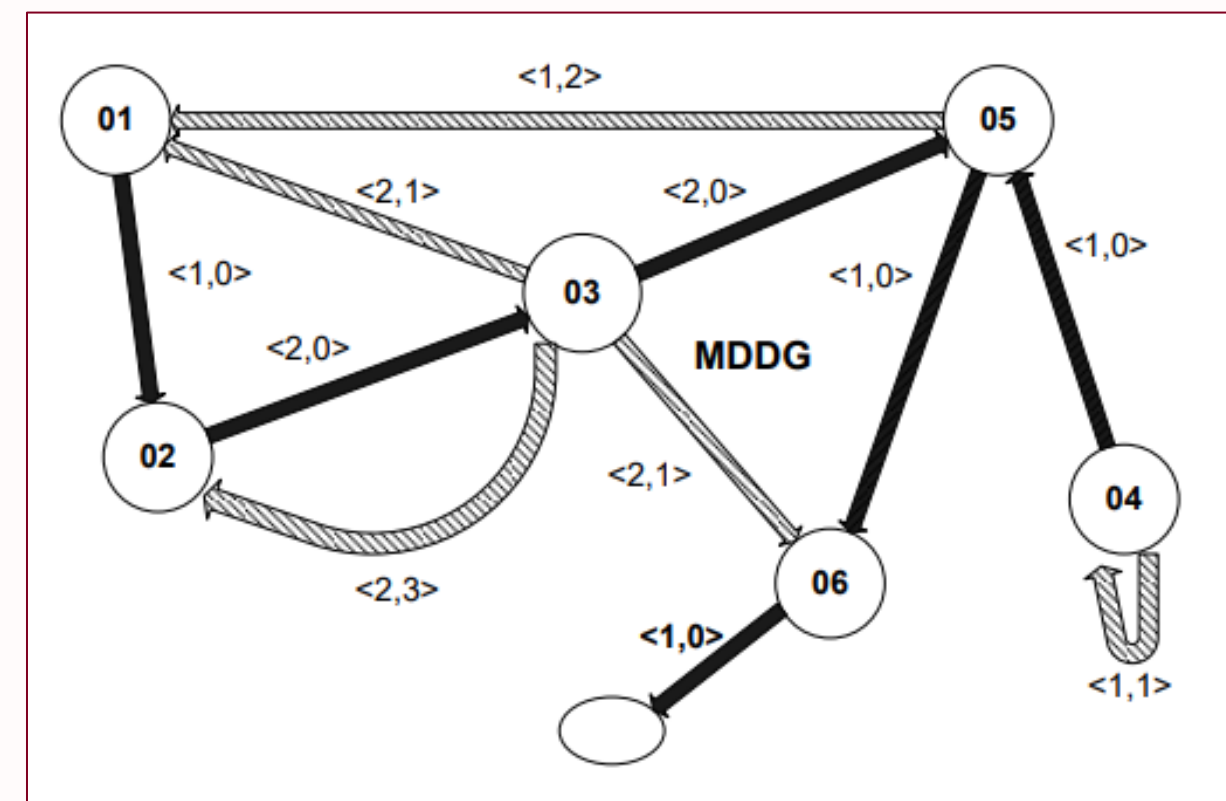
- Zavisnosti po podacima sa fiksnom iteracionom distancom.
- Obeležavanje grafa zavisnosti po podacima programske petlje sa fiksnim iteracionim distancama:
 - Svakoj grani se pridružuje par vrednosti (trajanje operacije, iteraciona distanca zavisnosti).
 - Trajanje operacije ~ kašnjenje zavisnosti.



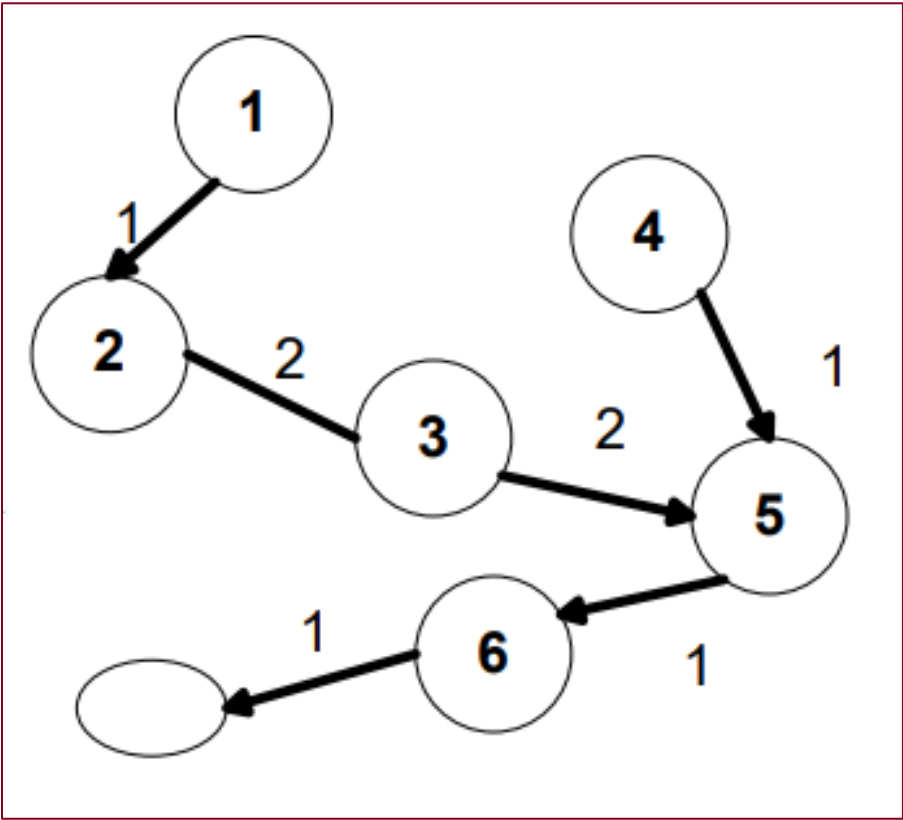
Zavisnosti po podacima u programskim petljama (5)

- Operacije ekvivalentne po telu petlje u razmotanom grafu.
- Predstavljanjem operacije ekvivalentne po telu petlje jednim čvorom i stapanjem grana, formira se modifikovani graf zavisnosti po podacima (MDDG).
 - Može da bude ciklični graf.

```
DO I = 3,100  
O1:      A(I) := C(I-1) + E(I-2)  
O2:      B(I) := A(I) * C(I-3)  
O3:      C(I) := B(I) * K1  
O4:      D(I) := D(I-1) + K2  
O5:      E(I) := C(I) + D(I)  
O6:      F(I) := C(I-1) + E(I)  
  
END
```



Zavisnosti po podacima u programskim petljama (6)



Graf jedne iteracije

1	O1 , O4
2	O2
3	
4	O3
5	
6	O5
7	O6

Najranije izvršavanje po ciklusima

DoAll i DoAcross petlje

DoAll

- Sve iteracije petlje se mogu obaviti u paraleli.
- Programske petlje kod kojih ne postoje zavisnosti po podacima za operacije iz različitih iteracija uvek su DoAll.
- Paralelizam je limitiran raspoloživim resursima.

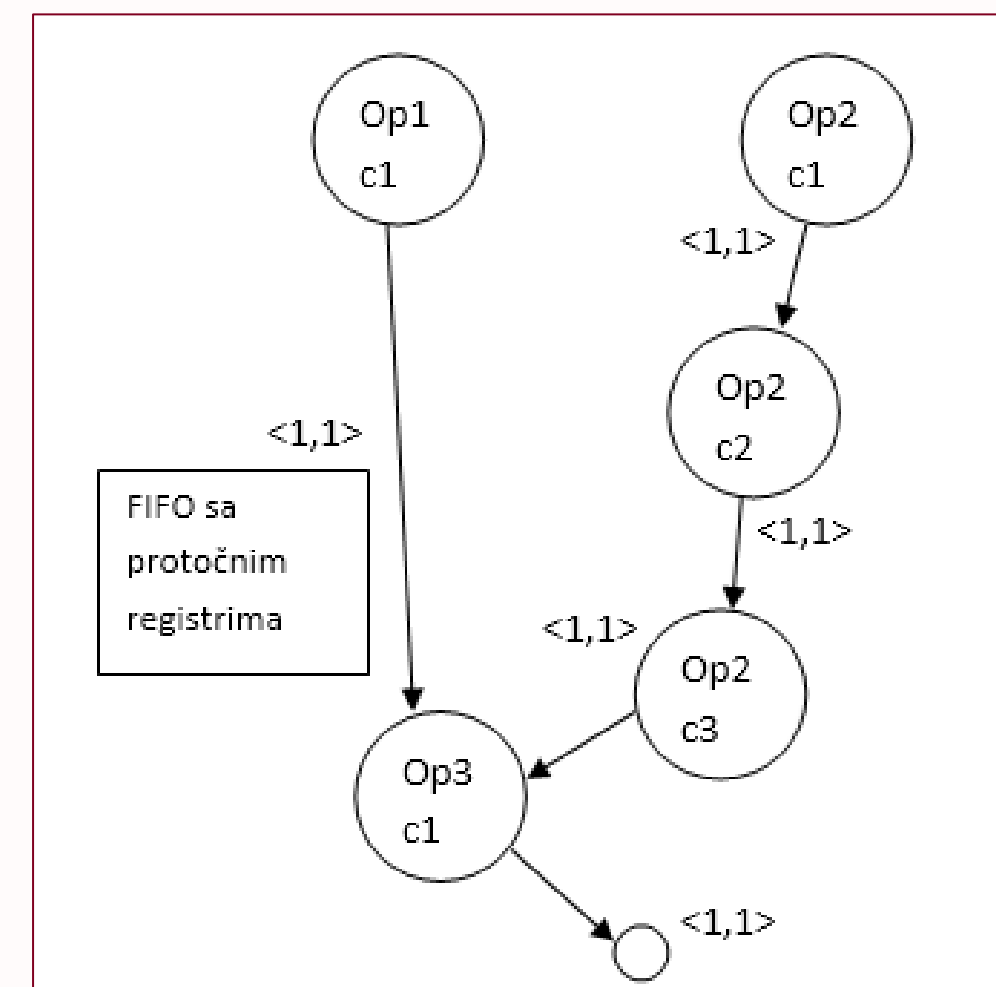
DoAcross

- Nikakvom transformacijom petlje nije moguće izvršiti sve iteracije u paraleli.
 - Ograničenje u paralelizmu.
 - U grafu beskonačno razmotane petlje postoje putevi čija je dužina srazmerna broju iteracija petlje.
- Postiže se veći nivo paralelizma, ali postoje limiti na osnovu zavisnosti po podacima.

Primer DoAll petlje

```
DO I = 1,100  
Op1:    C(I) := A(I) + B(I)  
Op2:    F(I) := D(I) * E(I)  
Op3:    H(I) := C(I) + F(I)  
END
```

Kod DoAll petlje

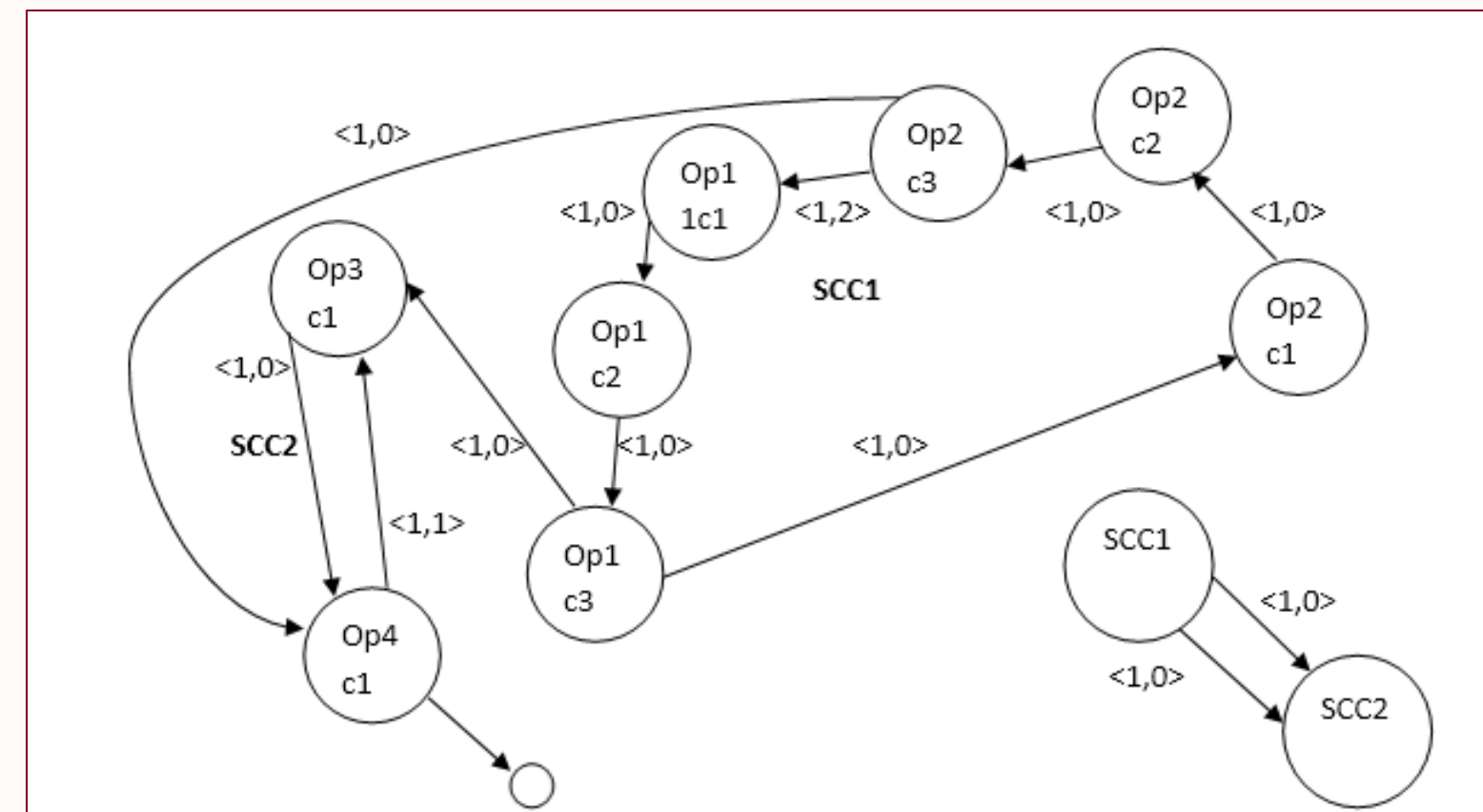


DDG graf petlje

Primer DoAcross petlje

```
DO I = 3,100  
Op1:    C(I) := A(I-2) * B(I)  
Op2:    A(I) := D(I) * C(I)  
Op3:    E(I) := C(I) + F(I-1)  
Op4:    F(I) := E(I) + A(I)  
END
```

Kod DoAcross petlje



DDG graf petlje

Softverska protočnost (1)

- **Software pipelining**
- Po ugledu na hardversku protočnost.
- Formira se nova iteracije petlje (kernel).
- Delimično preklapanje originalnih iteracija petlje u cilju postizanja paralelnijeg koda.
- Stepen preklapanja ograničen je zavisnosti po podacima za operacije iz različitih iteracija i resursima mašina.

$$L^k = \alpha K^m \Omega$$

- **Predpetlja** (*prolog*) – kod koji u novoj modifikovanoj petlji prethodi iteraciji.
- **Postpetlja** (*postlog*) – kod koji sledi nakon poslednje iteracije modifikovane petlje.
- Analogno popunjavanju i pražnjenju protočnog niza.

Softverska protočnost (2)

- U novoj petlji, preklopljene su operacije iz n originalnih iteracija petlje.
- Predpetlja i postpetlja sadrže delove $n - 1$ iteracija petlje.
- **Nivo softverske protočnosti (LSP)** – opseg originalnih iteracija petlje u novoj petlji = $n - 1$.
- Predpetlja i postpetlja – generalizovano ljuštenje petlje.
 - Ljuštenje delova iteracija u cilju optimizacije nove iteracije petlje.
- Softversku protočnost moguće je realizovati i kada broj iteracija petlje nije poznat u vreme prevodjenja.
 - Obnavljanje grananja i spojeva kako bi se zaobišle nepotrebne iteracije petlje.
 - Mora se izvršiti komplement izvršenih iteracija.

Softverska protočnost (3)

- Formiranjem nove iteracije petlje, moguće je promeniti iteracione distance sa grana originalne petlje.
- Zavisnosti preko granica iteracije mogu se pretvoriti u zavisnosti u okviru iste iteracije.
 - Izmenjen je graf iteracije.
- Optimizacija – traženje nove iteracije čiji aciklični graf ima minimalan kritičan put.
 - Minimizovati stepen softverske protočnosti u cilju smanjenja eksplozije koda.

Softverska protočnost (4)

- Generalniji pristup formiranja petlje softverske protočnosti:
 - Na osnovu grafa zavisnosti po podacima, moguće je modifikovati redosled operacija koje pripadaju iteracijama.
 - Nove iteracije dobijaju se na osnovu grafa zavisnosti.
 - Dalje ih je moguće optimizovati.
- Semantička ispravnost nove petlje:
 - Nijedna operacija iz predpetlje ne sme da zavisi od operacija iz iteracije petlje ili postpetlje.
 - Nijedna operacija iz iteracije petlje ne sme da zavisi od operacija iz kasnijih iteracija petlje ili postpetlje u novoformiranoj petlji softverske protočnosti.

Primer DoAll petlje - zadatak (1)

- Dat je kod DoAll petlje.
- Napraviti petlju softverske protočnosti koja se izvršava za jedan ciklus.
- Pretpostaviti neograničene hardverske resurse, uključujući registre.
- Pretpostaviti da su sve funkcionalne jedinice međusobno povezane i da se rezultati jedne jedinice sprovode do ulaza u drugu jedinicu za 1 ciklus.
- Množači/delitelji izvršavaju operacije za 3 ciklusa, a ALU jedinice za 1 ciklus. Pretpostaviti da su na raspolaganju FIFO memorije sa ulaznim nizovima. Za pamćenje rezultata mogu se koristiti pomoćni registri (kada treba zakasnuti rezultat) i pamćenje traje najmanje 2 ciklusa, i FIFO sa protočnim registrima na krajevima i kontrolnim signalom za čitanje gde je minimalno kašnjenje takođe 2 ciklusa.
- Prikazati graf petlje nakon primene softverske protočnosti. Prikaz rasporeda dati tako da se za operaciju navode svi njeni ciklusi. Podrazumeva se da se ostali ciklusi izvršavaju u protočnim stepenima funkcionalne jedinice. Raspored napraviti za minimalan nivo softverske protočnosti.

Primer DoAll petlje - zadatak (2)

- Prilikom prikaza rasporeda usvojiti:
 - Operacije traju koliko i kašnjenja zavisnosti.
 - Operacije iz originalnih prvih iteracija u predpetlji se označavaju notacijom $Op_x i, j$, gde je x redni broj operacije u iteraciji, i indeks originalne iteracije, a j ciklus izvršavanja u pipeline-u za tu operaciju.
 - Operacije iz originalnih prvih iteracija u postpetlji se označavaju notacijom $Op_x i, j$, gde je x redni broj operacije u iteraciji, i indeks originalne iteracije, a j ciklus izvršavanja u pipeline-u za tu operaciju.
 - Istu notaciju koristiti i za novu iteraciju, samo indekse iteracije predstaviti pomoću indeksa petlje i , a pritom definisati opseg indeksa petlje.
 - Pretpostaviti da je u istom ciklusu moguće izvesti povratni skok i ispitivanje da li treba da se iskoči iz petlje.

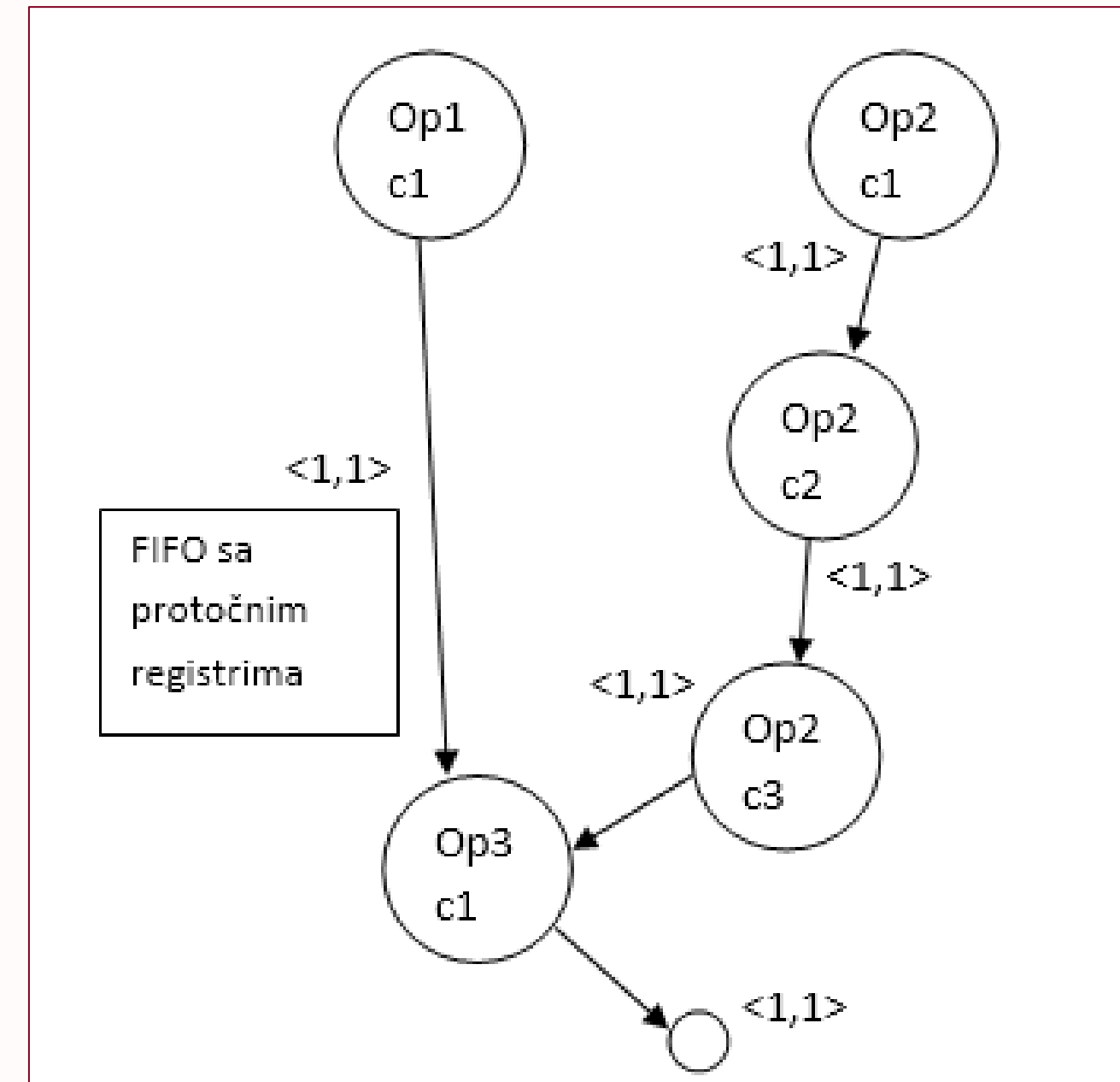
Primer DoAll petlje - zadatak (3)

```
        DO I = 1,100  
Op1:      C(I) := A(I) + B(I)  
Op2:      F(I) := D(I) * E(I)  
Op3:      H(I) := C(I) + F(I)  
        END
```

Kod DoAll petlje

Primer DoAll petlje - rešenje (1)

- Da bi petlja mogla da se izvrši za 1 ciklus, treba uvesti iteracione distance 1 za svaki ciklus svake operacije.
- Zato će novi DDG graf petlje (razmicanje komponenti jake povezanosti) izgledati kao na slici gde c1, c2 i c3 predstavljaju prvi drugi i treći ciklus operacije Op2.



Primer DoAll petlje - rešenje (2)

Op1 1,1	Op2 1,1					
Op1 2,1	Op2 2,1	Op1 1,1 → FIFO	Op2 1,2			
Op1 3,1	Op2 3,1	Op1 2,1 → FIFO	Op2 2,2	Op2 1,3	FIFO → Op3 1,1	
Op1 i,1	Op2 i,1	Op1 (i-1),1 → FIFO	Op2 (i-1),2	Op2 (i-2),3	FIFO → Op3 (i-2),1	Op3 (i-3),1
		Op1 100,1 → FIFO	Op2 100,2	Op2 99,3	FIFO → Op3 99,1	Op3 98,1
				Op2 100,3	FIFO → Op3 100,1	Op3 99,1
						Op3 100,1

Primer DoAll petlje - rešenje (3)

- Dužina kritičnog puta u grafu je 4.
- Nivo softverske protočnosti je 3 (što je takodje i minimalni nivo), kao i broj ciklusa u predpetlji.
- Ovo rešenje podrazumeva da operacije počinju najranije što mogu, što je samo jedan od mogućih rasporeda, npr. najkasniji kada Op1 1,1 kreće od trećeg ciklusa.
- Nije uneto inkrementiranje indeksa petlje i, ni ispitivanje uslova za iskakanje iz petlje, što se naravno radi u svakoj iteraciji (ciklusu).
- Takodje, nije prikazano ubacivanje ulaznih argumenata iz FIFO-a koji sadrže nizove, jer je trivijalno.

Primer DoAcross petlje - zadatak (1)

- Dat je kod DoAcross petlje.
- Napraviti petlju softverske protočnosti koja se izvršava za najmanji broj ciklusa.
- Pretpostaviti neograničene hardverske resurse, uključujući registre.
- Pretpostaviti da su sve funkcionalne jedinice medjusobno povezane i da se rezultati jedne jedinice sprovode do ulaza u drugu jedinicu za 1 ciklus.
- Množači/delitelji izvršavaju operacije za 3 ciklusa, a ALU jedinice za 1 ciklus. Pretpostaviti da su na raspolaganju FIFO memorije sa ulaznim nizovima. Za pamćenje rezultata mogu se koristiti pomoćni registri (kada treba zakasniti rezultat) i pamćenje traje najmanje 1 ciklus, i FIFO sa protočnim registrima na krajevima i kontrolnim signalom za čitanje gde je minimalno kašnjenje takođe 1 ciklus. Izlaz Op4 se pamti u izlaznom FIFO registru.
- Prikazati graf petlje nakon primene softverske protočnosti. Prikaz rasporeda dati tako da se za operaciju navode svi njeni ciklusi. Podrazumeva se da se ostali ciklusi izvršavaju u protočnim stepenima funkcionalne jedinice. Raspored napraviti za minimalan nivo softverske protočnosti.

Primer DoAcross petlje - zadatak (2)

- Prilikom prikaza rasporeda usvojiti:
 - Operacije traju koliko i kašnjenja zavisnosti.
 - Operacije iz originalnih prvih iteracija u predpetlji se označavaju notacijom $Op_x i, j$, gde je x redni broj operacije u iteraciji, i indeks originalne iteracije, a j ciklus izvršavanja u pipeline-u za tu operaciju.
 - Operacije iz originalnih prvih iteracija u postpetlji se označavaju notacijom $Op_x i, j$, gde je x redni broj operacije u iteraciji, i indeks originalne iteracije, a j ciklus izvršavanja u pipeline-u za tu operaciju.
 - Istu notaciju koristiti i za novu iteraciju, samo indekse iteracije predstaviti pomoću indeksa petlje i , a pritom definisati opseg indeksa petlje.

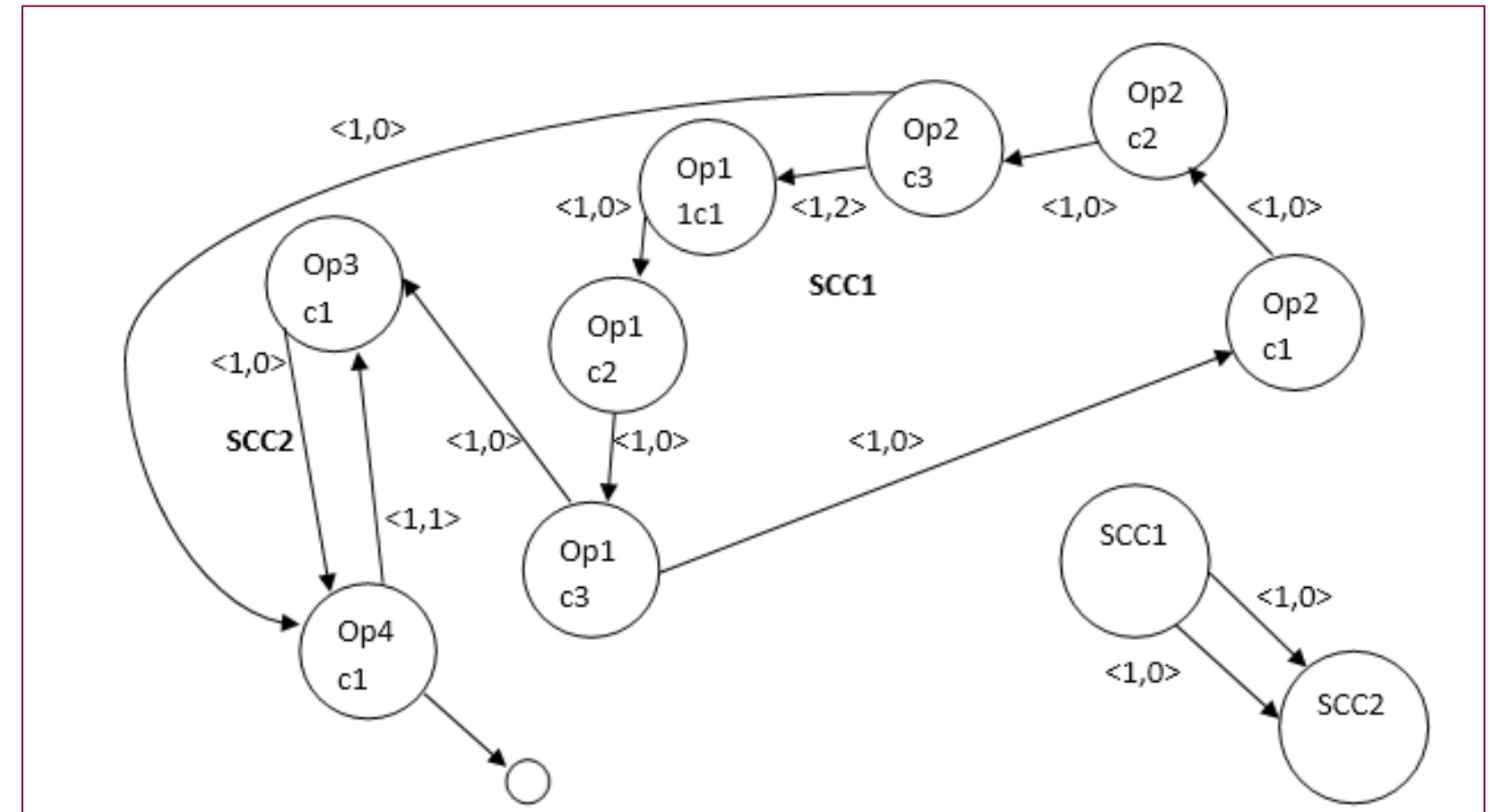
Primer DoAcross petlje - zadatak (3)

```
DO I = 3,100
Op1:    C(I) := A(I-2) * B(I)
Op2:    A(I) := D(I) * C(I)
Op3:    E(I) := C(I) + F(I-1)
Op4:    F(I) := E(I) + A(I)
END
```

Kod DoAcross petlje

Primer DoAcross petlje - rešenje (1)

- Pronaći komponente jake povezanosti i kritičan zatvoreni put.
- Za SCC1 minimalno trajanje iteracije je 3 ciklusa, a za SCC2 2 ciklusa.
- Izračunavanje SCC1 mora da prednjači za jedan ciklus u odnosu na SCC2 kako bi se eliminisale grane izmedju njih u acikličnom grafu.



Primer DoAcross petlje - rešenje (2)

Op1 1,1					
Op1 1,2					
Op1 1,3					
Op1 2,1	Op2 1,1	Op1 1,3 → FIFO1			
Op1 2,2	Op2 1,2		FIFO1 → Op3 1,1		
Op1 2,3	Op2 1,3			Op3 1,1	
Op1 i,1	Op2 (i-1),1	Op1 (i-1),1 → FIFO1			Op4 (i-2),1
Op1 i,2	Op2 (i-1),2		FIFO1 → Op3 (i-1),1		Zakasniti rezultat
Op1 i,3	Op2 (i-1),3			Op3 (i-1),1	
	Op2 100,1	Op1 100,1 → FIFO1			Op4 99,1
	Op2 100,2		FIFO1 → Op3 100,1		Zakasniti rezultat
	Op2 100,3			Op3 100,1	
					Op4 100,1

Primer DoAcross petlje - rešenje (3)

- U SCC1 smanjiti iteracionu distancu izmedju Op2 c3 i Op1 c1 na 1, a grana izmedju Op1 c3 i Op2 c1 dobija iteracionu distancu 1.
- Nije uneto inkrementiranje indeksa petlje i, ni ispitivanje uslova za iskakanje iz petlje, što se naravno radi u svakoj iteraciji.
- Takodje, nije prikazano ubacivanje ulaznih argumenata iz FIFO-a koji sadrže nizove, i izbacivanje rezultata Op3 u izlazni FIFO, jer je trivijalno.
- Zakašnjavanje rezultata Op4 za 1 ciklus je neophodno da se zatvoreni put iz ciklusa SCC2 prilagodi minimalnom mogućem trajanju od 3 ciklusa za iteraciju.

Primer DoAcross petlje - rešenje (4)

- Zatvoreni put SCC2 prilagodjava se minimalnom trajanju od 3 ciklusa za iteraciju zakašnjavanjem rezultata operacije 4 za jedan ciklus.
- Minimalan nivo softverske protočnosti za izvršavanje je posledica udaljavanja komponenti jake povezanosti za jednu iteraciju i sečenja zatvorenog puta u SCC1, pa je morao da bude 2.
- Predpetlja i postpetlja nisu mogle biti drugačije ako ne želimo da povećamo broj ciklusa u postpetlji. Ako je dozvoljeno povećavanje broja ciklusa u postpetlji, mogao je da se zadrži rezultat Op1 još za 1 ciklus, pa da se sve operacije vezane za SCC2 zakasne za 1 ciklus.

Primer DoAcross petlje - rešenje (5)

- Bar jedan rezultat izmedju komponenti jake povezanosti se morao čuvati u registrima ili FIFO baferu, jer rezultati za dve grane između SCC stižu u različitim trenucima – potrebno je zakasniti rezultat koji stiže dva ciklusa pre.
- Da li je petlja mogla da bude sa nepoznatim brojem iteracija u vreme prevodjenja, a da traje 1 ciklus, ako Op4 generiše flag za iskakanje iz petlje?
 - Ne, jer bi se svakog ciklusa u predpetlji moralo ispitivati da li treba da se iskače iz predpetlje, a flag nije na raspolaganju. Moralo bi da se primeni predikatsko izvršavanje sa 3 bita predikata i da se izvršavaju operacije predpetlje i svih mimoilazećih grana za iskakanje iz petlje, što nije praktično.