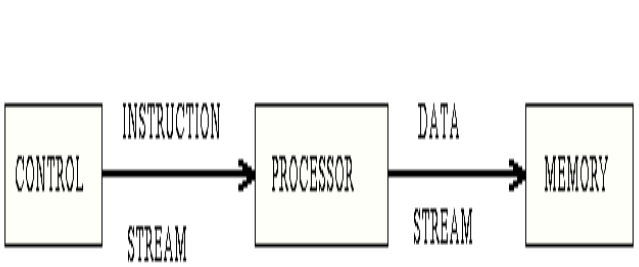
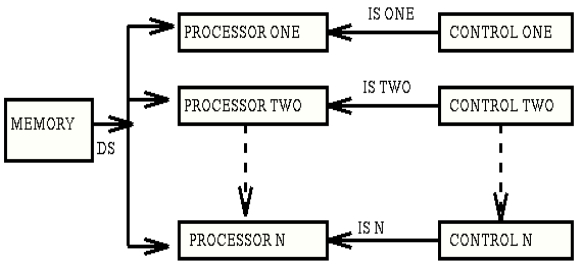
PARALELNI SISTEMI, 9.2016., Jovana

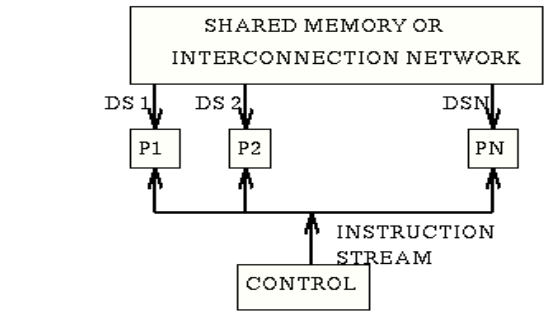
2016. (sept, jun, april, kol, jan), 2015. cela, 2014. (+1)

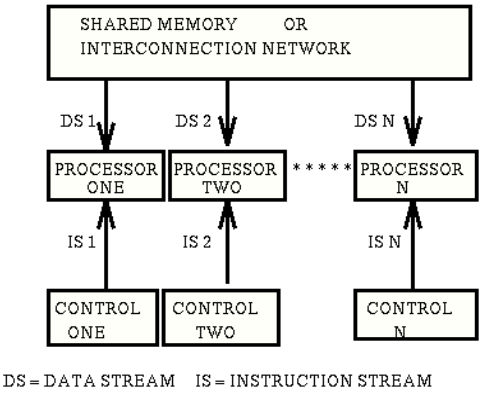
1. **Navesti Flynn-ovu klasifikaciju računarskih arhitektura. Ukratko objasniti karakter. svake klase i nacrtati koncepcijski dijagram arhitekture.**

   Kod Flinove klasifikacije Von Neumanovih računara podela je izvršena prema broju jednovremenih tokova instrukcija i tokova podataka na:
   - SISD (Single Instruction Single Data stream) – računari sa jednim tokom instrukcija i jednim tokom podataka
   - MISD (Multiple Instruction Single Data stream) – računari sa višestrukim tokom instrukcija i jednim tokom podataka
   - SIMD (Single Instruction Multiple Data stream) – računari sa jednim tokom instrukcija i višestrukim tokom podataka
   - MIMD (Multiple Instruction Multiple Data stream) – računari sa višestrukim tokom instrukcija i višestrukim tokom podataka









MISD:
N procesora, svaki sa sopstvenom upravljačkom jedinicom koji dele pristup zajedničkoj memoriji. Postoji N tokova instrukcija i jedan tok (niz) podataka. Paralelizam se postiže tako što više procesora izvršava različite instrukcije u isto vreme nad istim podatkom. MISD mašine su od koristi kod onih izračunavanja gde isti podatak treba obraditi na više načina .

SIMD:
Postoji N procesora čijim radom upravlja jedinstvena upravljačka jedinica (control). Ovo je isto kao da svaki procesor sadrži identičnu kopiju programa. Postoji N tokova podataka, po jedan za svaki procesor, što znači da se različiti podaci mogu koristiti u svakom procesoru. Procesori u sistemu rade sinhrono: svi izvršavaju istu instrukciju nad različitim podatkom.
MIMD:
To je najopštija i najmoćnija klasa računara. Postoji N procesora, N nizova instrukcija i N nizova podataka. Procesori rade asinhrono (mogu izvršavati različite instrukcije nad različitim podacima u isto vreme)

2. Ukratko objasniti ubrzanje i efikasnost. Šta je karakteristika efektivnih paralelnih algoritama? x2

Kod arhitektura kod kojih je uveden bilo koji vid poboljšanja, može se dati ocena o dobijenom poboljšanju sa stanovišta performansi korišćenjem mere UBRZANJE (S).

- S=vreme izvršenja programa na arhitekturi bez poboljšanja/vreme izvršenja programa na arh. sa izvedenim poboljšanjem
- Za paralelni sistem sa n procesora

  S(n)= T(1)/ T(n)

  T(1) vreme izvršenja programa na jednoprocesorskom sistemu

  T(n) vreme izvršenja programa na n-procesorskom sistemu

$$T(1) = T_s + T_p$$

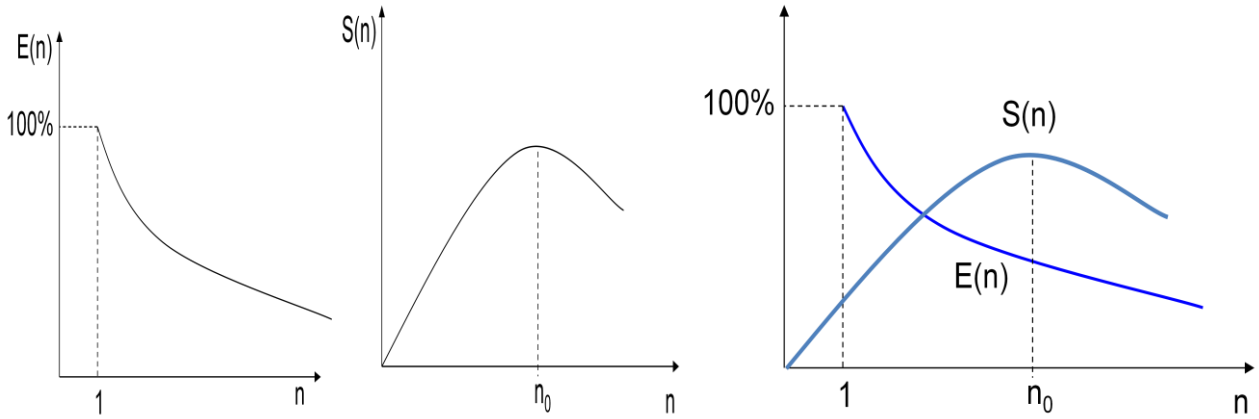
$$T(n) = T_s + \frac{T_p}{n} + T_o(n)$$

Efikasnost se definiše kao odnos ubrzanja i broja procesora.

- E(n)=S(n)/n, $0 \le E \le 1$
- Daje ocenu o srednjoj iskorišćenosti procesora u sistemu kada su svi procesori angažovani na rešavanju istog problema
  - ako se zanemare U/I aktivnosti, efikasnost jednoprocesorskig sistema je 1 (100%)

Ubrzanje i efikasnost ne treba posmatrati odvojeno

- sa porastom broja procesora ubrzanje raste (do odredjene granice)
- sa porastom broja procesora efikasnost opada
- S(n)*E(n) je najbolja ocena



Karakteristika efektivnih paralelnih algoritama - Efektivni algoritmi imaju linearno ubrzanje.

$$\lim_{m\to\infty} S(n) = \lim_{m\to\infty} \frac{n}{1+(n-1)\alpha(m)} = n$$

$\alpha = \alpha(m)$, m – obim problema

lim $\alpha(m)=0$ (m->beskonačno),

Kod najvećeg broja naučno-tehničkih aplikacija sa porastom obima problema α (Amdahlova frakcija – udeo vremena izvršenja sekvencijalnog dela programa u ukupnom vremenu izvršenja) teži 0.

Iz ove formule zaključujemo da ako alfa(m) teži nuli – sa povećanjem broja procesora n povećava se i ubrzanje S(n).

Ovo znači da se sa povećanjem broja procesora povećava brzina izvršavanja programa.

3. Za upravljanje izvršenjem instrukcija Scoreboard koristi nekoliko tabela. Koje su to tabele? Ukratko objasniti značenje pojedinih polja u svakoj tabeli. x5

Na osnovu evidencije koju vodi, Scoreboard kontroliše napredovanje instrukcije kroz protočni sistem. Postoje tri tabele preko koje Scoreboard koristi za upravljanje izvršenjem instrukcija:

1) Status instrukcije (Instruction status):
   Govori u kojoj od 4 faze izvršenja se instrukcija nalazi.

2) Status funkcionalne jedinice (Functional unit status):
   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)

3) 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.

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 ima četiri koraka koji zamenjuju ID, EX i WB:
4) Issue – detektuje strukturne i WAW hazarde
5) Read operands – rešavaju se RAW hazardi dinamički
6) Execution – izvršenje nakon pribavljanja operanada
7) Write results – proverava WAR hazarde

4. Preko kojih tabela Tomasulov algoritam upravlja izvršenjem instrukcija i kako one izgledaju? Objasniti značenje svakog polja u tabelama. x3

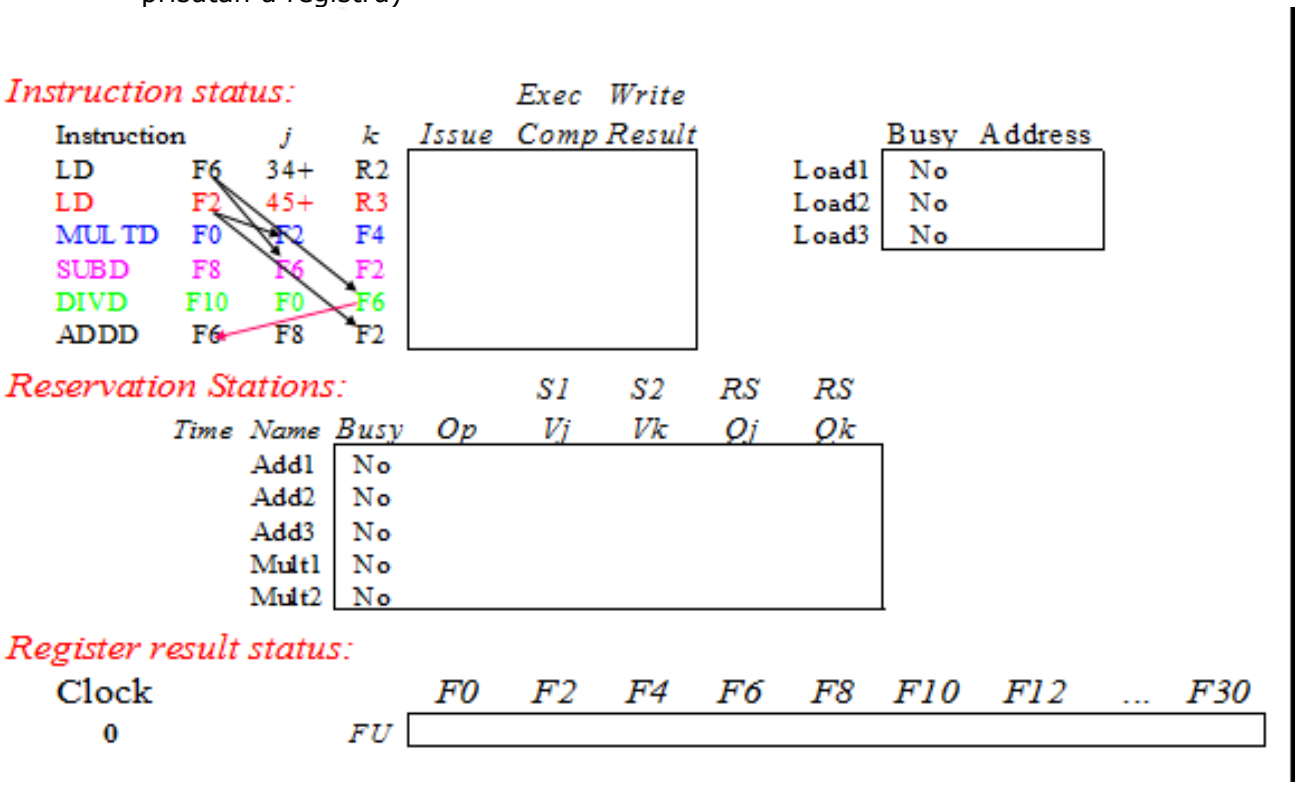
Tomasulov algoritam upravlja izvršenjem instrukcija preko četiri tabele:

1) Status instrukcije (Instruction status):
   Govori u kojoj od 3 faze izvršenja se instrukcija nalazi.

2) Rezervacione stanice (Reservation Stations):
   Svaka rezervaciona stanica ima 6 polja:
   - Busy – označava da li je RS i odgovarajuća FU slobodna
   - Op – operacija koja treba da se izvrši (npr. + ili -)
   - Vj, Vk – vrednosti izvornih operanada S1 i S2
     - Store baferi imaju smo jedno V polje u kome se pamti rezultat koji treba da se upiše u memoriju
   - Qj, Qk – rezervacione stanice koje koje generišu izvorne operande
     - nema ready flegova kao kod Sc; Qj,Qk=0 => ready
     - store baferi imaju samo jedno Q polje za oznaku RS koja proizvodi rezultat

3) Za Load i Store:
   A – adresna informacija za load ili store. U startu sadrži neposredni operand, zatim efektivnu adresu kada se izračuna

4) Tabela statusa registra rezultata (Register result status):
   Qi ukazuje koja funkcionalna jedinica će izvršiti upis (ako postoji)
   - blanko ili 0 ako nema aktivne instrukcije koja će upisati rezultat (tj. podatak je prisutan u registru)



Tomasulo ima tri koraka koji zamenjuju ID, EX i WB:
1) Issue - eliminišu se **WAR** i **WAW** hazardi
2) Execution – razrešavaju se **RAW** hazardi
3) Write results

5. **Zaokružiti šta od navedenog važi za Scoreboard (SC) a šta za Tomasulov algoritam (TA).**

   - **izvršenje instrukcija van redosleda (SC, TA)**
     oba obavljaju izv.inst.van redosleda, to je i poenta dinamičkog preuređivanja instrukcija instrukcije se izdaju van redosleda pribavljanja
   - **koriste pribavljanje unapred (bypassing, forwarding) (SC, TA)**
     SCOREBOARD NE KORISTI PRIBAVLJANJE UNAPRED POŠTO SE OPERANDI INSTR. ČITAJU TEK POŠTO SU OBA DOSTUPNA
   - **koriste Common Data Bus (SC, TA)**
   - **vrše preimenovanje registara (SC, TA)**
   - **centralizovano upravljanje (SC, TA)**
   - **omogućavaju dinamičko odmotavanje petlje (SC, TA) x4**
     Začto Tomasulo može da preklapa iteracije petlje?
     Preimenovanje registara
     - *Različite iteracije koriste različite fizičke destinacije za registre (dinamičko odmotavanje petlje)*

6. **Koje aktivnosti se obavljaju u ISSUE fazi kod**
   a. **Scoreboard tehnike**
   b. **Tomasulovog algoritma**

Kod Scoreboard tehnike u ISSUE fazi se obavljaju sledeće aktivnosti:

Ako je funkcionalna jedinica slobodna i ni jedna druga aktivna instrukcija nema isti odredišni registar, Scoreboard 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

Kod Tomasulovog algoritma u ISSUE fazi se obavljaju sledeće aktivnosti:

- Ako je u pitanju FP operacija, instrukcija se izdaje, ako postoji slobodna rezervaciona stanica i šalju se operandi, ako su u registrima.
- Ako je u pitanju load ili store instrukcija, ona se izdaje ako postoji slobodan load ili store bafer.
- Ako nema slobodne rezervacione stanice ili load/store bafera onda postoji strukturni hazard i instrukcija se zaustavlja u ovom koraku se vrši i proces preimenovanja registara.

U ovoj fazi, instrukcije se izdaju za izvršavanje ako su svi operandi i rezervacione stanice spremne ili ako su zaustavljene. Registri su preimenovani u ovom koraku, eliminišući WAR i WAW opasnosti.

7. **Koje faze u izvršenju instrukcija postoje i koje se aktivnosti u njima obavljaju kod**
   a. **Scoreboard tehnike**
   b. **Tomasulovog algoritma**

Kod Scoreboard tehnike postoje četiri faze:

1) Issue, Read operands, Execution, Write results

U ISSUE fazi se obavljaju sledeće aktivnosti:

(prethodno pitanje)

Read operands faza:

Scoreboard 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.
- Scoreboard rešava RAW hazarde dinamički u ovom koraku i instrukcija može biti poslata u izvršenje van redosleda.

Execution (izvršenje):

Funkcionalna jedinica (FU) otpočinje izvršenje nakon pribavljanja operanada. Kada je rezultat spreman FU obaveštava Scoreboard.

Write results (upis rezultata):

Kada je Scoreboard 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.

Kod Tomasulovog algoritma postoje tri faze:

Issue, Execution, Write results

U ISSUE fazi se obavljaju sledeće aktivnosti:

(prethodno pitanje)

EXECUTION (izvršenje):

Ako neki operand nije dostupan, nadgleda se CDB (Common data bus).

- Kada operand postane dostupan smešta se u odgovarajuću RS (rezervacionu stanicu).
- Kada su oba operanda dostupna, izvršava se FP (floating point) operacija.
- Vrši se provera RAW hazarda (čekanjem da operandi postanu dostupni razrešavaju se RAW hazardi).
- Load i store zahtevaju dvostepeno izvršenje:
  - u prvom koraku se izračunava efektivna adresa. a zatim se pamti u load/store bafer
  - Pribavljanje u load bafer se obavlja čim je memorijska jedinica raspoloživa
  - upisi u store bafere čekaju na podatak pre nego što se obavi upis u memoriju
- (Ako jedan ili više operanada nije dostupan sačekaj da operand postane dostupan.
- Kada su svi operandi dostupni, onda ako je instrukcije čitanje ili pisanje
  - izračunaj efektivnu adresu kada je bazni registar dostupan, i postavi ga u bafer za čuvanje i vađenje
    - Ako je instukcija uvežena, izvrši je čim memorijska jedinica postane dostupna
    - Ako nije, sačekaj da vrednost bude postavljena pre nego što je pošalješ u memorijsku jedinicu
- U suprotnom slučaju, instrukcija je ALU operacija i izvrši je na odgovarajućoj funkcionalnoj jedinici.)

WRITE RESULTS (upis rezultata):

- Ako je instrukcija bila ALU operacija
  - Ako je rezultat dostupan, napiši ga u CDB i odatle u registre i bilo koju rezervacionu stanicu koja čeka ovaj rezultat.
- U suprotnom slučaju, piši podatke u memoriju.

8. U kojoj od faza izvršenja kod Scoreboard i Tomalusovog algoritma se razrešavaju WAR/WAW/RAW hazardi? x2+1

Kod Scoreboard tehnike imamo četiri faze (Issue, Read operands, Execution I Write results).
U Issue fazi se detektuju **WAW** hazardi.
U Read operands fazi se **RAW** hazardi rešavaju dinamički.
U Write results fazi se razrešavaju **WAR** hazardi.

Kod Tomasulovog algoritma postoje tri faze (Issue, Execution i Write results).
U Issue fazi se eliminišu **WAR** i **WAW** hazardi.
RAW hazardi se razrešavaju u Execution fazi.

9. Kontrolni hazardi mogu dovesti do smanjena performansi protočnog sistema. Koje se hardverske tehnike koriste da bi se smanjili gubici uzrokovani kontrolnim hazardima? x4

Hazardi su situacije koje sprečavaju da izvršenje instrukcije otpočne u predvidjenom clock ciklusu. Svi hazardi se detektuju u ID fazi.

Postoje tri vrste hazarda:
- Strukturni
- Po podacima (RAW, WAR, WAW)
- Kontrolni

Kontrolni hazardi nastaju u pipeline procesorima prilikom izvršavanja instrukcija uslovnog grananja, dakle zbog instrukcija koje mogu promeniti sadržaj PC (branch, jump, call, return). Kontrolni hazardi se mogu rešiti uz pomoć softverskih i hardverskih tehnika.

Tehnike koje se koriste da bi se smanjili gubici uzrokovani kontrolnim hazardima:
- Predviđanje da nema skoka *(kompajlerska tehnika)*
- Predviđanje da ima skoka *(kompajlerska tehnika)*

Softverske tehnike uz pomoć kojih se smanjuju gubici uzokovani kontrolnim hazardima:
- Statičke šeme:
  - Odmotavanje petlje (*kompajlerska tehnika*)
  - Zakašnjeno grananje (*kompajlerska tehnika*)

Hardverske tehnike koje se koriste da bi se smanjili gubici uzrokovani kontrolnim hazardima:
- Dinamičke šeme (koristi ponašanje grananja u toku izvršenja da predvidi ishod):
  - BPB
  - BTB
  - Korelacioni predikatori
  - Hibridni

10. Navesti i ukratko objasniti hardverske tehnike za smanjenje kašnjenja urzokovanog kontrolnim hazardima. x2

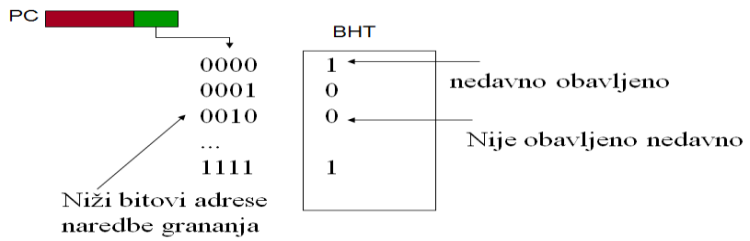
Dinamička predikcija grananja se razlikuje od statičke jer koristi ponašanje grananja u toku izvršenja programa da predvidi ishod grananja.

Hardverske tehnike koje se koriste da bi se smanjili gubici uzrokovani kontrolnim hazardima:
- Dinamičke šeme (koristi ponašanje grananja u toku izvršenja da predvidi ishod):
  - BPB
  - BTB
  - Korelacioni predikatori
  - Hibridni

BPB (Branch Prediction Buffer):

Tabela koja sadrži jedno ili dvo-bitne prediktore. BPB je mala memorija indeksirana nižim bitovima naredbe grananja.

Najjednostavnija varijanta koristi samo jedan bit za predikciju koji ukazuje da li se grananje nedavno obavilo (1) ili ne (0).



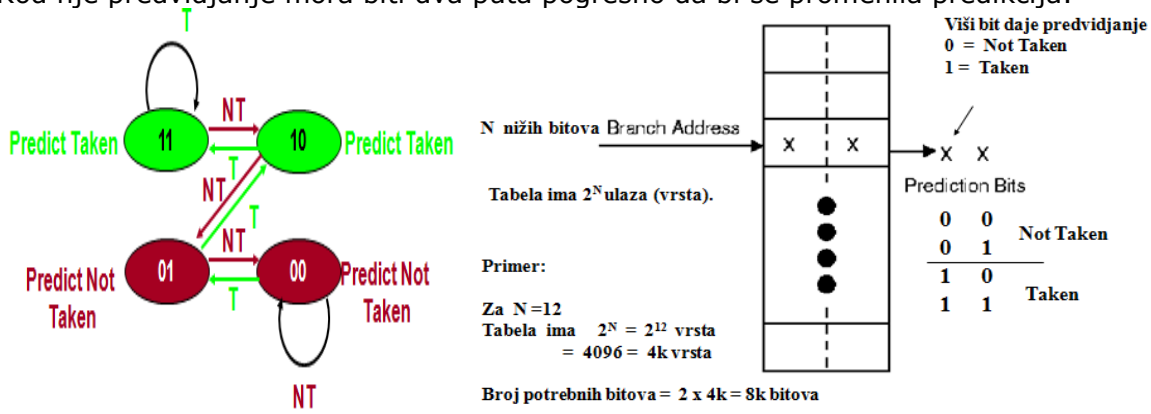

```
Loop1
        Loop2

    end Loop2

end Loop1
```

BPB je od koristi u sistemima kod kojih je adresa grananja poznata ranije od uslova grananja.
BHT se pristupa u ID fazi (kada se obavi dekodiranje).
S obzirom da se za indeksiranje BHT koriste niži bitovi (a ne cela) adrese naredbe grananja, predvidjanje koje se koristi može biti i od neke druge naredbe grananja koja je imala iste niže bitove adrese.
Problemi BHT:
Čak i ako se grananje skoro uvek obavlja (petlje), ova šema će imati dva pogrešna predvidjanja (umesto jednom). (slika desno gore)
Predvidjanje će biti pogrešno kod prve i poslednje iteracije Loop2:
Pogrešno predvidjanje kod poslednje iteracije je neizbežno (jer će bit biti postavljen na 1 – prethodno grananje se obavilo.
Pogrešno predvidjanje kod prve iteracije nastaje jer je bit bio postavljen na 0 prethodnim izvršenjem poslednje iteracije petlje, jer se grananje nije obavilo u toj iteraciji.

Da bi se poboljšala pouzdanost predvidjanja umesto 1-bitne koristi se 2-bitna šema.
Kod nje predvidjanje mora biti dva puta pogrešno da bi se promenila predikcija.



BTB (Branch Target Buffers):

Da bi se redukovali zastoji zbog naredbi grananja potrebno je znati sa koje adrese treba pribaviti instrukciju na kraju IF faze. To znači da moramo znati da li je još nedekodirana instrukcija instrukcija grananja, i ako jeste, koji je sledeći sadržaj programskog brojača. Na taj način bi se zastoji uzrokovani naredbom grananja sveli na 0. Bafer koji pamti predvidjene adrese za sledeću instrukciju nakon naredbe grananja zove se BTB.
BTB se pristupa u toku IF faze, korišćenjem adrese pribavljene instrukcije (moguće branch) da bi se pristupilo baferu:

Ako postoji pogodak, onda se zna adresa sledeće instrukcije na kraju IF faze, što je 1 clk ciklus ranije nego kod BHT (BPB).

Korelacioni predikatori:

Pouzdanost predvidjanja moguće je poboljšati ako se posmatra ponašanje i drugih naredbi grananja, a ne samo one čije ponašanje želimo da predvidimo (kao kod BTB i BPB). Prediktori koji koriste ponašanje drugih naredbi grananja da obave predvidjanje za tekuću naredbu grananja zovu se korelacioni ili 2-nivovski prediktori**.**

(1,1) korelacioni prediktor se može shvatiti kao da svako grananje ima dva posebna predikciona bita:

- jedno je predvidjanje koje se koristi ako se poslednje grananje nije obavilo (not taken NT)
- drugo predvidjanje koje se koristi ako se poslednje grananje obavilo (taken T)

Par predikcionih bitova se beleži zajedno:

- prvi bit predstavlja predvidjanje koje se koristi ako se poslednje grananje nije obavilo
- drugi bit predstavlja predvidjanje koje se koristi ako se poslednje grananje obavilo

predikcioni bitovi	predvidjanje ako poslednje grananje nije obavljeno	predvidjanje ako je poslednje grananje obavljeno
NT/NT (00)	NT	NT
NT/T (01)	NT	T
T/NT (10)	T	NT
T/T (11)	T	T

Korelacioni prediktori imaju veću pouzdanost predvidjanja od 2-bitnih prediktora a zahtevaju trivijalni dodatni hardver.

Hibridni:

Koriste kombinaciju dve ili više šema (obično dve) za predikciju grananja.

11. Objasnite rad brach prediction bafera i branch target bafera (BTB i BPB). x1+1

prethodno pitanje

12. U kojoj od faza izvršenja instrukcija se pristupa: **x2+1** (i KOL12015)

- **Baferima predikcije grananja (BPB)**
  BHT (BPB) se pristupa u ID fazi (kada se obavi dekodiranje).

- **Baferima ciljne adrese grananja (BTB)**
  BTB se pristupa u toku IF faze, korišćenjem adrese pribavljene instrukcije (moguće branch) da bi se pristupilo baferu.
  Ako postoji pogodak, onda se zna adresa sledeće instrukcije na kraju IF faze, što je 1 clk ciklus ranije nego kod BHT (BPB).

13. Program P sadrži dve naredbe uslovnog grananja CB-1 i CB-2. Ishodi grananja zavise od vrednosti promenljive A u programu. U sledećoj tabeli dati su ishodi naredbi grananja kada se program P izvršava za različite vrednosti promenljive A (T – grananje se obavlja, NT – grananje se ne obavlja) x3

Vrednost A	CB-1	CB-2
1	T	T
2	NT	NT
3	NT	T

a. Ako se koriste 1-bitni prediktori koji su inicijalizovani na ''NT'' popuniti tabelu pri čemu CB-P označava predikciju, CB-A akciju, a CB-U novu vrednost prediktora

A	CB-1:P	CB-1:A	CB-1:U	CB-2:P	CB-2:A	CB-2:U
2	NT	NT	NT	NT	NT	NT
1	NT	T	T	NT	T	T
3	T	NT	NT	T	T	T

U polje A samo prepišemo iz gornje tabele akciju koja se dogodi za datu vrednost.
U polje P (predikcija) na početku predviđamo Not Taken (i za CB-1 i za CB-2) (tako rečeno u zadatku), a onda to predviđanje u polju U (nova vrednost prediktora) menjamo ukoliko je akcija koja se izvršila različita od našeg predviđanja.
Naša predviđanja ćemo prepisati iz polja U iz jednog reda u polje Predikcije koje se nalazi u sledećem redu, za naredbu uslovnog grananja za koju to radimo.
Postupak nastavljamo do kraja popunjavanja tabele.

b. Ako se koristi (1,1) korelacioni prediktori koji su inicijalizovani na (1,0) i poslednje grananje se nije obavilo, popuniti tabelu pri čemu CB-P označava predikciju, CB-A akciju, a CB-U novu vrednost prediktora

A	CB-1:P	CB-1:A	CB-1:U	CB-2:P	CB-2:A	CB-2:U
2	T/NT	NT	NT/NT	T/NT	NT	NT/NT
1	NT/NT	T	T/NT	NT/NT	T	NT/T
3	T/NT	NT	T/NT	NT/T	T	T/T

Kod korelacionih prediktora se polje A popunjava na isti način.
U polje P se inicijalno stavlja (1,0) odnosno T/NT. Prvi bit predstavlja predvidjanje koje se koristi ako se poslednje grananje nije obavilo. Drugi bit predstavlja predvidjanje koje se koristi ako se poslednje grananje obavilo.
Gledamo da li se poslednje grananje obavilo ili ne i u odnosu na to menjamo drugi, odnosno prvi bit u vrednost onoga šta se jeste dogodilo (iz polja A). Prvi, odnosno drugi bit ostaje nepromenjen. To upisujemo u polje U (nova vrednost prediktora).
Novu vrednost prediktora prepisujemo u polje predikcije za sledeću vrednost promenljive A.
Postupak nastavljamo dok ne popunimo tabelu.
Idemo iz reda u red i u odnosu na prošlu A biramo da li menjamo 1. ili 2. bit za trenutnu A.

predikcioni bitovi	predvidjanje ako poslednje grananje nije obavljeno	predvidjanje ako je poslednje grananje obavljeno
NT/NT (00)	NT	NT
NT/T (01)	NT	T
T/NT (10)	T	NT
T/T (11)	T	T

Poslednja nar. gran. koja se izvršila ne mora biti i naredba za koju se vrši predikcija!

14. Program P sadrži tri naredbe uslovnog grananja CB-1 i CB-2 i CB-3. Ishodi grananja zavise od vrednosti promenljive A u programu. U sledećoj tabeli dati su ishodi naredbi grananja kada se program P izvršava za različite vrednosti promenljive A (T – grananje se obavlja, NT – grananje se ne obavlja) +1

Vrednost A	CB-1	CB-2	CB-3
1	T	T	T
2	NT	T	NT
3	NT	T	T

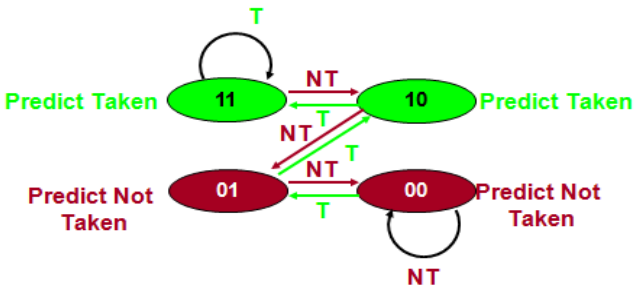
a. Ako se koriste 1-bitni prediktori koji su inicijalizovani na "NT" popuniti tabelu pri čemu CB-P označava predikciju, CB-A akciju, a CB-U novu vrednost prediktora.

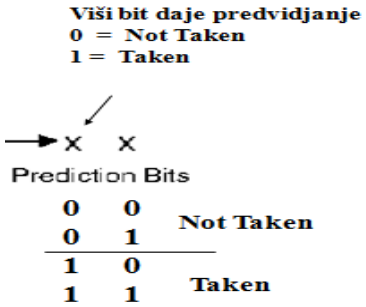
A	CB-1:P	CB-1:A	CB-1:U	CB-2:P	CB-2:A	CB-2:U	CB-3:P	CB-3:A	CB-3:U
2	NT	NT	NT	NT	T	T	NT	NT	NT
1	NT	T	T	T	T	T	NT	T	T
3	T	NT	NT	T	T	T	T	T	T

b. Ako se koriste 2-bitni prediktori koji su inicijalizovani na "00" popuniti tabeli pri čemu CB-P označava predikciju, CB-A akciju, a CB-U novu vrednost prediktora

A	CB-1:P	CB-1:A	CB-1:U	CB-2:P	CB-2:A	CB-2:U	CB-3:P	CB-3:A	CB-3:U
2	NT 00	NT	NT 00	NT 00	T	NT 01	NT 00	NT	NT 00
1	NT 00	T	NT 01	NT 01	T	T 10	NT 00	T	NT 01
3	NT 01	NT	NT 00	T 10	T	T 11	NT 01	T	T 10

Kod 2-bitne šeme predvidjanje mora biti dva puta pogrešno da bi se promenila predikcija. Dok popunjavamo tabelu, gledamo u šemu.





15. Zaokružiti šta od navedenog važi za Superskalarne (SP), a šta za VLIW procesore.

Planiranje izvršenja instrukcija se obavlja:

- **statički uz pomoć kompajlera (SP, VLIW)**
- **dinamički u fazi izvršenja programa (SP, VLIW)**
- **moguće su obe varijante (SP, VLIW) x3**

Kod SS su moguće obe varijante, dok je kod VLIW proc. isključivo STATIČKO planiranje izvršenja instrukcija.

16. Navesti osobine vektorskih instrukcija. x2

Vektorske instrukcije kao operande imaju linearna polja (vektore).

- Jedna vektorska instrukcija specificira veliki posao.
  - Ekvivalentana je izvršenju cele petlje u kojoj se u svakoj iteraciji izračunava jedan element vektora, ažuriraju indeksi i vrši grananje na početak petlje.

- Izračunavanje svakog elementa je nezavisno od prethodnog rezultata, što dozvoljava korišćenje veoma dubokih protočnih sistema bez šansi za nastupanje hazarda po podacima.
  - Odsustvo hazarda je odredjeno od strane kompajlera ili programera, kada je odlučeno da se može iskoristiti vektorska instrukcija.

- Pošto je cela petlja zamenjena vektorskom instrukcijom, kontrolni hazardi koji bi normalno nastupli zbog grananja na početak petlje ne postoje!

- Vektorske instrukcije koje pristupaju memoriji imaju poznate oblike pristupa.
  - Adekvatnim načinom smeštanja podataka može se kompenzovati velika latentnost pristupa memoriji.
  - Visoka latentnost pristupa glavnoj memoriji u odnosu na keš je amortizovana jer se jedan pristup inicira za ceo vektor, a ne za jednu reč (latentnost je vidljiva samo jednom za ceo vektor, a ne za svaki element vektora).

- Dobro vektorizovanim programskim kodom moguće je postići 10 do 20 puta veće brzine obrade od ekvivalentnog niza skalarnih instrukcija.

17. Koje osobine moraju da zadovolje matrice transformacije? Koje se elementarne transformacije mogu obavljati nad indeksnim promenljivim u petljama? x1

- Ako vektorizacija nije moguća po unutrašnjoj petlji, može se pokušati sa zamenom petlji ili nekim drugim transformacijama indeksnih promenljivih.
- Postoje tri elementarne transformacije koje se mogu obavljati nad indeksnim skupovima, tj. nad petljama.
  - Ove transformacije se opisuju pomoću matrica transformacija, T.

- Ove matrice moraju da poseduju sledeće osobine:
  - To su kvadratne matrice, što znači da vrše preslikavanje n-dimenzionalnog indeksnog prostora u n-dimenzionalni indeksni prostor.
  - To su celobrojne matrice.
  - $|\det T| = 1$

  - Da bi jedna transformacija mogla da se primeni nad indeksnim skupom a da to ne utiče na korektnost izračunavanja matrica transformacije T ne sme da menja znak vektora zavisnosti**.**
    - Ako je *d* vektor zavisnosti, T matrica transformacije, tada novi vektor zavisnosti $\hat{d}$, koji se dobija kada se T primeni na *d*, tj.

      $$\hat{d} = T \cdot d$$

      mora biti istog znaka kao i *d*.
      - Ako je d>0, tada mora i $\hat{d}$ >0 ili ako je d < 0, tada i $\hat{d}$ mora <0
      - **Za vektor se kaže da je pozitivan (negativan) ako mu je prvi ne-nulti element pozitivan (negativan).**

18. Da li je na sledećem gnezdu petlji dozvoljeno

```
for i=1,n
    for j=1,n
        c(i,j)=c(i+1,j-1)+1
endfor {i,j}
```

primeniti

a. **permutaciju petlji i i j**
b. **obrtanje po indeksnoj promenljivoj j**
c. **kompoziciju trasformacija pod a.i b.**

Svaki odgovor obrazložiti. U slučaju da je dozvoljeno izvršiti navedenu transformaciju prikazati kako izgleda transformisano gnezdo petlji. x3

Primeniti permutaciju petlji i i j nije dozvoljena.
Primeniti obrtanje po indeksnoj promenljivoj j nije dozvoljeno.
Kompozicija transformacija je dozvoljena.

Da bi neka transformacija mogla da se primeni nad indeksnim skupom ona ne sme da menja znak vektora zavisnosti da bi se sačuvale zavisnosti koje postoje u redosledu izračunavanja. Ako je *d* vektor zavisnosti, T matrica transformacije, tada novi vektor zavisnosti $\hat{d}$, koji se dobija kada se T primeni na *d*, tj. $\hat{d} = T \cdot d$ mora biti istog znaka kao i *d*.

Gledamo gnezdo.

Za permutaciju imamo matricu kojom množimo: $T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Primenom transformacije permutacije nad indeksnim skupom (I, J)^T dobijamo nove indeksne promenljive U i V na sledeći način

$$\begin{bmatrix} U \\ V \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} I \\ J \end{bmatrix} = \begin{bmatrix} J \\ I \end{bmatrix}$$

pri čemu je U = J, a V = I.
Granice za U i V odredjujemo na osnovu granica za indeksne promenljive J i I, respektivno.

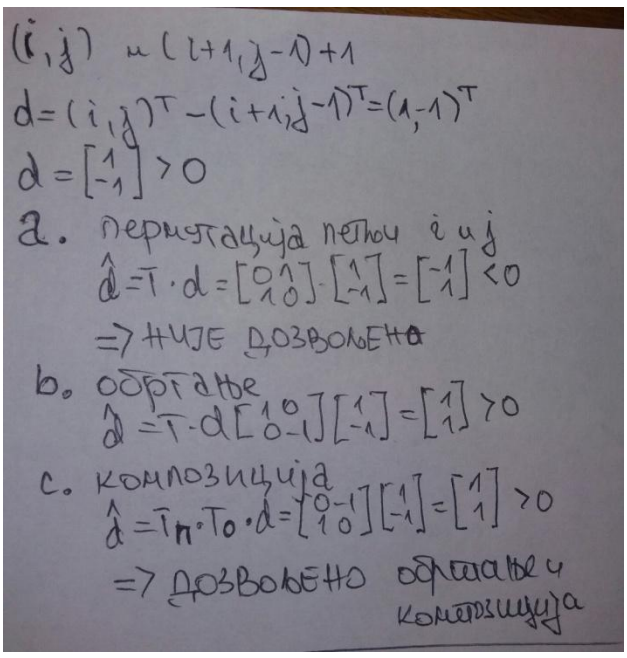
Za obrtanje po indeksnoj promenljivoj j matrica je: $T = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

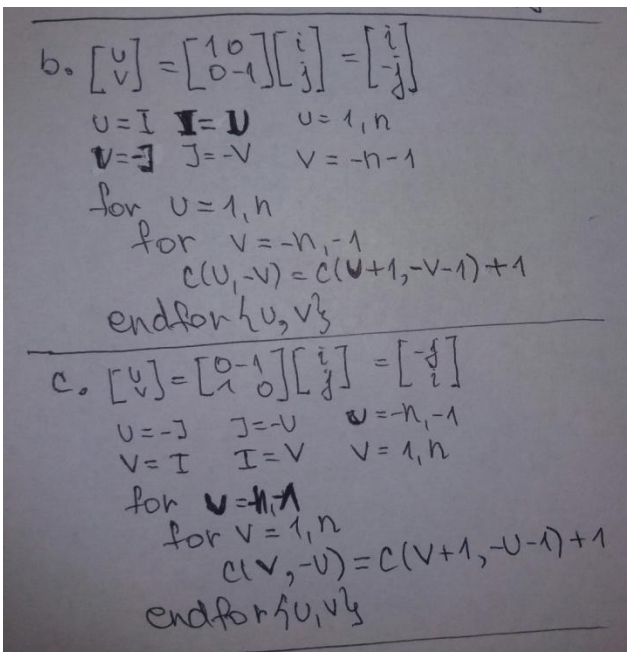
Novi indeksni skup dobijamo na sledeći način:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} i \\ -j \end{bmatrix}$$

Kompozicija transformacija matrica: $T = T_{\text{permutacija}} T_{\text{obrtanje}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} -j \\ i \end{bmatrix}$$





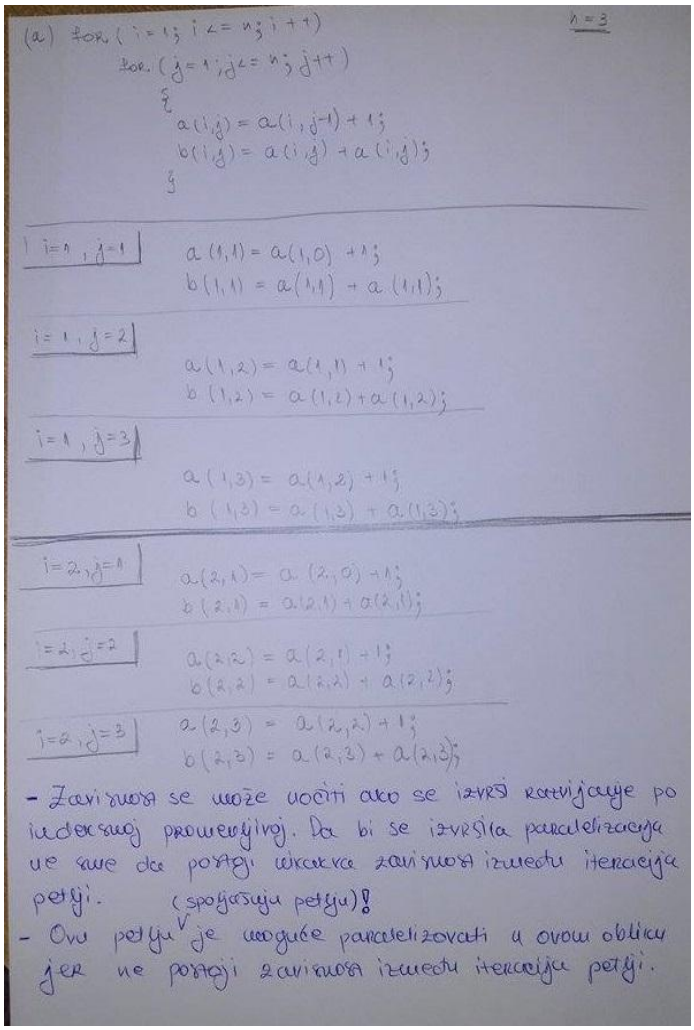
Najpre uočimo sve parove koji su zavisni.
Onda napravimo vektore zavisnosti tako što oduzmemo od prvog drugi.
Vektore zavisnosti poređamo paralelno jedan do drugog i dobijemo matricu zavisnosti D.
Zatim permutacionu/matricu obrtanja/kompozicionu matricu pomnožimo sa našom matricom zavisnos.
Na kraju gledamo da li su vektor(matrica) zavisnosti i vektor(matrica) koji dobijemo nakon permutacije/obrtanja/kompozicije veći ili manji od nula. Ako im se znakovi razlikuju, znači da bismo primenom transformacije narušili zavisnosti koje postoje u redosledu izračunavanja i to znači da određena transformacija nije dozvoljena.

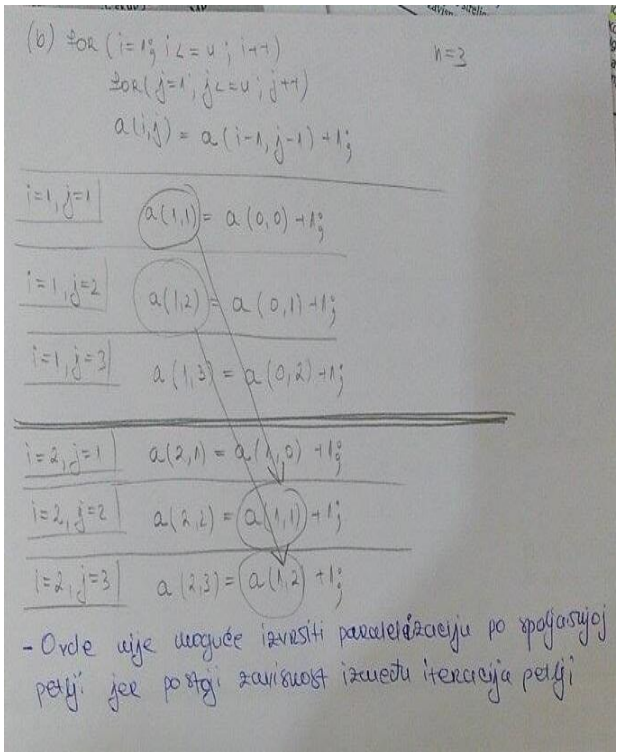
Za vektor se kaže da je pozitivan (negativan) ako mu je prvi ne-nulti element pozitivan (negativan).

19. Za svako od navedenih gnezda petlji odrediti da li se spoljašnja petlja može paralelizovati bez trasformacije tela petlje. Obavezno obrazložiti odgovor.

(a)
```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
    {
        a(i,j) = a(i, j-1) + 1;
        b(i,j) = a(i,j) + a(i,j);
    }
```

(b)
```
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        a(i,j) = a(i-1, j-1) + 1;
```
x4+1





20. Dato je sledeće gnezdo petlji:

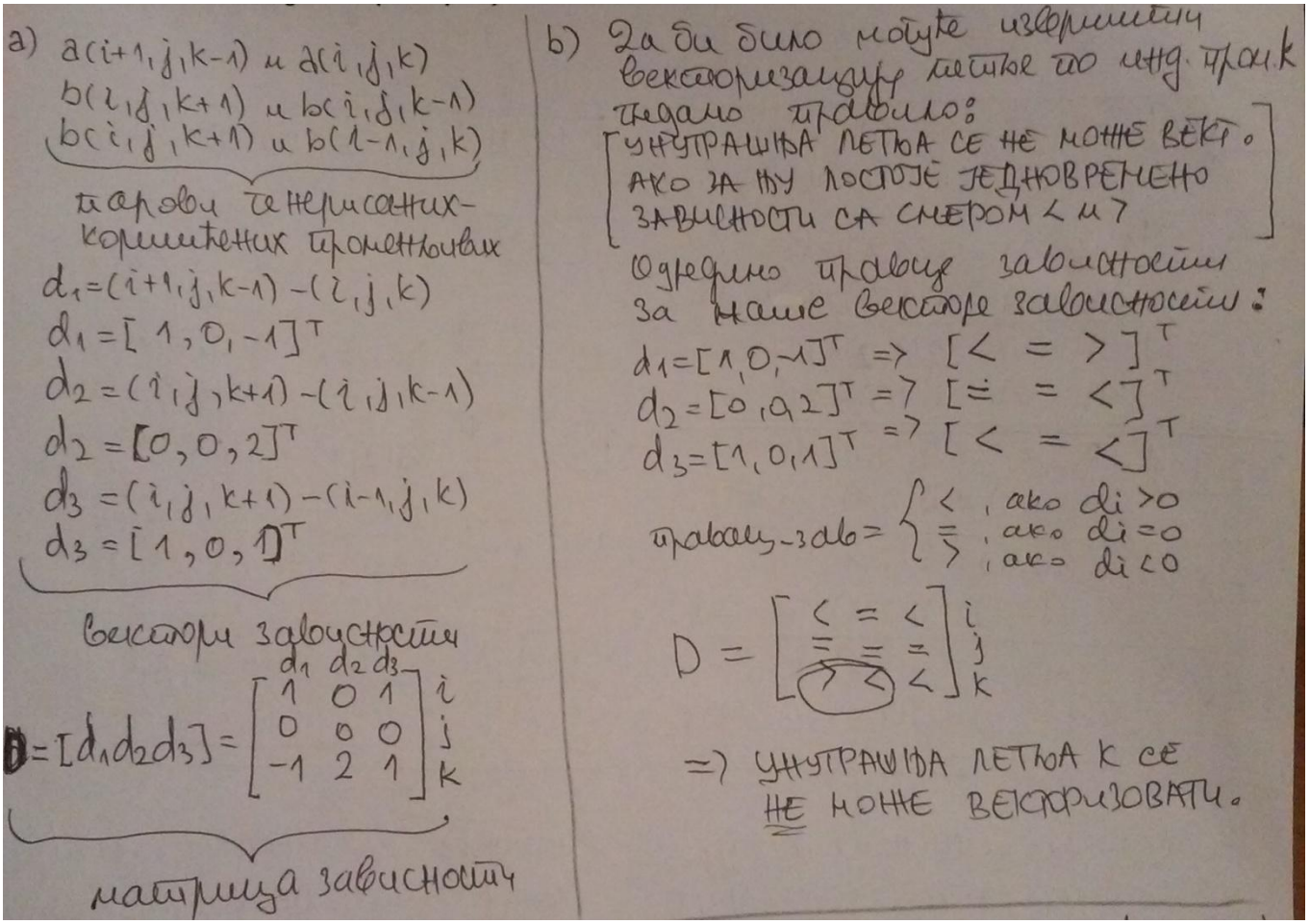
```
for i=1 to N
    for j=1 to M
        for k=1 to L
            a(i+1, j, k-1)=b(i,j,k-1)*2;
            b(i,j,k+1)=a(i,j,k)+b(i-1,j,k);
```

a) Naći matricu zavisnosti po podacima

b) Da li je moguće izvršiti vektorizaciju petlje po indeksnoj promenljivoj k?

c) Po kojoj indeksnoj promenljivoj se može obaviti paralelizacija?

Odgovore pod b) i c) obavezno obrazložiti.



Paralelizacija i vektorizacija nisu isti pojmovi. Kada se nešto izvršava u vektorskom obliku znači da se koristi protočnost a kod paralelizacije paralelizam u radu.
Kada je reč o ugnježđenim petljama pravilo je da se **unutrašnja vektorizuje a spoljašnja paralelizuje.**

Da bi se petlja paralelizovala po nekoj indeksnoj promenljivoj neophodno je da svi **pravci zavisnosti** budu = za tu indeksnu promenljivu. Odmotavanje je samo provera. Sve se radi preko vektora zavisnosti po podacima!
Vektorizacija može da se uradi SAMO po unutrašnjoj petlji, a provera se vrši preko **pravca zavisnosti kod vektora zavisnosti**.

21. Ako je latenost memorije 8 cik ciklusa a vektorski korak 5, koliko je najmanje memorijskih banaka potrebno da bi se elementima vektora pristupilo bez konflikata ako broj banaka mora biti stepen dvojke? x3+1

Konflikt kod pristupa memorij. banci NE nastupa ako važi sledeći uslov:

$$\frac{NZS(k,broj_{banaka})}{k} > latentnost$$

$$k=10 - vektorski\ korak,\ latentnost = 8,$$

$$\frac{NZS(10,2^x)}{10} > 8 \Rightarrow NZS(10,2^x) > 80$$

$$\begin{array}{ll} 10,16 & 2 \\ 5,8 & 2 \\ 5,4 & 2 \\ 5,2 & 2 \\ 5,1 & 5 \\ 1,1 & =80 \end{array} \Rightarrow x = 5 \Rightarrow broj_{banaka} = 2^5 = 32$$

Ovo sa 2kama je način za nalaženje NZS.
(NZS je najmanji zajednički sadržilac.)
Ovde umesto 10 imamo 5, pa će nam rešenje biti:
NZS>40 mora da bude
NZS(5,16)=80
80>40 =>
2^x=2^4=16
Najmanje memorijskih banaka potrebno da bi se elementima vektora pristupilo bez konflikta je 16.

22. Kojim sprežnim funkcijama su definisane sledeće dinamičke sprežne mreže

- **mešanje-zamena (shuffle-exchange)**
- **kub**
- **rešetka (mesh) x3+1+1**

Dinamičke SM se definišu skupom sprežnih funkcija. Sprežna funkcija Sf preslikava adresu X jednog PE u adresu Sf(X) drugog PE sa kojim se ostvaruje povezivanje. Kada se sprežna funkcija primeni na adresu jedne komponente sistema, dobija se adresa druge komponente sistema koja može direktno primiti podatak. Sprežne f-je su deo SIMD programa.

Mešanje-zamena (S-E):

- F-ja mešanja odgovara savršenom mešanju špila karata.

$$S(b_{m-1},b_{m-2},\ldots,b_1,b_0) = b_{m-2},\ldots,b_1,b_0,b_{m-1}$$

- F-ja zamene:

$$E(b_{m-1},b_{m-2},\ldots,b_1,b_0) = b_{m-1},b_{m-2},\ldots,b_1,\bar{b}_0$$

gde su b m-1 b m-2 ... b 1 b 0 – adrese PE-ova u binarnom obliku.

Kub:
Definisana sa $m=\log_2 N$ sprežnih f-ja

$$C_i(b_{m-1},\ldots,b_i,\ldots.b_1,b_0) = b_{m-1},\ldots,\bar{b}_i,\ldots.b_1,b_0,$$
$$i = 0,1,\ldots,m-1$$

Rešetka (mash): (dijametar O(√n), r = √N, N – broj PE)
Definisana sa 4 sprežne f-je:

- $M_{+1}(x)=(x+1) \bmod N$
- $M_{-1}(x)=(x-1) \bmod N$
- $M_{+r}(x)=(x+r) \bmod N$
- $M_{-r}(x)=(x-r) \bmod N$, r = √N

23. Kako se određuje zaglavlje poruke koje definiše put kroz višestepenu sprežnu mrežu generalizovani kub u slučaju da se vrši emisija?

Kod generalizovanog kuba put poruka se određuje:

- jedan na jedan:
  - određivanje puta pomoću EX-OR zaglavlja
  - određivanje puta pomoću odredišne adrese
- jedan izvor u 2^k odredišta (EMISIJA):
  - varijanta EX-OR

Odredjivanje puta pomoću EX-OR zaglavlja

- Poruci se dodaje zaglavlje dužine m bitova (2^m PE-ova).
  - zaglavlje se dobija primenom EX-OR operacija na binarnim adresama izvora i odredišta
  - adresa izvora $S=s_{m-1}s_{m-2}...s_1s_0$
  - adresa odredišta $D=d_{m-1}d_{m-2}...d_1d_2$
  - zaglavlje $T=S\oplus D=t_{m-1}t_{m-2}...t_1t_0$
  - da bi se odredio položaj KE (direktno ili ukršteno) u stepenu i, potrebno je ispitati t_i

$$t_i = \begin{cases} 0, \ diredktno \\ 1, \ ukrsteno \end{cases}$$

Određivanje puta za slučaj emisije:
Koristi se proširena šema sa EX-OR zaglavljem.

- Da bi emisija bila moguća *broj odredišta mora biti stepen dvojke (2 na i).*
- *Odredišne adrese se moraju razlikovati na j pozicija, a preklapati na m-j pozicija.*
- Da bi se odredilo zaglavlje za slučaj emisije potrebno je poznavati informaciju o putu pre i posle grananja, i tačke grananja.
- Zaglavlje za slučaj emisije određeno je skupom {R, B}:
  R = rm-1…r1r0 sadrži informaciju o putu,
  B = bm-1...b1b0 sadrži informacije o tačkama grananja.
- *PRIMER*
  - S adresa izvora, E i F adrese odredišta
  - Individualna zaglavlja za destinacije E i F, $T_E=S\oplus E$, $T_F=S\oplus F$
  - S=101 (5), E=100 (4), F=110 (6), T_E=001, T_F=011
  - Zaglavlja se slažu u svim bit pozicijama izuzev na mestu gde se razlikuju odredišne adrese
- I T_E i T_F sadrže informaciju o putu, pa se za R može odabrati bilo koji, npr. R= T_E.
- Da bi se odredilo $B=b_{m-1}...b_1b_0$ potrebno je naći $B=T_E\oplus T_F=E\oplus F$
  - Za S=101 (5), E=100 (4), F=110 (6), {R=001, B=E ⊕F=010}
- Da bi se odredilo u koiji položaj se postavlja KE u stepenu i potrebno je ispitati r_i i b_i.
  - ***ako je b_i=1, r_i se ignoriše i obavlja se emisija (grnja ili donja)***
  - ***b_i=0, r_i se koristi da se odredi položaj KE***

24. KOL2-2015 U statičkoj hiperkub mreži potrebno je poslati poruku iz čvora (0,0,0,0,0,0) u čvor (1,0,1,0,1,1). Prikazati preko kojih čvorova će poruka biti preneta do odredišta. x1+1

s(i)=(0,0,0,0,0,0) izvor
d(i)=(1,0,1,0,1,1) odredište

Ide preko čvorova:
(0,0,0,0,0,0) -> (1,0,0,0,0,0) -> (1,0,1,0,0,0) -> (1,0,1,0,1,0) -> (1,0,1,0,1,1)

25. U <u>hiperkub</u> sprežnoj mreži sa 16 procesora potrebno je poslati poruku iz čvora sa adresom 1001 u čvor 0110. Preko kojih čvorova će poruka biti prosležena do odredišnog čvora? x2

s(i)=(1,0,0,1) izvor
d(i)=(0,1,1,0) odredište

Ide preko čvorova:
(0,0,0,1) -> (0,1,0,1) -> (0,1,1,1) -> (0,1,1,0)

Rutiranje u hiperkubu:
Algoritam:

- *Neka je s(i) adresa izvora, d(i) adresa odredišta i σ(i) adresa čvora koji je na putu od s(i) do d(i)*
- *Uporediti bitove adresa d(i) sa σ(i) s leva u desno*
- *Identifikovati prvu bit poziciju na kojoj se ove dve adrese razlikuju*
- *Proslediti paket susedu n(i) tako da se adrese n(i) i σ(i) razlikuju samo na uočenoj bitskoj poziciji.*
    - *Npr. s(i)=(1,0,1,0), d(i)=(0,1,0,1) ide preko čvorova*
    - *(1,0,1,0) –> (0,0,1,0) –> (0,1,1,0) -> (0,1,0,0) -> (0,1,0,1)*

26. U <u>dinamičkoj hiperkub</u> mreži potrebno je poslati poruku iz čvora sa adresom 18 u čvorove sa adresama 5,7,13 i 15.

- **Da li je emisija moguća? Obrazložiti odgovor.**
- **Za slučaj da je emisija moguća kako izgleda zaglavlje koje se dodaje poruci?**

Ovde se misli na dinamičku višestepenu kub mrežu – GENERALIZOVANI KUB.

Generalizovani kub se sastoji od od $\log_2 N=m$ stepena i N/2 komutacionih elemenata.

Da bi emisija bila moguća broj odredišta mora biti stepen dvojke 2^j
4 odredista => j=2
Odredišne adrese se moraju razlikovati na j pozicija, a poklapati na m-j pozicija.
$\log_2 N=m$ => $\log_2 32=5$ (N - broj procesora, m - broj stepena)
m=5 => m-j=5-2=3
Odredišne adrese se moraju razlikovati na 2 pozicije, a poklapati na 3 pozicije.

D5=5 = 00101
D7=7 = 00111
D13=13 = 01101
D15=15 = 01111
S=18 = 10010

D5 =5 =00101, T5 =S ⊕D5 =10111
D7 =7 =00111, T7 =S ⊕D7 =10101
D13=13=01101, T13=S ⊕D13 =11111
D15=15=01111, T15=S ⊕D15 =11101

S obirom da je broj odredišta stepen dvojke i da se adrese odredišta poklapaju na 3 pozicije, a razlikuju na 2 – emisija je moguća.

Za zaglavlje poruke u slučaju emisije se koristi proširena šema sa EX-Or zaglavljem.
Zaglavlje za slučaj emisije odredjeno je skupom {R,B}.
R=rm-1...r1r0 - sadrži informaciju o puta,
B=bm-1...b1b0 - informacije o tačkama grananja.
Za R se odabira bilo koji od puteva.
Da bi se odredilo $B=b_{m-1}...b_1b_0$ potrebno je naći $B=T_E \oplus T_F=E \oplus F$ ($T_E=S \oplus E$). DAKLE nađemo EXOR prvog i poslednjeg kome šaljemo.

R=S⊕D5=10111 na primer
B=T5⊕T15=10111⊕11101=01010 ILI
B=D5⊕D15=00101⊕01111=01010
{R,B}={10111, 01010}

27. (KOL2-2015) U <u>dinamičkoj hiperkub</u> mreži potrebno je poslati poruku iz čvora sa adresom 18 u čvorove sa adresama 9,13,15 i 27.

- **Da li je emisija moguća? Obrazložiti odgovor.**
- **Za slučaj da je emisija moguća kako izgleda zaglavlje koje se dodaje poruci?**

I ovde se misli na dinamičku višestepenu kub mrežu – GENERALIZOVANI KUB.

Što se teorije tiče – sve isto kao u prošlom zadatku. Emisija ovde nije moguća jer se adrese odredišta poklapaju na 2, a razlikuju na 3 pozicije.

D9=9 = 01001
D13=13= 01101
D15=15= 01111
D27=27= 11011
S=18 = 10010

D9=9 =01001, T9 =S ⊕D9 =11011
D13=13=01101, T13=S ⊕D13 =11111
D15=15=01111, T15=S ⊕D15 =11101
D27=27=11011, T27=S ⊕D27 =10001

R=01001
B=D9⊕D27=01110
{R,B}={01001, 01110} - za slučaj da jeste moguća emisija.

28. (2013. god) U sistemu postoji 16 procesora. Za povezivanje se koristi višestepena sprežna mreža <u>generalizovani kub</u>. Procesor P1 treba da pošalje poruku procesorima P2, P3, P6, P7, P10, P11, P14 i P15

- **Da li je emisija moguća? Zašto?**
- **Kako izgleda zaglavlje koje se dodaje poruci?**

Da bi emisija bila moguća broj odredišta mora biti stepen dvojke 2^j
8 odredista => j=3
Odredišne adrese se moraju razlikovati na j pozicija, a poklapati na m-j pozicija.
$\log_2 N=m$ => $\log_2 16=4$ (N - broj procesora, m - broj stepena)
m=4 => m-j=4-3=1
Odredišne adrese se moraju razlikovati na 3 pozicije, a poklapati na 1 poziciji.

D2=2 =0010
D3=3 =0011
D6=6 =0110
D7=7 =0111
D10=10=1010
D11=11=1011
D14=14=1110
D15=15=1111
S=1 =0001

D2=2 =0010, T2=S ⊕D2 =0011
D3=3 =0011, T3=S ⊕D3 =0010
D6=6 =0110, T6=S ⊕D6 =0111
D7=7 =0111, T7=S ⊕D7 =0110
D10=10=1010, T10=S ⊕D10 =1011
D11=11=1011, T11=S ⊕D11 =1010
D14=14=1110, T14=S ⊕D14 =1111
D15=15=1111, T15=S ⊕D15 =1110

S obirom da je broj odredišta stepen dvojke i da se odredišne adrese razlikuju na 3 pozicije a poklapaju na jednoj emisija je moguća.

Za zaglavlje poruke u slučaju emisije se koristi proširena šema sa EX-Or zaglavljem.
Zaglavlje za slučaj emisije odredjeno je skupom {R,B}.
R=rm-1...r1r0 - sadrži informaciju o puta,
B=bm-1...b1b0 - informacije o tačkama grananja.
Za R se odabira bilo koji od puteva.
Da bi se odredilo $B=b_{m-1}...b_1b_0$ potrebno je naći $B=T_E \oplus T_F=E \oplus F$ ($T_E=S \oplus E$). DAKLE nađemo EXOR prvog i poslednjeg kome šaljemo.

R=S⊕D2=0011 na primer
B=T2 ⊕T15=0011⊕1110=1101 ILI
B=D2⊕D15=0010⊕1111=1101
{R,B}={0011, 1101}

29. 32 PE povezana su <u>hiperkub višestepenom sprežnom mrežom</u>.
Koliko stepena ima mreža i koliko komutacionih elemenata?
Iz PE-a sa adresom 21 potrebno je poslati poruku PE-ovima sa adresama 9, 13, 25 i 29. Da li je moguće obaviti emisiju? Ako jeste kako izgleda zaglavlje koje se dodaje poruci?

Generalizovani kub se sastoji od od $\log_2 N=m$ stepena i N/2 komutacionih elemenata.
N=32, $\log_2 N=m$, 2^m=N, N=5 => 5 stepena i 16 komutacionih elemenata.

Teorija ista kao gore. S obzirom da se adrese odredišta poklapaju na 3 pozicije, a razlikuju na 2, emisija je moguća.

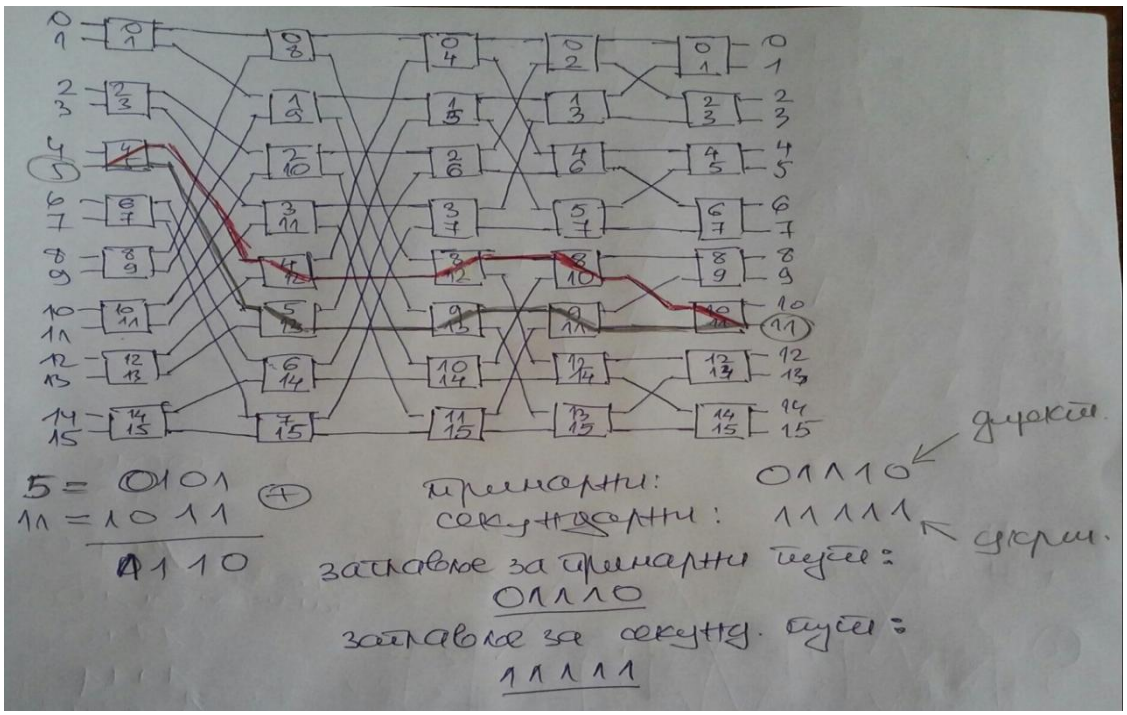
D9=9 = 01001
D13=13= 01101
D15=25= 11001
D27=29= 11101
S=18 = 10010

D9=9 =01001, T9 =S ⊕D9 =11011
D13=13=01101, T13=S ⊕D13 =11111
D25=25=11001, T15=S ⊕D15 =01011
D29=29=11101, T27=S ⊕D27 =01111

R=01001
B=D9⊕D29=10100
{R,B}={01001, 10100}

30. U <u>ESC</u> (Extra Stage Cube) višestepenoj sprežnoj mreži koja povezuje 16 PE-ova potrebno je poslati poruku iz čvora sa adresom 5 u čvor sa adresom 11. Kako izgleda zaglavlje poruke za primarni i sekundarni put? x3

- *Zaglavlje za primarni put* $T_p = 0t_{m-1}...t_1t_0$
- *Zaglavlje za sekundarni put* $T_s = 1t_{m-1}...t_1\bar{t}_0$



Extra Stage Cube rešava problem nastupanja greške na nekom komunikacionom kanalu dodavanjem ekstra stepena na ulaz mreže. Dodaju se i MUX i DMUX na ulaze ekstra stepena (m) i poslednjeg stepena (0) (zaboravila sam da ih dodam na slici). Dodavanjem ekstra stepena dobijaju se dva puta između svakog para izvor-odredište.
Zaglavlje poruke za primarni i sekundarni put dobijaju se primenom EX-OR operacija nad binarnim adresama izvor i odredišta, s tim što se za primarni put dodaje <u>nula na početak</u>, a za sekundarni se <u>dodaje jedinica i poslednja cifra se komplementira</u>.

31. Objasniti arbitražu na magistrali koja se zasniva na deljivim zahtevima i lančanom zahvatanju. x2+1+1

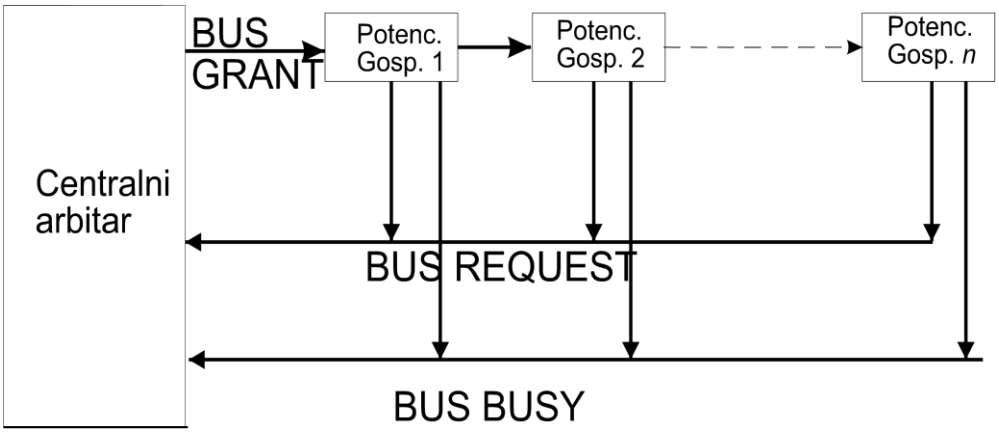
Arbitracioni mehanizmi omogućavaju da se dodela magistrale obavi bez konflikata. Postoje statička i dinamička arbitraža.

Harver za arbitražu kod dinamičke arbitraže može biti realizovan kao:
- centralizovan
- distribuiran

Kod centralizovane arbitraže hardver je koncentrisan na jednom mestu.

Hardverski mehanizmi koji se koriste za dodelu i zahvatanje magistrale kod centralizovane arbitraže mogu se realizovati kao:
- deljivi zahtevi i lančano zahvatanje
- nezavisni zahtevi i zahvatanja

Kod arbitraže na magistrali koja se zasniva na deljivim zahtevima i lančanom zahvatanju svaki potencijalni gospodar magistrale izdaje zahtev za dodelu magistrale preko linije BUS REQUEST. Svi zahtevi su povezani žičanom ILI logikom.



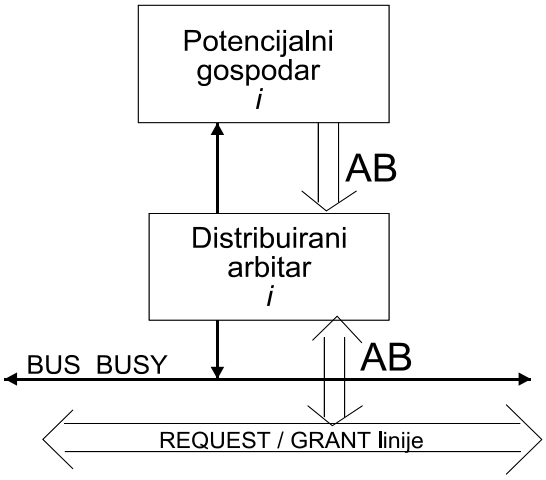
Kada CENTRALNI ARBITAR primi zahtev, on predaje signal BUS GRANT potencijalnom gospodaru magistrale označenim sa 1.
- Linija BUS GRANT povezuje sve potencijalne gospodare u lanac (daisy chain), tako da gospodar1 predaje signal gospodaru2, itd...
- Gospodar koji je izdao zahtev ne prenosi signal dalje (prekida lanac) i aktivira liniju BUS BUSY čime ukazuje da je zahvatio magistralu.
- Kada tekući gospodar završi sa prenosom oslobađa liniju BUS BUSY i naredni ciklus arbitraže može da počne.

Ovde važi politika zasnovana na fiksnim prioritetima (politike dodele magistrale mogu biti po prioritetu, nepristrasnosti i kombinovana). Gospodar koji je bliži centralnom arbitru ima viši prioritet.

Prednost ove šeme je jednostavnost izvodjenja i mali broj linija zahtevanih za arbitržu.

Nedostatak je što je priorite odredjen fizičkom pozicijom:
- može se desiti da gospodar sa nižim prioritetom ne dobiju magistralu ako se često izdaju zahtevi od strane gospodara sa višim prioritetom

32. Objasniti distribuiranu arbitražu na magistrali kod multiprocesorskih sistema? x1+1

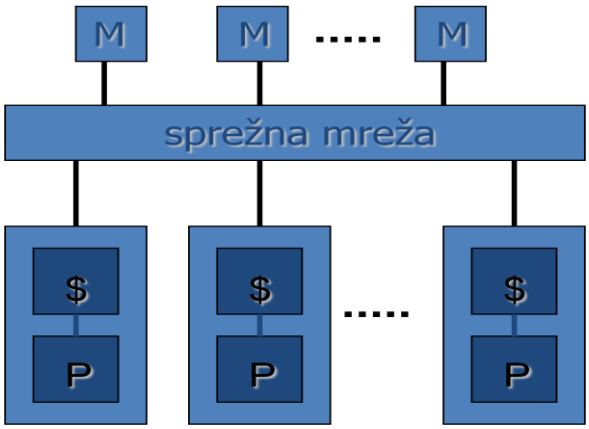


Kod distribuirane arbitraže na magistrali kod multiprocesorskih sistema hardver za arbitražu je raspodeljen po potencijalnim gospodarima magistrale.

- Svaki potencijalni gospodar ima sopstveni arbitar i jedinstveni arbitracioni broj koji se koristi da razreši sudare kod jednovremenih zahteva za korišćenje magistrale.
- Kada dva ili više uređaja konkurišre za korišćenje magistrale, pobednik je onaj koji ima viši arbitracioni broj.
- Svaki potencijalni gospodar može poslati svoj arbitracioni broj (AB) na deljive request/grant linije preko svog arbitra.
- Od svih zahteva se formira zbirni arbitracioni broj #AB.
- Nakon ovoga, svaki arbitar poredi svoj AB sa zbirnim, počev od bita najveće težine
  - Ako je njegov broj niži od zbirnog to znači da je njegov prioritet niži i zahteva se uklanjanje.
  - Na kraju na linijama ostaje arbitracioni broj onog gospodara koji ima najviši prioritet i njemu se dodeljuje magistrala.

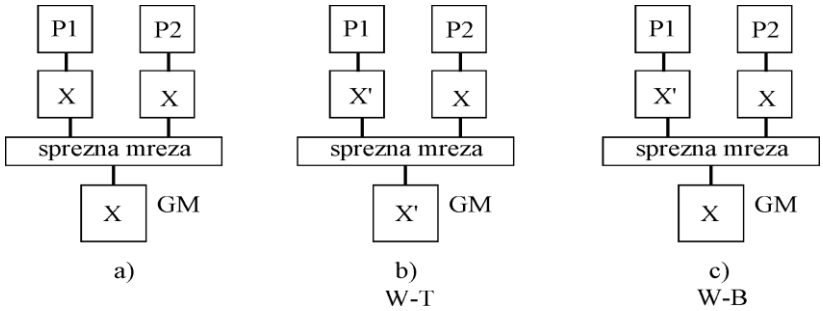
33. Koje aktivnosti mogu dovesti do nekonzistentnosti memorijskog sistema kada se koriste privatne keš memorije?

Privatne keš memorije koje se pridružuju svakom procesoru koriste se da bi se kod multiprocesorskih sistema otklonili problemi (sudari kod pristupa zajedničkoj memoriji, konflikti pri komunikaciji, latentnost pristupa kroz sprežnu mrežu).

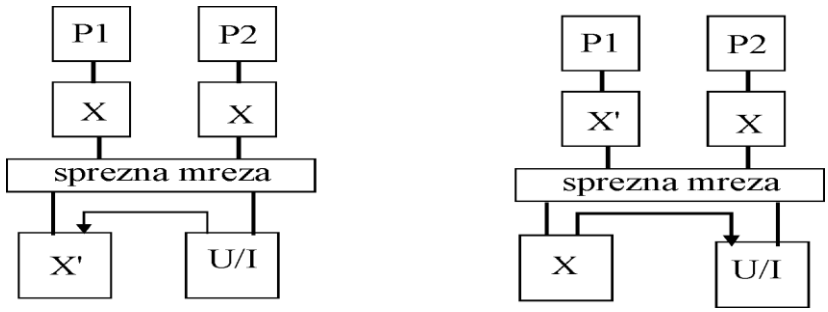


Do nekonzistentnosti memorijskog sistema kada se koriste privatne keš memorije mogu dovesti sledeće aktivnosti:

- Može se desiti da u jednom trenutku više keš memorija poseduje kopiju istog podatka iz glavne memorije. Bilo koja lokalna modikacija kopije podatka u kešu dovešće do nekonzistentnosti (neusaglašenosti) memorijskog sistema.
  Primer: Neka multiprocesorski sistem sa dva procesora, svaki sa privatnim kešom, koji imaju kopiju istog podatka X
  - *Ako jedan od procesora, recimo P1, modikuje sadržaj lokacije X na X' u svom kešu, ista vrednost biće automatski upisana u deljivu memoriju ako se koristi write through (W-T) politika za ažuriranje glavne memorije.*
    - *Kod ove politike svaki upis u keš automatski modifikuje i sadržaj glavne memorije.*
    - *U ovom slučaju nekonzistentnost postoji izmedju dve kopije keša.*
  - *Ako se koristi write-back (W-B) politika kod ažuriranja glavne memorije, onda će i sadrzaj glavne memorije biti nekonzistentan.*
    - *Kod W-B politike glavna memorija se ažurira tek kod zamene bloka*



- Do nekonzistentnosti može doći i zbog U/I aktivnosti i migracije procesa u sistemima sa privatnim keš memorijama.
  - Kod većine multiprocesorskih sistema organizovanih oko zajedničke magistrale, U/I procesor je takodje spregnut na magistralu.
  - U slučaju W-T politike kod ažuriranja glavne memorije do nekonzistentnoski može doći kada U/I procesor puni glavnu memoriju
  - U slucaju W-B politike kod ažuriranja glavne memorije do nekonzistentnoski može doći kada se podaci direktno šalju iz memorije u U/I processor



- Neki sistemi dozvoljavaju migraciju procesa, tj. da procesi budu rasporedjeni različitim procesorima u toku svog života da bi se izbalansiralo opterećenje izmedju procesora.
  - Ako se ova mogućnost koristi u sprezi sa privatnim keš memorijama takođe može doći do nekonzistentnosti podataka.
    - Npr. proces A koji se izvršava na procesoru P1 može promeniti podatke u svom kešu pre nego što bude suspendovan.
    - Ako kasnije proces A migrira na procesor P2 pre nego što glavna memorija bude ažurirana, proces A moze uzeti ustajalu vrednost iz memorije.
    - Evidentno je da se samo W-T politikom kod ažuriranja glavne memorije neće održati konzistentnost memorijskog sistema, jer se ovom tehnikom ne žuriraju kopije podataka u drugim keševima.

34. Navesti tehnike za postizanje keš koherencije. x1+1(2013)

Postoji više prilaza za rešavanje ovog problema.

- Hardverski implementirani protokoli za postizanje keš koherencije:
  Hardverskim tehnikama detektuju se uslovi koji mogu dovesti do nekonzistentnosti keš memorija i shodno tome obavljaju akcije koje održavju koherentnost memorijskog sistema. U ovu grupu protokola spadaju:
  - Snoopy keš protokoli (Njuškala)
    - Write-once
    - Firefly
  - Direktorijumske šeme:
    - Protokoli sa potpuno preslikanim adresarima (full-map directories)
    - Protokoli sa ograničenim adresarima (limited directories)
    - Protokoli sa ulančanim adresarima (chained directories)

- Softverske tehnike:
  Nekonzistentnost memorijskog sistema se može sprečiti tako sto se neće "keširati" deljivi podaci koji se mogu menjati upisom. Instrukcije i drugi podaci se mogu kopirati u keš.
  To znači da se podaci na neki nacin moraju označiti.
  - To moze uciniti korisnik korišćenjem viših programskih jezika kao što su Ada, Modula 2, Concurrent Pascal, itd., deklarišući podatke kao deljive (shared) ili nedeljive (local).
  - Alternativno, multiprocesorski kompajler može automatski klasikovati podatke kao deljive ili ne.

  Nedostatak ove tehnike je netransparentnost multiprocesorske arhitekture za korisnika ili kompajler.
  Efikasnost ovog pristupa zavisi od mogucnosti jezika da specificira podatke kao deljive ili ne, ili od kompajlera da detektuje takve podatke.
  Pošto u prakticnim implementacijama čitava stranica mora biti deklarisana kao "cachable" ili ne, može doći do interne fragmentacije memorije, ili će se zabraniti kešovanje i onih podataka koji se mogu kopirati u keš.

  Snoopy keš protokoli:
  - Mnogi današnji komercijalno raspoloživi multiprocesori koriste memorijske sisteme organizovane oko zajedničke magistrale.
  - Magistrala je pogodna za odrzavanje keš koherencije jer, kao jedinstveni sprežni put, dozvoljava da svi procesori u sistemu nadgledaju memorijske transakcije.
  - Ako transakcija na magistrali ugrožava konzistentnost podataka u lokalnom kešu, specijalni hardver (keš kontroler) može preduzeti akcije da poništi lokalnu kopiju (tj. proglasi je nevažećom).
  - Protokoli koji koriste ovaj mehanizam za postizanje koherentnosti zovu se snoopy (njuškala) protokoli jer svaki keš kontroler "njuška" transakcije drugih keševa.

  Direktorijumske šeme:
  - Kada se koriste višestepene sprežne mreže za gradnju multiprocesorskih sistema sa velikim brojem procesora, snoopy protokoli se moraju modikovati da bi se prilagodili mogućnostima mreže.
  - Pošto je broadcast (emisiju) veoma skupo obaviti kod višestepenih mreža, komande konzistentnosti se šalju samo onim keševima koji imaju kopiju bloka.
  - Da bi se to učinilo potrebne su tačne informacije o tome koje keš memorije imaju kopiju odredjenog bloka.
  - Ove informacije se smeštaju u direktorijumima (adresarima).
    - Protokoli koji koriste informacije zapamćene u direktorijumima za održavanje koherentnosti memorijskog sistema zovu se direktorijumske šeme.

35. Objasniti direktorijumske šeme sa potpuno preslikanim adresarima.

Kada se koriste višestepene sprežne mreže za gradnju multiprocesorskih sistema sa velikim brojem procesora, snoopy protokoli se moraju modikovati da bi se prilagodili mogućnostima mreže.
posto je broadcast (emisiju) veoma skupo obaviti kod višestepenih mreža, komande konzistentnosti se šalju samo onim keševima koji imaju kopiju bloka.
Da bi se to učinilo potrebne su tačne informacije o tome koje keš memorije imaju kopiju odredjenog bloka.
Ove informacije se smeštaju u direktorijumima (adresarima).
protokoli koji koriste informacije zapamćene u direktorijumima za održavanje koherentnosti memorijskog sistema zovu se direktorijumske šeme.
Svaki memorijski modul sadrži poseban direktorijum koji beleži stanje i prisustvo memorijskog bloka u odredjenom kešu.
Svaki direktorijumski ulaz (linija) sadrži

- *veći broj pokazivača koji ukazuju na keš koji sadrži kopiju bloka i*
- *jedan "dirty" bit kojim se ukazuje kada odredjeni keš ima, ili ne, dozvolu da modikuje podatak..*

Direktorijumske šeme su hardverski implementirani protokoli za postizanje keš koherencije.
Postoje tri tipa direktorijumskih protokola:

- protokoli sa potpuno preslikanim adresarima (full-map directories)
- protokoli sa ograničenim adresarima (limited directories)
- protokoli sa ulančanim adresarima (chained directories)

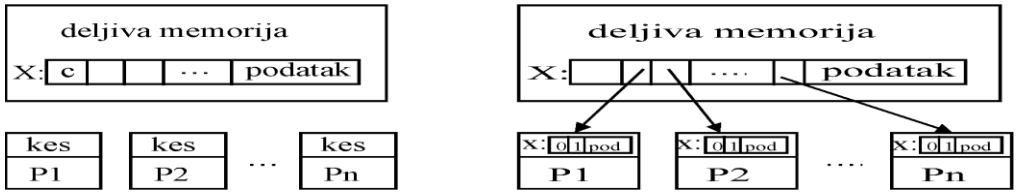
Kod protokola sa potpuno preslikanim adresarima svaki direktorijumski ulaz u glavnoj memoriji sadrži po jedan bit za svaki procesor u sistemu i jedan "dirty" bit.

- Svaki bit pridružen procesoru ukazuje na status bloka u odgovarajućem procesorskom kešu (prisutan ili ne).
- Ako je "dirty" bit postavljen, tada je jedan i samo jedan procesorski bit postavljen i taj procesor ima pravo da vrši upis u kešovani blok (tj. da ga modifikuje).

Svaki keš direktorijum sadrži dva bita stanja za svaki kešovani blok.

- Jedan bit ukazuje da li je blok važeći.
- Drugi bit ukazuje da li se u važeći blok moze vršiti upis.

Keš koherentni protokol mora držati bitove stanja u memorijskom direktorijumu i u kešu konzistentnim.



Slika a) ni jedan keš ne sadrži kopiju lokacije X.
Slika b) procesori P1, P2 i Pn zahtevali kopiju podatka X.

- *Tri bita u odgovarajućem adresarskom ulazu u glavnoj memoriji koji ukazuju na prisustvo bloka u odredjenom kešu se postavljaju.*
- *U keš adresaru se postavljaju odgovarajući bitovi stanja da ukažu na važecu kopiju.*
- *U adresaru u glavnoj memoriji "dirty" bit je obrisan (c), i ukazuje da ni jedan procesor nema pravo da modikuje kopiju podataka u svom kešu*

Šta se dešava ako jedan procesor, recimo Pn, izda komandu za upis u svoj keš (Cn):

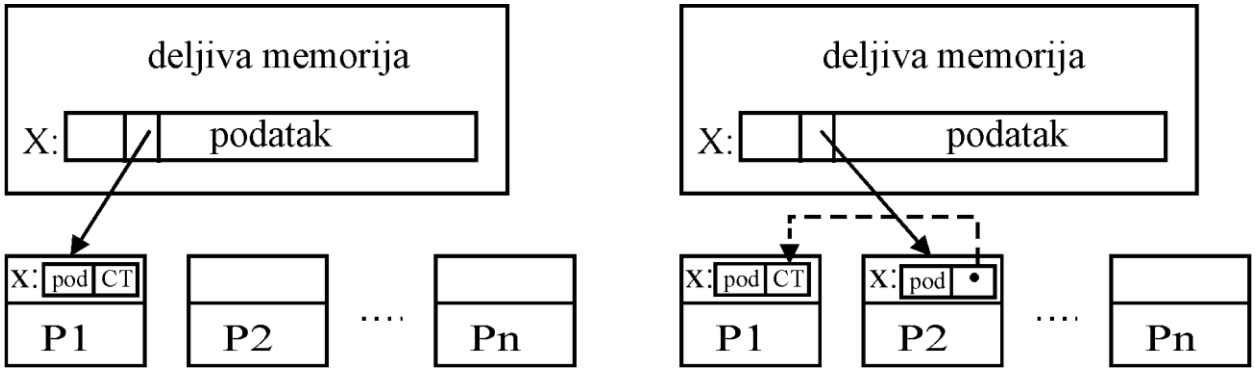
- *Keš Cn detektuje da je blok koji sarži lokaciju X važeci, ali da procesor nema pravo upisa.*
- *Keš kontroler Cn upućuje zahtev za modikacijom odgovarajućem memorijskom modulu koji sadrži lokaciju X i zaustavlja svoj procesor, Pn.*
- *Memorijski modul izdaje zahtev za invalidacijom keševima C1 i C2.*
- *Keševi C1 i C2 primaju invalidacione zahteve, postavljaju odgovarajuce bitove koji ukazuju da je blok koji sadrži lokaciu X nevažeci i vraćaju potvrdu o priznavanju zahteva nazad memorijskom modulu.*
- *Memorijski modul prima potvrdu o invalidaciji, postavlja "dirty" bit, briše pokazivače na keševe C1 i C2 i predaje dozvolu za upis kešu Cn.*
- *Keš Cn prima poruku o dozvoli upisa, ažurira stanje u kešu i aktivira procesor Pn.*

Ovakvim redosledom akcija protokol garantuje da će memorijski sistem ostati konzistentan.

36. Objasniti kako izgleda formiranje direktorijumskog lanca ako se koristi direktorijumska šema sa ulančanim adresarima. x2+2

Dozvoljavaju proširljivost sistema bez ograničavanja broja kopija jednog memorijskog bloka. Ovaj tip adresarske šeme zove se "ulančani adresari" jer čuva trag o kopijama bloka preko lanca adresarskih pokazivaca.
Formiranje adresarskog lanca :

- Pretpostavimo da u sistemu ne postoji kopija bloka sa lokacijom X.
- Ako P1 čita lokaciju X, glavna memorija šalje kopiju bloka kešu procesora P1 i ujedno pokazivač na kraj lanca CT (Chain Termination).
- U u adrsaru glavne memorije čuva se pokazivač na procesor P1
- Ako P2 zahteva čitanje lokacije X, glavna memorija šalje kopiju bloka kešu procesora P2 i pokazivac na P1.
- Sada se u adrsaru glavne memorije čuva se pokazivač na procesor P2
- Ponavljanjem ovih koraka svi keševi mogu dobiti kopiju bloka.



Neka jedan od procesora, recimo P3, zahteva da modikuje sadržaj lokacije X.

- Da bi se održala konzistentnost memorijskog sistema potrebno je da se komanda invalidacije prosledi kroz lanac.
- Kontroler glavne memorije ne daje dozvolu modikacije procesoru P3 sve dok ne primi potvrdu o invalidaciji od procesora koji sadrži CT pointer.
- S obzirom da se informacija o invalidaciji prenosi od jednog do drugog kesa u lancu, protokol se zove i "gossip" (tračarenje) protokol.

37. Ako nastupi promašaj pri upisu kod obraćanja lokalnom kešu koje se akcije preduzimaju i koje je rezultujuće stanje lokalnih i udaljenih kopija ako se koristi:

- **Write-once protokol**
- **Firefly protokol x3+1**

U slučaju da nastupi promašaj pri upisu kod obraćanja lokanom kešu:
Write-once protokol:

- Prvo se pribavlja kopija iz glavne memorije ili iz keša koji ima dirty kopiju.
  - To se postiže slanjem Read_Inv komande koja će pribaviti i ažurirati blok i invalidirati sve ostale keš memorije.
- Lokalna memorija se posle ovoga nalazi u Dirty stanju.

Firefly protokol:
Kopija dolazi iz drugih keševa ili glavne memorije.

- Ako dolazi iz memorije stanje kopije nakon upisa je Dirty.
- U ostalim slučajevima sve kopije se ažuriraju, a novo stanje je Shared.

38. Koja stanja keš kopije postoje kod Write-once a koja kod Firefly protokola? Koje aktivnosti se dešavaju kod ovih protokola kada nastupi događaj promašaj pri upisu? x2

Write-once protokol:
Koristi W-T (write through) politiku kod ažuriranja glavne memorije pri prvoj modifikaciji keš bloka. Nakon prvog upisa keš memorija se ažurira korišćenjem W-B (write back) politike.
Protokol razlikuje četiri stanja kopije keš bloka:

- Valid (važeća) – U kešu je kopija koja je jednaka sa memorijskom kopijom i kopijama u drugim keševima (ako ih ima).
- Ivalid (nevažeća) – Kopija je poništena.
- Reserved (rezervisano) – Podatak u kešu je izmenjen samo jednom i keš kopija je jednaka sa kopijom u glavnoj memoriji, koja je jedina druga (sve ostale kopije su invalidirane ako su postojale).
- Dirty (prljava) – Keš memorija je izmenjena više puta i kopija je jedina u sistemu (u sistemu može postojati samo jedna Dirty kopija).

Firefly protokol:
Firefly protokol koristi write-back (W-B) politiku za ažuriranje privatnih blokova (blokovi koji su u stanju Valid-exclusive i Dirty), a W-T za deljive (shared) blokove.
Postoje tri moguća stanja keš kopije:

- Valid-exclusive – Postoji samo jedna kopija koja je jednaka memorijskoj kopiji.
- Shared – Postoji više kopija i sve su jednake.
- Dirty – Postoji samo jedna kopija i ona nije jednaka sa momorijskom kopijom.

U slučaju da nastupi promašaj pri upisu kod obraćanja lok. kešu: odg. na prethodno pitanje.

39. Prikazati sadržaje keš memorija i glavne memorije i stanje keš blokova nakon svake operacije navedene u tabeli za slučaj da se koristi write-once protokol.

korak	Događaj	Glavna mem.	P keš		Q keš	
		Lokacija X	Lokacija X	Stanje bloka	Lokacija X	Stanje bloka
0	početna vredn.	5	x	X	x	x
1	P čita X	5	5	Val	x	x
2	Q čita X	5	5	Val	5	Val
3	Q ažurira X X:=X+5	10	5	Inval	10	Rsrvd
4	Q čita X	10	5	inval	10	Val
5	Q ažurira X X:=X+5	10	5	Inv	15	Dirty
6	P ažurira X X:=X+5	15	10	Dirty	15	Inval
7	Q čita X	10	10	Val	10	Val

40. Multiprocesorski sistem se sastoji od tri procesora. Svaki procesor ima svoju keš memoriju. Prikazati kako se vrši promena stanja keš blokova i glavne memorije ako se za održavanje keš koherencije koristi:

- **Write-once protokol**
- **Firefly protocol**

rešeni zadaci ispod

Write-once protokol:
Ovaj protokol koristi W-T (write through) politiku kod ažuriranja glavne memorije pri prvoj modifikaciji keš bloka. Nakon prvog upisa keš memorija se ažurira korišćenjem W-B (write back) politike.
Protokol razlikuje četiri stanja kopije keš bloka:

- Valid (važeća) – U kešu je kopija koja je jednaka sa memorijskom kopijom i kopijama u drugim keševima (ako ih ima).
- Ivalid (nevažeća) – Kopija je poništena.
- Reserved (rezervisano) – Podatak u kešu je izmenjen samo jednom i keš kopija je jednaka sa kopijom u glavnoj memoriji, koja je jedina druga (sve ostale kopije su invalidirane ako su postojale).
- Dirty (prljava) – Keš memorija je izmenjena više puta i kopija je jedina u sistemu (u sistemu može postojati samo jedna Dirty kopija).

Firefly protokol:
Firefly protokol koristi write-back (W-B) politiku za ažuriranje privatnih blokova (blokovi koji su u stanju Valid-exclusive i Dirty), a W-T za deljive (shared) blokove.
Postoje tri moguća stanja keš kopije:

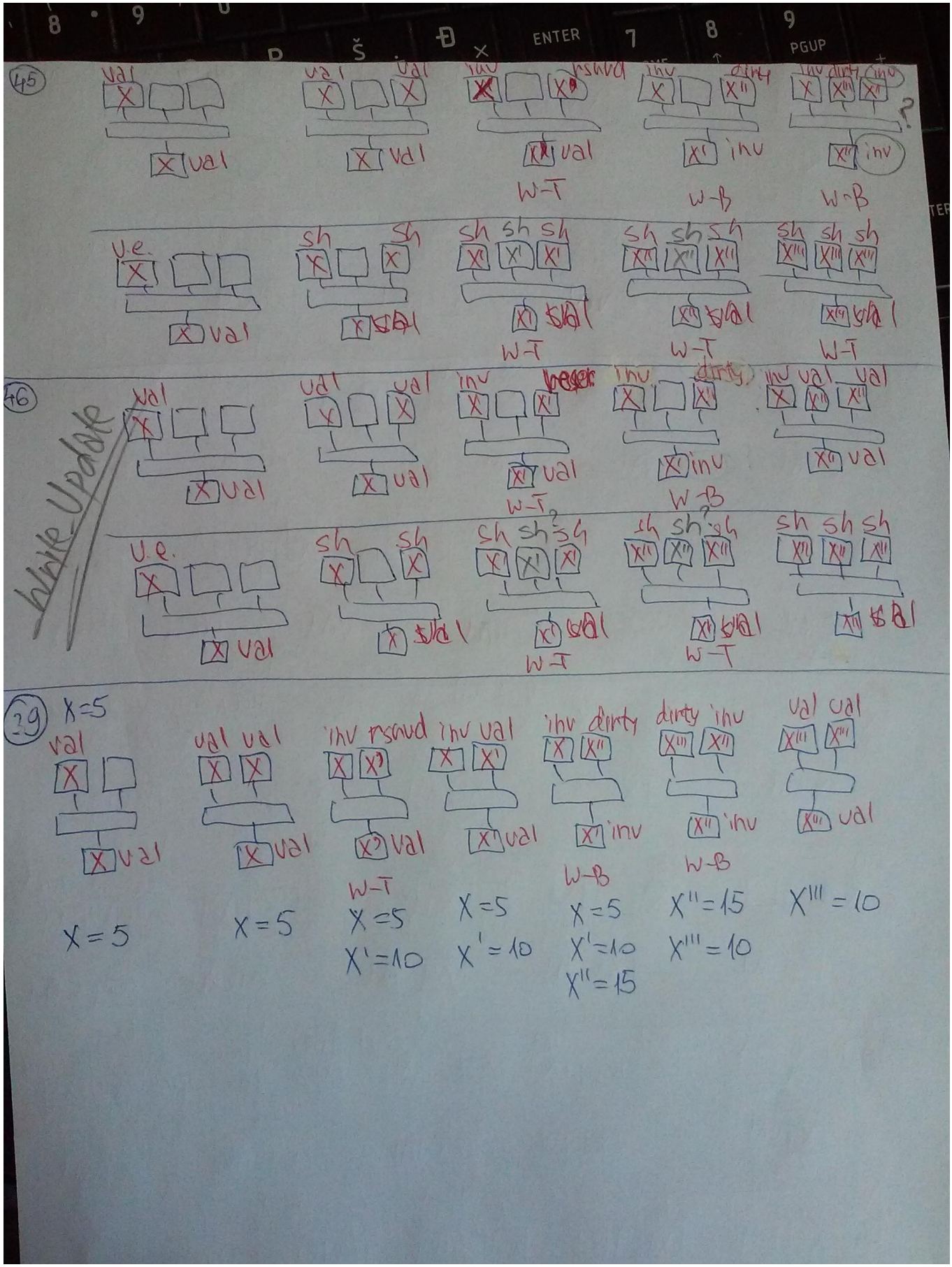
- Valid-exclusive – Postoji samo jedna kopija koja je jednaka memorijskoj kopiji.
- Shared – Postoji više kopija i sve su jednake.
- Dirty – Postoji samo jedna kopija i ona nije jednaka sa momorijskom kopijom.

Da bi se održala konzistentnost kod modifikacije deljivog (shared) bloka koristi se Write_Update komanda koja ažurira sve kopije u sistemu.

Write-through: write is done synchronously both to the cache and to the backing store.
Write-back (also called write-behind): initially, writing is done only to the cache. The write to the backing store is postponed until the cache blocks containing the data are about to be modified/replaced by new content.

Nisam napisala rešenja svih primere u tabelama, ali sam ih ovako rešila.


(41)
reserved ?
val
W-T
dirty
inv
W-B
sh
(42)
val
rsrvd
dirty
inv
W-T
W-B
v.e.
val.
(43)
reserved
in
dirty
val.e.
W-B
(44)


45
val
inv
dirty
W-T
W-B
sh
46
Write Update
inv
dirty
W-B
W-T
29
X=5
val
inv rsnud
inv dirty
dirty inv
W-T
W-B
X=5
X=10
X=15
X=10

41. Multiprocesorski sistem se sastoji od tri procesora. Svaki procesor ima svoju keš memoriju. Prikazati kako se vrši promena stanja keš blokova i glavne memorije ako se za održavanje keš koherencije koristi: x4+2

- **Write-once protokol**
- **Firefly protokol**

Inicijalno su keš memorije prazne. Procesori pristupaju deljivoj promenjivoj X. Akcije procesora navedene su u prvoj koloni tabele. Popuniti ostale ćelije tabele.

Akcija procesora	Stanje keš kopije u P1		Stanje keš kopije u P2		Stanje keš kopije u P3		Podatak se pribavlja iz GM ili keša		Stanje kopije u glavnoj memoriji	
	a)	b)	a)	b)	a)	b)	a)	b)	a)	b)
P2 upisuje u X										
P3 čita X										
P1 čita X										
P3 upisuje u X										
P2 upisuje u X										

42. Multiprocesorski sistem se sastoji od tri procesora. Svaki procesor ima svoju keš memoriju. Prikazati kako se vrši promena stanja keš blokova i glavne memorije ako se za održavanje keš koherencije koristi:

- **Write-once protokol**
- **Firefly protokol**

Inicijalno su keš memorije prazne. Procesori pristupaju deljivoj promenjivoj X. Akcije procesora navedene su u prvoj koloni tabele. Popuniti ostale ćelije tabele.

Akcija procesora	Stanje keš kopije u P1		Stanje keš kopije u P2		Stanje keš kopije u P3		Podatak se pribavlja iz GM ili keša		Stanje kopije u glavnoj memoriji	
	a)	b)	a)	b)	a)	b)	a)	b)	a)	b)
P3 čita X										
P3 čita X										
P1 čita X										
P3 upisuje u X										
P1 upisuje u X										
P2 čita X										

43. Multiprocesorski sistem se sastoji od tri procesora. Svaki procesor ima svoju keš memoriju. Prikazati kako se vrši promena stanja keš blokova i glavne memorije ako se za održavanje keš koherencije koristi:

- **Write-once protokol**
- **Firefly protokol**

Inicijalno su keš memorije prazne. Procesori pristupaju deljivoj promenjivoj X. Akcije procesora navedene su u prvoj koloni tabele. Popuniti ostale ćelije tabele.

Akcija procesora	Stanje keš kopije u P1		Stanje keš kopije u P2		Stanje keš kopije u P3		Podatak se pribavlja iz GM ili keša		Stanje kopije u glavnoj memoriji	
	a)	b)	a)	b)	a)	b)	a)	b)	a)	b)
P1 čita X										
P3 čita X										
P1 upisuje u X										
P3 upisuje u X										
P2 upisuje u X										

44. Multiprocesorski sistem se sastoji od tri procesora. Svaki procesor ima svoju keš memoriju. Prikazati kako se vrši promena stanja keš blokova i glavne memorije ako se za održavanje keš koherencije koristi:

- **Write-once protokol**
- **Firefly protokol**

Inicijalno su keš memorije prazne. Procesori pristupaju deljivoj promenjivoj X. Akcije procesora navedene su u prvoj koloni tabele. Popuniti ostale ćelije tabele.

Akcija procesora	Stanje keš kopije u P1		Stanje keš kopije u P2		Stanje keš kopije u P3		Podatak se pribavlja iz GM ili keša		Stanje kopije u glavnoj memoriji	
	a)	b)	a)	b)	a)	b)	a)	b)	a)	b)
P1 čita X	Val	v.e.	X	X	Xs	X	GM	GM	Val	Val
P3 čita X	Val	Sh	X	X	Val	Sh	GM	Gm	Val	Val
P3 upisuje u X	Val	Sh	X	Sh	Rsrvd	Sh	Keš	Keš	Val	Val
P1 upisuje u X	Dirty	Sh	X	Sh	Inval	Sh	Keš	Keš	Inval	Val
P2 čita X	Val	Sh	Val	Sh	Inval	Sh	Gm	GM	Val	Val

45. Multiprocesorski sistem se satoji od tri procesora. Svaki procesor ima svoju keš memoriju. Prikazati kako se vrši promena stanja keš blokova i glavne memorije ako se za održavanje keš koherencije koristi:
a) Write-once protokol
b) Firefly protokol
Procesori pristupaju deljivoj promenljivoj X. Akcije procesora navedene su u prvoj koloni tabele. Popuniti ostale ćelije tabele.

Akcija procesora	Stanje keš kopije u P1		Stanje keš kopije u P2		Stanje keš kopije u P3		Podatak se pribavlja iz GM ili keša		Stanje kopije u glavnoj memoriji	
	a)	b)	a)	b)	a)	b)	a)	b)	a)	b)
P1 čita X										
P3 čita X										
P3 upisuje u X										
P3 upisuje u X										
P2 upisuje u X										

46. Multiprocesorski sistem se satoji od tri procesora. Svaki procesor ima svoju keš memoriju. Prikazati kako se vrši promena stanja keš blokova i glavne memorije ako se za održavanje keš koherencije koristi:
c) Write-once protokol
d) Firefly protokol
Procesori pristupaju deljivoj promenljivoj X. Akcije procesora navedene su u prvoj koloni tabele. Popuniti ostale ćelije tabele.

Akcija procesora	Stanje keš kopije u P1		Stanje keš kopije u P2		Stanje keš kopije u P3		Podatak se pribavlja iz GM ili keša		Stanje kopije u glavnoj memoriji	
	a)	b)	a)	b)	a)	b)	a)	b)	a)	b)
P1 čita X	Val	Val. e.	X	X	X	X	Gm	GM	Val	Val
P3 čita X	Val	Sh	X	X	Val	Sh	GM	Gm	Val	Val
P3 upisuje u X	Inval	Sh	X	Sh	Rsrvd	Sh	Keš	Keš	Val	Val
P3 upisuje u X	Inval	Sh	X	Sh	Dirty	Sh	Keš	Keš	Inval	Val
P2 čita X	Inval	Sh	Val	Sh	Val	Sh	GM	GM	Val	Val

47. Koje se tehnike koriste za sinhronizaciju pristupa deljivim resursima (kritičnim sekcijama) kod multiprocesorskih sistema? x2+1

Multiprocesorski sistemi su MIMD.
Kodni segment (deo programa) koji se obraća deljivoj promenljivoj predstavlja tvz. kritičnu sekciju. Tenike koje se koriste za sinhronizaciju pristupa deljivim resursima su:

1) Deljivoj promenljivoj može se pridružiti ključ
2) Semafori
3) Monitori

1) Deljivoj promenljivoj D može se pridružiti ključ *k* nad kojim se mogu obavljati dve operacije LOCK(k) i UNLOCK(k) (zaključaj(k) i otključaj(k)) koje postavljaju k na 1 (zaključavanje) odnosno na 0 (otključavanje).
   Proces može izvršiti LOCK(k) samo ako je k otvoren (tj.postavljen an 0).
   Rezultat izvršenja LOCK(k) je postsvljanje k na 1, i na taj način se sprečava da drugi proces obavi LOCK(k) operaciju.
   - Proces koji naidje na zaključanu bravu, mora da čeka.
   - Operacijom UNLOCK(k) vrednost k se postavlja na 0.
2) Semafor je celobrojna promenljiva koja može uzeti samo nenegativne vrednosti.
   Jedine operacije, izuzev inicijalizacije, koje su dozvoljene nad semaforima su P i V opracije (WAIT i SIGNAL).
   Ove operacije su nedeljive, tako da se ne može desiti da više procesa izvršava ove operacije nad istim semaforom u isto vreme.
   Svakom semaforu pridružena je lista procesa koji čekaju na ulazak u kritičnu oblast.
3) Sve kritične sekcije, tj.naredbe koje se obraćaju deljivim promenljivim, sadržane su u monitoru.
   Deklaracija monitora sadrži brojne procedure koje definišu operacije nad deljivim resursima (promenljivim).
   Opšti oblik deklaracije monitora:
   monitor ime;
   deklaracija lokalnih promenljivih za monitor
   deklaracija procedura lokalnih za monitor
   begin
   inicijalizacija lokalnih promeljivih
   end ime;
   Monitorske procedure su kritične sekcije. Monitorske procedure mogu pristupati samo promenljivim koje su lokalne za dati montor. Ovim lokalnim promenljivim ne sme se pristupati van monitora. Ako se ovo poštuje, može se garantovati uzajamna isključivost kod pristupa deljivim promenljivim.
   Kada proces želi da pristupi deljivom resursu, mora pozvati odgovarajuću monitorsku proceduru.
   U jednom trenutku samo jedan proces može izvršavati neku monitorsku proceduru i ona se mora završiti pre nego što se dozvoli da neki drugi prices pozove neku monitorsku proceduru.

48. Zadatak T1 računa x=a*b i šalje vrednost zadatku T2 koji je prihvata i štampa. Napisati na programskom jeziku Ada kako izgledaju zadaci T1 i T2 ako
   a. se entry deklaracija nalazi u T1
   b. se entry deklaracija nalazi u T2 x2+1

49. Objasniti sinhronizaciju procesa korišćenjem Randevua u Adi x3

50. Sinhronizacija ili koordinacija akcija niti (thread) je nekad neophodna da bi se obezbedio korektan pristup deljivim (zajendičkim) podacima i da bi se sprečilo narušavanje podataka. OpenMP nudi relativno mali skup jednostavnih sinhronizacionih mehanizama. Koje su konstrukcije na raspolaganju u OpenMP za sinhronizaciju između niti? Ukratko objasniti sintaksu i dejstvo svake od njih. x4

Za sinhronizaciju niti, pored barrier, OpenMP nudi i konstrukcije:

1. ordered
2. critical
3. atomic
4. lock
5. master

U OpenMP su na raspolaganju sledeće direktive:

- Parallel direktiva
- Direktive za podelu posla
  - for direktiva
  - section direktiva
  - single direktiva
  - task
- Direktive za sinhronizaciju niti:
  1. Barrier
  2. Ordered
  3. Critical
  4. Atomic
  5. Master
  6. Bibliotečke funkcje – lock

Kod direktiva za sinhronizaciju niti svaka direktiva opciono može sadržati i veći broj odrebi (klauzula).Ove direktive omogućavaju da se sinhronizuje pristup deljivim promenljivim od strane više niti.
Algoritam može zahtevati da se orkestrira (upravlja) redosledom pristupa deljivim promenljivim da bi se obezbedilo da više niti ne pristupa jednovremeno deljivoj promenljivoj radi upisa.

- Barrier:
  Barijera je tačka u izvršenju programa u kojoj niti čekaju jedna drugu tako da ni jedna nit ne može da produži sa izvršenjem dok sve niti iz tima niti ne stignu u barijeru.

  Sintaksa u C/C++:
  #pragma omp barrier

- Direktiva ordered:
  Omogućava da se strukturni blok u okviru paralelne petlje izvrši sekvencijalno. Kod van ovog bloka se izvršava paralelno.

  Sintaksa direktive u C/C++:
  #pragma omp ordered
  strukturni blok

  Ukoliko se koristi direktiva ordered uz nju mora da se iskoristi odredba ordered. Njena svha je da informiše kompajler o prisustvu ordered direktive.

- Direktiva critical:
  Omogućava da niti pristupaju kritičnoj sekciji uzajamno isključivo. Kritičnoj sekciji se opciono može dati ime. Kada neka nit naiđe na konstrukciju *critical* ona čeka da druga nit okonča pristup kritičnoj sekciji pre nego što ona uđe u kritičnu sekciju.

  Sintaksa**:**
  #pragma omp critical *[(ime)]*
  strukturni blok

- Direktiva atomic:
  Omogućava da više niti ažurira deljivu promenljivu bez interferencije. Za razliku od ostalih konstrukcija, ona se primenjuje samo na jednu naredbu dodeljivanja koja neposredno sledi iza konstrukcije atomic. Atomic direktiva omogućava efikasno ažuriranje deljivih promenljivih od strane više niti na hardverskoj platformi koja podržava atomične operacije.

  Sintaksa**:**
  #pragma omp atomic
  naredba

- Direktiva master:
  Definiše blok naredbi koji će izvršiti samo master nit. Slična je *single* direktivi. Ova direktiva nema implicitnu barijeru na ulasku ili nakon okončanja strukturnog bloka. Zbog toga, ako je neophodno, treba eksplicitno ubaciti barrier konstrukciju da bi se postigla željena sinhronizacija.

  Sintaksa**:**
  #pragma omp master
  strukturni blok

- Runtime funkcije za sinhronizaciju:
  Pored navedenih direktiva OpenMP ima i skup run time funkcija koje su po načinu rada slične semaforima. Ove funkcije kao argument imaju promenljivu posebne namene, *lock* promenljivu kojoj se može pristupati samo preko ovih funkcija.

  Opšti oblik ovih funkcija je:
  void omp_*func*_lock (omp_lock_t *lck)

  Dva tipa locks – simple locks i nestable locks (analogno binarnimi i brojnim semafor).

51. Uz OpenMP direktive mogu se koristiti određenje odredbe (klauzule). Popuniti donju tabeli navodeći oznaku x tamo gde se određena klauzula može koristiti navedenu direktivu. x1+1

	Direktiva	Direktiva	Direktiva
Klauzula (odredba)	**#pragma omp parallel**	**#pragma omp for**	**#pragma omp sections**
shared	**x**		
private	**x**	**x**	**x**
lastprivate		**x**	**x**
firstprivate	**x**	**x**	**x**
nowait		**x**	**x**
schedule		**x**	
if	**x**		
num_threads	**x**		
ordered		**x**	
reduction	**x**	**x**	**x**

Sintaksa OpenMP direktiva je sledeća
#pragma omp ime_direktive [odredba[[,] odredba]...]

- *omp ključna reč govori da je reč o OpenMP pragmi koji kompajlira samo OpenMP kompajler, a ignorišu ne OpenMP kompajleri*
- *ime_direktive ime direktive (npr. parallel, for, section, ...)*
- *odredbe (klauzule, eng. clauses) se koriste da daju dodatne informacije direktivi*

- Parallel direktivom se kreira tim niti i definiše paralelni region u kome će taj tim raditi.
  Sa parallel direktivom mogu se koristiti sledeće odredbe:

if *(scalar-expression)*	(C/C++)
if *(scalar-logical-expression)*	(Fortran)
num_threads *(integer-expression)*	(C/C++)
num_threads *(scalar-integer-expression)*	(Fortran)
private *(list)*	
firstprivate *(list)*	
shared *(list)*	
default(none|shared)	(C/C++)
default(none|shared|private)	(Fortran)
copyin *(list)*	
reduction *(operator:list)*	(C/C++)
reduction *({operator | intrinsic_procedure_name}:list)*	(Fortran)

  - *samo jedna if odredba se može pojaviti u parallel direktivi*
  - *samo jedna num_threads odredba se može pojaviti u direktivi. Izraz mora vratiti pozitivan ceo broj*

- For direktiva uzrokuje da iteracije petlje budu izvršene paralelno.
  Sa for direktivom mogu se koristiti sledeći tipovi odredbi:

private *(list)*	
firstprivate *(list)*	
lastprivate *(list)*	
reduction *(operator:list)*	(C/C++)
reduction *({operator | intrinsic_procedure_name}:list)*	(Fortran)
ordered	
schedule *(kind[,chunk_size)]*	
nowait	

- Sections/section direktiva najčešće se koristi za paralelno izvršenje funkcija ili potprograma.
  Odredbe uz sections:

private *(list)*	
firstprivate *(list)*	
lastprivate *(list)*	
reduction *(operator:list)*	(C/C++)
reduction *({operator | intrinsic_procedure_name}:list)*	(Fortran)
nowait	

52.U čemu je razlika između #pragma omp single i #pragma omp master direktiva? x2

#pragma omp single
strukturni blok

#pragma omp master
strukturni blok

Single direktivom se postiže da se strukturni blok izvršava od strane samo jedne niti.
- Ona ne kaže koja je to nit, već samo da se taj blok izvršava sekvencijalno.
- Druge niti čekaju na barijeri da se okonča izvršenje bloka koji se nalazi u okviru single direktive

Master direktiva definiše blok naredbi koji će izvršiti samo master nit. Slična je single direktivi.
Ova direktiva nema implicitnu barijeru na ulasku ili nakon okončanja strukturnog bloka.
- Zbog toga, ako je neophodno, treba eksplicitno ubaciti barrier konstrukciju da bi se postigla željena sinhronizacija.
  - Npr. ako se master konstrukcija koristi da inicijalizuje podatke, mora se voditi računa da se inicijalizacija okonča pre nego što ostale niti u timu krenu da koriste podatke.

53.(KOL2-2015)Objasniti dejstvo firstprivate i lastprivate odredbi.

Privatnim promenljivima se ne može pristupati nakon okončanja paralelnog regiona (petlje ili sekcije). Ako je promenljiva potrebna nakon okončanja paraelnog regiona. Odredba *lastprivate* (lista_promenljivih):

- Obezbeđuje da vrednost promeljivih navedenih u listi budu definisane nakon izlaska iz paralelnog regiona.
- Poslednja vrednost (lastprivate) kod sekvencijalnog izvršenja je nedvosmislena.
  - Međutim, kod paralelnog izvršenja je potrebno objasniti šta znači „poslednja" vrednost.
    - U slučaju da se radi o paralelnoj for petlji, promenljiva će nakon okončanja paralelne petlje imati vrednost koju bi imala nakon poslednje iteracije sekvencijalne petlje.
  - Ako se odredba *lastprivate* koristi sa *section* direktivom onda promenljiva dobija vrednost koju ima na kraju u leksikografski poslednjoj section konstrukciji.

Privatne promenljive paralelnog regiona nisu inicijalizovane na ulasku u paralelni region.

- Odredbom *firstprivate*(lista_promenljivih) se promenljive navedene u listi promenljivih deklarišu kao privatne, ali se inicijalizuju na vrednost promenljive sa istim imenom koja se nalazi van paralelnog regiona.
- Inicijalizaciju obavlja polazna nit pre izvršenja paralelne konstrukcije.

54.U čemu je razlika između private i firstprivate odredbi?

Odredba private omogućava da se definiše koje će promenljive biti privatne za svaku nit u timu. *private* (lista_promenljivih)

- Svaka promenljiva u listi se replicira tako da svaka nit u timu ima ekskluzivno pravo pristupa svojoj kopiji promenljive.
- Promene koje učini jedna nit nisu vidljive drugoj niti.

Privatne promenljive paralelnog regiona nisu inicijalizovane na ulasku u paralelni region.
Odredbom *firstprivate* (lista_promenljivih):

- se promenljive navedene u listi promenljivih deklarišu kao privatne, ali se inicijalizuju na vrednost promenljive sa istim imenom koja se nalazi van paralelnog regiona.
- Inicijalizaciju obavlja polazna (master) nit pre izvršenja paralelne konstrukcije

Razlika između private i firstprivate odredbe, primer:

```
A = 1, B = 1, C = 1
#pragma omp parallel private(B) firstprivate(C)
```

- Da li su A,B,C lokalne za svaku nit ili su deljive unutar paralelnog regiona?
- Koje su inicijalne vrednosti ovih promenljivih na početku i kraju paralelnog regiona?
- Unitar paralelnog regiona ...
  - "A" je deljiva za sve niti i njena vrednost na ulasku u paralelni region je 1.
  - "B" i "C" su lokalne promenljive za svaku nit.
  - vrednost promenljive B na ulasku u paralelni region nije definisana
  - inicijalna vrednost promenljive C u paralelnom regionu je 1
- Nakon izlaska iz paralelnog regiona...
  - B i C će se vratiti na vrednosti pre ulaska u paralelni region, tj. imaće vredn. 1
  - A će imati vrednost koju je dobio unutar paralelnog regiona (ili 1 ako joj vrednost nije promenjena u paralelnom regionu)

Dakle, razlika je u tome što vrednost promenljive B (private) na ulasku u paralelni region nije definisana, dok C (firstprivate) ima inicijalnu vrednost.

55. a) U čemu je razlika u dejstvu direktive threadprivate i odredbe private?
b) Ako su kreirane 4 niti prikazati kako izgledaju izlazi iz dole navedenog programa dobijeni sa *printf.*

```
#include<omp.h>
int a,b,i,tid; float c;
#pragma omp threadprivate(a,c)
main ()
{
        omp_set_dynamic(0);
        #pragma omp parallel private(b, tid)
        {
                tid=omp_get_thread_num(b,tid);
                a=tid;
                b=2*tid;
                c=3.0*tid;
                printf("Thread %d: a, b, c = %d %d %f\n", tid, a, b, c);
                }
        #pragma omp parallel private(b, tid)
                {
                        tid=omp_get_thread_num(tid);
                        printf("Thread %d: a, b, c = %d %d %f\n", tid, a, b, c);
                }
}
```

You can make named common blocks private to a thread, but global within the thread, by using the THREADPRIVATE directive.

Each thread gets its own copy of the common block with the result that data written to the common block by one thread is not directly visible to other threads. During serial portions and master sections of the program, accesses are to the master thread copy of the common block.

56. Na koje se sve načine može definisati broj niti koji se kreira u paralelnom regionu i koji su njihovi prioriteti? x4+1+1

Kod u okviru paralelnog regiona se izvršava od strane svih niti.
Za neke aplikacije je važno da može da se upravlja brojem niti koje izvršavaju paralelni region.
OpenMP omogućava programeru da specificira ovaj broj:

- pre izvršenja programa korišćenjem odgovarajuće promenljive okruženja (environment variable),
- nakon početka izvršenja korišćenjem izvršne bibliotečke funkcije, ili
- na početku paralelnog regiona korišćenjem odgovarajuće odredbe (klauzule).

Ako se ovo ne uradi, onda broj niti zavisi od implementacije.
OpenMP svakoj niti u timu dodeljuje jedinstveni identifikator koji se kreće u garnicama od 0 do broj niti-1.

Postoji više načina da se definiše koliko broj niti će biti u paralelnom regionu.
Po prioritetu:

- IF odredba
- NUM_THREADS odredba
- omp_set_num_threads() bibliotečka funkcija
- OMP_NUM_THREADS promenljiva okruženja
- Default: zavisno od implementacije

57. Potrebno je paralelno izvršiti 100 iteracija petlje. Ako su kreirane 4 niti, prikazati kako će izgledati distribucija iteracija po nitima ako se koriste odredbe

- **schedule (static, 10)**
- **schedule (dynamic, 10) i**
- **schedule (guided, 10)?**

Odgovore dati popunjavanjem tabele oblika: x3+1

	Nit 0	**Nit 1**	**Nit 2**	**Nit 3**
iteracije	**Od x do y**	**Od x do y**	**Od x do y**	**Od x do y**
iteracije	**...**	**...**	**...**	**...**

isti zadatak kao ispod

58. Potrebno je paralelno izvršiti 100 iteracija petlje. Ako su kreirane 4 niti, prikazati kako će izgledati distribucija iteracija po nitima ako se koriste odredbe

a. **schedule (static, 10)**
b. **schedule (dynamic, 10) i**
c. **schedule (guided, 10)?**

Odgovore dati popunjavanjem tabele oblika:

	Nit 0	**Nit 1**	**Nit 2**	**Nit 3**
Iteracije	**Od x do y**	**Od x do y**	**Od x do y**	**Od x do y**
a.	Od 0 do 9	Od 10 do 19	Od 20 do 29	Od 30 do 39
	40 - 49	50 - 59	60 - 69	70 - 79
	80 - 89	90 - 99		
b.	70-79	20-29	90-99	0-9
	50-59	60 - 69	10 - 19	80 - 89
	40 - 49	30 - 39		
c.	0 - 24	43 - 56	25 - 42	57 - 66
	97 - 99	77 - 86	87 - 96	67-76

schedule (static, 10)
10 je broj koji nam kaze koliko iteracija dobija svaka od niti. Posto je static, dodeljujemo svakoj niti po 10 iteracija i to redom - nit 0, 1, 2, 3, 0, 1, 2, 3...

schedule (dynamic, 10)
10 je opet broj koji nam kaze koliko iteracija dobija svaka od niti. Ali sada posto je dynamic, nove iteracije se dodeljuju niti cim ona zavrsi prethodni set iteracija. Obzirom da ne znamo koliki je posao u odredjenom setu iteracija, ne znamo kojim redosledom niti dobijaju iteracije, sto znaci da treba da napisemo neki random raspored. Npr 0, 3, 3, 2, 1, 2, 1, 0, 3, 0, 0, 1… samo je važno da je random.

schedule (guided, 10)
Kod guided, 10 je broj koji nam kaze koliko najmanje iteracija moze da se nadje u jednom chunk-u. Znaci ne sme da postoji chunk koji ima manje iteracija od 10. Kod guided se velicina chunk-a menja kroz vreme, tako sto se smanjuje. Znaci prva nit dobije najvise iteracija, pa sledeca manje, itd. Sto se tice rasporeda iteracija po nitima, raspored je isti kao kod dynamic – RANDOM, tako da se ne brine o rasporedu (samo toliko da izgleda random) nego samo o velicini chunkova.

Kako racunamo velicinu chunkova? Prvi je broj_iteracija / broj_niti. U ovom slucaju

100 / 4 = 25 (to su iteracije 0 - 24, ostalo je jos 75 iteracija)

Sledeca nit dobija ostatak iteracija podeljen sa brojem niti:

75 / 4 = 18.75 (racunas 18, to su iteracije 25 - 42, sada ostaje jos 75-18 = 57 iteracija)

Pa onda opet, sledeca nit dobija ostatak iteracija podeljen sa brojem niti:

57 / 4 = 14.25 (racunas 14, to su iteracije 43 - 56, sada nam ostaje 57 - 14 = 43 iteracije)

Opet isto:

43 / 4 = 10.75 (racunas 10, to su iteracije 57 - 66, sada nam ostaje 43 - 10 = 33)

Ovde vidmo da ne mozemo vise da primenjujemo formulu kao u prethodnim koracima, jer bi to dovelo do chunk-a manjeg od 10 iteracija. Tako da rasporedjujemo po 10 iteracija na dalje (broj koji smo dobili kao broj iteracija u prethodnom koraku). Tako da ce biti:

67 - 76
77 - 86
87 - 96
97 - 99
U poslednjem koraku vidimo da imamo samo 3 iteracije i to je ok, jer tako pise u teoriji: "sa izuzetkom poslednjeg bloka koji može biti manji"

Sada samo uzmemo i popunimo ove brojeve iteracija u tabelu, random redoledom, i to je to. Mada mislim da prva nit treba da dobije najveci chunk, tako je na slici dole.

(SA PREZENTACIJA:)

Odredba schedule se koristi da se upravlja načinom raspoređivanja (distribucije) iteracija petlje između niti.
*Sintaksa**:***

- ***schedule** (kind [, chunk size]).*

Odredba schedule definiše kako se iteracije petlje dodeljuju nitima u timu.

Kod distribucije static:
Iteracije se dodeljuju nitima u blokovima veličine chunk_size. Blokovi se dodeljuju nitima statički kružno (na round robin način) po redosledu brojeva niti. Poslednji blok koji se dodeljuje niti može biti manji, tj. imati manji broj iteracija. Svakoj niti se dodeljuje bar jedan blok.

Distribucija dynamic:

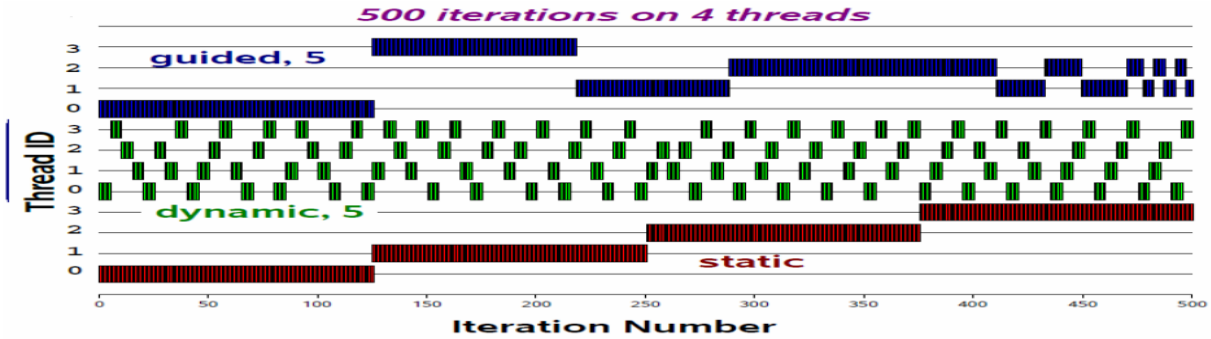
- *Iteracije se dodeljuju nitima po zahtevu niti.*
- *Nit izvršava blok iteracija čija je veličina definisana vrednošću parametra chunk_size, zatim zahteva sledeći blok sve dok ima blokova koji treba da se obrade.*
- *Poslednji blok može imati manje iteracija od chunk_size.*

Distribucija guided:
Varijacija distribucije dynamic.

- *Iteracije se dodeljuju nitima po zahtevu niti.*
- *Nit izvršava blok iteracija čija je veličina definisana parametrom chunk_size.*
- *Kada okonča izvršenje bloka, nit zahteva sledeći blok, itd.*

Ako parametar čhunk_size ima vrednost k (k >1), veličina bloka se određuje na isti način sa ograničenjem da blok ne može imati manje od k iteracija (sa izuzetkom poslednjeg bloka koji može biti manji).

Primer:



za static distribuciju nije navedena vrednost chunk_size pa je iterativni prostor od 500 iteracija podeljen na 4 niti od po 125 iteracija
dynamic i guided koriste chunk_size=5
kod dynamic prvoj niti se dodeljuje 125 iteracija, drugoj (500-125)/4, itd dok se ne dodje do veličine chunk_size=5

Kolokvijumi 2015. godina

59. KOL1 Zaokružiti tačan odgovor. Hazardi su

- **svojstvo arhitekture**
- **svojstvo programa**
- **svojstva arhitekture i programa**

Hazardi su svojstvo protočnog sistema.

60. KOL1 Zaokružiti tačan odgovor. Zavisnosti po podacima su

- **svojstvo arhitekture**
- **svojstvo programa**
- **svojstva arhitekture i programa**

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 ukazuje na potencijalni hazard, ali stvarni hazard i dužina zastoja zavise od konkretnog protočnog sistema.

61. KOL1 U kojoj od faza izvršenja kod Scoreboard se razrešavaju x2+1

- **WAR hazardi –** u Write results fazi
- **WAW hazardi –** u Issue fazi
- **RAW hazardi? –** u Read operands fazi

62. KOL1 U kojoj fazi izvršenja se kod Tomasulovog algoritma obavlja preimenovanje registara?

Kod Tomasulovog algoritma preimenovanje registara se obavlja u ISSUE fazi.
Registri su preimenovani u ovom koraku, eliminišući WAR i WAW opasnosti.

63. KOL1 Opisati rad n-bitnog prediktora za predikciju grananja x1+1 (i 2014. bilo)

2-bitni prediktor je specijalni slučaj n-bitne šeme koja koristi n-btni zasićeni brojač za predikciju grananja.
Vrednosti koje brojač može uzeti su u opsegu $0\text{-}2^n\text{-}1$.
Ako je vrednost brojača $\geq 2^{n-1}$ predvidjanje je da će se grananjae obaviti (taken), u suprotnom da se ne obavlja (not taken).
Brojač se inkrementira svaki put kada se grananje obavi, a dekrementira kada se grananje ne obavi

- Kada je vrednost brojača $2^n\text{-}1$, nema inkrementiranja
- Kada je vrednost brojača 0, nema dekrementiranja

Proučavanja n-bitnih prediktora su pokazala da se 2-bitni prediktori ponašaju skoro isto dobro kao n-bitni (n>2) prediktori.

- zbog toga većina sistema koristi 2-bitne prediktore

64. KOL1 Ukratko objasniti razlike između VLIW i superskalarne organizacije procesora.

Ideja je kod obe organizacije je izdati više od jedne instrukcije u jednom clk ciklusu.

SS procesori mogu da izdaju različit broj instrukcija po clk ciklusu.
VLIW izdaje fiksni broj instrukcija koje su formatirane kao jedna velika instrukcija ili kao fiksni instrukcioni paket.

SS procesori mogu da koriste i statičko i dinamičko (zasnovano na SC ili Tomasulo) planiranje izvršenja instrukcija.
VLIW su sa statičkim planiranjem izvršenja instrukcija.

65. KOL1 Pomoću NZD testa ustanoviti da li u sledećoj petlji postoje loop carry zavisnosti

```
for i=3,50
        x(i)=y(i-2)*z(i)
        y(i)=y(i)+z(i)
endfor
```

Da li je petlju moguće vektorizovati preuređenem instrukcija? Ako jeste, napisati vektorski kod.

Pretpostavimo da se u petlji vrši upis u element polja sa indeksom $a*i+b$, a da se pristupa elementu sa indeksom $c*i+ d$, gde je i indeks petlje koji se kreće u granicama od m do n.
Loop-carry zavisnost postoji ako važi sledeće:

- Postoje dva iterativna indeksa $\boldsymbol{j}$ i $\boldsymbol{k}$ ($\boldsymbol{j} < \boldsymbol{k}$), oba unutar granica petlje, tako da se u petlji pamti rezultat u element sa indeksom $\boldsymbol{a*j+b}$, a zatim se pribavlja (čita) isti element kada je indeksiran sa $\boldsymbol{c*k+d}$, tj. kada je $\boldsymbol{a*j+b} = \boldsymbol{c*k+d}$.

Jednostavan i dovoljan test koji se koristi za detekciju zavisnosti je NZD (najveći zajednički delilac) test. (GCD)

Ako loop-carry zavisnost postoji tada NZD(c, a) mora deliti (d – b) bez ostatka (baš (d – b) jer je j < k).

NZD test je dovoljan da garantuje da nema nikakve zavisnosti.
Može se desiti da NZD test da odgovor da zavisnost postoji a da nema nikakve zavisnosti.
Ovo može da nastupi zato što NZD test ne uzima u obzir granice petlje.

a=1, b=0, c=1, d=-2
d-b=-2
NZD(1,1)=1
1 deli -2 bez ostatka => loop cary zavisnost postoji.

2014. godina

66. Navesti osnovne karakteristike multiprocesora i multiračunara.

• **Multiprocesori (čvrsto spregnuti sistemi)**

- Multiprocesor sadrži dva ili više homogenih procesora podjednakih mogućnosti.
- Svi procesori dele pristup zajedničkoj (deljivoj) memoriji preko sprežne mreže.
- Svaki procesor može imati malu lokalnu memoriju koja se može koristiti za pamćenje jezgra OS-a.
- Komunikacija između procesora ostvaruje se preko deljive memorije korišćenjem zajedničkih (deljivih) promenljivih.
- Razmene upravljačkih informacija može se ostvariti i preko procesor-procesor sprežne mreže, mada njeno postojanje nije obavezno.
- Svi procesori dele pristup U/I kanalima i uređajima, DMA kontrolerima preko zajedničke mreže.
- Radom celokupnog sistema upravlja jedinstveni OS koji obezbeđuje interakciju između procesora i njihovih programa na nivou posla, zadatka, skupa podataka i elementa podatka.
- Sprežne mreže multiprocesora su dinamičkog tipa, što znači da se veze između procesora i memorija uspostavljaju po zahtevu u toku izvršenja programa.

• **Multiračunari (slabo spregnuti sistemi)**

- Svaki procesor ima lokalnu memoriju i skup U/I uređaja mada mogu deliti neke periferije.
- Procesor sa svojom memorijom i U/I uređajima zove se računarski modul ili čvor.
- Sprega procesora sa lokalnom i memorijom i lokalnim U/I uređajima ostvaruje se preko lokalne magistrale.
- Skup procesora može biti heterogen.
- Kod slabo spregnutih sistema ne postoji zajednička memorija.
- Komunikacija između procesora ostvaruje se slanjem poruka kroz sprežnu mrežu.
- Sprega čvora sa sprežnom mrežom ostvarena preko odgovarajućeg interfejsa.
- Sprežne mreže su statičkog tipa.

67. Koji se hazardi javljaju kod protočnih sistema? Ukratko ih objasniti.

Hazardi su situacije koje sprečavaju da izvršenje instrukcije otpočne u predvidjenom clok ciklusu. Oni redukuju idealne performanse protočnog sistema (izvršenje jedne instrukcije po klok ciklusu).

Hazardi se mogu klasifikovati u tri grupe:

- Strukturni hazardi:
  Nastaju zbog jednovremenih zahteva za korišćenjem istog hardverskog resursa.

- Hazardi po podacima:
  Nastupaju zato što je redosled pristupa operandima izmenjen uvodjenjem protočnosti u odnosu na sekvencijalno izvršenje instrukcija. Postoje:
  - RAW (Read After Write)
  - WAW (Write After Write)
  - WAR (Write After Read) hazardi

- Kontrolni hazardi:
  Nastupaju zbog zavisnosti u redosledu izvršenja instrukcija. Izazivaju ih instrukcije koje mogu promeniti sadržaj PC (call, branch, jump, return).

68. Navesti razlike izmedju Scoreboard tehnike i Tomasulovog algoritma.

- Tomasulo kombinuje ključne elelmente Sc. šeme i tehnike preimenovanja registara radi eliminacije WAR i WAW hazarda.
- Tomasulo ima tri faze, Scoreboard četiri.
- Kod Tomasulovog alg. upravljanje je distribuirano po Rezervacionim stanicama naspram centralizovanog upravljanja kod Scoreboard.
- Tomsulo koristi pribavljanje unapred, SC ne koristi (operandi instr.se čitaju tek kad su oba dostupna).
- Tomasulo koristi Common Data Bus.
- Tomasulo vrši preimenovanje registara.
- Kod Tomasulovog alg. rezultati instrukcija se prosledjuju FU iz RS preko *Common Data Bus (CDB)* (*a ne preko registara* kao kod SC), preko kojeg se rezultati distribuiraju svim FU.
- Kod Tomasula Load i Store se tretiraju kao FU sa svojim RS.
- U različitim fazama im se razrešavaju WAW, WAR i RAW hazardi.
- Tomasulo omogućava dinamičko odmotavanje petlje.

69. Objasniti rad korelacionih prediktora.

Dinamička predikcija grananja se razlikuje od statičke jer koristi ponašanje grananja u toku izvršenja programa da predvidi ishod grananja.

Pouzdanost predvidjanja moguće je poboljšati ako se posmatra ponašanje i drugih naredbi grananja, a ne samo one čije ponašanje želimo da predvidimo (kao kod BTB i BPB). Prediktori koji koriste ponašanje drugih naredbi grananja da obave predvidjanje za tekuću naredbu grananja zovu se korelacioni ili 2-nivovski prediktori**.**

(1,1) korelacioni prediktor se može shvatiti kao da svako grananje ima dva posebna predikciona bita:

- jedno je predvidjanje koje se koristi ako se poslednje grananje nije obavilo (not taken NT)
- drugo predvidjanje koje se koristi ako se poslednje grananje obavilo (taken T)

Par predikcionih bitova se beleži zajedno:

- prvi bit predstavlja predvidjanje koje se koristi ako se poslednje grananje nije obavilo
- drugi bit predstavlja predvidjanje koje se koristi ako se poslednje grananje obavilo

predikcioni bitovi	predvidjanje ako poslednje grananje nije obavljeno	predvidjanje ako je poslednje grananje obavljeno
NT/NT (00)	NT	NT
NT/T (01)	NT	T
T/NT (10)	T	NT
T/T (11)	T	T

Korelacioni prediktori imaju veću pouzdanost predvidjanja od 2-bitnih prediktora a zahtevaju trivijalni dodatni hardver.

70. Data je matrica zavisnosti po podacima $D = [d_1 \quad d_2 \quad d_3] = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 2 \end{bmatrix}$ **i matrica transformacije** $T = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$**. Da li je dozvoljeno izvršiti transformaciju petlji primenom transformacije T? Zašto?**

Da bi jedna transformacija mogla da se primeni nad indeksnim skupom, a da to ne utiče na koreknost izračunavanja, matrica transformacije T ne sme da menja znak vektora zavisnosti.

$$D = [d_1 \quad d_2 \quad d_3] = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 2 \end{bmatrix}$$

matrica zavisnosti

$$T = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

matrica transformacie

rešenje:

$$\hat{D} = T \cdot D = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 2 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Nije dozvoljeno izvršiti transformaciju petlji, zato što T menja znak vektoru zavisnoti d_1.

71. Izvršiti komparativnu analizu sledećih sprežnih mreža u odnosu na cenu, performanse i pouzdanost:

a. **Magistrala (bus)**
b. **Hiperkub**
c. **Rešetka (mreža)**
d. **Potpuno povezana**
e. **Višestepena dinamička SM**
f. **Krosbar**
g. **Zvezda**

72. Za postizanje keš koherencije u multiprocesorskim sistemima koriste se dve politike:

a. **invalidacija pri upisu**
b. **ažuriranje pri upisu**

Ni za jednu od politika se ne može reći da je bolja od one druge. Navesti primere kada politika ''invalidacija pri upisu'' ima bolje performanse od politike ''ažuriranje pri upisu'' i obrnuto.

2013.godina

73. Navesti politike dodele magistrale kod multiprocesorskih sistema.

Arbitraža se može izvesti kao statička i dinamička.
Kod statičke arbitraže, raspored transakcija na magistrali između potencijalnih gospodara vrši se po unapred definisanom rasporedu (obično je redosled dodeljivanja kružni). Dinamička arbitraža se može podeliti po:

- Prioritetu – svakom potencijalnom gospodaru dodeljen je fiksni prioritet. U slučaju više zahteva magistrala se dodeljuje gospodaru sa višim prioritetom.
- Nepristrasnosti – potencijalni gospodari imaju jednak prioritet. Svakom gospodaru koji je izdao zahtev se mora garantovati dodela magistrale pre nego što se bilo kom drugom gospodaru dodeli magistrala po drugi put. Gospodar koji je poslednji koristio magistralu dobija nanjniži prioritet.
- Kombinovana – politika zasnovana na prioritetu i nepristrasnosti. Obično se U/I zahtevima vrši dodela po prioritetu, a procesorskim zahtevima po principu nepristrasnosti. Multiprocesori koriste ovaj tip politike.

74. Kako se može obaviti maskiranje procesnih elemenata u SIMD? x3

Svaka maska deli skup PE-ova na podskup aktivnih I podskup pasivnih. Može se obaviti:
*Statički

- Direktno
- Pomoću adrese PE

*Dinamički

- Maskiranje podacima

75. Koje osobine moraju da zadovolje matrice transformacije? Koje se elementarne transformacije mogu obavljati nad indeksnim promenljivim u petljama?

…znaš to..

76. Kako se određuju vektori zavisnosti po podacima?

odgovoreno gore