

## 1. Osnovne zakonitosti algebre logike.

Algebra logike predstavlja strukturu  $\{S, \wedge, \vee, \neg\}$ , gde je  $S=\{T, \perp\}$ ,  $\wedge$  binarna operacija konjukcije,  $\vee$  binarna operacija disjunkcije i  $\neg$  unarna operacija negacije.

Osnovni zakoni:

- |                       |   |   |
|-----------------------|---|---|
| 1) komutacija:        | $A \cdot B = B \cdot A$                       | $A + B = B + A$                           |
| 2) asocijacija:       | $(A \cdot B) \cdot C = A \cdot (B \cdot C)$   | $(A + B) + C = A + (B + C)$               |
| 3) distribucija:      | $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ | $A + (B \cdot C) = (A + B) \cdot (A + C)$ |
| 4) neutralni element: | $1 \cdot A = A$                               | $0 + A = A$                               |
| 5) inverzni element:  | $A \cdot \neg A = 0$                          | $A + \neg A = 1$                          |

Identiteti i pravila:

- |                          |                                     |                                     |
|--------------------------|-------------------------------------|-------------------------------------|
| 1) nula:                 | $A \cdot 0 = 0$                     | $A + 1 = 1$                         |
| 2) idempotencija:        | $A \cdot A = A$                     | $A + A = A$                         |
| 3) apsorpcija:           | $A \cdot (A + B) = A$               | $A + (A \cdot B) = A$               |
| 4) dvostruka negacija:   | $\neg \neg A = A$                   |                                     |
| 5) De Morganova pravila: | $\neg(A \cdot B) = \neg A + \neg B$ | $\neg(A + B) = \neg A \cdot \neg B$ |

## 2. Sta je logicka funkcija i koliko ima logickih funkcija reda n?

Logicka funkcija je svaka funkcija sa domenom  $S^n$  i kodomenom  $S$ ,  $f: S \times S \times \dots \times S \rightarrow S$ .

Neka su  $A_1, A_2, \dots, A_n$  logicke promenljive. Svaka funkcija  $f: (A_1, A_2, \dots, A_n) \rightarrow \{0, 1\}$  se naziva *logicka funkcija*.

Broj razlicitih funkcija od  $n$  argumenata je  $2^{2^n}$ .

## 3. Sta je pun sistem funkcija? Navesti bar tri primera.

Skup  $F = \{f_1, f_2, \dots, f_n\}$  funkcija algebre logike se naziva pun sistem funkcija ako se proizvoljna funkcija algebre logike moze predstaviti pomocu funkcija iz ovog skupa.

*Pun sistem funkcija* - skup funkcija na osnovu kojih mogu da se izvedu sve ostale funkcije. Ako se iz nekog sistema funkcija mogu izvesti sve funkcije punog sistema funkcija, onda je i taj sistem pun sistem funkcija.

Primeri:  $\{\wedge, \vee, \neg\}$ ,  $\{\vee, \neg\}$ ,  $\{\wedge, \neg\}$ ,  $\{\uparrow\}$ ,  $\{\downarrow\}$

## 4. Normalne forme funkcija.

Elementarna konjunkcija - logicki izraz koji ne sadrzi operaciju disjunkcije

Elementarna disjunkcija - logicki izraz koji ne sadrzi operaciju konjukcije

Savrsena elementarna konjunkcija - elementarna konjunkcija koja sadrzi sve promenljive iz skupa promenljivih od kojih se grade logicki izrazi

Savrsena elementarna disjunkcija - elementarna disjunkcija koja sadrzi sve promenljive iz skupa promenljivih od kojih se grade logicki izrazi

Savrsena disjunktivna normalna forma - disjunktivna forma u kojoj su sve funkcije savrsene elementarne konjunkcije

Savrsena konjunktivna normalna forma - konjunktivna forma u kojoj su sve funkcije savrsene elementarne disjunkcije

## 5. Koji su uobicajeni logicki elementi?

NOT-element (negacija), AND-element (konjukcija), OR-element (disjunkcija), NAND-element, (Seferova f-ja), NOR-element (Pirsova f-ja), XOR-element (ekskluzivno ILI)

## 6. Sta je minimizacija logickih funkcija? Koji su osnovni metodi minimizacije?

Minimizovanje logicke funkcije je pronalazenje njenog najjednostavnijeg zapisa.

Osnovni metodi minimizacije su:

algebarske transformacije, Karnoove mape, tablicna metoda Kvin MekKlaskog

## 7. Na primeru objasniti metod algebarskih transformacija.

$$\begin{aligned} F &= A'BC' + A'BC + ABC' \rightarrow F = A'BC' + A'BC + ABC' + A'BC' \rightarrow F = A'B(C'+C) + BC'(A+A') \\ &\rightarrow F = A'B + BC' \rightarrow F = B(A' + C') \end{aligned}$$

## 8. Objasniti nacin upotrebe Karnoovih mapa.

Pravi se visedimenziona mapa. Na svaku dimenziju navode se najvise po dva argumenta funkcije. Vrednosti argumenata se navode takvim redom da se menja po tacno jedan bit (Hamingova distanca) - 00, 01, 11, 10. Osnovu za uproscavanje predstavlja pravilo:

$A_1A_2...A_k...A_n + A_1A_2...A_k'...A_n = A_1A_2...A_{k-1}A_{k+1}...A_n$ . Kako se susedni kvadrati uvek razlikuju za po najvise 1 bit, ako odgovaraju istim vrednostima funkcije onda se odgovarajuci bit moze eliminisati. I kvadrati na krajnjim suprotnim stranama se smatraju za susedne. KDNF ima po disjunkt za svaki kvadrat u mapi koji sadrzi jedinicu. Ako dva susedna kvadrata sadrže 1, dva disjunkta kojima oni odgovaraju se mogu zameniti jednim, koji ne sadrzi argument koji se razlikuje. Isti postupak se moze primeniti i na grupe od  $2^n$  susednih kvadrata.

Pocinje se posmatranjem sto je moguće veće grupe. Odabere se sve grupe koje su susedne. Ako neki od kvadrata koji sadrži 1 ostane nezaokružen, tada se posmatraju i manje grupe. Dozvoljeno je da jedan kvadrat koji sadrži 1 pripada većem broju grupa. Minimizovana verzija funkcije uključuje odgovarajući broj promenljivih na mesto svake od odabranih grupa, pri čemu se odabrana grupa koja je potpuno obuhvaćena drugim odabranim grupama ignorise kao redundantna.

U nekim slučajevima određena kombinacija vrednosti promenljivih se nikada ne pojavljuje, pa samim tim ne može ni da bude prisutna u rezultatu. Ove vrednosti se označavaju kao "nebitno". Za svaku od ovih kombinacija u odgovarajući kvadrat se unosi slovo "N" koje može da se koristi ili kao 0 ili kao 1, po potrebi.

## 9. Objasniti nacin upotrebe metode Kvin MekKlaskog.

Na početku treba predstaviti funkciju preko DNF, tako da se nedefinisane vrednosti tretiraju kao jedinice. Potrebno je grupisati konjunkcije po broju negacija. Porediti svake dve susedne grupe i napraviti sva moguća sažimanja implikanata tako da su upareni samo ako se razlikuju po stanju jedne promenljive. Označiti štikliranjem implikante koji su učestvovali u sažimanju. Postupak nastaviti sa novodobijenim implikantima sve dok je neko sažimanje moguće. Važno je nastaviti sa svim mogućim sažimanjima iako su svi implikanti već štiklirani. Kada sažimanje više nije moguće, pravi se tabela koja u zaglavlju kolona sadrži konjunkcije koje odgovaraju samo ulazima za koje funkcija ima vrednost jedan. Tabela u zaglavlju vrsta treba da ima sve neuparene implikante. Za svaki implikant iz druge tabele, upisati X u svakoj koloni koja odgovara konjunkciji koja sadrži taj implikant. Ovo označi da taj implikant može da zameni početnu konjunkciju koja je učestvovala u DNF. Potrebno je i dovoljno da svaku konjunkciju zameni po jedan implikant. Prvo označavamo X koje je jedino u koloni, zato što je ono jedina zamena za konjunkciju koja odgovara toj koloni. Sve ostale X-eve iz te vrste takođe označavamo, zato što implikant koji odgovara tom X-u sigurno učestvuje u konačnoj, minimalnoj formi, pa će biti zamena i za sve ostale konjunkcije u kojima se sadrži. Nakon toga biramo implikante koji mogu biti zamena za više konjunkcija (nalaze se u više kolona). Kada je po bar jedan x iz svake kolone označen na bilo koji način, konačni, minimizovani oblik funkcije se sastoji od svih implikanata koji su označeni.

## 10. Sta je kombinatorna mreza?

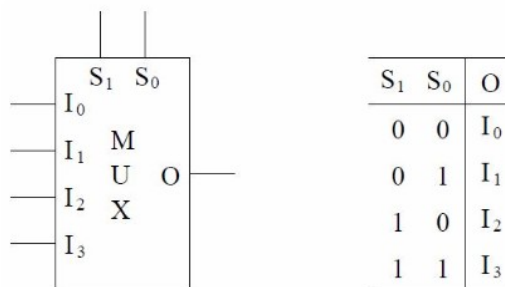
Kombinatorne mreze - predstavljaju skup medjusobno povezanih elemenata ciji je izlaz u nekom trenutku funkcija koja zavisi od vrednosti ulaza u tom istom trenutku. Predstavljaju visi nivo apstrakcije logickih kola. Omogucavaju upotrebu slozenijih logickih elemenata pri projektovanju logickih kola. Njihovom upotrebom se smanjuje broj potrebnih elemenata pri izgradnji logickih kola.

## 11. Navesti najvaznije kombinatorne mreze.

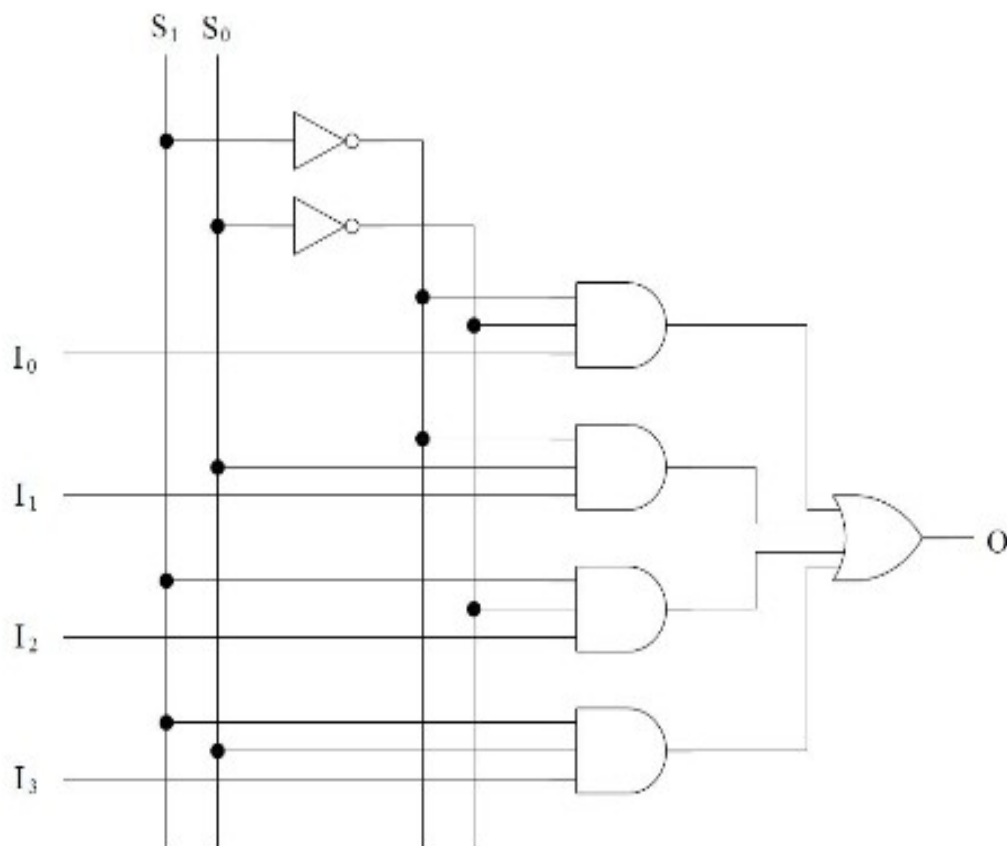
Multipleksori, demultipleksori, dekoderi, enkoderi, komparatori, sabiraci

## 12. Sta je multipleksor? Predstaviti ga grafickim simbolom i tablicom.

Multipleksori – kombinatorne mreze koje imaju  $2^n$  ulaza,  $n$  selektorskih ulazaz i 1 izlaz. Vrednost izlaza odgovara vrednosti ulaza koji je odredjen vrednoscu selektorskih ulaza.



## 13. Nacrtati logicko kolo implemetacije multipleksora 4-1.



#### 14. Kako se multipleksor upotrebljava za implementaciju logickih funkcija?

Osim osnovne primene kombinovanja vise ulaza u jedan izlaz, biranjem ulaznih vrednosti se multipleksorski mogu upotrebljavati za implementiranje funkcija od selektorskih ulaza.

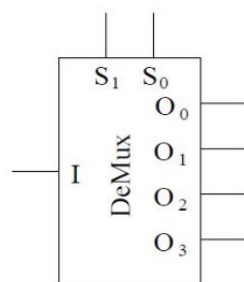
*I nacin:* na ulaze se dovedu konstantne vrednosti, tako da odgovaraju vrednostima funkcije za odgovarajuće selektorske ulaze

*II nacin:* procesom redukcije se multipleksorom može implementirati funkcija sa  $n+1$  argumenata.

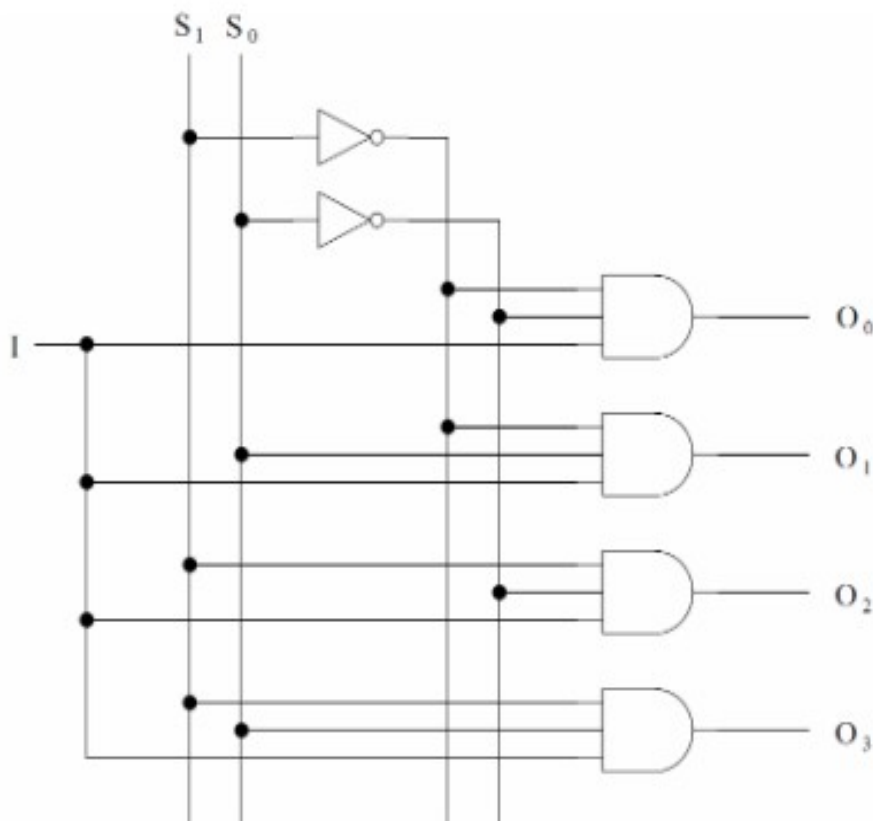
Ideja redukcije je da se jedan od argumenata pretoci u rezultat funkcije: izabere se jedan argument ( $A_k$ ), prepoznaju se slucajevi u kojima vazi  $F=A_k$  ili  $F=\neg A_k$ , u ostalim slucajevima se funkcija predstavi tako da ne zavisi od  $A_k$

#### 15. Sta je demultipleksor? Predstaviti ga grafickim simbolom.

Demultipleksori – kombinatorne mreze koje imaju  $n+1$  ulaza (1 ulazna vrednost i  $n$  selektorskih ulaza) i  $2^n$  izlaza. Obavljaju inverznu funkciju multipleksora: tacno na jedan izlaz se preslikava vrednost ulaza, a svi ostali izlazi imaju vrednost 0.



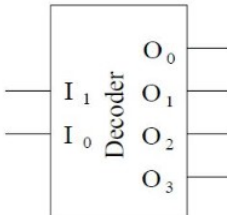
#### 16. Nacrtati logicko kolo implementacije demultipleksora 1-4.



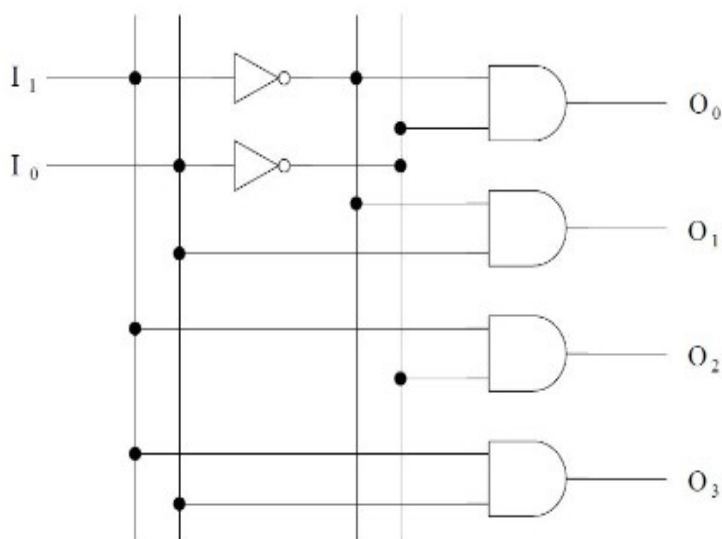
### 17. Sta je dekodler? Predstaviti ga grafickim simbolom i tablicom.

Dekoderi – kombinatorne mreze koje imaju  $n$  ulaza i  $2^n$  izlaza. U svakom trenutku je aktivan najviše jedan izlaz, a u zavisnosti od ulaza.

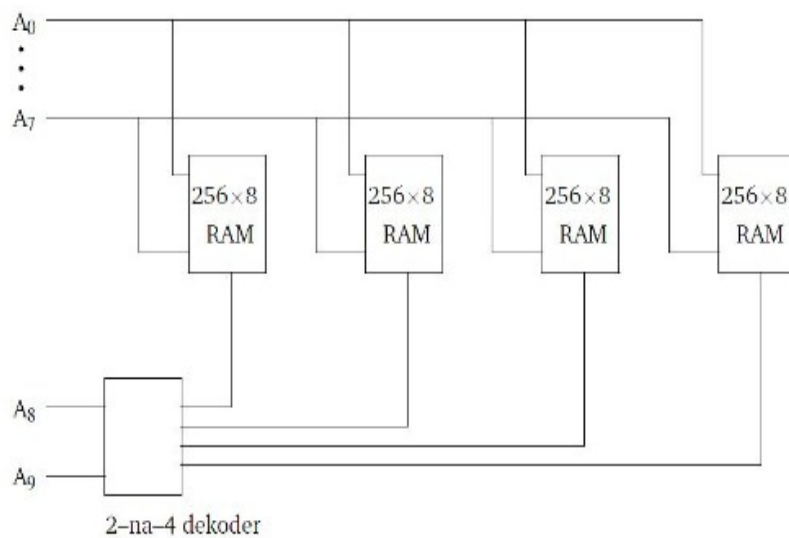
$I_1$	$I_0$	$O_3$	$O_2$	$O_1$	$O_0$
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



### 18. Nacrtati logicko kolo implementacije dekodera 2-4.



### 19. Kako se dekoderi 2-4 mogu implementirati za prosirivanje adresnog prostora na vise cipova?

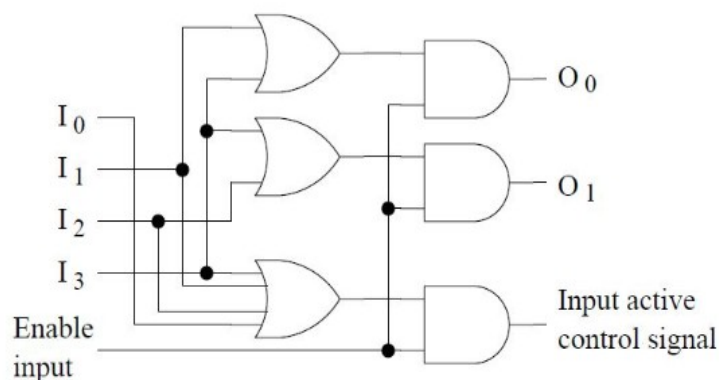


## 20. Sta je enkoder? Nacrtati tablicu vrednosti enkodera 4-2.

Enkoderi – kombinatorne mreze koje imaju  $2^n$  ulaza i  $n$  izlaza. Predstavljaju inverznu operaciju dekodera: u svakom trenutku je aktivan najviše jedan ulaz i izlaz je određen aktivnim ulazom. Obično se dodaju kontrolni ulaz koji uključuje enkoder i kontrolni izlaz koji je aktivan ako su aktivni kontrolni ulaz i bar jedan ulazni bit.

Enable input	$I_3$	$I_2$	$I_1$	$I_0$	$O_1$	$O_0$	Input active control signal
0	X	X	X	X	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	1
1	1	0	0	0	1	1	1

## 21. Nacrtati logicko kolo enkodera 4-2.



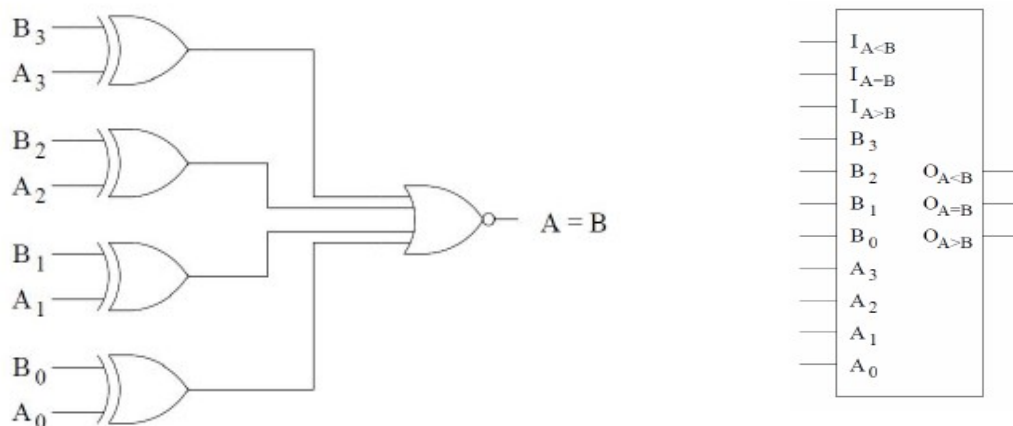
## 22. Sta je komparator? Navesti osnovne vrste komparatora.

Komparatori – mreze koje porede dva niza bitova na odgovarajući način. Imaju  $2 \times n$  ulaza i odgovarajući broj izlaza, zavisno od vrste poredjenja.

Primeri: • poredjenje jednakosti (dovoljan 1 izlaz)

- poredjenje velicine broja (dovoljna su 2 izlaza ali se koriste 3)
- poredjenje sa prosirenjem (ako se porede brojevi koji imaju  $m \times n$  bitova, ulazna informacija obuhvata i rezultat poredjenja prethodne grupe od  $n$  bitova)

## 23. Nacrtati logicko kolo 4-bitnog komparatora jednakosti i logicki simbol uopstenog 4-bitnog komparatora.



## 24. Sta su sabiraci?

Sabiraci – kombinatorne mreže koje se koriste pri implementaciji sabiranja.

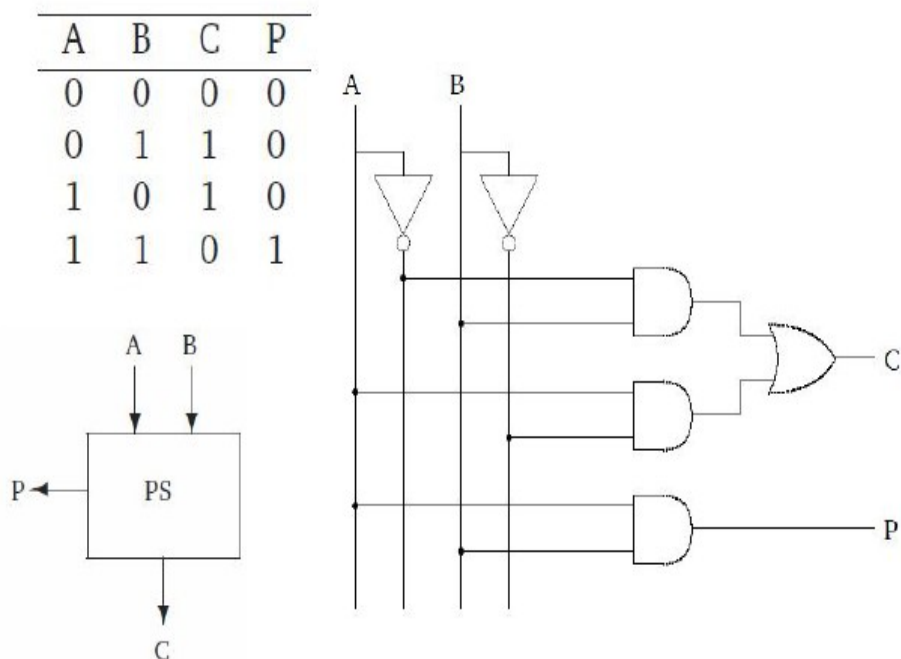
Binarni polusabirac – kombinatorna mreža koja sabira dva jednocifrena binarna broja. Ulazi su dva jednocifrena broja, a izlazi su jedan bit rezultata i jedan bit prenosa

Binarni sabirac – kombinatorna mreža koja sabira 2 jednocifrena binarna broja i dati prenos. Ulazi su 2 jednocifrena broja i 1 bit prenosa, a izlazi su 1 bit rezultata i 1 bit prenosa

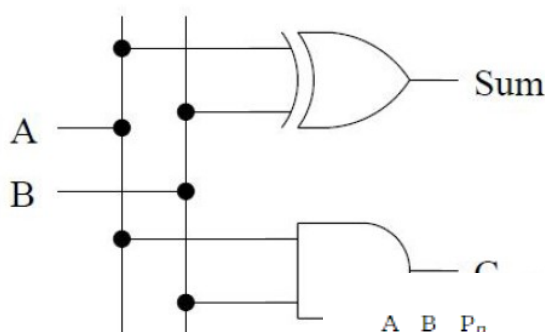
Slozeni sabirac – kombinatorna mreža koja sabira 2 visecifrena binarna broja i dati prenos.

Implementira se pomocu vise binarnih sabiraca.

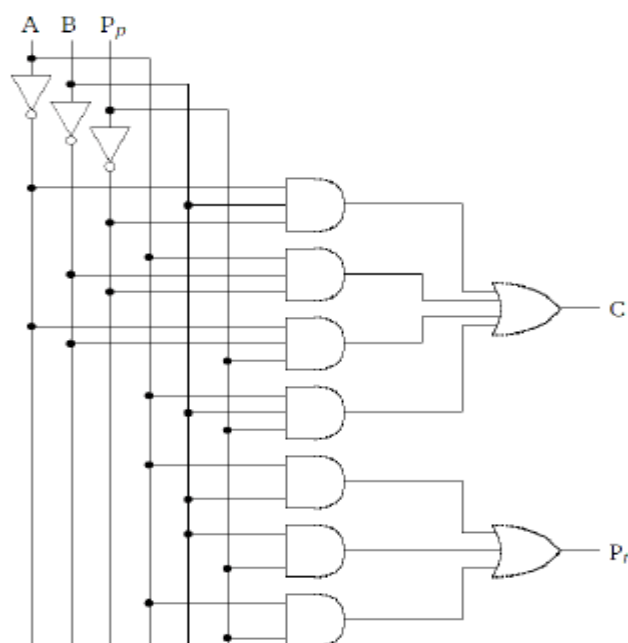
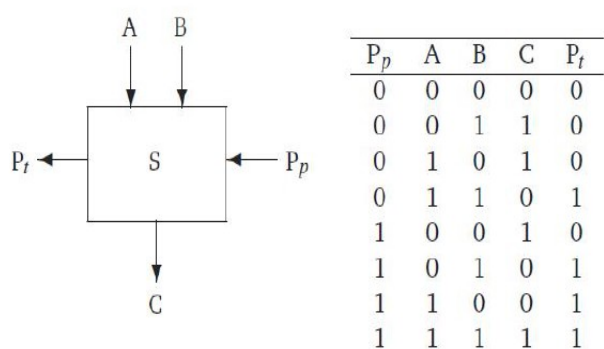
## 25. Nacrtati istinitosnu tablicu i logicko kolo binarnog polusabiraca.



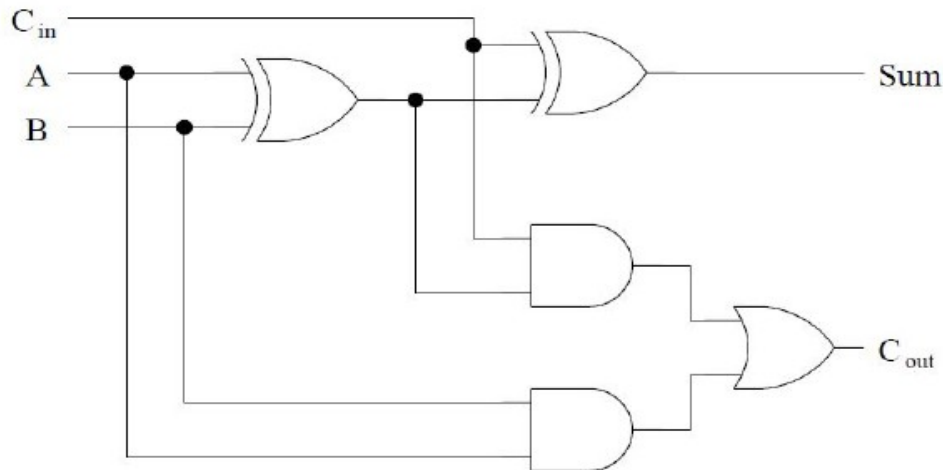
## 26. Nacrtato logicko kolo optimizovanog binarnog polusabiraca.



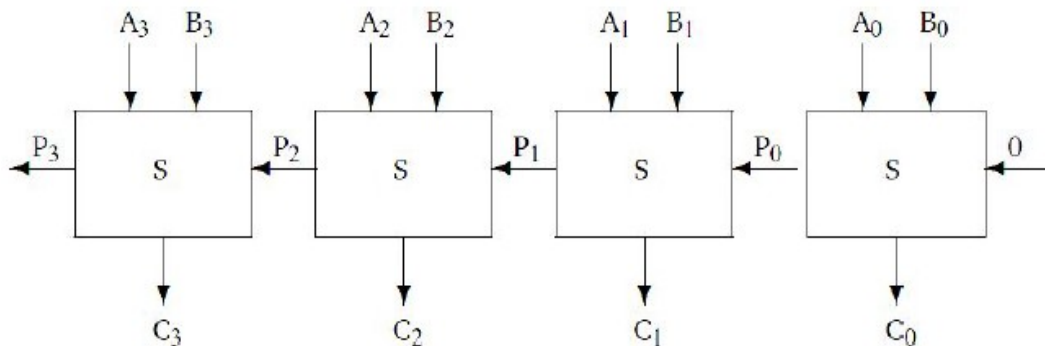
## 27. Nacrtati istinitosnu tablicu, logicki simbol i logicko kolo binarnog sabiraca.



## 28. Nacrtati logicko kolo optimizovanog binarnog sabiraca.



## 29. Nacrtati primer slozenog sabiraca.



## 30. Sta je programabilni niz logickih elemenata? Kako se implementira?

Programabilni niz logickih elemenata – cip opste namene koji relativno jednostavno moze da se prilagodi raznim specifcnim potrebama. Pociva na primeni SDNF. Sadrzi skup I, ILI i NE elemenata. Najopstija formula je da se: svaki ulaz "poveze" sa NE elementom, svaki ulaz i njegova negacija "poveze" sa I elementima, svaki izlaz iz I elemenata se "poveze" sa ILI elementima, izlazi ILI elemenata predstavljaju izlaze logickog kola.

Ako ima  $n$  ulaza i  $m$  izlaza, potrebno je: do  $2^n$  I elemenata sa po  $2n$  ulaza i  $m$  ILI elemenata sa do  $2n$  ulaza.

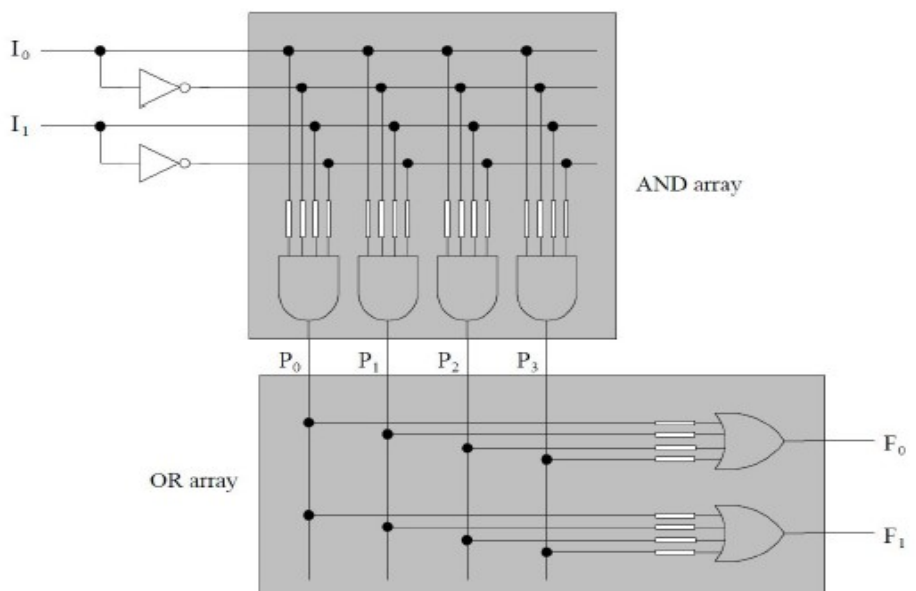
## 31. Nacrtati i objasniti primer programabilnog niza logickih elemenata.

Na svakom ulazu u I i ILI elemente postoji po prekidac.

Programiranje se postize iskljucivanjem prekidaca.

Prekidaci se po pravili iskljucuju jednokratno, uobicajeno je njihovo spaljivanje propustanjem jake el. struje.

Implementacijom se obezbedjuje da se iskljuceni prekidaci ponasaju kao aktivni ulazi za I kola i neaktivni ulazi za ILI kola.

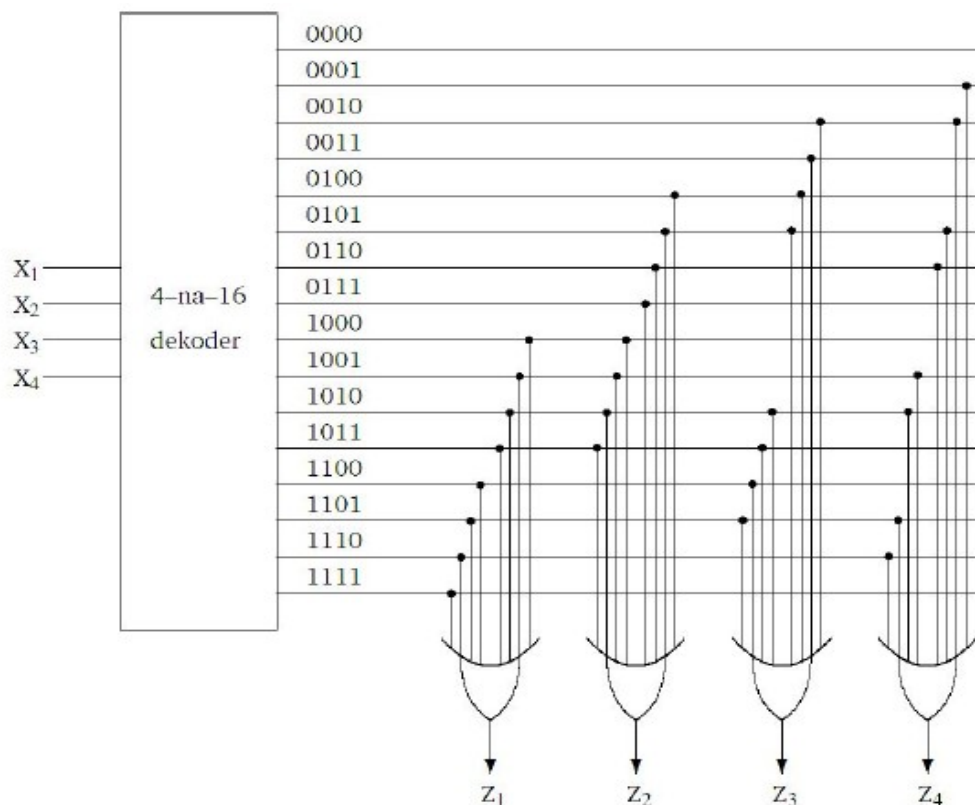




### 32. Kako se pomocu kombinatornih mreza implementira samocitajuca memorija (ROM)?

Kombinatorne mreze se nazivaju i "mreze bez memorije", zato sto rezultati zavise iskljucivo od ulaza. Ipak, mogu se upotrebljavati za implementaciju memorija koje sluze samo za citanje (ROM). Memorija samo za citanje je funkcija koja preslikava adresu, uvek ce vracati isti izlaz za odgovarajuci ulaz jer ne moze da se menja. Mozemo da je implementiramo pomocu SDNF. Ako je duzina adrese  $n$  bitova, a podatka  $m$  bitova, onda se implementira pomocu:

- dekodera  $2^n$ , koji prepoznaje adresu
- niza od  $m$  ILI elemenata, koji daju bitove podatka u zavisnosti od adrese



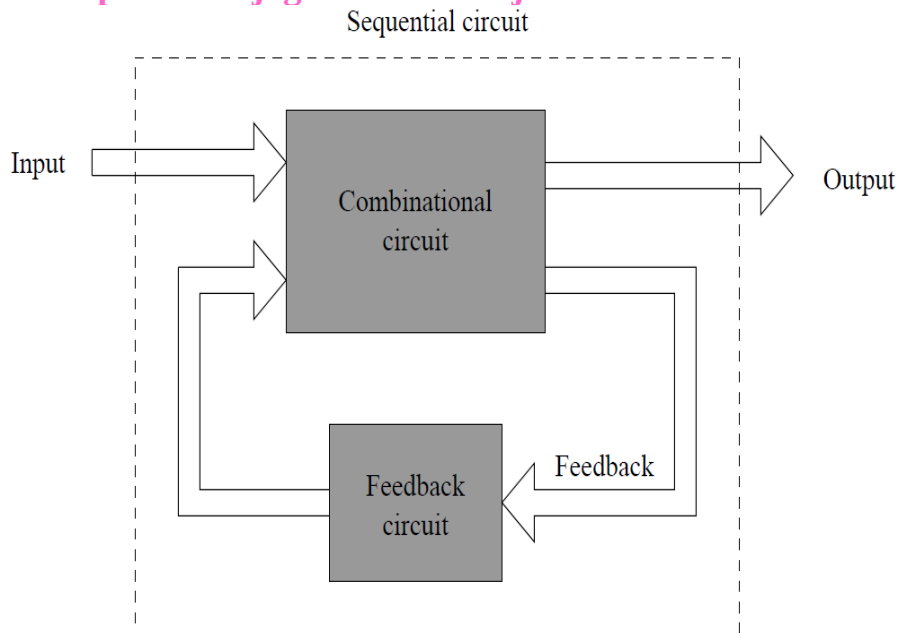
### 33. Implementirati aritmeticko logicku jedinicu pomocnu kombinatornih mreza.

Treba implementirati kolo koje u zavisnosti od ulaza  $F_0$  i  $F_1$  racuna konjukciju, disjunkciju, zbir ili razliku dva ulazna bita A i B. Rezultat se racuna pomocu multipleksora koji ima kao ulaz rezultate svih podrzanih operacija, a kao selektorske ulaze  $F_0$  i  $F_1$ . Konjukcija i disjunkcija se racunaju neposrednom primenom elemenata I i ILI. Za sabiranje se upotrebljava sabirac sa ulazima A i B, a za oduzimanje sabirac sa ulazima A i B' (dopuna do punog komplementa ce doci kao prethodni prenos). Jednostavnije je da se jedan sabirac koristi i za sabiranje i za oduzimanje. Ulaz B se negira u zavisnosti od ulaza  $F_0$ , ako je  $F_0 = 1$  onda se B negira, inace ne. Resava se pomocu XOR kola. Dopuna do potpunog komplementa se resava kao prenos.

### 34. Sta je sekvencijalna mreža?

Sekvencijalna mreža – skup povezanih logickih elemenata čiji izlaz u nekom trenutku zavisi od tekućeg stanja elemenata mreže i vrednosti ulaza u “tom istom” vremenskom trenutku.

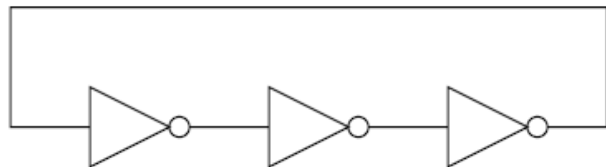
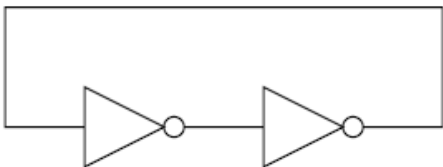
### 35. Nacrtati konceptualni dijagram sekvencijalne mreže.



### 36. Sta su stabilni i nestabilni sistemi? Nacrtati primere.

Za sistem kažemo da je stabilan ako se za nepromenljiv ulaz dobijaju nepromenljivo stanje i nepromenljiv izlaz.

Sistem nije stabilan ako se za nepromenljiv ulaz dobijaju promenljivo stanje ili promenljiv izlaz.



### 37. Objasniti razliku između sinhronog i asinhronog režima funkcionisanja.

Asinhroni režim: • digitalna kola funkcionisu nezavisno jedna od drugih

- trenutak odvijanja promena u jednom kolu ne zavisi od trenutka odvijanja promena u drugom
- nisu svi izlazi i ulazi ispravni u istom trenutku
- asinhroni rad je problematičan ako izlaz jednog kola mora da predstavlja ulaz za neko drugo kolo

Sinhroni režim: • sva kola u sistemu menjaju svoja stanja u precizno definisanim trenucima

- trenuci promena su određeni signalom časovnika
- posledica je da brzina rada zavisi od časovnika

### 38. Objasniti ulogu časovnika i elemente ciklusa.

Casovnik – signal koji predstavlja sekvencu naizmeničnih vrednosti 0 i 1. Uglavnom se signal predstavlja kao da se prelazak stanja časovnika sa 0 na 1 (i obrnuto) odvija trenutno, mada taj prelazak ima neko kratko trajanje.

Period (trenutak) menjanja stanja se naziva “rub”. Prelazak sa 0 na 1 je “izlazni rub”. Prelazak sa 1 na 0 je “silazni rub”. Osnovna uloga časovnika je globalna sinhronizacija signala u sistemu. Svaki ciklus ima 3 dela: početak ciklusa, kraj ciklusa i središnja promena signala časovnika. Druga uloga je merenje vremena u obliku broja ciklusa.

### 39. Objasniti aspekte trajanja stanja signala casovnika i tipove casovnika.

Iako je uobicajeno da trajanje svakog nepromenljivog stanja signala casovnika bude jednako (*simetrican casovnik*), moze se upotrebljavati casovnik kod koga trajanje stanja 0 i 1 nije jednako (*asimetrican casovnik*).

Ciklus casovnika - period izmedju dva uzastopna uzlazna/silazna ruba

Brzina casovnika - broj ciklusa u sekundi (Hz)

### 40. Koje su osnovne vrste sekvencijalnih mreza? Objasniti osnovna svojstva.

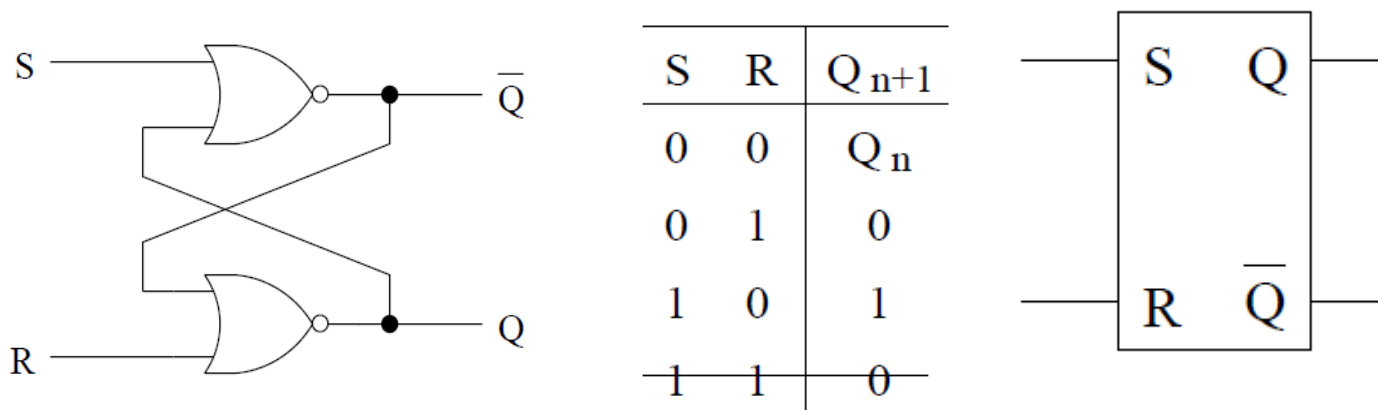
Postoje dve vrste elementarnih sekvencijalnih mreza:

1. reza – kolo koje reaguje na nivo signala, bez obzira na tip promene; cuva 1 bit stanja
2. flip-flop – kolo koje reaguje samo na promene na uzlaznom ili silaznom rubu ciklusa; cuva 1 bit stanja

### 41. Sta je SR reza? Nacrtati implementaciju, tablicu i logicki simbol.

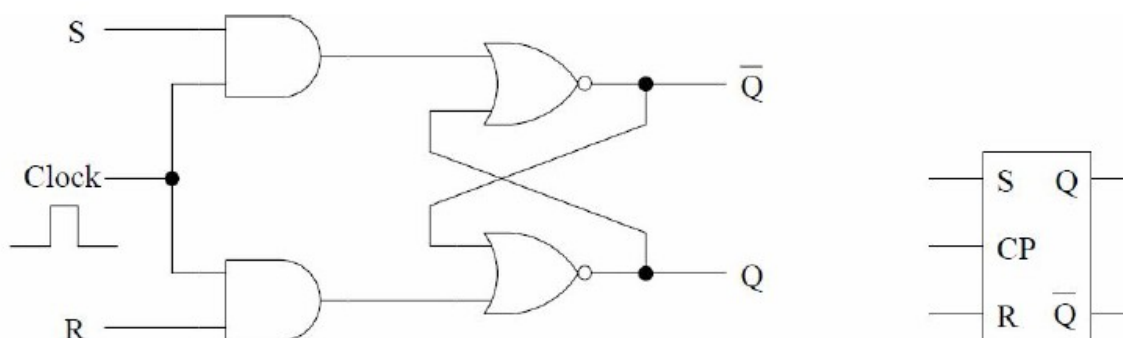
Ima dva ulaza: S i R, i dva izlaza: Q i Q'. Implementira se pomocu dva NILI elementa. Ponasanje:

- ako su oba ulaza neaktivna, cuva predhodno stanje
- ako je samo R ulaz aktivan, postavlja stanje na 0 ( $Q=0$ )
- ako je samo S ulaz aktivan, postavlja stanje na 1 ( $Q=1$ )
- izlaz SR reze se menja asinhrono u odnosu na ulaz u zavisnosti od brzine NILI elemenata



### 42. Sta je SR reza sa casovnikom? Nacrtati implementaciju.

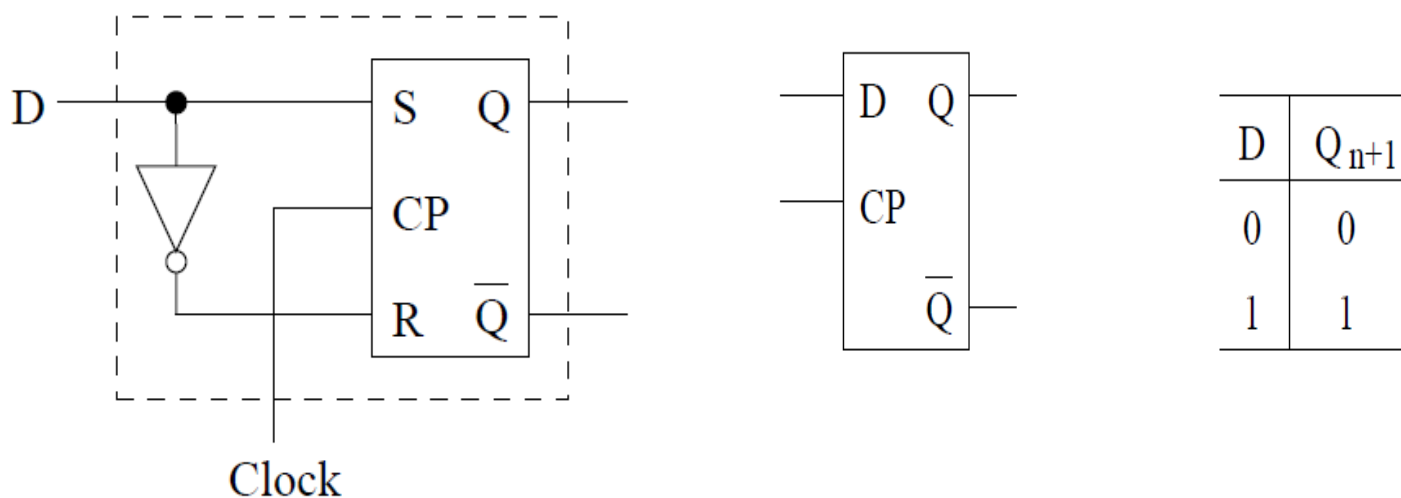
Sinhronizacija se ostvaruje dodavanjem ulaznog signala casovnika u kolo. Tako ulazni signali ne uticu na eventualnu promenu sve dok signal casovnika ne dostigne visok nivo.



#### 43. Sta je D reza? Nacrtati implementaciju, tablicu i logicki simbol.

Problem sa svim vrstama SR reza je u tome sto mora da se izbegava par vrednosti (1, 1) na ulazu. To se moze resavati primenom D reze. Zbog same implementacije nikada ne dolazi do slucaja (1, 1). Novo stanje zavisi uvek od ulaza i pozitivnog (aktivnog) stanja casovnika.

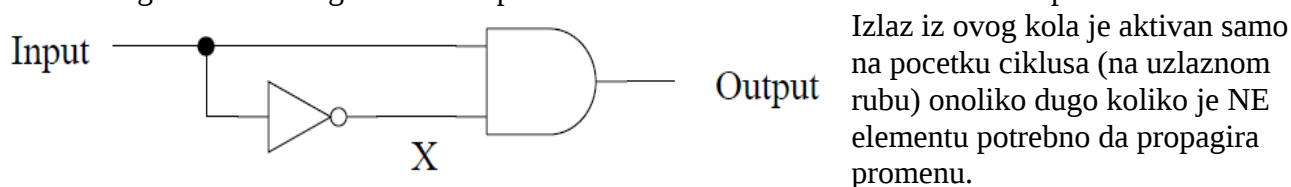
Uvek se pravi sa casovnikom, bez casovnika pravi kasnjenje i ponasa se kao SR reza. Sve dok je signal casovnika neaktivan, promene na ulazu nemaju uticaja na rezultat, tj. rezultat je isti kao i ranije. U trenutku aktiviranja signala casovnika (ili kontrolnog signala), stanje ulaza se propagira na izlaz.



#### 44. Sta je flip-flop? Kako se obezbedjuje stabilnost flip-flopa?

Flip-flop – sekvencijalna mreza kod koje se vrednosti ulaza upotrebljavaju samo na jednom rubu ciklusa casovnika (obicno na uzlaznom rubu). Time se omogucava da se u ostalim fazama ciklusa promene ulaza prakticno ignorisu i ne remete rad kola.

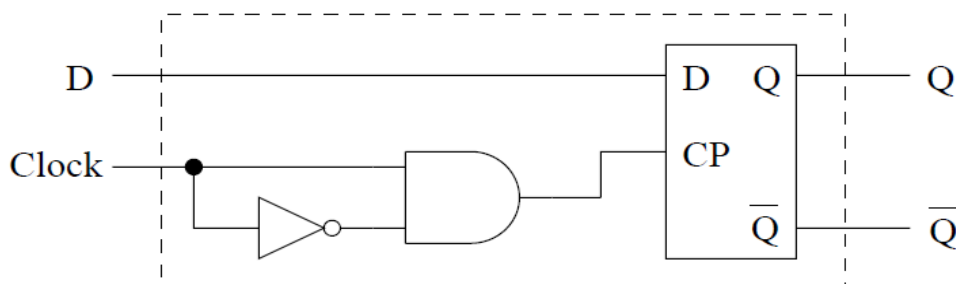
Ulazni signal se moze ograniciti na upotrebu samo na uzlaznom rubu casovnika primenom kola:



Izlaz iz ovog kola je aktivan samo na pocetku ciklusa (na uzlaznom rubu) onoliko dugo koliko je NE elementu potrebno da propagira promenu.

#### 45. Sta je D flip-flop? Nacrtati implementaciju.

D flip-flop se pravi pomocu D reze, ogranicavanjem kontrolnog signala na uzlazni rub ciklusa:



- clock=0, D=X → Q = prethodno stanje
- clock=1, D=0 → Q = 0
- clock=1, D=1 → Q = 1

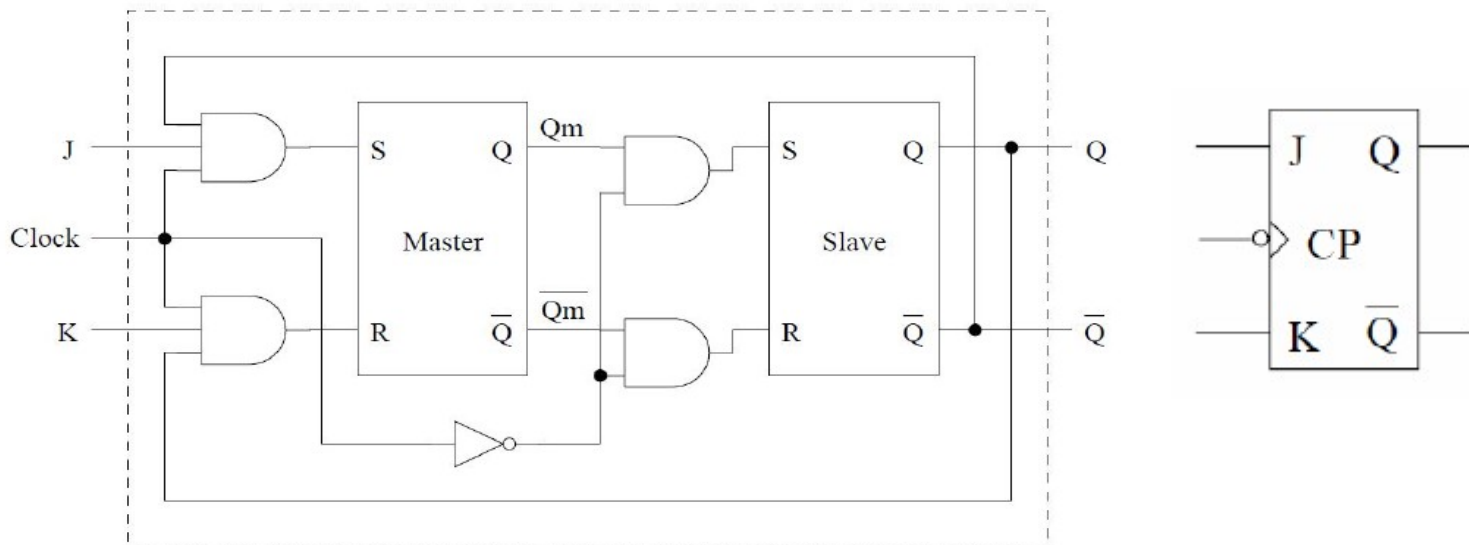
#### 46. Sta je JK flip-flop? Nacrtati implementaciju.

JK flip-flop se ponasa kao SR flip-flop, samo sto stanje 1,1 koristi za inverziju stanja. Pravi se pomocu dve SR reze (glavne i podredjene):

- glavna – aktivira se tokom aktivnog dela ciklusa
- izlaz glavne SR reze se prenosi na izlaz tokom neaktivnog dela ciklusa

Tokom trajanja aktivnog signala casovnika, glavna SR resa proizvodi izlaz koji predstavlja ili set ili reset ulaz za podredjenu SR rezu. Izlaz je stabilan zato sto na njega utice tek izlaz iz podredjene reze, koji se ne menja tokom aktivnog dela ciklusa.

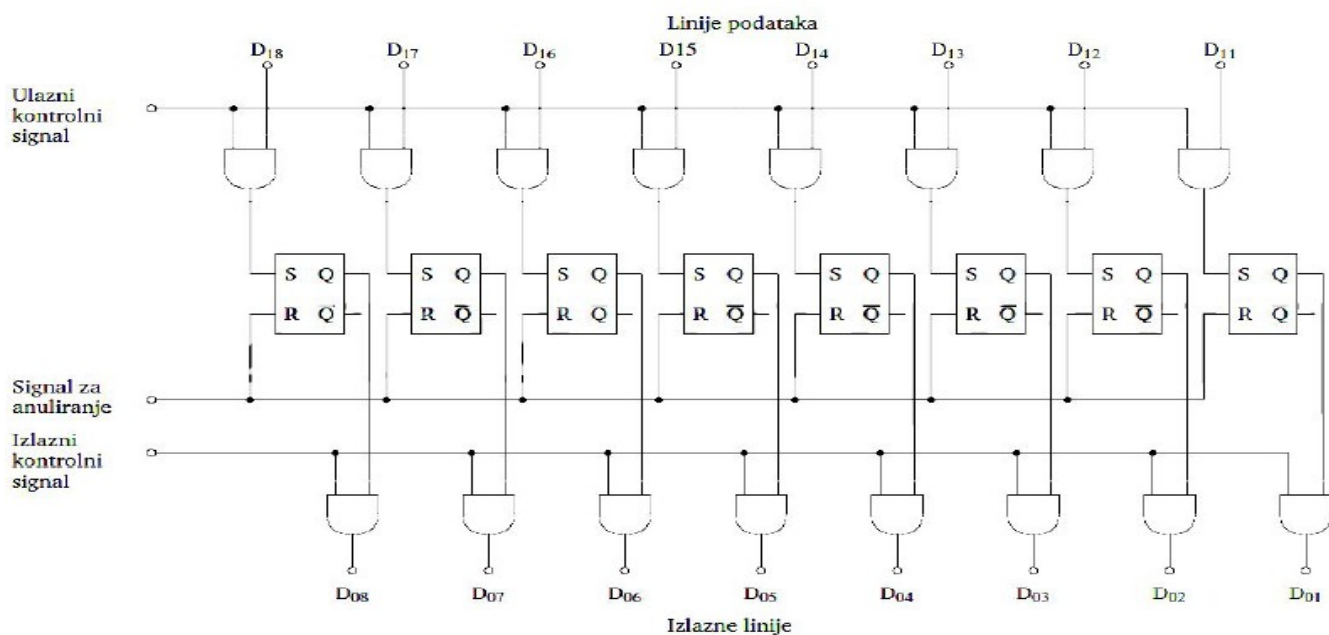
Na silaznom rubu sikhlasa se signal propagira kroz podredjenu rezu. Ne ostvaruje se odmah uticaj na glavnu rezu, zato sto ona dopusta promene tek tokom pozitivnog dela ciklusa.



#### 47. Sta je paralelni registar? Kako se implementira? Nacrtati primer.

Registri su koja se koriste za cuvanje jednog ili vise bitova podataka.

Paralelni registar – sastoji se od skupa 1-bitnih memorijskih jedinica ciji se sadrzaj moze istovremeno citati ili menjati. Moze se implementirati pomocu reza ili flip-flova.



Na slici je prikazana implementacija 8-bitnog paralelnog registra pomocu SR flip-flop elemenata. Omogucava tri osnovne operacije:

- *anuliranje* – ako je aktivan kontrolni signal za anuliranje
- *upisivanje* – ako je aktivan ulazni kontrolni signal za upisivanje; uz pretpostavku da je prethodno izvršeno anuliranje
- *citanje* – ako je aktivan izlazni kontrolni signal

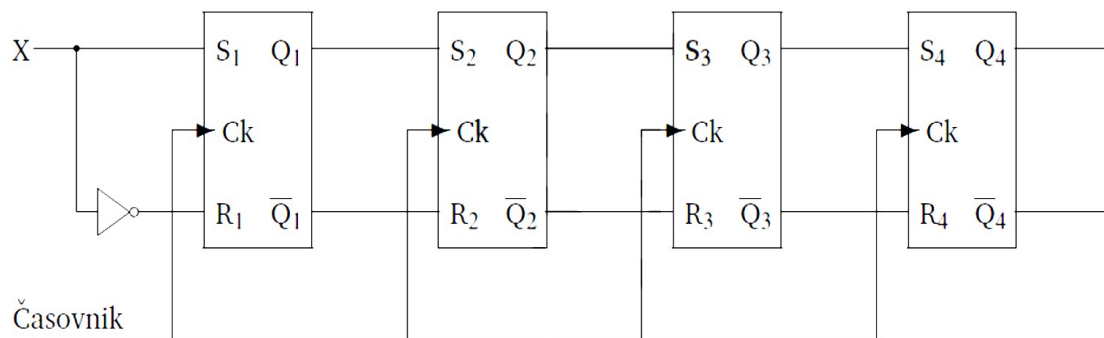
#### 48. Sta je pomeracki registar i koji su tipovi pomeranja?

Pomeracki registri – pomeraju niz bitova ulevo ili udesno sa svakim ciklusom casovnika. Mogu da se upotrebljavaju za konvertovanje iz paralelnog u serijski vid komunikacije i obrnuto.

Logicko pomeranje – ne postoji ulazni podatak. Umesto ulaza dopisuje se nula (*linijsko pomeranje*) ili vrednost “izbacenog” bita (*ciklicno pomeranje*).

Aritmeticko pomeranje – Ne postoji ulazni podatak. Umesto ulaza dopisuje se nula ako je pomeranje ulevo ili najvisi bit ako je pomeranje udesno.

#### 49. Nacrtati logicko kolo i objasniti ponasanje 4-bitnog pomerackog registra sa serijskim ulazom i izlazom.



U svakom ciklusu: sadržaj se pomera za 1 mesto udesno; sleva se dopisuje 1 novi bit sa ulaza; na izlazu se cita 1 “izbacen” bit.

#### 50. Sta je binarni brojac? Nacrtati logicko kolo i objasniti ponasanje 3-bitnog binarnog brojaca.

Brojaci – sekvencijalne mreže koje sa svakim ciklusom povećavaju vrednost registra za 1.

Binarni brojac – koristi se za brojanje registra sa B bitova i omogucava brojanje od 0 do  $2^B - 1$ .

Nakon “prekoracenja” se pocinje ponovo od 0 (brojac po modulu  $2^B$ ).

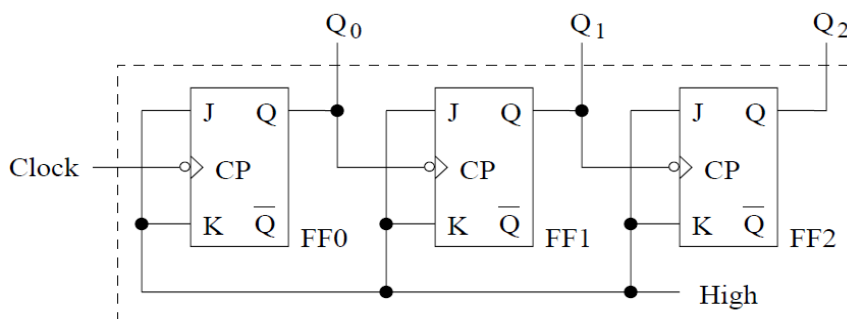
Brojaci se mogu upotrebljavati: za merenje vremena, za brojanje podataka na ulazu ili izlazu, za generisanje casovnika sa sporijim taktom itd.

Implementacija brojaca po modulu 8 (3-bitni):

- niz bitova se posmatra kao niz binarnih cifara
- najnizi bit se menja u svakom ciklusu
- visi bit je potrebno promeniti svaki put kada se prethodni nizi bit promeni iz 1 u 0

Ideja implementacije je:

- na niz JK flip-floпова se dovedu na ulaze aktivni signali (kada se ulazi na JK flip-flopu postave na 1 onda se izlaz menja u svakom ciklusu)
- na kontrolni ulaz prvog (koji odgovara najnižem bitu) se dovede signal casovnika
- na kontrolne ulaze ostalih se dovedu izlazni signali prethodnih



## 51. Sta je ISA (arhitektura)?

ISA je arhitektura skupa instrukcija. Apstrakcija arhitekture skupa instrukcija pruža programerima informacije o racunaru neophodne za razvoj programa i predstavlja interfejs izmed ju hardvera i najnižeg nivoa softvera.

## 52. Koje su osnovne komponente racunarskog sistema?

Centralna jedinica za obradu (CPU), memorijska jedinica, U/I uređaji i njihovo povezivanje

## 53. Koje su osnovne komponente procesora?

Kontrolna jedinica - čita instrukcije iz glavne memorije, dekodira ih i prepoznaje tip, upravlja radom procesora

Registri - lokalni memorijski prostor procesora, načelno su svi iste veličine

Aritmeticko-logicka jedinica (jedna ili više) - implementacija konkretnih a.l. operacija

## 54. Sta je magistrala? Koje su osnovne komponente magistrale?

Magistrala - podsistem koji povezuje osnovne komponente racunarskog sistema.

Komponente magistrale:

- adresna magistrala - prenosi podatke o adresnom prostoru, njena sirina određuje veličinu adresnog prostora
- magistrala podataka - prenosi podatke, njena veličina određuje veličinu podataka koji se prenose
- kontrolna magistrala - prenosi kontrolne signale (kodirane operacije)

## 55. Sta je sistemska magistrala? Od čega se sastoji i sta povezuje?

Sistemska magistrala - mreža koja povezuje komponente racunarskog sistema. Upotrebljavaju se i termin *interna (unutrasnja) magistrala*. Sastoji se od tri osnovne komponente - adresne magistrale, kontrolne magistrale i magistrale podataka. Nalazi se unutar procesorskog sistema i povezuje procesorske jedinice sa memorijom i ulazno/izlaznim podsistemom.

## 56. Sta je spoljasnja magistrala? Sta povezuje?

Spoljasnja magistrala - podsistem koji povezuje uređaje koji su van procesorskog sistema (USB, FireWire, serijski interfejs, paralelni interfejs).

## 57. Kako se ostvaruje deljenje magistrale?

Magistrala je deljeni resurs. Svaka komponenta povezana sa magistralom je korisnik magistrale. Pri deljenju magistrale postoji mogućnost istovremenih aktivnosti na magistrali. Istovremena upotreba magistrale od strane više komponenti dovodi do neispravnosti.

## 58. Sta je transakcija magistrale?

Transakcija magistrale - celovit niz postupaka na magistrali. Primeri aktivnosti su: čitanje iz memorije, pisanje u memoriju, čitanje sa ulaznog uređaja, pisanje na ulaznom uređaju...

Jedna transakcija može da obuhvati više operacija (npr. agresivno čitanje).

U okviru jedne transakcije prepoznaju se: glavni korisnik (zapocinje transakciju) i podređeni korisnik (odgovara na zahtev glavnog korisnika).

U jednom trenutku postoji najviše jedna transakcija na magistrali. Svaka transakcija ima tačno jednog glavnog korisnika.

Neki uređaji mogu biti samo podređeni korisnici magistrale, dok drugi mogu biti glavni ili podređeni (ali ne u isto vreme).

## 59. Navesti najvaznije vrste kontrolnih signala magistrale.

Radom magistrale se upravlja posredstvom kontrolnih signala.

- *Memory Read* , *Memory Write* – transakcija je jedna od operacija sa memorijom
- *I/O Read* , *I/O Write* – transakcija obuhvata U/I operaciju
- *Ready* - ovaj signal obavestava glavnu komponentu da je potrebno jos vremena; glavna operacija obicno reaguje prelaskom u stanje cekanja
- *Bus request* - pre svake transakcije komponente najpre moraju zahtevati da dobiju magistralu
- *Bus grant* - arbitar magistrale bira ko ce da dobije magistralu i salje mu signal
- *Clock* – signal koji služi za sinhronizaciju rada svih komponenti
- *Reset* – signal koji inicijalizuje rad sistema

## 60. Koje su osnovne karakteristike magistrale?

Sirina magistrale - odnosi se na magistralne adrese i podataka. Sira magistrala podataka podize performanse, a sira adresna magistrala povecava adresni prostor

Tip magistrale - posvecena ili multipleksirana.

Operacije magistrale - citanje, pisanje, prenos blokova, citanje sa menjanjem i prekidi

Arbitraza - moze da bude centralizovana i distribuirana

Podesavanje vremena - moze biti sinhrono i asinhrono

## 61. Sta je sirina magistrale? Koji kriterijumi uticu na odlucivanju o izboru sirine?

Sirina magistralne podataka odredjuje velicinu podataka koji se prenose magistralom. Osnovna motivacija za prosirivanje je podizanje propusnosti magistrale, a time i performansi. A motivacija za suzavanje je smanjivanje slozenosti i smanjivanje troskova.

Sirina adresne magistrale odredjuje velicinu adresnog prostora. Ako sirina magistrale ima  $n$  adresnih linija, broj adresabilnih lokacija je  $2^n$ . Jedna adresablina lokacija sadrzi jednu memorijsku rec, koja je obicno sirine 1 bajt, ali ne mora biti tako. Osnovna motivacija za prosirivanje je povecanje adresnog prostora. A motivacija za suzavanje je smanjivanje slozenosti i troskova.

## 62. Koji su tipovi magistrala? Objasniti razlike.

Posvecena magistrala - posvecena jednoj ulozi (npr. služi samo za prenosenje adresa), ima vecu propusnost, ali je teza za implementaciju

Multipleksirana magistrala - ista magistrala prenosi adrese, podatke i kontrolne signale, jednostavnija je za implementaciju, ali ima nizu propusnost. Multipleksiranjem se smanjuje efikasnost magistrale (operacije se usporavaju zbog veceg broja koraka)

## 63. Objasniti postupke citanja iz memorije i pisanja u memoriju u slucaju multipleksirane magistrale.

Citanje iz memorije:

Procesor najpre stavlja na magistralu adresu. Memorijska jedinica cita adresu i pristupa lokaciji. U medjuvremenu procesor uklanja adresu sa magistrale. Memorijska jedinica na magistralu postavlja procitan podatak.

Pisanje u memoriju:

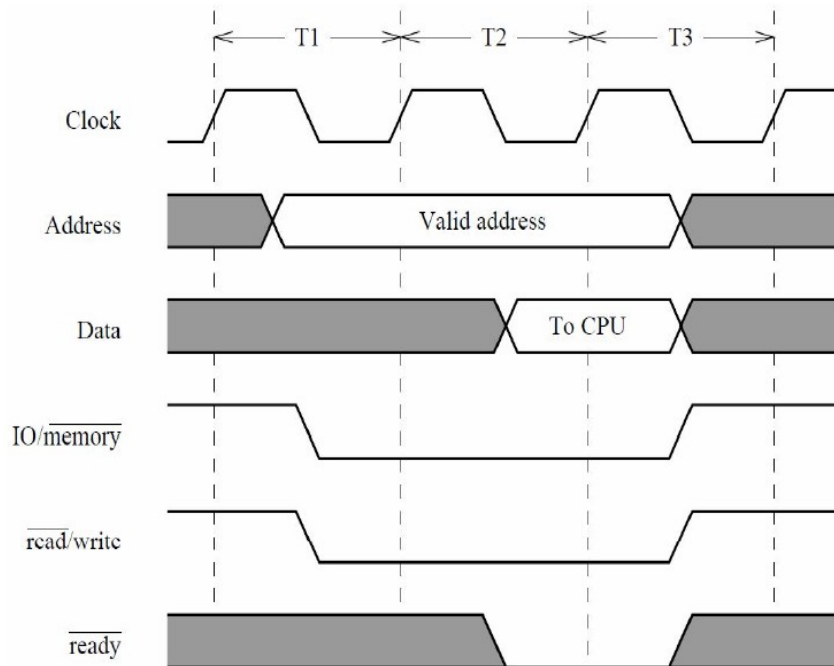
Procesor najpre stavlja adresu na magistralu. Memorijska jedinica cita adresu i pristupa lokaciji. Procesor uklanja adresu sa magistrale i postavlja podatak. Memorijska jedinica cita podatak i upisuje ga u memoriju.



#### 64. Objasniti i predstaviti vremenskim dijagramom izvršavanje operacije čitanja iz memorije na primeru sinhronizirane magistralne procesora Intel Pentium.

Kod sinhronizirane magistralne sinhronizovanje svih postupaka na magistrali obezbeđuje časovnik. Promene drugih signala se odvijaju u odnosu na ulazne i silazne rubove časovnika.

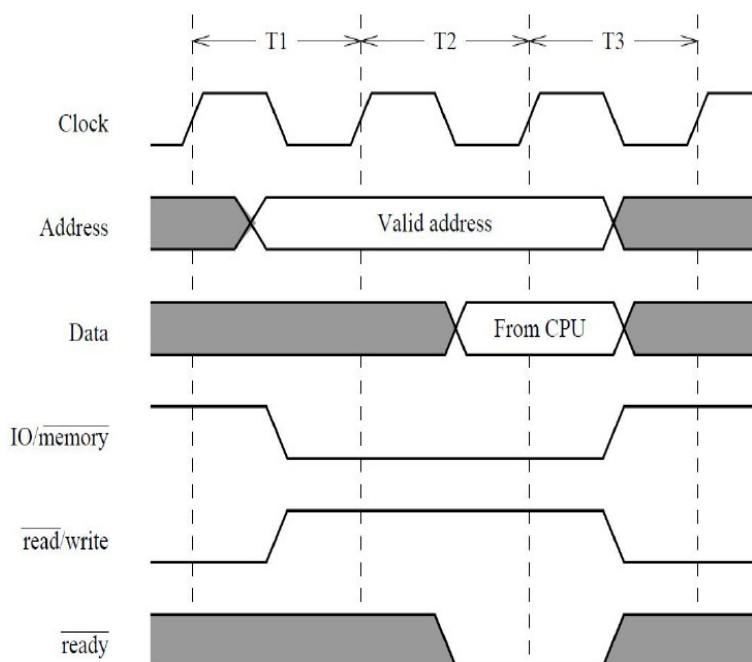
Operacija čitanja iz memorije se sastoji od tri osnovna postupka: procesor postavlja zahtev za čitanje, memorija izvršava operaciju, procesor preuzima procitane podatke



U ciklusu T1 procesor postavlja zahtev za čitanje. Tokom aktivnog stanja ciklusa T1 procesor postavlja na adresnu magistralu ispravnu adresu memorijske lokacije sa koje je potrebno citati. Nakon toga procesor postavlja dva kontrolna signala za identifikovanje vrste operacije: *signal IO/memory* se postavlja na nisko stanje, što oznacava memorijsku operaciju i *signal read/write* se postavlja na nisko stanje, što oznacava operaciju čitanja. Memorija izvršava operaciju čitanja od trenutka postavljanja kontrolnog signala. Memorija čita adresu sa adresne magistrale i postavlja na magistralu podataka procitanu vrednost. Memorija završava operaciju čitanja najranije na silaznom rubu ciklusa T2. Ako je operacija izvršena, postavlja nisko stanje *signala ready*, a ako je memorija sporija, ona oznacava da operacija jos nije završena održavanjem aktivnog stanja signala *ready*, sve dok ne postavi podatke (može da traje i više ciklusa). Počev od neaktivnog stanja ciklusa T2 procesor proverava signal *ready*. Nisko stanje oznacava da su podaci procitani i spremni za preuzimanje, aktivno znaci da memorija zahteva dodatno vreme (bar jos jedan ciklus) da bi postavila podatke na magistralu. Ako su podaci prisutni (nisko stanje signala *ready*) procesor čita podatke sa magistralne podataka, sklanja adresu sa adresne magistrale, deaktivira signale *IO/memory* i *read/write*. Operacija je završena najranije na silaznom rubu ciklusa T3.

## 65. Objasniti i predstaviti vremenskim dijagramom izvršavanje operacije pisanja iz memorije na primeru sinhronne magistrale procesora Intel Pentium.

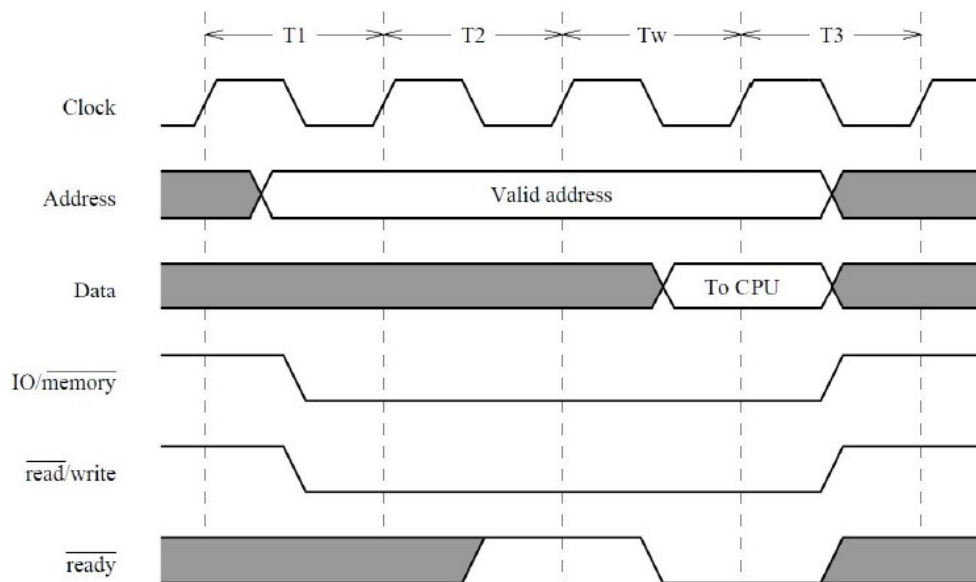
Procesor postavlja zahtev za pisanje, memorija izvršava operaciju pisanja, po potvrđenom pisanju procesor nastavlja rad



U ciklusu T1 procesor postavlja zahtev za pisanje. Tokom aktivnog stanja ciklusa T1 procesor postavlja na adresnu magistralu ispravnu adresu memorijske lokacije na koju je potrebno pisati. Nakon toga procesor postavlja dva kontrolna signala za identifikovanje vrste operacije: *signal IO/memory* se postavlja na nisko stanje, što označava memorijsku operaciju i *signal read/write* se postavlja na aktivnoo stanje, što označava operaciju pisanja. Kasnije, tokom ciklusa T2, procesor postavlja podatke na magistralu podataka. Memorija izvršava operaciju pisanja pocev od silaznog ruba ciklusa T1. Memorija cita adresu sa adresne magistrale i obavlja pripreme za upisivanje. Zatim cita podatke sa magistrale podataka i upisuje ih na odgovarajucoj lokaciji. Memorija oslobadja magistralu najranije na silaznom rubu ciklusa T2. Ako je operacija pisanja izvršena (ili ce biti izvršena tokom ciklusa), postavlja se nisko stanje signala ready, a ako je memorija sporija, ona označava da operacija jos nije izvršena održavanjem aktivnog stanja signala ready sve dok ne postavi procitane podatke (potencijano i vise ciklusa). Procesor prima potvrdu o upisivanju. Pocev od neaktivnog stanja ciklusa T2 (odmah po postavljanju podataka) procesor proverava signal ready. Nisko stanje označava da ce memorija tokom ciklusa završiti upisivanje, dok aktivno stanje označava da memorija zahteva dodatno vreme (bar jos jedna ciklus) da bi upisala podatke. Ako je operacija izvršena (nisko stanje signala ready) procesor sklanja adresu sa adresne magistrale i podatke sa magistrale podataka, deaktivira signale IO/memory i read/write. Operacija je završena najranije na silaznom rubu ciklusa T3.

## 66. Sta je stanje cekanja? Kada se i kako upotrebljava? Objasniti operaciju citanja sa stanjem cekanja.

Rad procesora je cesto suviše brz da bi ga memorija ili U/I uredaji mogli pratiti. Zbog toga se uvode stanja cekanja i kontrolni *signal ready*. Ako je stanje signala ready aktivno, procesor ne sme da pretpostavi da je uredjaj izvršio operaciju. U tom slucaju procesor ceka jos jedan ciklus i ponovo proverava stanje signala ready.



### 67. Sta je “prenosenje blokova podataka”? Kako se i zasto upotrebljava?

Prenosenje blokova podataka podrazumeva da se jednom slozenom operacijom prenosi veca kolicina podataka. Koristi se za popunjavanje kes memorije.

### 68. Kako se sinhronizuje rad na asinhronoj magistrali? Objasniti signale i tok aktivnosti (cetvorofazno rukovanje).

U slucaju asinhronne magistrale ne upotrebljava se casovnik za sinhronizaciju vec se upotrebljavaju operacije rukovanja i dodatni sinhronizacioni signali. Za asinhronne magistrale je uobicajeno *cetvorofazno rukovanje*. Dodatni signali su *glavna sinhronizacija* (MSYN) i *podredjena sinhronizacija* (SSYN).

Cetvorofazno rukovanje obuhvata sledece faze:

- glavni uredjaj inicijalizuje komunikaciju tako sto postavlja sve potrebne podatke na magistralu i postavlja glavni sinhronizacioni signal (MSYN)
- podredjeni uredjaj reaguje na MSYN, cita upucene podatke, obavlja operaciju, postavlja odgovor na magistralu, postavlja podredjeni signal (SSYN)
- glavni uredjaj reaguje na SSYN, cita odgovor sa magistrale, iskljucuje signal MSYN
- podredjeni uredjaj reaguje na iskljucivanje MSYN i iskljucuje signal SSYN

### 69. Objasniti rad asinhronne magistrale na primeru operacije citanja.

Prvo procesor postavlja adresu, kontrolne signale i signal MSYN. Zatim memorija cita upucene podatke, obavlja operaciju, postavlja podatke na magistralu i postavlja signal SSYN. Procesor cita odgovor sa magistrale, ponistava adresu magistralu i kontrolne signale i iskljucuje signal MSYN. Memorija ponistava magistralu podataka i iskljucuje signal SSYN.

### 70. Objasniti razlike izmedju sinhronne i asinhronne magistrale.

Sinhrona magistrala je jednostavnija od asinhronne. Sinhroni rad zahteva da svi uredjaji rade u skladu sa casovnikom. Ako neki rade brze moraju da sacekaju da se završi ciklus.

Asinhrona je fleksibilnija u pogledu trajanja operacija. Operacije ne moraju da traju ceo ciklus. Fleksibilnije su u pogledu uredjaja, brzina rada se prilagođava brzini uredjaja. Slozenije su za implementaciju. Asinhronne su pogodnije za uredjaje cije su brzine razlicite.

## 71. Sta je arbitraza magistrale? Koje su osnovne vrste arbitraze?

Magistrale koje mogu imati vise potencijalnih glavnih uredjaja moraju imati mehanizam arbitraze. Taj mehanizam služi za dodeljivanje magistrale glavnom uredjaju. Na sistemskoj magistrali glavni uredjaj je najcesce procesor, ali to moze biti i kontroler DMA. Vrste:

- staticka arbitraza - raspodela medju glavnim uredjajima odigrava se na unapred odredjen nacin
- dinamicka arbitraza - odlucuje na osnovu zahteva pristiglih od uredjaja. Vecina implementacija pociva na dinamickoj arbitrazi

## 72. Nabrojati i objasniti politike dodeljivanja magistrale.

- Politike fiksnih prioriteta - svakom glavnom uredjaju se dodeli fiksni prioritet. Kada vise glavnih uredjaja zahteva magistralu, dobija je onaj sa najvisim prioritetom. Veoma je vazno da se prioriteti pazljivo odrede, u suprotnom uredjaj sa visim prioritetom moze vecito da uzima magistralu od drugih uredjaja (tzv. izgladnjivanje). Ova politika se obicno upotrebljava za U/I uredjaje i za usluge DMA.

- Politike rotirajucih prioriteta - prioriteti predstavljaju funkciju vremena cekanja na magistralu. Sto uredjaj duze ceka, to mu je veci prioritet. Ovakvom politikom se izbegava izgladnjivanje. Podvarijanta ove politike je da se uredjaju koji je upravo dobio magistralu spusti prioritet. Ako se pri tome spusti na najnizi, dobija se raspodela prioriteta u krug (*round robin*).

- Ravnopravne politike – ravnopravnost je vazan kriterijum dodeljivanja (sprejava izgladnjivanje). Ravnopravne politike ne moraju da upotrebljavaju prioritete. Ravnopravnost se moze definisati na vise nacina. Svi zahtevi u predefinisanoj vremenu moraju biti zadovoljeni pre odobravanja zahteva u narednom prozoru. Ili, zahtev ne sme da ceka duze od nekog odredjenog broja sekundi.

- Hibridne politike - zasnivaju se na kombinovanoj upotrebi prioriteta i pravila ravnopravnosti. Nazivaju se i kombinovane politike. Mozemo da podelimo uredjaje u klase i unutar klase primenjujemo jednu politiku, a izmedju klasa neku drugu.

## 73. Navesti i ukratko objasniti politike oslobadjanja magistrale.

Odnose se na uslove pod kojima trenutni glavni uredjaj oslobadja magistralu za druge uredjaje. Deli se na:

- Politike bez planiranja - glavni uredjaj, koji upotrebljava magistralu, oslobadja magistralu dobrovoljno. Deli se na:

- politike zasnovane na transakcijama
- politike zasnovane na zahtevima

- Politike sa planiranjem - omogucavaju da odredjena transakcija bude prinudno prekinuta u odredjenim slucajevima.

## 74. Objasniti moguće načine organizacije arbitraze magistrale.

Arbitraza se implementira centralizovano i distribuirano.

U slucaju centralizovane implementacije, jedan centralni arbitar prima zahteve od svih glavnih uredjaja i na osnovu politike dodeljivanja dodeljuje magistralu jednom. Po zavrsetku transakcije, glavni uredjaj oslobadja magistralu u skladu sa politikom oslobadjanja.

U slucaju distribuirane implementacije, hardver za arbitrazu je distribuiran po glavnim uredjajima. Distribuiran algoritam se upotrebljava za odredjivanje glavnog uredjaja kome ce se dodeliti magistrala.

## 75. Objasniti mehanizam ulancavanja kod centralizovane arbitraze.

U lancavanje koristi jednu liniju za zahteve, koju dele svi glavni uredjaji. Kada centralni arbitar primi zahtev, on salje odobrenje za upotrebu magistrale prvom glavnom uredjaju u lancu. Uredjaj u lancu prosledjuje signal ako nije zahtevao magistralu, a ne prosledjuje ga ako jeste. Tako se signal za odobravanje prosledjuje niz lanac sve dok ne dodje do nekog od uredjaja koji su zahtevali magistralu. Prvi takav uredjaj u lancu je dobija.

Prednosti: Jednostavna implementacija, zahteva samo tri kontrolne linije po uredjaju. Arbitar ne ogranicava broj uredjaja niti njegova implementacija zavisi od broja uredjaja.

Nedostaci: Implementira politiku fiksnih prioriteta. Trajanje arbitraze je proporcijalno broju glavnih uredjaja. Shema nije otporna na otkazivanje, ako neki uredjaj otkaze, ni jedan uredjaj nizeg prioriteta ne moze da dobije magistralu.

## 76. Objasniti mehanizam nezavisnih zahteva kod centralizovane arbitraze.

Arbitar se povezuje sa svakim uredjajem putem posebnih linija za zahteve i odobravanje. Kada glavni uredjaj zahteva magistralu, salje zahtev putem svoje linije zahteva. Kada arbitar primi zahtev, na osnovu politike dodeljivanja odredjuje koji uredjaj ce dobiti magistralu.

Prednosti: Mogu se implementirati razlicite politike dodeljivanja magistrale. Kratko (konstantno) vreme dodeljivanja, nezavisno od broja uredjaja.

Nedostaci: Slozenija implementacija. Broj uredjaja je ogranicen brojem linija.

---

## 77. Sta je memorija?

Memorija - uređaj koji omogućava čuvanje (zapisivanje) i čitanje podataka u računaru. Memorija računara se deli na unutrašnju i spoljašnju memoriju.

## 78. Kojim se osobinama opisuju memorije?

Karakteristika memorije određuje način i mesto njene upotrebe u računarskim sistemima. Osim toga, različiti kriterijumi podele memorije se uglavnom zasnivaju na nekoj od karakteristika memorije. Značajne karakteristike su:

trajanje zapisa, tip nosioca, kapacitet, jedinica prenosa, adresibilnost, cena, mogući načini pristupa, performanse, mogućnost promene sadržaja

## 79. Objasniti trajanje zapisa memorije.

Prema trajanju zapisa informacije koje čuvaju, memorije se mogu podeliti na memorije sa stalnim zapisom i memorije sa privremenim zapisom.

Memorije sa privremenim zapisom gube sadržaj po prestanku električnog napajanja.

Memorije sa stalnim zapisom čuvaju sadržaj sve dok ne dodje do njegove namerne promene. Obično ne zahtevaju električno napajanje za čuvanje informacija.

## 80. Objasniti tip nosioca zapisa memorije.

Memorije se mogu podeliti na osnovu tehnologije koja se koristi za zapis informacije. Danas su najviše u upotrebi poluprovodničke memorije, memorije sa magnetnom površinom (diskovi, trake), optičke memorije (CD-ROM, CD-R, CD-RW, DVD-ROM).

## 81. Objasniti kapacitet memorije.

Kapacitet predstavlja količinu informacija koju memorija može da sadrži. Kapacitet *interne memorije* se obično izražava u bajtovima ili u recima. Dužina reci zavisi od tipa procesora; najčešće dužine reci su 8, 16, 32, 64, 128 bitova. Kapacitet *spoljašnje memorije* se izražava u bajtovima (KiB, MiB, GiB, TiB).

## 82. Objasniti adresibilnost memorije.

Adresibilnost predstavlja svojstvo memorije da joj se može pristupiti pomoću adrese. Memorije mogu biti:

- adresibilne - ako se može adresirati svaka pojedinačna memorijska lokacija (reci)
- poluadresibilne - ako se pomoću adrese može pristupiti grupi bajtova (većoj od reci)
- neadresibilne - ako se posredstvom adrese ne može pristupiti sadržaju memorije

## 83. Navesti moguće načine pristupa memoriji.

- Sekvencijalan pristup - podaci su organizovani u jedinice (slogove) koji se međusobno razdvajaju kontrolnim informacijama koje se koriste pri čitanju podataka; piše se redom i čita se redom, kako je vršeno pisanje; da bi se izvršilo pozicioniranje na željeni slog moraju se pročitati svi slogovi koji mu prethode, zbog toga je vreme pristupa proizvoljnom slogu relativno veliko (primer: magnetna traka)
- Neposredan pristup - postoji zavisnost adrese sloga i njegove fizičke lokacije (ne mora da bude puna); na osnovu adrese se pristupa neposredno slogu ili njegovoj okolini; vreme pristupa nije fiksno (primer: magnetni disk)
- Proizvoljan pristup - svaka adresibilna lokacija ima jedinstven mehanizam pristupa podacima; vreme pristupa je fiksno (primer: glavna memorija računara)
- Asocijativni pristup - podvrsta memorije sa proizvoljnim pristupom; podatku se pristupa na osnovu nekog uzorka adrese ili podatka (primer: keš memorija)

#### 84. Objasniti hijerarhiju memorije.

Sto je krace vreme pristupa, cena je veca. Sto je veci kapacitet, vreme pristupa je duze. Sto je veci kapacitet, cena po bitu je niza. Nove tehnologije donose nizu cenu po bitu uz ocuvanje prethodnih odnosa.

#### 85. Sta je ROM? Kakve vrste postoje?

Memorija samo za citanje (ROM) predstavlja oblik memorije ciji je sadrzaj stalan i ne moze da se menja. ROM se koristi za mikroprogramiranje i cuvanje sistemskih programa. Ne zahteva napajanje za odrzavanje sadrzaja. Moze cuvati sadrzaj dok je racunar iskljucen. Obicno se upotrebljava za podizanje racunarskog sistema (boot).

Vrste:

- *fabricki programiran* - pravi se u slucaju masovne potrebe
- *programabilan (PROM)*
- *visokratno programabilan (erasable programmable ROM - EPROM)*
- *visokratno programabilan sa el. brisanjem (EEPROM)*

#### 86. Sta je RAM? Kakve vrste postoje?

Od svih vrsta poluprovodnicke memorije najcesce se koristi RAM (eng. Random Access Memory). Sadrzaj ove memorije moze i da se cita i da se upisuje proizvoljan broj puta. I upis i citanje podataka se vrši pomocu elektricnih signala. RAM je nestalna memorija, tj. gubi svoj sadrzaj po prestanku elektricnog napajanja.

Vrste:

- *staticki (SRAM)* – implementira se pomocu reze ili flip-flopa. Ne zahteva osvezavanje da bi cuvao sadrzaj. Prednosti: jednostavnost upotrebe i brzina. Koristi se za kes memorije.
- *dinamicki (DRAM)* - pravi se od celija koje cuvaju vrednosti kao naboje u kondenzatorima. Prisustvo, odnosno odsustvo elektricnih naboja implementira se kao 1, odnosno 0. Kako kondenzatori imaju prirodnu tendenciju praznjenja, dinamicki RAM zahteva periodicno osvezavanje naboja da bi zadrzao neizmenjen sadrzaj. Citanje narušava sadrzaj, neophodno je pisanje posle citanja. Prednosti: niza cena, manje zagrevanje, veca gustina pakovanja. Upotrebljava se za radnu memoriju racunara.

#### 87. Objasniti tehnologije izrade dinamicke RAM memorije.

DRAM se pravi kao matrica bitova. Svakom bitu moze da se pristupi preko adrese vrste i kolone u kojima se nalazi. Ove adrese generise kontroler memorije koji se nalazi u racunarskom sistemu. DRAM se deli na asinhroni i sinhroni DRAM.

U klasicnoj DRAM organizaciji sa asinhronim interfejsom, odredjuje se minimalni period vremena koji mora da protekne da bi bili sigurni da ce neka operacija biti završena. Ukoliko se novi takt casovnika dogodi pre isteka ovog vremenskog intervala, procesor mora da saceka najmanje jos jedan takt casovnika pre pocetka nove operacije. Ako casovnik radi brze, uvode se stanja cekanja.

Sinhroni DRAM radi pod kontrolom sistemskog casovnika. Adresa i informacije o kontrolnim linijama se predaju DRAM-u i do završetka operacije, koja moze da potraje i nekoliko ciklusa casovnika, procesor moze da obavlja druge poslove. Zna se broj ciklusa za svaku od operacija. Brzina se meri brzinom casovnika i brojem ciklusa.

#### 88. Sta je SDRAM?

To je memorija koja sadrzi SRAM kes na DRAM cipu. SDRAM moze da se koristi ili kao kes ili kao bafer sto znatno ubrzava rad sa takvom memorijom.



## 89. Sta je DDR SDRAM?

DDR SDRAM (Double Data Rate SDRAM) - memorija koja omogućuje dvostruko veću brzinu rada memorije u odnosu na klasični JEDEC. Povećanje brzine je omogućeno dozvolom da se aktivira izlazna operacija na cipu na granici početnog dela ciklusa časovnika. Podaci se isporučuju po dva puta u ciklusu: na uzlaznom rubu, na silaznom rubu.

## 90. Objasniti bafer sa tri stanja.

Uredjaji sa tri stanja imaju 3 a ne samo 2 stanja (za razliku od ostalih reza i flip-flopova). Imaju dodatni kontrolni signal. Ako je on aktivan, izlaz je sa visokom impedancijom (aktivan) nezavisno od ulaza.

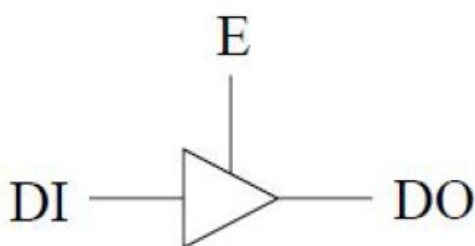
Bafer je aktivan element. Ako je na ulazu *pozitivan potencijal*, on obezbedjuje da je na izlazu *pun pozitivan potencijal*. Ako je na ulazu *priblizno nulti potencijal*, on obezbedjuje da je na izlazu *nulti potencijal*. Ponasa se kao pojacavac signala. Svaki potencijal *iznad* praga funkcionisanja tranzistora pojacava se do punog intenziteta pozitivnog potencijala. Svaki potencijal *ispod* praga funkcionisanja tranzistora "pojacava" se do nultog potencijala.



Upotreba bafera obezbedjuje da elementi D1-D4 dobiju pun potreban potencijal . Tri stanja:

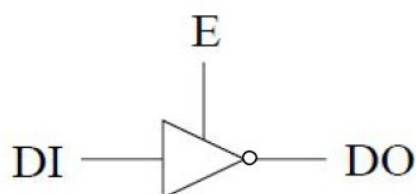
- 0 - na izlazu se postavlja nulti potencijal (uzemljenje) i omogućava protok struje
- 1 - na izlazu se postavlja pozitivan potencijal (napajanje) i omogućava protok struje
- Z - ne utice se na stanje na izlazu i onemogućava se protok struje

Bafer sa tri stanja se ponasa poput ventila. Ako se na "ventil" E koji dopusta protok dovede 0, onda se ne utice na stanje na izlazu. Ako se na "ventil" E koji dopusta protok dovede 1, onda se signal sa ulaza X propagira na izlaz.



Inputs		Output
E	DI	DO
1	0	0
1	1	1
0	X	Z

Invertor sa tri stanja se ponasa kao negacija sa ventilom. Ako se na "ventil" E koji dopusta protok dovede 0, onda se ne utice na stanje na izlazu. Ako se na "ventil" E koji dopusta protok dovede 1, onda se signal sa ulaza X invertuje i propagira na izlaz.



Inputs		Output
E	DI	DO
1	0	1
1	1	0
0	X	Z



## 91. Kako se implementira RAM od flip-flopa uz upotrebu bafera sa 3 stanja?

Koristi se kao registar sirine 8 bita. Interno koristi 8 D flip-flopa.

Izlaze flip flopa salje na izlaze cipa kroz invertujuce bafere sa tri stanja (tj. invertore sa tri stanja). Kontrolni signal OE# (output enable) kontrolise izlaze. Ako je OE=0 (OE#=1), propusta izlaze invertora na izlaze cipa.

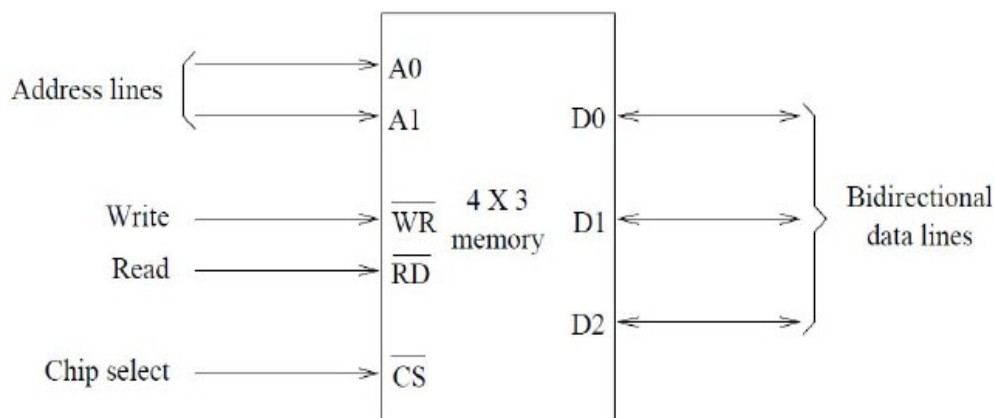
Kontrolni signal LE kontrolise pisanje (menjanje stanja). Ako je LE=1 (LE#=0), omogucava pisanje.

Zbog upotrebe reze, izlaz je jednak ulazu sve dok je LE=1 i OE=0.

## 92. Nacrtati i objasniti blok dijagram memorije 4x3 koji koristi bafer sa tri stanja.

Dodajemo selektorski ulaz, ulazni signal koji odredjuje da li se memorijski blok upotrebljava ili ne. Povezuje se kao ulaz na konjukcije za izbor adrese. Spajamo ulazne i izlazne signale podataka. Ako se pise, onda pomocu bafera sa tri stanja usmeravamo podatke sa magistrale na ulaze flip-flopa. Ako se cita, onda pomocu bafera sa tri stanja usmeravamo izlaze iz flip-flopa na magistralu. Potreban je dodatni kontrolni signal koji oznacava operaciju citanja. Selektorski ulaz ukljucuje/iskljucuje kontrolne signale za citanje i pisanje. Baferima sa tri stanja se:

- signal sa magistrale podataka propusta do ulaza na flip-flobove, akko su aktivni i selektorski signal i kontrolni signal operacije citanja
- signal sa izlaza flip-flopa se propusta na magistralu podataka, akko su aktivni i selektorski signal i kontrolni signal pisanja
- iskljuceni bafer (kontrolni signal 0) ima visoku impedancu i ne predstavlja smetnju funkcionisanju ukljucenih bafera sa istim izlazom/ulazom



## 93. Kako se od memorijskih blokova prave vece memorije? Pr.: 2x16 od 1x8.

Prvi korak je pravljenje nezavisne memorijske jedinice koja nije cvrsto vezana za specificne adrese u adresnom prostoru. Drugi korak je vezivanje ovakvih nezavisnih memorijskih jedinica za konkretan adresni prostor. Pomocu 4 cipa dimenzija 1x8 moze se napraviti memorijski blok 2x16. Jedan cip moze da cuva 8 bita, pa se koristi matrica 2x2 cipa. Povezivanjem vise cipova povecava se sirina memorijske reci (horizontalna ekspanzija). Kontrolni ulazi dva cipa se vezuju zajedno kako bi predstavljali 16-bitnu celinu. Ulaze i izlaze svakog cipa vezujemo na odgovarajuce linije magistralne podataka. Dodavanjem redova povecavamo velicinu memorije (vertikalna ekspanzija). Svaki red cuva po jednu rec. Pomocu dekodera se vrši odabir aktivnog reda. Na kontrolne signale *izlaza* cipova se vezuje konjukcija (kontrolnog selektora, kontrolnog signala citanja, izlaza dekodera). Zbog invertovanog ulaza umesto konjukcije se primenjuje disjunkcija invertovanih ulaza. Na kontrolne signale *ulaza* cipova se vezuje konjukcija (kontrolnog selektora, kontrolnog signala pisanja, izlaza dekodera).

#### 94. Dati primere manjih memorijskih cipva i njihovu organizaciju u izgradnji vecih memorija. Objasniti na primeru memorije 256MB (64Mx32b).

SRAM: • 8mb cip u tri konfiguracije (512Kx18, 256Kx32, 256Kx36)

- dodatni bitovi sluze za prepoznavanje i otklanjanje gresaka
- vreme pristupa 3.5ns
- cip 512Kx18 ima 19 adresnih linija
- cipovi 256K imaju po 18 adresnih linija

DRAM: • 256Mb cip u tri konfiguracije (64Mx4 – 26 adr. lin., 32Mx8 – 25 a.l., 16Mx16 – 24 a.l.)

- trajanje ciklusa je oko 7ns

Osnovno pitanje je da li je memorijski adresni prostor adresibilan na nivou pojedinacnih bajtova ili ne. Zatim se odlucuje o konfiguraciji cipova. Ako je cilj memorija MxN, koriste se cipovi DxW (broj kolona je N/W, broj redova je M/D, broj cipova je (MxN)/(DxW)).

Za memoriju od 256Mib ciljna konfiguracija je 64Mix32b, koriste se cipovi 16Mix16b, a matrica cipova je 4x2. Vezivanje na magistralu podataka je neposredno, svakom cipu odgovara deo linija podataka. Ako se u jednom koraku cita N bitova, Z najnižih bitova adrese se ignorisu ( $Z = \log_2(N/8)$ ). 256M se adresira sa 28 bitova adrese; jedan cip ima 24 bita adrese (1 cip 16Mix16b); najniza 2 bita adrese se ne koriste; na cipeve se vezuju 24 bita; bitovi adrese  $A_{26}$  i  $A_{27}$  se koriste za biranje reda cipova. Kontrolni signali za citanje i pisanje svih cipova se povezuju medjusobno i na kontrolnu magistralu.

#### 95. Objasniti preslikavanje memorijskih adresa.

Preslikavanje adresa - postupak kojim se fizicka memorija locira u adresnom prostoru racunara. Preslikavanje moze biti *puno* i *delimicno*. Adresne linije se dele u tri grupe:

- X - najviše adresne linije koje odredjuju cip
- Y - adresne linije koje se prosledjuju cipovima
- Z - najniže adresne linije koje se zanemaruju

#### 96. Objasniti puno preslikavanje memorijskih adresa.

Puno preslikavanje adresa - preslikavanje kod koga je funkcija preslikavanja memorijskih adresa u memorijske lokacije "1-1". Za svaku memorijsku lokaciju postoji najviše jedna adresa koja joj odgovara.

Sve adresne linije X se koriste pri dekodiranju radi dobijanja signala za izbor modula. Sve adresne linije se dele u dve grupe:

- linije Y i Z odredjuju bajt u memorijskom modulu
- linije X se koriste za izracunavanje signala za izbor cipa CS (chip selector)

#### 97. Navesti i objasniti dijagram primera implementacije punog preslikavanja.

Npr. koristimo 2 modula 16Mi x 32 u adresnom prostoru od 4GiB. Delimo 32 adresne linije na dve grupe:

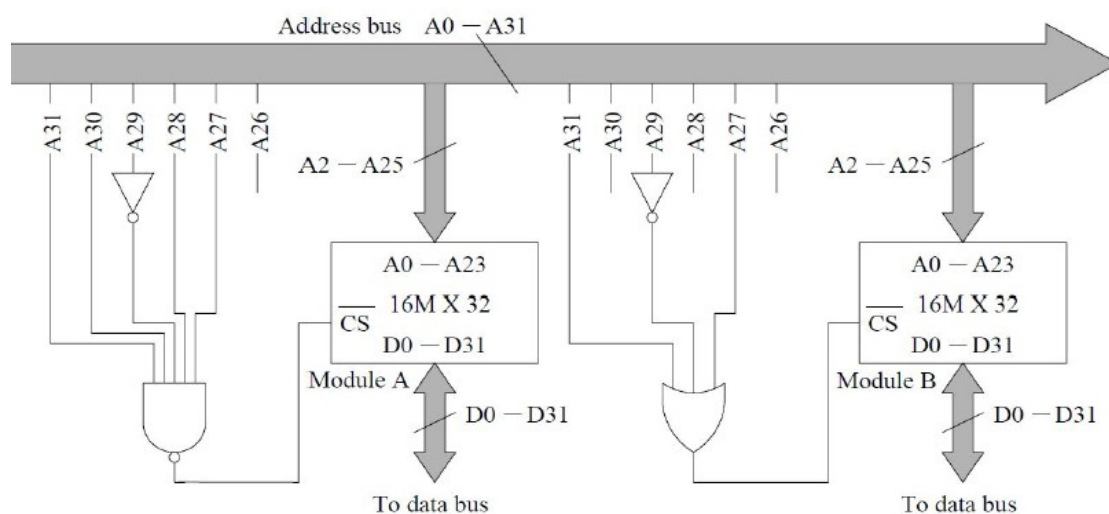
- nizih 26 linija (Y, Z) se koriste za odredjivanje bajta u okviru modula 16Mi x 32b
- visih 6 linija (X) se koriste za odredjivanje signala CS
  - modulu A odgovaraju (na primer) adrese X=110110
  - opseg adresa: D8000000H - DBFFFFFFH
  - modulu B odgovaraju (na primer) adrese X=001001
  - opseg adresa: 24000000H - 27FFFFFFH
  - ostale vrednosti adrese X ne odgovaraju nijednom modulu
  - opsezi se ne preklapaju, pa je ovo puno preslikavanje

## 98. Objasniti delimicno preslikavanje memorijskih adresa.

Delimicno preslikavanje adresa - ono kod koga funkcija preslikavanja memorijskih adresa u memorijske lokacije nije "1-1". Nekim memorijskim lokacijama može da odgovara više adresa. Cilj je pojednostavljivanje logike određivanja selektorskih signala. Može se primenjivati kada je broj memorijskih lokacija znatno manji od broja adresibilnih lokacija.

**99. Navesti i objasniti dijagram primera implementacije delimicnog preslikavanja.**

Slicno prethodnom primeru. Modulima se dodeljuju višestruke adrese. Modulima A odgovaraju (na primer) adrese X=110110 i X=110111. Skraćeno X = 11011d. Dvostruki opseg adresa: D8000000H - DBFFFFFFH, DC000000H – DFFFFFFFH . Modulu B odgovaraju (na primer) adrese X=1d0d1d, opsezi se preklapaju, pa je ovo delimično preslikavanje.



## 100. Objasniti poravnanje podataka.

Ako implementacija memorije počiva na 32-bitnim modulima onda: ako je adresa poravnata sa 32-bitnim recima, citanje 32-bitne reci se odvija u jednom ciklusu. Modul može vratiti celu 32-bitnu rec odejdanput. Ako adresa nije poravnata sa 32-bitnim recima, citanje 32-bitne reci se odvija u dva ciklusa. Modul ne može vratiti celu 32-bitni rec odjedanput.

## 101. Sta su isprepletane memorije?

Uobicajeno je da se visih  $r$  adresnih linija upotrebljava za prepoznavanje modula, a nizih  $m$  za adresiranje u modulu. Kod *isprepletanih memorija* to se menja, kako bi se uzastopne reci nalazile u razlicitim modulima.

Svaki zahtev se izvršava tokom više ciklusa (npr. 4). Ako bismo zeleli da procitamo 8 uzastopnih reci, to bi zahtevalo  $8 \times 4 = 32$  ciklusa. Isprepletane memorije omogucavaju da se skрати vreme citanja uzastopnih reci. Memorijski moduli se u kontekstu preplitanja nazivaju memorijskim bankama. Adrese se dodeljuju naizmenicno: neka imamo  $B$  banaka, banka se odredjuje kao  $addr \bmod B$ . Implementiraju se na dva nacina: *sinhronizovanim pristupom* i *nezavisnim pristupom*.

**102. Objasniti sinhronizovan pristup implementaciji isprepletanih memorija.**

Gornjih  $r$  adresnih linija se simultano dovode svim modulima. Sve banke istovremeno zapocinju svoje operacije. Nakon 4 ciklusa, svaka od memorija je dovrсила citanje (pretpostavimo da citanje zahteva 4 ciklusa).

### 103. Objasniti nezavisan pristup implementaciji isprepletanih memorija.

Problemi sa sinhronizovanim pristupom je da donosi ubrzanja samo u slucaju sekvencijalnog pristupa. Nezavisan pristup omogućava preklopljeno izvršavanje operacija na proizvoljnim adresama. Svakoj banci se dodaje memorijski registar adrese koji cuva adresu koji koristi ta banka. Nisu potrebni dodatni registri podataka, vec se podaci citaju neposredno iz memorije.

#### 104. Objasniti namenu i osnovni princip rada kesa.

Procesor sadrži registre, kao najefikasniju memoriju u sistemu. Radna memorija računara je relativno velika i spora. Razlika u brzini ova dva sloja je dovoljno velika da značajno oslabi performanse sistema.

Osnovna funkcija *kes memorije* je povećanje performansi računarskog sistema. Po hijerarhiji ova vrsta memorije se nalazi između glavne memorije i registara procesora. Svrha kes memorije je da premosti razlike između brzina procesora i glavne memorije. Ako je razlika prevelika dodaje se više slojeva kes memorije.

Kada procesor zahteva neki podatak iz memorije, tada se blok u kome se on nalazi prenosi iz glavne memorije u kes. Ubrzanje rada se postiže zahvaljujući principu lokalnosti (zahtevanje podataka iz radne memorije unapred (prefetch), pre nego što zaista zatrebaju procesoru), tj. ako se osnovni podatak nalazi u bloku koji je prenet u kes, tada je velika verovatnoca da će i podaci potrebni u narednim izračunavanjima biti takodje u tom bloku. Ako je uspesno predviđeno koji će podaci biti potrebni procesoru u bliskoj budućnosti, njih će procesor citati iz kesa, a ne iz memorije. Kako je kes znatno brza memorija od glavne memorije, vreme potrebno za naredne prenose podataka do procesora će biti neuporedivo kraće.

#### 105. Objasniti citanje kesa u slučaju pogotka.

Ako se potrebni podaci nalaze u kesu, onda se odatle i citaju. U slučaju pogotka, linije adrese i podataka prema memoriji se blokiraju. Razmena informacija se odvija isključivo sa kesom. Citanje sa pogotkom je značajno brže od citanja iz memorije bez primene kesa.

#### 106. Objasniti citanje kesa u slučaju promasaja.

Ako se potrebni podaci ne nalaze u kesu, onda se citaju iz memorije i istovremeno upisuju u kes. U slučaju promasaja, linije adrese i podataka prema memoriji su aktivne. Odvija se uobičajeno (kao da nema kesa) citanje podataka iz memorije. Dodatno se procitani podaci upisuju i u kes. Citanje sa promasajem je nešto sporije od citanja iz memorije bez primene kesa, zbog neophodnog proveravanja da li podatak postoji u kesu ili ne.

#### 107. Objasniti pisanje kesa u slučaju pogotka.

U slučaju pogotka postoje dve osnovne mogućnosti: pisanje se obavlja samo u kes ili pisanje se obavlja i u kes i u memoriju.

#### 108. Objasniti pisanje kesa u slučaju promasaja.

U slučaju promasaja podaci se upisuju samo u memoriju, zato što ne postoje u kesu.

#### 109. Koji su osnovni preduslovi za uspesnu primenu kesa (zasto kes radi)?

Prvi faktor za uspesnu primenu kesa je *ponovljena upotreba* istih podataka ili delova koda. Naredba u petlji se ponavlja  $M \times N$  puta. Ako je naredba zapisana u kesu, njeno izvršavanje je značajno ubrzano zato što se cita iz brzog kesa, a ne iz spore memorije.

Drugi faktor je plansko punjenje kesa podacima i instrukcijama pre nego što ih procesor zatraži. Plansko punjenje podize performanse iz dva razloga: maskira kasnjenje sporije glavne memorije i punjenje se odvija u blokovima, a prenos blokova podataka je višestruko brži nego prenos pojedinačnih podataka.

U slučaju većine programa se ispoljava bar jedan od faktora za uspesnost kesa: ponavljanje i/ili predvidivo pristupanje podacima.

## 110. Objasniti princip lokalnosti. Sta je prostorna, a sta vremenska lokalnost?

Princip lokalnosti referisanja - tvrdi da programi imaju tendenciju da u nekom datom periodu vremena referisu samo neki podskup podataka i instrukcija i to cesto uz ponavljanje. Razlikujemo vremensku i prostornu lokalnost.

Prostorna lokalnost - programi imaju tendenciju da podatke i instrukcije koriste sekvencijalno. Lokalni podaci u funkcijama su zapisani na steku, blisko jedni drugima. Slozeni podaci zauzimaju sekvencijalne oblasti u memoriji. Vecinu vremena instrukcije se citaju i izvorsavaju sekvencijalno. Skokovi remete sekvencijalnost, ali izmedju dva skoka obicno je vise instrukcija.

Vremenska lokalnost - programi imaju tendenciju da ponovljeno koriste podatke i instrukcije u odredjenim periodima vremena. Tipican primer su petlje - iste instrukcije se izvorsavaju vise puta, iste lokalne promenljive se koriste vise puta, neki delovi slozenih podataka se koriste vise puta.

## 111. Objasniti preslikavanje adresa kesa i navesti vrste preslikavanja.

Kada je potrebno pristupiti sadrzaju neke memorijske lokacije, najpre se proverava da li je vec u kesu ili nije. Ako jeste, potrebno je ustanoviti gde se u kesu nalazi tj. izracunati njegovu adresu u kesu. Ovaj postupak se naziva preskilavanje adresa. Adresa memorije se preslikava u adresu kesa.

Vrste (funkcije) preslikavanja:

- neposredno prelikvanje
- asocijativno preslikavanje
- skup-asocijativno preslikavanje

## 112. Objasniti neposredno preslikavanje adresa kesa i dati primer.

Preslikavanjem se odredjuje tacno jedna linija kesa za svaki memorijski blok. Obicno se koristi funkcija  $c = i \bmod C$ , gde je  $c$  linija kesa,  $i$  memorijski blok, a  $C$  broj linija kesa (velicina kesa u linijama). Jednostavno je za implementaciju, ali niska fleksibilnost moze da oslabi performanse.

Delovi memorijske adrese:

$B$  - velicina bloka (najnižih  $b = \log_2 B$  bitova cine polozej u bloku),  $C$  – broj linija kesa (narednih  $c = \log_2 C$  bitova cine liniju kesa), najvisih  $t$  bitova cine oznaku kesa

Za svaku liniju kesa moraju postojati podaci o:

sadrzaju kesa ( $B$  bajtova koji predstavljaju sadrzaj linije kesa, kopija preslikanog memorijskog bloka), bitu ispravnosti (oznacava da li je sadrzaj linije ispravan), oznaci kesa ( $t$  bitova koji oznacavaju kom memorijskom bloku odgovara trenutni sadrzaj linije kesa)

Postupak preslikavanja:

Da bi se pronasla odgovarajuca linija kesa, primenjuje se funkcija preslikavanja po modulu. Ako je linija kesa ispravna i oznaka kesa odgovara memorijskom bloku, to znaci da sadrzaj linije kesa vec odgovara datom memorijskom bloku. Ako blok nije u kesu, onda se cita iz memorije i upisuje u odgovarajucu liniju kesa. Ako je linija kesa bila popunjena, njen sadrzaj se prethodno obradjuje u skladu sa politikom pisanja.

Primer:

Neka je velicina kesa 4 linije x 4 bajta. Neka se pristupa redom memorijskim adresama: 0, 16, 0, 32, 0, 32, 0, 16, 0, 16, 0, 16. Adrese 0, 16 i 32 odgovaraju blokovima 0, 4 i 8. Svim ovim blokovima odgovara linija kesa 0. Stepni pogodak je 0. Ovo je primer najgoreg scenarija.

### 113. Objasniti asocijativno preslikavanje adresa kesa i dati primer.

Ova vrsta preslikavanja nema ogracenja - svaka linija kesa moze biti korisćena za bilo koji memorijski blok. Fleksibilnost je maksimalna, ali je veoma slozena za implementaciju.

#### Delovi adrese:

najnižih  $b = \log_2 B$  bitova adrese cine polożaj u bloku. Svih preostalih  $t$  bitova cine oznaku kesa.

Nemamo informaciju o liniji kesa jer podatak moze da bude zapisan u bilo kojoj liniji.

#### Problemi sa asocijativnim preslikavanjem:

Duzina oznake kesa je veca nego kod neposrednog preslikavanja. Ovo nije najvažniji problem, ali ima uticaja na kolicinu potrebnih podataka za liniju kesa. Provera da li je memorijski blok vec zapisan u nekoj liniji kesa je značajno složenija. Umesto da se proverava samo jedna linija kesa, potrebno je da se oznake svih ispravnih linija kesa uporede sa oznakom datog bloka. To zahteva  $2^c$  poredjenja duzine  $t+c$  bitova. Posledica je da je implementacija značajno složenija i skuplja.

#### Primer:

Neka je velicina kesa 4 linije x 4 bajta. Neka se pristupa redom memorijskim adresama: 0, 16, 0, 32, 0, 32, 0, 16, 0, 16, 0, 16. Svaki blok moze da koristi bilo koju liniju kesa. Blok 0 se zapisuje u prvoj slobodnoj liniji, blok 4 u narednoj slobodnoj liniji, blok 8 u narednoj. Stepeni pogodak je  $9/12 = 0.75$ . To je najviši stepen koji se moze postići na datom primeru.

### 114. Objasniti skup-asocijativno preslikavanje adresa kesa i dati primer.

Kompromis između neposrednog i asocijativnog preslikavanja. Preslikavanje se odvija slično neposrednom, ali se za 1 memorijski blok određuje skup linija kesa umesto samo 1 linije kesa. Velicina skupa je obično neki manji stepen broja 2.

#### Delovi adrese:

najnižih  $b = \log_2 B$  bitova adrese cine polożaj u bloku. Narednih  $s = \log_2 S$  bitova određuju skup linija kesa. Najviših  $t$  bitova cine oznaku kesa.

Prednosti u odnosu na neposredno su slične kao i prednosti asocijativnog - omogućavanje višeg stepena pogodak

Prednosti u odnosu na asocijativno su manja dužina oznake kesa, manji broj kandidata koji je potrebno proveravati, umesto  $2^c$  poredjenja po  $t$  bitova potrebno je  $2^s$  poredjenja po  $t-s$  bitova.

Slabost - nizi stepen pogodaka nego u slučaju asocijativnog preslikavanja.

#### Primer:

Neka je velicina kesa 4 linije x 4 bajta. Neka se pristupa redom memorijskim adresama: 0, 16, 0, 32, 0, 32, 0, 16, 0, 16, 0, 16. Svaki od blokova 0, 4 i 8 se preslikava u skup linija 0. Blok 0 se zapisuje u prvoj slobodnoj liniji skupa 0, blok 4 u narednoj slobodnoj. Pri prvom pristupanju bloku 8 sve linije skupa 0 su već zauzete: primenjuje se politika zamenjivanja kako bi se odabrala linija čiji će se sadržaj zameniti. Stepeni pogodak je  $8/12 = 0.67$ .

### 115. Sta su i čemu služe politike zamenjivanja kesa? Nabrojati ih.

Politika zamenjivanja se primenjuje radi odabira linije kesa čiji će sadržaj biti zamenjen sadržajem novog memorijskog bloka. Zavisi od primenjenog preslikavanja. U slučaju neposrednog preslikavanja ne postoje politike zamenjivanja zato što nema izbora. U slučaju skup-asocijativnog preslikavanja razlikujemo:

- politiku zamenjivanja najduže nekoriscene linije kesa
- politika zamenjivanja pseudo-najduže-nekoriscene linije kesa
- politika proizvoljnog zamenjivanja
- FIFO - prva procitana se prva zamenjuje
- LFU - najmanje puta korisćena



### 116. Objasniti politiku zamenjivanja najduze nekoriscene linije kesa.

Idealno bi bilo da se zameni onaj memorijski blok koji najduze nece biti ponovo koriscen.

Predvidjanje se zasniva na principu lokalnosti. Zamenjuje se sadrzaj one linije kesa koja najduze nije bila koriscena. *Implementacija:*

- za skupove velicine 2 dovoljan je jedan bit po skupu, za oznacavanje poslednje koriscene linije
- za vece skupove se prave odgovarajuci konacni automati:
  - za skup velicine 4 ima 4! mogucih kombinacija redosleda, potrebno 5 bitova po skupu linija
  - za skup velicine 8 ima 8! mogucih kombinacija redosleda, potrebno 16 bitova po skupu linija

Ova politika je najbolje logicki zasnovana, ali je komplikovana za implementaciju, pa se umesto nje koristi politika zamenjivanja pseudo-najduze nekoriscene linije kesa.

### 117. Sta su i cemu sluze politike pisanja kesa? Nabrojati ih.

Kada se podaci menjaju mora se uzeti u obzir da postoje dve kopije podataka, kopija u glavnoj memoriji i kopija u kesu. Postoje dve osnovne politike pisanja:

- pisanje se obavlja samo u kes (tzv. politika pisanje sa prepisivanjem)
- pisanje se svaki put obavlja i u kes i u memoriju (tzv. pisanje sa propustanjem)

### 118. Objasniti politiku pisanja kesa sa propustanjem.

Pri pisanju se menjaju obe kopije podataka. Menja se kopija u kesu i menja se kopija u memoriji. Dobro je sto su podaci u glavnoj memoriji i kesu uvek uskladjeni (posebno vazno u slucaju viseprosesorskih sistema). Mana je sto se pri svakoj operaciji pisanja angazuje memorijska magistrala (i memorija).

### 119. Objasniti politiku pisanja kesa sa prepisivanjem.

Pri pisanju se menja samo kopija u kesu. Kopija u glavnoj memoriji se azurira u trenutku zamenjivanja linije kesa. Zamenjivanje je dvostruko duze nego u slucaju politike sa propustanjem. Optimizacija se postize evidentiranjem da li je bilo pisanja tako da se pisanje u glavnoj memoriji obavlja samo ako su podaci u kesu izmenjeni. Za svaku liniju kesa se vodi dodatni bit koji oznacava da li je sadrzaj kesa razlicit od sadrzaja glavne memorije (dirty bit). Ovakva optimizacija je prilicno efikasna zato sto pisanje cine svega oko 15% svih pristupa memoriji. Dobro je sto se pisanje obavlja samo jedanput po liniji kesa. Mane - zamenjivanje je dvostruko duze.

### 120. Objasniti vrste promasaja kesa.

Neizbezni promasaji - prvi pristupi nekom memorijskom bloku moraju biti promasaji.

Promasaji usled kapaciteta - zbog manje velicine kesa, mora doci do zamenjivanja sadrzaja linija kesa. Mogli bi se izbeci kada bi bilo vise prostora.

Promasaji usled sudara - desavaju se usled neposrednog ili skup-asocijativnog preslikavanja, kada u kesu ima prostora, ali ne za konkretne blokove.

### 121. Objasniti razdvajanje kesa podataka od kesa instrukcija.

Osnovna ideja je da instrukcije i podaci predstavljaju razlicite vrste sadrzaja. Na razlicit nacin im se pristupa (instrukcije se obicno ne menjaju), razlikuje se ponavljanje upotrebe instrukcija i podataka. Ako se instrukcije i podaci posmatraju odvojeno lokalnost je tacnije i buduci pristupi lakse su predvidivi. Mana je sto se ne moze dinamicki (prema potrebi) menjati proporcije ova dva kesa.



## 122. Objasniti arhitekture visestepenog kesa.

Ekskluzivan kes - svaki blok je na najviše jednom nivou kesa; smanjeno ponavljanje znaci veci prostor ; svaki nivo mora da ima istu velicinu linija

Inkluzivan kes - kes viseg nivoa sadrzi blokove koji su u kesu nizeg nivoa; pri zamenjivanju blokova jednog nivoa vrsi se pisanje samo na sledecem nivou ; kes viseg nivoa moze da koristi vece linije .

Uglavnom inkluzivan kes - kes viseg nivoa ne mora da sadrzi blokove koji su u kesu nizeg nivoa

*Rad visestepenog kesa (inkluzivan slucaj):*

- Procesor pokusava da pristupi podacima u kesu L1: ako su podaci pronadjeni, oni se citaju iz kesa L1 (pogodak kesa L1) , ako nisu (promasaj kesa L1), podaci se moraju citati iz kesa L2 ili iz glavne memorije
- U slucaju promasaja kesa L1, kes kontroler pokusava da pristupi podacima u kesu L2: ako su podaci pronadjeni u kesu L2, oni se citaju odatle (pogodak kesa L2) - rec se prosledjuje procesoru i blok se upisuje u kes L1 , ako nisu (promasaj kesa L2), podaci se citaju iz glavne memorije - rec se salje procesoru, blok (ili blokovi razlicitih velicina) se upisuju u oba kesa

*Performanse:*

Namena kesa L2 je da “uhvati” promasaje kesa L1. Ako je stepen pogodaka kesa L1 90% i stepen pogodaka kesa L2 takodje 90%, onda je ukupan stepen promasaja svega 1%.

## 123. Objasniti odnos velicine kesa i performansi.

Sto je veca brzina prenosa memorije to je manji kapacitet kesa.

## 124. Objasniti odnos velicine bloka kesa i performansi.

Za fiksnu velicinu bloka, ako menjamo velicinu linije postoji neka tacka u kojoj je broj promasaja minimalan.

## 125. Objasniti odnos asocijativnosti i performansi.

Veci je broj promasaja ako se koristi direktno preslikavanje nego kad se koristi asocijativno.

## 126. Sta je virtualna memorija i objasniti osnovne funkcije?

Koncept *virtualne memorije* omogućava da programi u racunarskom sistemu upotrebljavaju veci memorijski prostor nego sto je stvarna velicina fizicki prisutne memorije. Tehnika virtualne memorije automatizuje staranje o memoriji i oslobadja programera suvisnog staranja o fizickim resursima, automatski se podaci i delovi programa prebacuju iz fizicke memorije na disk i obratno. Dve *osnovne funkcije* koje ostvaruje tehnika virtuelna memorija su:

- *premestanje* - svaki program koristi svoj virtualni adresni prostor, tokom izvorsavanja taj adresni prostor se moze preslikavati u razlicite fizicke memorijske lokacije. Detalji upravljanja ovim preslikavanjem nemaju nikakve veze sa implementacijom programa, svu brigu oko preslikavanja vode procesor i operativni sistem
- *zastita* - razdvojenost adresnih prostora programa pruza medjusobnu izolovanost programa i zastitu podataka i koda.

## 127. Objasniti koncept stranica virtualne memorije.

Osnovi koncepta virtualne memorije su: organizacija memorije po stranicama , preslikavanje virtualnih i fizickih adresa i stranicenje pomocu diska.

Celokupan virtualan adresni prostor se deli na *virtualne stranice*. Bitovi virtualne adrese se dele na *broj virtualne stranice* i *adresu u stranici*. Slicno tome, fizicka memorija se deli na *fizicke stranice* (ili okvire za stranice). Bitovi fizicke adrese se dele na *broj fizicke stranice* i *adresu u stranici*.

## 128. Objasniti preslikavanje adresa virtualne memorije. Primer.

Pri preslikavanju virtualnih adresa u fizicke pretpostavlja se da su fizicke i virtualne stranice iste velicine. Adresa u okviru stranice se ne menja preslikavanjem. Broj virtualne stranice se preslikava u broj odgovarajuce fizicke stranice. Uobicajena velicina stranice je 4KiB. Za preslikavanje je zaduzena jedinica za upravljanje memorijom.

Primer: Preslikavanje 32-bitne virtualne u 24-bitnu fizicku memoriju, sa stranicom od 4KiB

## 129. Kako virtualna memorija upotrebljava disk?

Iz ugla operativnog sistema svaka stranica virtualne memorije je u glavnoj memoriji ili na disku. Stranice koje su u glavnoj memoriji imaju svoju sliku na disku. Ako program zahteva vise memorije nego sto ima, pravi se virtualna memorija koja je veca od fizicke, nekativni podaci se sklone na disk i oslobodi se fizicka memorija. Posao se deli izmedju OS-a i procesora jer procesor ne moze sam da obavi sve, ali mora da obavi deo posla da bi bilo efikasno. Operativni sistem mora da omoguci da procesi mogu da komuniciraju medjusobno na neki nacin. Radna memorija se ponasa kao kes izmedju virtualne memorija i diska. Svaka dodeljena virtualna stranica ima mesto u glavnoj memoriji, a ako ih ima vise nego mesta u glavnoj memoriji, onda je na disku.

## 130. Sta je stranicenje po zahtevu?

Stranicenje je premestanje stranice sa diska u glavnu memoriju i obrnuto (cesce ovo prvo zovemo stranicenje, ali jedno bez drugog ne ide).

Ako se pokusa pristupiti sadrzaju stranice koja nije u glavnoj memoriji, tada nastaje *greska stranicenja*. U tom slucaju OS je zaduzen da odreaguje i ucita trazenu stranicu u glavnu memoriju i azurira podatke koji se odnose na preslikavanje stranica. Ovo se naziva *stranicenje po zahtevu*, zato sto se obavlja kada se zahteva data stranica.

Svaki prvi pristup stranici je promasaj. Postoji tablica koja govori gde je sta na disku, kao i za fizicke adrese, moze i ne mora da bude ista tablica. "Promasena" stranica se prebacuje sa memorije na disk. Ako je glavna memorija puna, primenjuje se *politika zamene stranice*. Osvezi se tabela stranica i vrati se na promasenu instrukciju.

### 131. Sta je implicitno stranicenje?

Operativni sistem moze da preduzima stranicenje i implicitno tj. bez eksplicitno iskazane potrebe za stranicom, ukoliko se proceni da se stranicenjem nece znacajno oslabiti performanse aktivnih procesa (npr. da se disk trenutno ne upotrebljava) i da se relativno visokom verovatnocom moze pretpostaviti da ce u skoroj buducnosti biti potrebne neke stranice koje trenutno nisu u fizickoj memoriji.

### 132. Sta su i kada se koriste politike zamenjivanja stranica?

Pri razmatranju politika zamenjivanja uzimaju se u obzir razlike izmedju virtualne memorije i kesa. Cena promasaja u slucaju virtualne memorije je visestruko veca nego u slucaju kesa, pa je veci znacaj dobre politike. Politike zamenjivanja kesa se implementiraju u hardveru, a u slucaju virtualne memorije u softveru, sto pruza vecu slobodu. Zbog toga se u slucaju virtualne memorije texti postizanju sto kvalitetnijeg odabira stranica, a po cenu slozenosti algoritma.

### 133. Objasniti politiku stranicenja FIFO.

Zamenjuje se ona stranica koja je najduze u glavnoj memoriji. Jednostavna implementacija. Mana je sto ne uzima u obzir upotrebu stranice, relativno slabe performanse, retko se primenjuje.

### 134. Objasniti politiku stranicenja retko upotrebljavana.

Zamenjuje se stranica koja je najmanje puta referisana. Relativno jednostavna implementacija. Svakoj stranici se dodaje brojac referisanja, operativni sistem povremeno ponistava brojace. Umereno znacajan doprinos performansama.

### 135. Objasniti politiku stranicenja najduze neupotrebljavana.

Zamenjuje se stranica ciji sadrzaj najduze nije referisan. Iako se implementira u softveru a ne u hardveru, ipak je neprakticna puna implementacija. Najcesce se aproksimira. Nacelno slicno kao u slucaju kesa, ali ipak se tezi boljoj aproksimaciji. Ovo je najcesce primenjivana politika.

### 136. Objasniti politike pisanja virtualne memorije.

U slucaju kesa razmatrali smo pisanje sa propustanjem i pisanje sa prepisivanjem. U slucaju virtualne memorije pisanje sa propustanjem nije opcija, zbog velike razlike u brzini diska i glavne memorije. Dodatno, u slucaju virtualne memorije postoji zastita na nivou stranica i procesa (poput zakljucavanja) pa je pisanje sa prepisivanjem bezbednije nego u slucaju kesa. Ne postoji opasnost da neko upotrebljava neazurirane podatke iz stranice na disku.

### 137. Objasniti znacaj velike stranice virtualne memorije i navesti primere.

Male stranice su dobre zbog:

- interne fragmentacije – velicina podataka, programa i steka nije ceo broj stranica, prosečno je pola stranice neupotrebljeno, što je stranica manja, iskoriscenost je veca
- boljeg pogadjanja – što je veca stranica, to će pri njenom učitavanju u glavnu memoriju zastupljenost nepotrebnih sadržaja biti veca

Velike stranice su dobre zbog:

- velicine tablice stranica – što su vece stranice, bice ih manje, pa su i tablice manje
- vremena pristupa disku – vreme pristupa disku je mnogo veco nego vreme citanja; reda 10ms po pristupu, 100M iB/s citanje; citanje 100 stranica od 4KiB traje  $100 \times (10\text{ms} + 4/100\text{ms}) = 10.04\text{s}$ ; citanje 25 stranica od 16KiB traje  $25 \times (10\text{ms} + 16/100\text{ms}) = 2.54\text{s}$ . Veca stranice redukuju broj pristupa disku i ubrzavaju rad

Primeri:

- Intel x86, Power PC, MIPS R4000

### 138. Kako se implementira preslikavanje adresa virtualne memorije?

Zbog visoke cene promasaja potrebno je da bude sto manji stepen promasaja. Zbog toga se u slucaju virtualne memorije uobicajeno primenjuje *puno asocijativno preslikavanje stranica*. Implementira se primenom *tablica translacija* (tablice stranica).

### 139. Sta su tablice stranica virtualne memorije?

Primitivna implementacija - u osnovnom obliku tablica sadrzi skup parova (broj virtualne stranice, broj fizicke stranice) ili (broj virtualne stranice, lokacija na disku). Po pravilu se pristupa po broju virtualne stranice. Ovakva implementacija zahteva suvisna uredjivanja.

Pojednostavljena implementacija - tablica se implementira kao niz indeksiran brojevima virtualnih stranica

Moze se modelirati sa dve tablice:

- *tablica fizickih stranica* – niz indeksiran brojevima virtualnih stranica (VPN); sadrzi brojeve fizickih stranica; sadrzi i dodatne kontrolne bitove
- *tablica adresa na disku* – niz indeksiran brojevima virtualnih stranica; sadrzi lokacije stranica na disku

Obicno se implementira kao jedna tablica koja sadrzi obe vrste podataka.

### 140. Sta sadrže tablice stranica virtualne memorije?

Svaka stavka tabele stranica (PTE) sadrzi:

- *broj fizicke stranice (PPN)* – lokacija date stranice u glavnoj memoriji; vodi se za stranice koje postoje u glavnoj memoriji
- *adresa stranice na disku* – lokacija date stranice na disku; vodi se za sve stranice
- *bit ispravnosti* – oznacava da li je stranica u memoriji ili ne
- *bit izmenjenosti* – oznacava da li je sadrzaj stranice menjan; ako je menjana, stranica se pri uklanjanju iz glavne memorije zapisuje na disku
- *bit referisanja* – koristi se za implementaciju algoritma pseudo-LRU; OS povremeno postavlja sve bitove referisanja na 0; pri pristupanju stranici bit se postavlja na 1
- *informacija o vlasniku* – OS mora da zna kom procesu pripada stranica
- *bitovi zastite* – oznacavaju tip pristupa koji vlasnik ostvaruje (samo citanje, citanje i pisanje, izvorsavanje,...)

### 141. Sta je bafer tablice stranica virtualne memorije? Sta sadrzi i kako se upotrebljava?

TLB se implementira u procesoru. Sadrzi podatke o poslednjem koriscenju PTE. Svaka stavka TLB sadrzi: broj virtualne stranice (VPN), odgovarajuci broj fizicke stranice (PPN), kontrolne bitove. Sve stavke u TLB se nalaze i u glavnoj memoriji i u okviru pune tablice stranica dopunjene dodatnim podacima, kao sto je lokacija na disku,... Kao sto kes moze biti podeljen, tako i TLB moze da se podeli prema nameni - za instrukcije i za podatke. Uobicajeno je da uz podeljen kes ide podeljen TLB. Po pravilu se implementira sa punim asocijativnim preslikavanjem.

Algoritam upotrebe:

najpre se podaci o stranici traze u TLB. Ako se pronadju, pomocu njih se racuna fizicka adresa, obicno je pravilo da se stranica referisana iz TLB ne zamenjuje, pa je ona vec u memoriji. Inace se podaci traze u tablici stranica. Ako stranica nije u memoriji, mora da se ucita. Racuna se fizicka adresa i pristupa se sadrzaju fizicke memorije.

Politika zamenjivanja stavki TLB je obicno jednostavna: politika slucajnog izbora ili politika pseudo najduze nekoriscene.

## 142. Gde se nalaze tablice stranica virtualne memorije? Primer organizacije u tri nivoa.

Tablica stranica može biti vrlo velika. Zato nije praktično (a ponekad ni ostvarivo) da se locira u fizičkoj memoriji, već se zapisuje u *virtualnoj memoriji* i tablica se deli na stranice i samo se potrebne stranice čuvaju u fizičkoj memoriji. Za ove stranice se pravi dodatna tablica drugog nivoa. Po potrebi se može praviti više nivoa, sve dok se ne dođe do tablice čija je veličina prihvatljiva za stalno lociranje u fizičkoj memoriji.

Primer: Neka su virtuelne adrese 40-bitne, veličina stranice 4KiB, a veličina stavke 4 bajta.

Tada postoji  $2^{28}$  stranica, veličina tablice prvog nivoa je 1GiB i zauzima  $2^{18}$  stranica.

Velicina tablice drugog nivoa je 1MiB i zauzima  $2^8$  stranica.

Velicina tablice trećeg nivoa je 1KiB i staje u jednu stranicu.

## 143. Sta su ulazno/izlazni uredjaji?

Racunarski sistem obično ima više različitih ulazih i izlaznih uredjaja. Obezbeđuju dve osnovne funkcije: *komunikaciju racunarskog sistema sa spoljnim svetom i cuvanje podataka.*

## 144. Objasniti principe rada ulazno/izlaznih uredjaja.

Nezavisno od vrste uredjaja, osnovni principi rada ulazno/izlaznih uredjaja su isti:

- svi U/I uredjaji se na sistemsku magistralu povezuju posredstvom odgovarajućeg U/I kontrolera
- prvi razlog je u različitosti uredjaja - različiti uredjaji imaju različite protokole komunikacije. Umesto da procesor i sistemski magistrala "uče" kako da komuniciraju sa različitim vrstama uredjaja, te specifičnosti se prepustaju kontrolerima. Kontroleri imaju ulogu mosta između centra i periferije.
- drugi razlog je u tehničkim ograničenjima - magistrala radi na visokim frekvencijama. Da se ne bi suviše grejalo, radi pod vrlo niskim naponima. Nizak napon i visoka frekvencija mogu da funkcionisu bez smetnji samo na vrlo kratkim rastojanjima. U/I uredjaji zahtevaju duže kablove, jaci napon i nizu frekvenciju.

## 145. Sta su U/I kontroleri?

Procesor se nikada ne obraća neposredno uredjajima, već samo odgovarajućim kontrolerima.

U/I kontroleri imaju ulogu mosta između *centra* (procesor, memorija i sistemski magistrala) i *periferije* (periferni uredjaji računarskog sistema). Kontroleri uobičajeno imaju tri vrste internih registara: registar podataka, komandni registar i statusni registar.

## 146. Sta podrazumeva upotreba U/I uredjaja putem memorijskog mapiranja?

Svi U/I uredjaji (portovi) se preslikavaju u memorijski adresni prostor. Nije potreban nikakav poseban interfejs procesora. Procesor koristi uredjaj kao memoriju. Svaki procesor može upotrebljavati uredjaje putem preslikavanja portova u memoriju. Neki procesori podržavaju samo ovakav način rada: PowerPC, MIPS.

## 147. Sta podrazumeva upotreba U/I uredjaja putem izolovanog U/I?

Izolovani U/I podrazumeva poseban U/I adresni prostor, nezavisan od memorijskog adresnog prostora. Intel x86 procesori podržavaju ovakav način rada. Sistemi zasnovani na procesorima koji podržavaju izolovani U/I omogućavaju izbor metoda (npr: stampaci se koriste putem izolovanog U/I, a grafički podsistem putem preslikavanja u memoriju).

## 148. Koje se osnovne tehnike koriste pri prenosu podataka između U/I uredjaja i procesora?

Tri značajne tehnike: *programirani U/I*, *neposredan pristup memoriji (DMA)*, *sistem prekida*

Ako se podaci prenose putem DMA, onda se obaveštavanje o kraju odvija putem sistema prekida, a ako se podaci prenose putem programiranog U/I, onda se i obaveštenje ostvaruje istim putem.

### 149. Objasniti tehnike programiranog U/I.

U osnovi programiranog U/I je petlja u kojoj se ceka da se dovrši zadati posao da bi se moglo nastaviti sa radom. Cekanje se izvodi kroz višestruko proveravanje da li je uređaj dostigao očekivano stanje.

Primer: Tastatura - očitava se registar PA. Ako bit 7 ima vrednost 1, znaci da nije pritisnut taster, pa se ponavlja prethodni korak. Ako bit 7 ima vrednost 0, znaci da je pritisnut taster i prekida se petlja. Zatim se procitani kod tastera može dalje upotrebljavati tj. prevoditi u kod karaktera.

### 150. Objasniti direktan pristup memoriji (DMA)?

U slučaju uređaja koji prenose veće količine podataka ili rade vremenski zahtevne poslove, programirani U/I vodi velikom utrosku vremena na cekanje. DMA ima za cilj da se procesor oslobodi staranja o prenosu podataka i posveti drugim stvarima.

### 151. Kako se implementira DMA? Kontroler DMA.

DMA se implementira pomoću kontrolera DMA. Kontroler se ponasa kao podređen uređaj u odnosu na procesor i prima instrukcije za prenos podataka od procesora. Zatim preuzima kontrolu nad magistralom i ostvaruje prenos podataka. Kontroler uobičajeno podržava veći broj uređaja i svaki se povezuje na poseban kanal DMA.

DMA kontroler je dodatni modul priključen na sistemsku magistralu. U sustini predstavlja specijalizovani procesor koji može da izvršava programirani U/I. Kada procesor treba da izvrši U/I operaciju, on upiše DMA kontrolni blok u memoriju.

### 152. Objasniti osnovne korake operacija DMA.

1. Inicijalizacija kanala - procesor inicira kontroler DMA i šalje mu: broj uređaja, adresu prostora u memoriji, broj bajtova koji se prenose, smer prenošenja podataka. Nakon inicijalizacije kanal je spreman za prenošenje podataka.
2. Prenosenje podataka - kada U/I uređaj bude spreman za prenošenje podataka, obavestava o tome kontroler DMA. Kontroler započinje operaciju prenosa: kontroler zahteva magistralu (i dobija je uobičajenim postupkom arbitraže), postavlja memorijsku adresu i odgovarajući signal (citanje ili pisanje) na magistralu, dovršava prenos i oslobadja magistralu, azurira adresu i brojac i na kraju, ako ima još podataka za prenošenje, ponavlja postupak.
3. Obavestavanje procesora - nakon dovršenog prenosa obavestava se procesor. Uobičajeno se za to koristi sistem prekida. Procesor zatim proverava stanje prenosa.

### 153. Navesti osnovne karakteristike interfejsa USB. Standardi i specifikacije.

Izvorno razvijen 1995. Najsira do sada sprovedena inicijativa za projektovanje interfejsa za povezivanje periferija. *Osnovni cilj* - omogućiti povezivanje računarskih periferija jednako jednostavno kao što se povezuju telefon ili pegla.

Standardi:

- 1.0 iz 1996. - niska brzina 1.5 Mbps, puna brzina 12 Mbps
- 1.1 iz 1998. - razreseni neki problemi sa protokolom. Prvi široko rasprostranjen standard
- 2.0 iz 2000. - visoka brzina, do 480 Mbps
- 3.0 iz 2008. - brzina do 4800 Mbps

## 154. Koji su bili osnovni ciljevi postavljeni pred USB interfejs i dodatne napredne osobine?

Osnovni ciljevi:

- Obezbeđivanje homogenog računarskog interfejsa - ideja je da se mnoštvo različitih i nekompatibilnih interfejsa zamene jednim univerzalnim interfejsom.
- Prevazilaženje problema sa resursima sistema - dodavanje novih uređaja donosi probleme razresavanja konflikata oko adresa ili prekida. USB uređaji ne zahtevaju ni memoriju ni adresni prostor ni prekide.
- Jednostavnija instalacija i konfiguracija - potpuna automatizacija za razliku od starijih uređaja, koji su često zahtevali manuelno konfigurisanje.
- Povezivanje tokom rada računara - stariji načini povezivanja uređaja su zahtevali isključivanje računara pre povezivanja. Povezivanje USB uređaja se može obavljati bez isključivanja.

Dodatne napredne osobine:

- Napajanje uređaja kroz interfejs - kabl za povezivanje ima linije napajanja, čime je omogućeno da se uređaji napajaju strujom napona +5V i jačine 100–500mA
- Dvosmerna kontrola uređaja - podaci mogu da teku u oba smera. Omogućena je puna kontrola uređaja, kao i upotreba uređaja za kontrolisanje računara.
- Prozirivost pomoću habova - na jedno priključno mesto se može povezati hab radi povećanja broja priključaka ili postavljanja priključaka na željeno mesto.
- Ušteda energije - USB uređaji automatski ulaze u primireno (suspendovano) stanje ako nema aktivnosti na magistrali 3ms. U primireom stanju zahtevaju svega oko 2.5mA
- Prepoznavanje i otklanjanje gresaka - koristi se algoritam CRC za prepoznavanje gresaka u komunikaciji. U slučaju prepoznavanja greske, transakcija se ponavlja.

## 155. Objasniti način enkodovanja podataka NRZI u slučaju interfejsa USB.

Kabl ima četiri linije - dve služe za napajanje uređaja, a druge dve za prenos signala. Za prenos podataka se upotrebljava shema enkodovanja NRZI (nonreturn to zero-inverted). Enkodovanje NRZI u slučaju USB-a: signal se menja ako je naredni bit 0, signal se ne menja ako je naredni bit 1, smer promene nije značajan.

## 156. Objasniti koncept umetanja bitova kod protokola USB.

Opisana shema enkodovanja NRZI rešava neke od problema:

nivo signala ne igra glavnu ulogu, posmatraju se samo tranzicije, otklanjaju se dugacki nizovi nepromenjenih stanja u slučaju nizova jedinica u originalnim podacima.

Preostaje problem: opstaju dugacki nizovi nepromenjenih stanja u slučaju jedinica.

Rešenje - preduzima se tzv. umetanje bitova. Nakon svakih 6 uzastopnih jedinica se umeće jedna 0. Umetanje je na nivou podataka, a ne signala.

## 157. Objasniti arhitekturu povezivanja USB uređaja.

Osnovni hardver čine :

- *matični kontrolor USB-a* (host controller) - služi da inicira transakcije
- *koreni hab* (root hub) - služi da uspostavi vezu kontrolera sa ciljnim uređajem



## 158. Sta je i kako funkcionise sistem prekida? Kakvi tipovi prekida postoje?

Sistem prekida predstavlja jedan od mehanizama za upravljanje tokom rada procesora.

Omogucavaju efikasniji rad racunara. Vecina osnovnih uredjaja je mnogo sporija od procesora.

Osnovne primene su komunikacija sa U/I uredjajima i upotreba osnovnih usluga OS-a.

Po nastupanju prekida, trenutno izvorsavani program se prekida i prelazi se na kontrolnu proceduru koja se naziva *rutina za obradu prekida* ili *opsluzilac prekida*. Kada se opsluzivanje prekida dovrši, program nastavlja sa izvorsavanjem od mesta na kome je prekinut.

Prekidi mogu biti hrdvreski ili softverski.

## 159. Objasniti hardverske prekide.

Proizvode se od strane spoljasnjih uredjaja. Nazivaju se i *neplanski* ili *asinhroni* prekidi. Sluze za *skretanje paznje* procesora na odgovarajuci uredjaj. Postoje dve vrste hardverskih prekida:

- nemaskirajuci - odmah se obradjuju od strane procesora. Prekid se izaziva dovodjenjem signala na nozicu NMI procesora. Procesor uvek odmah odreaguje na ovaj prekid . On se ne moze softverski maskirati . Njegov tip (broj) je 2. Svi ostali su maskirajuci.
- maskirajuci - mogu da se odloze do nekog specifcnog trenutka. Izazivaju se dovodjenjem signala na nozicu INTR procesora. Procesor reaguje samo ako je zastavica IF = 1. Moze se softverski maskirati

## 160. Objasniti softverske prekide.

Nazivaju se i *planski* ili *sinhroni* prekidi. Vecina procesora podrzava softverske prekide:

Intel x86 - instrukcija int , PowerPC - instrukcija sc (system call) , MIPS - instrukcija syscall.

Osnovne namene softverskih prekida su pristupanje U/I uredjajima i koriscenje OS-a ili BIOS-a.

## 161. Sta su izuzeci? Objasniti.

Posebna podvrsta hardverskih prekida su izuzeci. Mehanizam prekida se upotrebljava i za obradu prepoznatih neispravnosti pri izvorsavanju programa. Neki procesori prave razliku izmedju prekida i izuzetaka (Intel x86), a neki ne (MIPS, PowerPC).

## 162. Sta je vektor prekida? Kako se poziva opsluzilac prekida?

Ostvaruje se na dva osnovna nacina:

- Vektorski prekidi – svakom tipu prekida je dodeljenja adresa u memoriji . U slucaju nastupanja prekida, izvorsavanje se nastavlje od te adrese. Imaju ga Intel x86 i PowerPC
- Prekidi sa uzrokom – prekidi nastupaju uz oznacavanje uzroka prekida . Svi prekidi se obradjuju istim opsluziocem prekida . Opsluzilac proverava registar uzroka i na osnovu njegove vrednosti prepoznaje vrstu prekida i prenosi kontrolu odgovarajucem kodu OS-a . Ima ga MIPS

## 163. Objasniti sistem prekida Intel-ovih procesora. Vrste izuzetaka.

Prekidi se oznacavaju brojevima 0-255 . Adrese opsluzilaca se nalaze u tablici deskriptora prekida (IDT). Stavke tablice su velicine po 8 bajtova i predstavljaju 64-bitnu adresu opsluzioca. Tablica se moze nalaziti bilo gde u memoriji, a njena adresa mora biti upisana u registar procesora IDTR.

Postoje posebne instrukcije za punjenje i cuvanje tablice. Broj prekida se koristi kao indeks tablice.

- greske – javljaju se pri stanju koje je prethodilo izvorsavanju instrukcije koja je proizvela gresku . Nakon obrade prekida instrukcija ce biti ponovljena . Pr: promasaj stranice ili segmenta
- zamke – javljaju se sa uslovima nakon izvorsavanja instrukcije koja je proizvela gresku . Nakon obrade prekida, bice izvorsena naredna instrukcija. Pr: prekoracenje, korisnicki prekidi
- zaustavljanja – prijavljuju ozbiljne probleme, moguće je da ne može ni biti prepoznata instrukcija koja je proizvela prekid. Jedino sto opsluzilac moze da uradi jeste da prekine proces . Pr: hardverske neispravnosti, nekozistente vrednosti sistemskih tablica



## 164. Objasniti komponente i rad kontrolora prekida PIC 8259.

Ako je vise uredjaja u prilici da izaziva prekide, postoji mogucnost da vise prekida nastane u isto vreme. Problem je u tome sto se svi prekidi prijavljuju na isti nacin i putem istih signala. To se resava pomocu dodatnog uredjaja – kontrolera prekida. Kontroler prekida sluzi kao posrednik izmedju uredjaja i procesora pri izazivanju prekida.

### Kontroler prekida kod PIC 8259:

Moze da opsluzuje do 8 uredjaja. Svaki uredjaj ima po jednu liniju prema kontroleru, kojom pokusava da izazove prekid. Sa procesorom se povezuje preko linija INTR i INTA i magistrale podataka. U slucaju da vise uredjaja istovremeno zatrazi prekid, kontroler serijalizuje te prekide na osnovu prioriteta.

PIC 8259 ima dva registra - 8-bitni komandni registar (ICR) koji se koristi za programiranje kontrolera (podrazumevani prioriteti su od naviseg: 0, 1,..., 7) i 8-bitni registar maske (IMR) koji se koristi za dopustanje i onemogucavanje pojedinih prekida. Programiranje se ostvaruje komunikacijom procesora sa kontrolerom (putem magistrale podataka i adresa: 20H za registar ICR, 21H za registar IMR). Tipovi prekida se dodeljuju odredjivanjem najnizeg tipa (to je tip prekida 0, a ostali su u rasponu +1 do +7).

Blokiranje svih prekida se ostvaruje postavljanje zastavice IF = 0 . Blokiranje pojedinih prekida se ostvaruje postavljanjem registra IMR kontrolera (bit 0 - prekid je omogucen, bit 1 - prekid je onemogucen)

Svaki put pri zavrsetku obrade prekida koji su izazvani posredstvom kontrolera, neophodno je javiti kontroleru da je obrada zavrшена. To se radi posredstvom komandnog registra, slanjem koda 20H (naziva se EOI - end of interrupt), koji oznacava kraj obrade. Nakon toga kontroler moze da nastavi sa izazivanjem novih prekida.

### 165. Objasniti CISC arhitekturu.

CISC - procesori sa slozenim skupom instrukcija. Slozena arhitektura skupa instrukcija (ISA) (kodiranje sto slozenijih instrukcija u sto manje memorije). Raznovrsnost operacija. Raznovrsnost nacina adresiranja. Novi modeli procesora uvodili su sve vise i vise novih nacina adresiranja i novih instrukcija. Podrzan preveliki broj slozenih instrukcija – cak se dodaju neke instrukcije koje se nikada ne koriste pri programiranju na asmebleru, ali omogucavaju efikasnije prevodjenje programa pisanih na visim programskim jezicima. Otezano dekodiranje instrukcija.

### 166. Objasniti RISC arhitekturu.

RISC – procesori sa sredukovanim skupom instrukcija. Jednostavna arhitektura skupa instrukcija. Obezbedjivanje minimalnog skupa instrukcija i nacina adresiranja. Povecan broj registara koji se mogu koristiti za racunanje. Stalniji skup instrukcija. Jednostavno dekodiranje instrukcija. Skracivanje trajanja izvršavanja operacija. Jednostavnija implementacija procesora.

*Osnovni principi dizajna:*

Jednostavne operacije; operacije registar-u-registar; jednostavni nacini adresiranja (registarsko); veci broj registara; fiksna duzina i jednostavan format instrukcija

*Druge odlike:*

Udvajanje magistrale - posebno za podatke i instrukcije (Harvard arhitektura). Svi registri su ravnopravni po mogucnostima i performansama. Preklapanje izvršavanja instrukcija. Visoka propusnost sistemske magistrale.

### 167. Objasniti odnos arhitektura CISC i RISC.

RISC koncepti se razvijaju od 1975. godine. U aktuelnom stanju tehnologije proizvodnje procesora (CISC) bilo je skupo podrzavati slozene operacije na vecem broju registara, pa je zato postojalo malo registara. Vecina instrukcija je zahtevala slozenu implementaciju, pa i dugacke cikluse. RISC pristup pojednostavljuje implementacije i skracuje izvršavanje instrukcija u ciklusima. Slozena implementacija otezava podizanje radne frekvencije. RISC omogucava znacajno podizanje radne frekvencije.

Analize programa pokazuju da vecina instrukcija obavlja prenosenje podataka izmedju procesora i memorije. Implementacija slozenih instrukcija koje se retko izvršavaju znacajno usloznjava implementaciju procesora a donosi skromne dobitke u performansama. Od 1975. pa do sredine 1990. postoji tendencija da se proizvođači procesora priklanjaju RISC konceptima. Kasnije postepeno usavršavanje proizvodnih tehnologija smanjuje znacaj RISC arhitekture. Dolazi do spajanja elemenata arhitektura.

Danas su uobicajene arhitekture koje se odlikuju CISC odnosom prema skupu instrukcija, a imaju vecinu ostalih odlika RISC arhitektura - veliki broj registara, koji su prakticno ravnopravni; preklapanje izvršavanja instrukcija; napredne arhitekture kes memorija.

### 168. Po cemu je znacajan broj adresa u instrukcijama procesora?

Binarne operacije zahtevaju dva, a unarne jedan argument. Operacije najcesce imaju jedan izlaz, ali ih moze biti i vise. Uobicajena binarna operacija zahteva tri adrese: dve adrese argumenta, jednu adresu rezultata. Postoje procesori sa 3, 2, 1 adresom i bez adresa. Procesor koji podrzava neki broj adresa, obicno moze da podrzi i instrukcije sa manjim brojem adresa.

### 169. Sta su troadresni procesori? Primer i karakteristike.

Troadresni procesori eksplicitno adresiraju dva argumenta i rezultat operacije. Vecina savremenih procesora je troadresna.

*Primer instrukcija:*

add dest src1, src2 – sabiraju se vrednosti adresirane sa src1 i src2 i rezultat se smesta u dest

*Primer koda:*

Na troadresnom procesoru izraz :  $A=B+C*D-E+F+A$ , moze da se implementira kao:

```
mul T, C, D
add T, T, B
sub T, T, E
add T, T, F
add A, T, A
```

U praksi vecina instrukcija sadrzi ponovljenu jednu od adresa argumenata kao adresu rezultata. Skup instrukcija odgovara operacijama koje procesor moze da izvršava.

### 170. Sta su dvoadresni procesori? Primer i karakteristike.

Dvoadresni procesori eksplicitno adresiraju jedan argument i rezultat operacije.. Motivacija potice iz cinjenice da se relativno retko upotrebljavaju tri razlicite adrese. Procesori familije Intel x86 su dvoadresni.

*Primer instrukcija:*

add dest, src – sabiraju se vrednosti adresirane sa dest i src i rezultat se upisuje u dest

*Primer koda:*

Na dvoadresnom procesoru izraz :  $A=B+C*D-E+F+A$ , moze da se implementira kao:

```
load T, C
mul T, D
add T, B
sub T, E
add T, F
add A, T
```

U praksi vecina instrukcija ponavlja jednu istu ciljnu adresu. Skup instrukcija odgovara operacijama koje procesor moze da izvršava. Dodaje se instrukcija za prepisivanje podatka.

### 171. Sta su jednoadresni procesori? Primer i karakteristike.

Jednoadresni procesori eksplicitno adresiraju jedan argument. Rezultat se uvek upisuje u *akumulator* - jedini registar na kome mogu da se izvršavaju operacije. Motivacija potice iz cinjenice da se za vecinu operacija upotrebljava ponavljanje adresa cilja.

*Primer instrukcija:*

add addr – sabiraju se vrednost akumulatora i vrednost adresirana sa addr i rez. je u akumulatoru

*Primer koda:*

Na jednoadresnom procesoru izraz:  $A=B+C*D-E+F+A$ , moze da se implementira kao:

```
load C
mul D
add B
sub E
add F
add A
store A
```

Redukovan broj registara - samo jedan ima punu operativnu funkcionalnost. Skup instrukcija odgovara operacijama koje procesor moze da izvršava. Dodaju se instrukcije za prepisivanje podatka u akumulator i iz akumulatora.

## 172. Sta su bezadresni procesori? Primer i karakteristike.

Bezadresni procesori ne adresiraju nijedan argument eksplicitno, osim u posebnim instrukcijama koje stavljaju podatke na stek i uzimaju podatke sa steka. I argumenti i rezultat se uvek nalaze na steku. Motivacija potice iz cinjenice da je za vecinu operacija potrebno relativno malo podataka.

*Primer instrukcija:*

push addr – sadrzaj podatka adresiranog sa addr se stavlja na vrh steka

pop addr – podatak sa vrha steka se uklanja i upisuje na adresu addr

add – dva podatka sa vrha steka se sklanjaju sa steka i sabiraju, rezultat se stavlja na vrh steka

*Primer koda:*

Na bezadresnom procesoru izraz:  $A=B+C*D-E+F+A$ , moze da se implementira kao:

push E / C / D

mul

push B

add

sub

push F

add

push A

add

pop A

Ne postoje imenovani registri. Skup instrukcija odgovara operacijama koje procesor moze da izvršava. Dodaju se instrukcije za prepisivanje podataka na stek i sa steka. Obicno se implementiraju tako da se nekoliko poslednjih podataka na steku nalazi u *stek registrima* procesora. Broj stek registara se naziva *dubina steka*. Na taj nacin se znacajno ubrzavaju operacije sa stekom, zato sto nije potrebno pristupati memoriji.

## 173. Objasniti odnos performansi procesora i broja adresa.

Svaki od predstavljenih pristupa ima prednosti i mane. Sto se vise adresa navodi u instrukcijama broj pristupa memoriji je veci, zapis instrukcija je veci, programi se sastoje od manje instrukcija.

*Troadresni racunar* - svaka instrukcija zahteva 4 pristupa memoriji, jedan za instrukciju, dva za podatke, jedan za rezultat. 5 instrukcija → ukupno 20 pristupa memoriji.

*Dvoadresni racunar* - svaka operacija i dalje zahteva 4 pristupa memoriji, jedan za instrukciju, dva za podatke, jedan za rezultat. Instrukcija *load* zahteva tri pristupa. 5 operacija i jedno prepisivanje → ukupno 23 pristupa memoriji.

*Jednoadresni racunar* - svaka operacija zahteva 2 pristupa memoriji, jedan za instrukciju i jedan za podatke. Akumulator je registar, a ne memorija. Instrukcija *load* zahteva dva pristupa. 7 instrukcija → ukupno 14 pristupa memoriji.

*Bezadresni racunar* - svaka operacija zahteva 1 pristup memoriji, jedan za instrukciju.

Pretpostavljamo da je stek dovoljno dubok da je primer stao u stek registre.

Instrukcijw push i pop zahtevaju po dva pristupa. 5 operacija po 1 i 7

instrukcija push i pop → ukupno 19 pristupa memoriji.

## 174. Sta je arhitektura load/store?

Sve operacije se izvršavaju isključivo nad registrima procesora. Samo operacije *load* i *store* mogu da pristupaju memoriji.

RISC i vektorski procesori cesto koriste ovakvu arhitekturu. Znacajno se smanjuje velicina instrukcija. Znacajno se redukuje slozenost dekodiranja i implementiranja instrukcija. Omogucava se visok stepen preklapanja instrukcija. Duzina izvršavanja nije neposredno proporcionalna broju instrukcija i pristupa memoriji.

### 175. Objasniti arhitekturu registara procesora.

Registri procesora služe za čuvanje podataka, instrukcija i stanja procesora. Registri se dele na:

1. registre opšte namene - broj i vrsta registara opšte namene su obično povezani sa arhitekturom adresiranja. Bezadresni procesori ne zahtevaju registre opšte namene, mada imaju implicitne stek-registre. Kod dvo- i troadresnih procesora registri opšte namene nisu neophodni, uvode se zbog podizanja performansi. RISC procesori po pravilu imaju veći broj registara opšte namene.
2. posebne registre - primer registara posebne namene su registri za vođenje steka, brojac instrukcija, interni registar instrukcije koji sadrži tekucu instrukciju
  - posebni registri dostupni korisničkim programima
  - posebni registri rezervisani za sistemske potrebe

### 176. Objasniti kontrolu toka programa.

Brojac instrukcija (ili programski brojac) ima ulogu kontrolora toka. Sadrži adresu naredne instrukcije. Čim se instrukcija pročita, brojac se povećava tako da pokazuje na narednu instrukciju. Kod arhitekture sa fiksnom veličinom instrukcije, uvek se uvećava za fiksni broj. Program se u načelu izvršava sekvencijalno. Sekvencijalno izvršavanje se po potrebi može izmeniti grananjem i petljama.

### 177. Sta je grananje? Koje vrste grananja postoje?

Grananje se implementira instrukcijama grananja. One eksplicitno menjaju vrednost brojača instrukcija. Postoje dve vrste instrukcija grananja:

- beuslovne (ili eksplicitne) - eksplicitna i bezuslovna promena toka izvršavanja programa
- uslovne - eksplicitna promena toka izvršavanja programa u slučaju vazenja nekog navedenog uslova. Dve osnovne vrste uslovnog grananja su *postavi-pa-skoci* i *proveri-pa-skoci*

Granjanje može biti trenutno ili odloženo.

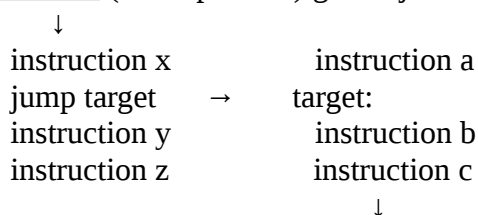
*Nacin navodjenja nove adrese:*

Nova adresa izvršavanja se navodi kao

- apsolutna - navodi se puna nova adresa; podržavaju praktično svi procesori
- relativna - navodi se razlika između nove i tekuće adrese; ne podržavaju svi procesori. Prednost je u pomerljivosti koda. Bez obzira na lokaciju u memoriji grananje je jednako ispravno. Ako razlika nije velika, ima kraći zapis nego navodjenje apsolutne adrese.

### 178. Objasniti bezuslovno grananje. Primer.

Beuslovno (ili eksplicitno) grananje - eksplicitna i bezuslovna promena toka izvršavanja programa.



### 179. Objasniti uslovno grananje, postavi-pa-skoci, proveri-i-skoci. Registri stanja.

Uslovno grananje - eksplicitna promena toka izvršavanja programa u slučaju vazenja nekog navedenog uslova. Dve osnovne vrste uslovnog grananja su

- *postavi-pa-skoci* - osnovna ideja je da se razdvoje proveravanje uslova i grananje. Najpre se posebnim instrukcijama proveravaju uslovi ili drugacije postavljaju odgovarajuće stanje procesora. Zatim se instrukcijama grananja samo proverava stanje procesora i po potrebi izvršava promena brojača instrukcija. Kod familije procesora Intel x86.

- *proveri-i-skoci* - osnovna ideja je da jedna ista instrukcija proverava uslov i promeni adresu. Ovakav vid grananja primenjen je kod većeg broja procesora, uključujući MIPS.

### Registri stanja:

Da bi bilo moguće implementirati grananje *postavi-pa-skoci*, neophodno je da procesor ima *registar stanja* koji čuva rezultate poredjenja i drugih operacija. Ovakvi registri se nazivaju i registri kodova poredjenja. Ovakve registre imaju Intel x86, SPARC, PowerPC; nema MIPS.

## **180. Objasniti pozivanje procedura.**

Grananja su jednosmerne promene toka izvršavanja. Dejstvo je ograničeno na jednu promenu toka izvršavanja.

Pozivanje procedura su dvosmerne promene, nakon inicijalnog pozivanja, kasnije sledi povratak na mesto odakle je izvršeno pozivanje. Da bi povratak bio moguć neophodne su dve stvari:

- *eksplicitna oznaka kraja procedure* (naredba *return*). Intel Pentium ima instrukciju *ret*. MIPS ima instrukciju *jr*
- *adresa povratka* - mora biti sačuvana pri pozivanju procedure

## **181. Objasniti tehnike cuvanja adrese povratka iz procedure i tehnike prenošenja parametara.**

### Cuvanje adrese povratka:

Može se čuvati u registru (MIPS) ili na steku (Intel x86). Čuva se adresa instrukcije pozivanja (SPARC) i adresa prve instrukcije nakon instrukcije pozivanja.

### Cuvanje adrese povratka u registrima:

Može se koristiti registar posebno namenjen za to ili bilo koji registar.

Na primer MIPS. Primer povratka, uz pretpostavku da je adresa povratka u registru \$ra: *jr \$ra*  
U slučaju rekurzije i pozivanja potprocedura je neophodno dodatno staranje o adresama povratka.

### Dve osnovne tehnike prenošenja parametara:

- *pomocu registrara* - parametri se zapisuju u registrima pre pozivanja. Ovaj metod je brzi. Ne omogućava rekurziju. Često zahteva čuvanje prethodnih vrednosti registrara. Koriscen kod RISC procesora (MIPS, PowerPC)
- *pomocu steka* - parametri se postavljaju na stek pre pozivanja. Fleksibilniji nacin. Manje efikasan. Koriscen kod CISC procesora (Intel x86)

## **182. Objasniti tipove operanada procesora.**

Uobičajeno je da procesori prepoznaju samo elementarne tipove podataka (*char*, *int*, *float*). Često iste instrukcije rade sa podacima razlicite velicine, u zavisnosti od nacina navodjenja operanada.

Na primer (Intel x86):  
*mov AL, addr;*      prepisuje 8-bitni podatak  
*mov AX, addr;*      prepisuje 16-bitni podatak  
*mov EAX, addr;*      prepisuje 32-bitni podatak

U slučaju RISC procesora uobičajeno je da se iz notacije instrukcije prepozna velicina operanda.

Na primer:  
*lb Rdest, address ;*      prepisuje 8-bitni podatak (bajt)  
*lh Rdest, address ;*      prepisuje 16-bitni podatak (pola reci)  
*lw Rdest, address ;*      prepisuje 32-bitni podatak (rec)  
*ld Rdest, address ;*      prepisuje 64-bitni podatak (dvostruka rec)

## **183. Objasniti osnovne nacine adresiranja koje podrzavaju svi procesori.**

Nacini adresiranja opisuju kako se određuje operand instrukcije. Operandi mogu biti konstante (rezim neposrednog adresiranja), u registrima (rezim registarskog adresiranja) i u memoriji (rezim memorijskog adresiranja). Svi procesori podržavaju bar dva osnovna nacina adresiranja:

- *rezim neposrednog adresiranja* - navodjenje konstantne vrednosti operanda. Ne postoji pristupanje memoriji, osim citanja instrukcije.
- *rezim registarskog adresiranja* - navodjenje registra koji sadrži vrednost operanda. Ne postoji pristupanje memoriji osim citanja instrukcije.

### 184. Objasniti razlike u načinu adresiranja između RISC i CISC procesora.

Razlika između RISC i CISC procesora je u podržanim načinima adresiranja podataka u memoriji. RISC procesori koriste arhitekturu *load/store*. Sve instrukcije, osim *load* i *store*, podržavaju samo neposredno i registarsko adresiranje. Podaci koji su u memoriji mogu se adresirati samo u okviru instrukcija *load* i *store*. Broj načina adresiranja je obično sasvim skroman. CISC procesori podržavaju mnoštvo načina adresiranja. Uobičajeno je da sve instrukcije podržavaju adresiranje podataka u memoriji. Broj načina adresiranja je obično veliki.

### 185. Objasniti instrukcije mikroprocesora za premještanje podataka.

Instrukcije za premještanje podataka podržavaju svi procesori. Dele se na instrukcije koje premestaju podatke između memorije i registara i između registara.

Kod RISC procesora je premještanje podataka između procesora i memorije strogo ograničeno na instrukcije *load* i *store*. Neki od RISC procesora ne omogućavaju neposredno premještanje podataka između registara, već samo u okviru drugih instrukcija (npr. sabiranje). Na primer:

```
add Rdest, Rsource, 0    /* Rdest = Rsource + 0 */
```

Kod CISC procesora se umesto dve obično implementira samo jedna instrukcija za premještanje podataka koja ravnopravno tretira registre i memoriju. Na primer, kod Intel x86:

```
mov dest, src, najviše jedan od argumenata može biti u memoriji.
```

### 186. Objasniti aritmetičke i logičke instrukcije mikroprocesora.

Aritmetičke instrukcije obuhvataju kako celobrojne tako i operacije u pokretnom zarezu. Većina procesora podržava bar 4 osnovne aritmetičke operacije. Logičke instrukcije podrazumevaju skup operacija na nivou bitova. Praktično svi procesori podržavaju *and* i *or*. Većina procesora podržava *not* i *xor*.

### 187. Objasniti instrukcije mikroprocesora za kontrolu toka programa.

Instrukcije za kontrolu toka programa su instrukcije grananja i instrukcije za pozivanje procedura (tu spadaju i instrukcije za vraćanje iz procedura).

### 188. Objasniti ulazno/izlazne instrukcije mikroprocesora.

Ulazno/izlazne instrukcije se značajno razlikuju između procesora. One postoje samo kod procesora koji podržavaju izolovano preslikavanje ulaza i izlaza. Ako procesor podržava samo memorijsko preslikavanje ulaza/izlaza, onda nema ove instrukcije. Uobičajene instrukcije:

```
in Reg, io port i out io port, Reg
```

Velicina instrukcije zavisi od podržane duzine adrese porta.

### 189. Koji su osnovni formati instrukcija?

Format instrukcije podrazumeva način kodiranja tj. binarnog zapisivanja instrukcije. Dva osnovna tipa formata instrukcija su:

- *format fiksne duzine instrukcija* – uobičajen za RISC procesore. MIPS, PowerPC, Sparc imaju instrukcije duzine 32 bita
- *format promenljive duzine instrukcija* – uobičajen za CISC procesore (Intel x86)

### 190. Nabrojati osnovne faze pri izvršavanju instrukcija mikroprocesora.

Da bi procesor mogao da izvrši instrukciju potrebno je da:

- adresira i pročita instrukciju iz memorije
- dekodira instrukciju
- adresira i pročita potrebne argumente
- izvrši odgovarajuću operaciju
- adresira i zapise izračunat rezultat



### 191. Objasniti namenu i nacin upotrebe pomocnih registara A i C pri izvršavanju instrukcija procesora sa jednom internom magistralom.

Zbog toga sto postoji samo jedna interna magistrala A, potrebni su pomocni registri:

- registar A cuva vrednost prvog operandu dok se drugi adresira, uvek je dostupan za citanje
  - registar C cuva rezultat dok se ne prenese dalje, uvek je dostupan prethodni rezultat za pisanje
- Implementiraju se pomocu D flip-flopova.

### 192. Objasniti ulogu, nacin povezivanja, nacin upotrebe i implementaciju registra PC i registra IR memorijskog interfejsa u slucaju jedne interne magistrale. Implementacija putanje podataka.

Registar PC je brojac instrukcija. Sadrzi adresu naredne instrukcije. Sadržaj stavlja na sistemsku adresnu magistralu radi citanja instrukcije iz memorije. Sadržaj stavlja na magistralu A radi omogucavanja relativnih skokova i pozivanja procedura. Moze simultano da ide na obe magistrale.

Registar IR - registar instrukcije. Sadrzi instrukciju koja se trenutno izvršava. Kontrolni signal IRbin postavlja vrednost registra, a kontrolni signal IRout stavlja vrednost registra na magistralu A.

### 193. Objasniti ulogu, nacin povezivanja, nacin upotrebe i implementaciju registra MAR i registra MDR memorijskog interfejsa u slucaju jedne interne magistrale. Implementacija putanje podataka.

Registar MAR - registar memorijske adrese. Sadrzi adresu operandu koji je u memoriji. Koristi se pri adresiranju podataka koji su u memoriji. Radi slicno registru PC: kontrolni signal MARin postavlja vrednost registra; kontrolni signal MARout stavlja vrednost registra na magistralu A; kontrolni signal MARbout stavlja vrednost registra na sistemsku adresnu magistralu.

Registar MDR - registar memorijskog podatka. Sadrzi vrednost operandu koji je u memoriji. Koristi se pri citanju operandu iz memorije. Adresa operandu je u registru MAR. Ima dvosmeran interfejs: kontrolni signal MDRin postavlja vrednost registra sa magistrale A; kontrolni signal MDRbin postavlja vrednost registra sa sistemske magistrale podataka; kontrolni signal MDRout stavlja vrednost na magistralu A; kontrolni signal MDRbout stavlja vrednost registra na sistemsku magistralu podataka.

### 194. Objasniti nacin povezivanja, nacin upotrebe i implementaciju registara opste namene u slucaju jedne interne magistrale. Dati i objasniti korake izvršenja instrukcije sabiranja.

Registri opste namene su vezani samo na internu magistralu A. Svaki od registara Gx ima po dva kontrolna signala: Gxin i Gxout.

Primer instrukcije (sabiranje): add %G9, %G5, %G7 /\*  $G9 = G5 + G7$  \*/

Najpre se sadržaj jednog registra mora sacuvati u pomocnom registru A, a zatim sabirati sa drugim.

K1: prepisujemo G5 u pomocni registar A. Postavlja se signal da bi se sadržaj registra G5 stavio na magistralu A. Postavlja se signal Ain da bi se podaci sa magistrale A upisali u pom. registar A

K2: stavljamo G7 na magistralu A i racunamo. Postavlja se signal G7out da bi se sadržaj registra G7 stavio na magistralu A. Postavlja se signal Cin da bi se rezultat upisao u registar C. AL jedinica se nalaze da izracuna rezultat.

K3: rezultat upisujemo u registar G9. Postavlja se signal G9in da bi se sadržaj registra G9 procitao sa magistrale A. Postavlja se signal Cout da bi se sadržaj registra C stavio na magistralu A



## 195. Opisati korake citanja jedne instrukcije mikroprocesora i signale kojima se implementiraju ti koraci.

Citanje instrukcija obuhvata:

- K1: Adresiranje instrukcije. Postavlja se signal PCbout da bi se sadržaj registra PC stavio na sistemsku adresnu magistralu. Postavlja se signal PCout da bi se sadržaj registra PC stavio na magistralu A. AL jedinici se daje instrukcija add4 da bi sabrala vrednost na svom ulazu B sa 4. Postavlja se signal Cin da bi se rezultat (PC+4) upisao u pomocni registar C.
- K2: Upisivanje nove adrese instrukcije. Cekamo (bar) jedan ciklus da memorija procita i dostavi instrukciju. Postavlja se signal Cout da bi se rezultat (PC+4) stavio na magistralu A. Postavlja se signal PCin da bi se rezultat (PC+4) sa magistrale A upisao u registar PC.
- K3: Citanje instrukcije. Postavlja se signal IRbin da bi se procitana instrukcija stavila u pomocni registar IR.

## 196. Sta je i kako se implementira postupak dekodiranja instrukcija procesora?

Dekodiranje instrukcije je zajednicko za sve vrste instrukcija. Sledi neposredno posle citanja instrukcije. U fazi dekodiranja prepoznaju se: kod instrukcije, broj i tipovi operanada i nacin adresiranja operanada.

## 197. Sta je i kako se moze implementirati kontrolna jedinica procesora?

Instrukcije se mogu opisati konacnim automatima. Svaki od konacnih automata se moze implementirati hardverski ili softverski.

## 198. Objasniti hardversku implementaciju kontrolne jedinice.

Ulaze u PLA (programabilni nizovi logickih elemenata) cine tri grupe signala:

- *kod operacije* (opcode) - operacija koja se implementira
- *statusni i uslovni kodovi* - neophodni za implementaciju uslovnih grananja i nekih AL operacija
- *casovnik* - služi za brojanje koraka i sinhronizaciju aktivnosti

## 199. Kada se bira softverska, a kada hardverska implementacija?

Ako su instrukcije jednostavne, onda je hardverska implementacija prihvatljivija varijanta - RISC. U slucaju slozenih instrukcija hardverska implementacija nije dobar izbor - CISC, tada se signalima upravlja programski.

## 200. Objasniti softversku implementaciju kontrolne jedinice. Sta je mikroinstrukcija, a sta mikrokod?

Pocetkom 1950-ih godina je prvi put predloženo da se kontrola izvršavanja instrukcija implementira softverski [Wilkes, Stringer]. Programski kod instrukcije se naziva mikrokod.

Ideja je da se konacni automat transformise u mikroinstrukcije koje upravljaju postavljanjem signala adom procesora.

## 201. Sta je uprosćena organizacija mikrokoda?

Linearna organizacija mikrokoda pocinje od citanja instrukcije. Nakon dekodiranja instrukcije nastavlja se od mikrorutine koja odgovara kodu operacije. Izvršavanje mikrorutine je sekvencijalno. Kada se dodje do signala *end*, ponavlja se citanje (naredne) instrukcije.

## 202. Sta je složena organizacija mikrokoda?

Linearna organizacija mikrokoda ima za posledicu da se neki zajednicki delovi mikrokoda ponavljaju vise puta. Slozena organizacija mikrokoda podrazumeva da se zajednicki delovi efikasnije organizuju u mikrorutine sa grananjima – svaka mikroinstrukcija se prosirojuje adresom naredne mikroinstrukcije. Zbog toga se povecava kontrolna rec, ali se stedi na duzini mikroprograma.

### 203. Objasniti horizontalni format mikroinstrukcija mikroprocesora.

Svakom signalu odgovara po jedan bit. Nema kodiranja i dekodiranja signala.

Prednosti – omogućava navodjenje velikog broja signala u jednoj instrukciji.

Mane - velicina mikroinstrukcije.

### 204. Objasniti vertikalni format mikroinstrukcija mikroprocesora.

Na duzini se moze uštedeti ako se bitovi kodiraju. Npr. umesto 64 bita za kontrolisanje 32 registra, dovoljno je 5 bitova za identifikovanje registra i 1 bit za izbor signala.

Prednosti - krace mikroinstrukcije.

Mane – u jednom koraku se moze navesti samo jedan registar (neophodno je prepisati sadrzaj jednog registra u nekoliko drugih)

### 205. Uporedite puteve podataka pri izvršavanju instrukcija procesora sa jednom, dve i tri interne magistrale.

Osnovna slabost puta podataka sa *jednom* internom magistralom je multipleksiranje te magistrale i veci broj koraka u implementaciji instrukcija. Put podataka moze imati i vise internih magistrala:

- sa 2 interne magistrale: magistrala A je za jedan operand, a magistrala C za rezultat
- sa 3 interne magistrale: magistrala A za jedan operand, magistrala B za drugi operand, magistrala C za rezultat