1. *Написати истинитоносне таблице основних логичких везника (НЕ, И, ИЛИ).*

x	$\overline{x}$
0	1
1	0

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

2. *Написати истинитоносне таблице изведених логичких везника (НИ, НИЛИ, ЕИЛИ).*

НИ:

x	y	$x \uparrow y$
0	0	1
0	1	1
1	0	1
1	1	0

НИЛИ:

x	y	$x \downarrow y$
0	0	1
0	1	0
1	0	0
1	1	0

ЕИЛИ:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

3. *Навести бар један начин на који се ЕИЛИ везник може представити помоћу основних логичких везника (НЕ, И, ИЛИ).*

$$x \oplus y = x\,\overline{y} + \overline{x}\,y$$

4. *Навести основне законе алгебре логике.*

   Аксиоме: асоцијативност, комутативност и дистрибутивност сабирања и множења (тј. И и ИЛИ), постојање неутралних елемената за сабирање и множење и закон комплементарности.
   Додатни закони алгебре логике: закон идемпотенције, закон апсорпције, закон двојне негације, Де Морганови закони и закони нуле и јединице.

5. *Због чега се алгебра логике користи као основа савремених рачунара?*

   Због једноставности израде електронских уређаја који имају само два стабилна стања, као и због једноставног изражавања бинарне аритметике преко логичких операција.

6. *Шта значи да су два логичка израза еквивалентна?*

   Два логичка израза су еквивалентна ако имају исте вредности за сваку валуацију, где је валуација функција која свим логичким променљивим у изразу додељује неку истинитосну вредност.

7. *Дефинисати појмове елементарне конјункције и дисјунктивне нормалне форме (ДНФ). Шта је савршена елементарна конјункција, а шта савршена ДНФ?*

Елементарна конјункција је израз који је конјукција литерала, тј. логичких променљивих или њихових негација. ДНФ је израз који се састоји од дисјункције елементарних конјункција. Елементарна конјункција је савршена ако садржи тачно један литерал за сваку логичку променљиву из посматраног скупа. ДНФ је савршена ако се састоји од савршених елементарних конјункција.

8. *Дефинисати појмове елементарне дисјункције и конјунктивне нормалне форме (КНФ). Шта је савршена елементарна дисјункција, а шта савршена КНФ?*

   Елементарна дисјункција је израз који је дисјункција литерала, тј. логичких променљивих или њихових негација. КНФ је израз који се састоји од конјункције елементарних дисјункција. Елементарна дисјункција је савршена ако садржи тачно један литерал за сваку логичку променљиву из посматраног скупа. КНФ је савршена ако се састоји од савршених елементарних дисјункција.

9. *Укратко описати поступак за свођење логичког израза на ДНФ.*

   Прво елиминишемо све примене логичких везника над 0 и 1, рачунајући те делове израза. Потом све негације сведемо на негације једне променљиве применом Де Морганових закона и закона двојне негације. На крају, применом дистрибутивних закона, добијамо ДНФ.

10. *Шта је логичка функција и колико има логичких функција реда n?*

    Логичка функција је било које пресликавање облика $f:\{0,1\}^n \to \{0,1\}$, тј. функција над n логичких променљивих. Логичких функција реда n има 2^{2^n}.

11. *Шта је потпун систем везника? Навести бар три примера потпуних система логичких везника.*

    Потпун систем везника је скуп везника помоћу којих се могу изразити све логичке функције. Примери: (НЕ, И, ИЛИ), (НЕ, И), (НЕ, ИЛИ), (НИ), (НИЛИ).

12. *Изразити НЕ, И и ИЛИ везник помоћу НИ везника.*

$$\overline{x} = \overline{xx} = x \uparrow x$$
$$xy = \overline{\overline{xy}} = \overline{\overline{xy} \cdot \overline{xy}} = (x \uparrow y) \uparrow (x \uparrow y)$$
$$x + y = (x \uparrow x) \uparrow (y \uparrow y)$$

13. *Укратко објаснити како се прозвољна логичка функција може изразити у облику израза у савршеној дисјунктивној нормалној форми.*

    За комбинацију вредности логичких променљивих за које је вредност логичке функције

1 додајемо у дисјункцију савршену елементарну конјукцију која за литорале има логичке променљиве које имају вредност 1 у посматраној комбинацији, односно негације логичких променљивих које имају вредност 0 у посматраној комбинацији.

14. *Шта је минимизација логичких израза и због чега нам је значајна?*

Минимизација логичког израза представља тражење логичког израза еквивалентног са датим, а који ће у себи да садржи најмањи могући број логичких везника. Минимизација је значајна због уштеда приликом производње и употребе рачунара.

15. *На примеру објаснити метод алгебарских трансформација за минимизацију логичких израза.*

Посматрајмо следећи пример:

$$F(x,y,z) = \overline{x}\,\overline{y}\,\overline{z} + \overline{x}\,\overline{y}\,z + \overline{x}\,yz + x\,\overline{y}\,zF(x,y,z)$$
$$= \overline{x}\,\overline{y}\,\overline{z} + \left(\overline{x}\,\overline{y}\,z + \overline{x}\,\overline{y}\,z + \overline{x}\,\overline{y}\,z\right) + \overline{x}\,yz + x\,\overline{y}\,zF(x,y,z)$$
$$= \overline{x}\,\overline{y}\left(\overline{z} + z\right) + \overline{x}\,z\left(\overline{y} + y\right) + \overline{y}\,z\left(\overline{x} + x\right)F(x,y,z) = \overline{x}\,\overline{y} + \overline{x}\,z + \overline{y}\,z$$

16. *Објаснити начин употребе Карноових мапа за минимизацију логичких израза. Пример.*

Циљ је да групишемо јединице у Карноовој мапи. Дакле, свака јединица треба да буде заокружена, док нуле не смеју бити заокружене. Дозвољено је више пута заокружити неку јединицу. Групе које се заокружују морају у себи имати број јединица који је једнак неком степену двојке. Заокружују се што веће групе. Након што су све јединице заокружене, треба проверити да ли ће све јединице остати заокружене иако се избаце неке од група. На крају, свака група представља једну елементарну конјукцију у минималној ДНФ.
Пример: У примеру $F(x,y,z) = \overline{x}\,\overline{y}\,\overline{z} + \overline{x}\,\overline{y}\,z + \overline{x}\,yz + x\,\overline{y}\,z$ групе представљају: $\overline{x}\,\overline{y}\,\overline{z}$ и $\overline{x}\,\overline{y}\,z$, $\overline{x}\,\overline{y}\,z$ и $\overline{x}\,yz$, $x\,\overline{y}\,z$ и $\overline{x}\,\overline{y}\,z$.

17. *Објаснити методу Квин—Мекласког за минимизацију логичких израза. Пример.*

Најпре се савршене елементарне конјункције из дате ДНФ сортирају растуће по броју неинвертованих литерала и поделе у класе по броју неинвертованих литерала. Затим се кроз итерације групишу елементарне конјункције тако што се пролази кроз суседне класе и ако се неке конјункције разликују за један литерал, оне се означавају као покривене, а њихова дисјункција се преноси у следећу итерацију. Са итерацијама се наставља све док је могуће извршити бар једно груписање. Када се заврши са итерацијама све конјункције које су остале непокривене означавају се као прости импликанти, који се даље преносе у другу фазу алгоритма. У другој фази правимо табелу чије су колоне савршене елементарне конјункције из дате ДНФ, а врсте су пренети прости импликанти. Означимо са + поља табеле где је прост импликант садржан у одговарајућој савршеној конјункцији. Ако се у некој колони налази само један плус, прост импликант из врсте у којој је тај плус називамо битним. Битни прости импликанти

ће се сигурно наћи у коначној минималној форми. Сада треба пронаћи који се још прости импликанти морају укључити да би се покриле савршене конјункције које нису покривене битним простим импликантима.
Пример: У примеру $F(x,y,z) = \overline{x}\ \overline{y}\ \overline{z} + \overline{x}\ \overline{y}\ z + \overline{x}\ yz + x\ \overline{y}\ z$ прва фаза алгоритма се може описати наредном табелом:

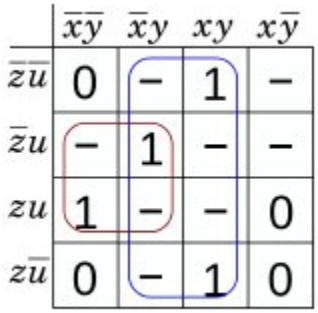
0	$\overline{x}\ \overline{y}\ \overline{z} \rightarrow$	$\overline{x}\ \overline{y}$
1	$\overline{x}\ \overline{y}\ z \rightarrow$	$\overline{x}\ z\ \overline{y}\ z$
2	$\overline{x}\ yz$ $\rightarrow x\ \overline{y}\ z$ $\rightarrow$	

Друга фаза алгоритма се може описати наредном табелом:

	$\overline{x}\ \overline{y}\ \overline{z}$	$\overline{x}\ \overline{y}\ z$	$\overline{x}\ yz$	$x\ \overline{y}\ z$
$\overline{x}\ \overline{y}$	+	+		
$\overline{x}\ z$		+	+	
$\overline{y}\ z$		+		+

Из прве, треће и четврте колоне видимо да су сви прости импликанти битни, а они покривају и другу савршену конјункцију, па се на крају добија $F(x,y,z) = \overline{x}\ \overline{y} + \overline{x}\ z + \overline{y}\ z$.

18. *Како се употребљавају Карноове мапе у присуству небитних вредности? Пример.*



Опет је потребно заокружити све јединице и није дозвољено заокруживање нула, али је сада дозвољено заокружити и небитне вредности ако ће то допринети већим заокруживањима. С друге стране, није потребно заокруживати небитне вредности.

19. *Како се метод Квин—Мекласког користи у присуству небитних вредности? Пример.*

Небитне вредности се у првој фази третирају као јединице, тј. додају се у класе, а другој фази се третирају као нуле, тј. не уписују се у табелу као колоне.

1	$\overline{x}\,\overline{y}\,\overline{z}u\surd$ $\overline{x}y\overline{z}\,\overline{u}\surd$ $x\overline{y}\,\overline{z}\,\overline{u}\surd$	$\overline{x}\,\overline{y}u\surd$ $\overline{x}\,\overline{z}u\surd$ $\overline{y}\,\overline{z}u\surd$ $\overline{x}y\overline{z}\surd$ $\overline{x}y\overline{u}\surd$ $y\overline{z}\,\overline{u}\surd$ $x\overline{y}\,\overline{z}\surd$ $x\overline{z}\,\overline{u}\surd$	$\overline{x}u$ $\overline{z}u$ $\overline{x}y\surd$ $y\overline{z}\surd$ $y\overline{u}\surd$ $x\overline{z}$	y
2	$\overline{x}\,\overline{y}zu\surd$ $\overline{x}y\overline{z}u\surd$ $\overline{x}yz\overline{u}\surd$ $x\overline{y}\,\overline{z}u\surd$ $xy\overline{z}\,\overline{u}\surd$	$\overline{x}zu\surd$ $\overline{x}yu\surd$ $y\overline{z}u\surd$ $\overline{x}yz\surd$ $yz\overline{u}\surd$ $x\overline{z}u\surd$ $xy\overline{z}\surd$ $xy\overline{u}\surd$	$yu\surd$ $yz\surd$ $xy\surd$	
3	$\overline{x}yzu\surd$ $xy\overline{z}u\surd$ $xyz\overline{u}\surd$	$yzu\surd$ $xyu\surd$ $xyz\surd$		
4	$xyzu\surd$			

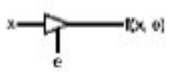
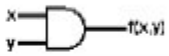
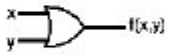
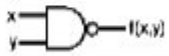
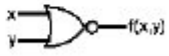
	$\overline{x}\,\overline{y}zu$	$\overline{x}y\overline{z}u$	$xy\overline{z}\,\overline{u}$	$xyz\overline{u}$
$\overline{x}u$	$\oplus$	$\boxplus$		
$\overline{z}u$		$+$		
$x\overline{z}$			$+$	
y		$\boxplus$	$\boxplus$	$\oplus$

20. *Шта је Петриков метод и која је његова улога у оквиру методе Квин—Мекласког? Навести пример.*

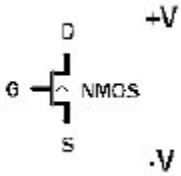
    Петриков метод служи за проналазак најмањег могућег подскупа простих импликаната које треба додати на битне да би се покриле све савршене конјункције. Најпре се из табеле из друге фазе обришу врсте у којима су битни прости импликанти, као и колоне у којима су покривене савршене конјункције. Сада i-том преосталом простом импликанту додељујемо логичку променљиву p_i, а j-тој преосталој колони додамо дисјункцију D_j која садржи све оне p_i где i-ти прост импликант покрива j-ту савршену конјункцију. Најзад, правимо КНФ облика $K=D_1D_2...D_m$, где је m број колона табеле. Сада се ова КНФ применом дистрибутивних закона пребаци у ДНФ у којој треба уочити елементарну конјункцију са најмањим бројем логичких променљивих. Ако таквих има више, бирамо ону чији одговарајући прости импликанти садрже најмањи број конјункција. Најзад, скуп простих импликаната који одговарају логичким променљивама у уоченој елементарној конјункцији представљају тражени скуп. Пример: Скрипта, страна 32-33.

21. *Елементарна логичка кола (гејтови) и њихове шематске ознаке.*

Елементарна логичка кола су: бафер, бафер са три стања, НЕ коло, И коло, ИЛИ коло, НИ коло, НИЛИ коло, ЕИЛИ коло и НЕИЛИ коло. Шематске ознаке:

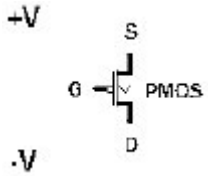
Назив кола	Функција кола	Шематска ознака
Бафер (енгл. *buffer*)	$f(x) = x$	
Бафер са три стања (енгл. *three-state buffer*)	$f(x,e) = \begin{cases} x, & \text{за } e = 1 \\ \mathbf{Z}, & \text{за } e = 0 \end{cases}$	
НЕ коло (енгл. *NOT*)	$f(x) = \overline{x}$	
И коло (енгл. *AND*)	$f(x,y) = x \cdot y$	
ИЛИ коло (енгл. *OR*)	$f(x,y) = x + y$	
НИ коло (енгл. *NAND*)	$f(x,y) = x \uparrow y$	
НИЛИ коло (енгл. *NOR*)	$f(x,y) = x \downarrow y$	
ЕИЛИ коло (енгл. *XOR*)	$f(x,y) = x \oplus y$	
НЕИЛИ коло (енгл. *XNOR*)	$f(x,y) = x \sim y$	

22. *Нацртати симбол и објаснити функцију NMOS транзистора.*



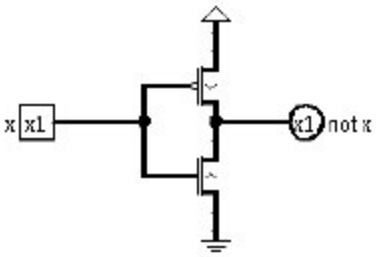
Код NMOS транзистора сорс је повезан на негативан напон, а дрејн на позитиван. NMOS транзистор проводи када је на гејту логичка јединица, а не проводи када је на гејту логичка нула.

23. *Нацртати симбол и објаснити функцију PMOS транзистора.*

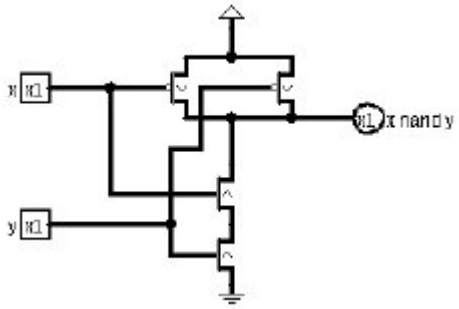


Код PMOS транзистора сорс је повезан на позитиван напон, а дрејн на негативан. PMOS транзистор проводи када је на гејту логичка нула, а не проводи када је на гејту логичка јединица.

24. *Имплементација НЕ кола у CMOS-у.*



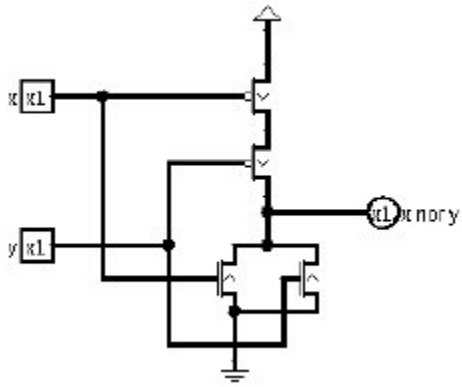
25. Имплементација НИ и И кола у CMOS-у.



НИ коло се имплементира тако што је у горњој мрежи паралелна веза два PMOS транзистора, док је у доњој мрежи редна веза два NMOS транзистора. Када су на улазу обе јединице, PMOS транзистори не проводе, док NMOS транзистори проводе, па ће на излазу бити нула. Иначе, неки од PMOS транзистора неће проводити, а неки од NMOS хоће, па ће на излазу бити јединица.

И коло се имплементира као комбинација НИ кола и НЕ кола, тј. надовеже се НЕ коло на излаз НИ кола.

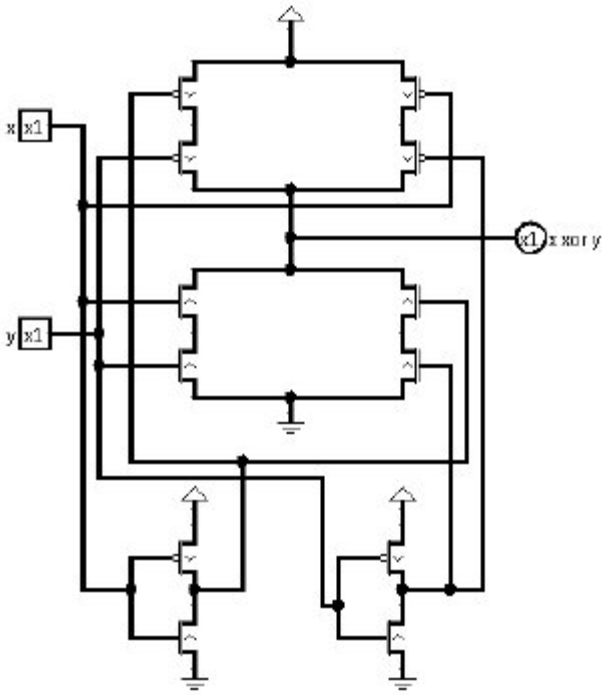
26. Имплементација НИЛИ и ИЛИ кола у CMOS-у.



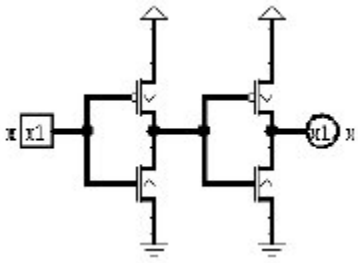
НИЛИ коло се имплементира тако што је у горњој мрежи редна веза два PMOS транзистора, док је у доњој мрежи паралелна веза два NMOS транзистора. Када су на улазу обе нуле, PMOS транзистори проводе, док NMOS транзистори не проводе, па ће на излазу бити јединица. Иначе, неки од PMOS транзистора ће проводити, а неки од NMOS неће, па ће на излазу бити нула.

ИЛИ коло се имплементира као комбинација НИЛИ кола и НЕ кола, тј. надовеже се НЕ коло на излаз НИЛИ кола.

27. Имплементација ЕИЛИ кола у CMOS-у.



28. *Пропусни транзистори и преносне капије. Функција и улога.*



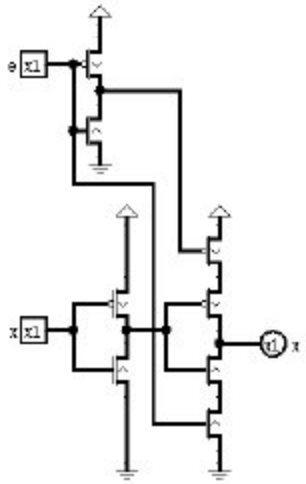
Пропусни транзистори служе да се неки сигнал преноси из једне тачке кола у другу у зависности од вредности неког другог сигнала Пропусни транзистори представљају NMOS или PMOS транзистор са контролним сигналом повезаним на гејт. Преносна капија представља комбинацију ова два. Оваква имплементација преносне капије се може користити у баферу са три стања.

29. *Шта је бафер са три стања и чему служи?*

Бафер са три стања је логичко коло које пропушта неки сигнал у зависности од вредности неког контролног сигнала.

30. *Имплементација бафера са три стања у CMOS-у.*

Бафер са три стања се у CMOS-у имплементира као преносна капија, уз евентуално НЕ коло ако није одмах доступан инфертовани контролни сигнал. Евентуално се пре преносне капије може додати обичан бафер (два НЕ кола) ако је потребно појачати сигнал.



31. *Шта је вредност високе импедансе и која је њена улога у логичким колима?*

    Вредност високе импедансе се може посматрати као да неко логичко коло не даје вредност на излазу. Ова вредност не утиче на друге вредности у колу. Ова вредност је омогућена да би се спречили кратки спојеви, а самим тим и кварови на колима.

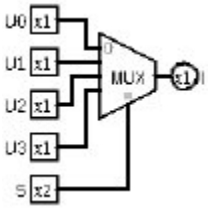
32. *Шта је комбинаторно коло?*

    Комбинаторно коло је логичко коло код којег се вредности на излазима у сваком тренутку могу изразити као логичке функције од вредности улаза у том истом тренутку.

33. *Навести најважније врсте основних комбинаторних кола.*

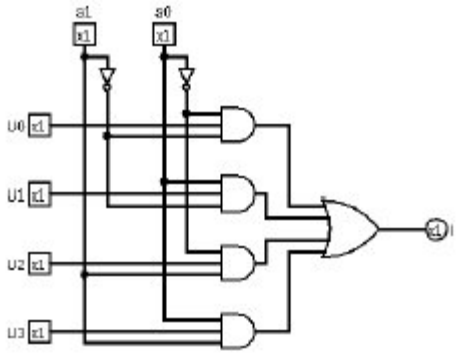
    Мултиплексер, демултиплексер, декодер, кодер.

34. *Шта је мултиплексер и која му је основна функција? Представити графичким симболом и таблицом мултиплексер 4-1.*



Мултиплексер је комбинаторно коло које омогућава избор једне од више понуђених вредности. Има 2^k улаза, као и додатних k селекционих улаза помоћу којих се врши избор.

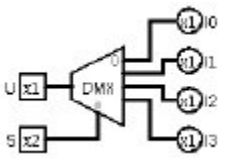
35. *Нацртати логичко коло имплементације мултиплексера 4-1.*



36. *Како се мултиплексер употребљава за имплементацију логичких функција?*

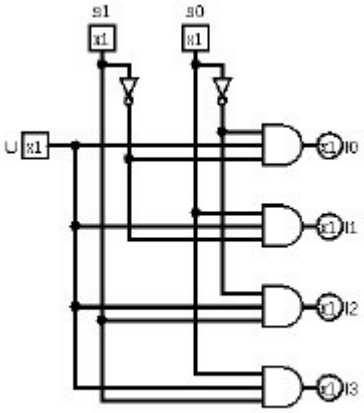
    Мултиплексери се могу користити за декомпозицију логичке функције на више једноставнијих функција тако што се посматрају функције за фиксиране вредности неких логичких променљивих, излази тих логичких функција се повежу на мултиплексер који ће на крају, у зависности од вредности променљивих које смо током декомпозиције фиксирали, изабрати тражену вредност.

37. *Шта је демултиплексер и која је његова основна функција? Представити графичким симболом и таблицом демултиплексер 1-4.*

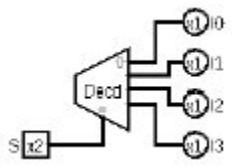


Демултиплексер је комбинаторно коло које има један улаз и 2^k излаза, при чему се улазни сигнал преусмерава на тачно један излазни уз помоћ вредности представљене кроз k селекционих улаза.

38. *Нацртати логичко коло имплементације демултиплексора 1-4.*

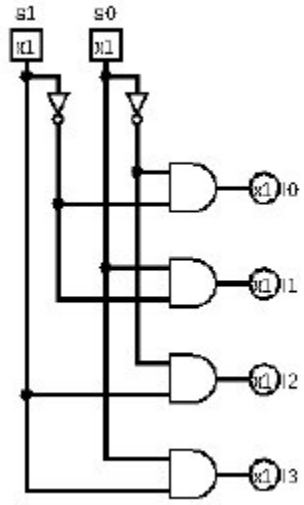


39. *Шта је декодер и која је његова основна функција? Представити графичким симболом и таблицом декодер 2-4.*



Декодер је комбинаторно коло које декодира бинарно записани број и на основу његове вредности активира одговарајући сигнал на излазу. Декодер има k улаза који представљају бинарни број једног од 2^k излаза које треба поставити на 1.

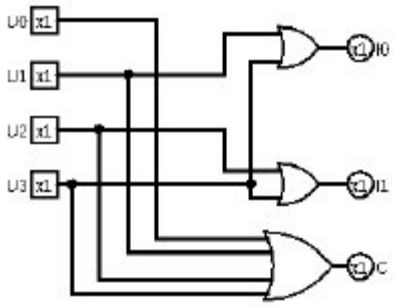
40. *Нацртати логичко коло имплементације декодера 2-4.*



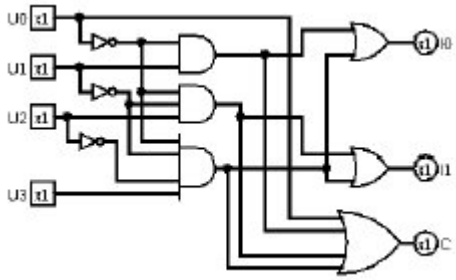
41. *Шта је кодер и где се обично користи? Шта је кодер са приоритетом?*

Кодер је логичко коло које има 2^k улаза, од којих највише један може узети вредност 1, као и k излаза који представљају бинарне цифре индекса улаза који има вредност 1. Додатно, имамо и контролни излаз који ће имати вредност 0 ако је вредност свих улаза 0, а иначе 1. Кодери се користе код обраде захтева за прекиде као и када знамо да се неки податак налази у највише једном од неких регистара у ком случају на улазе кодера прикључујемо излазе одговарајућих компаратора, а што се примењује у кеш меморијама. Кодер са приоритетом омогућава да више од једног улаза има вредност 1, при чему се сваком улазу тада додељује одређени приоритет, а на излазу ће бити индекс оног са највишим приоритетом.

42. *Нацртати логичко коло имплементације кодера 4-2.*



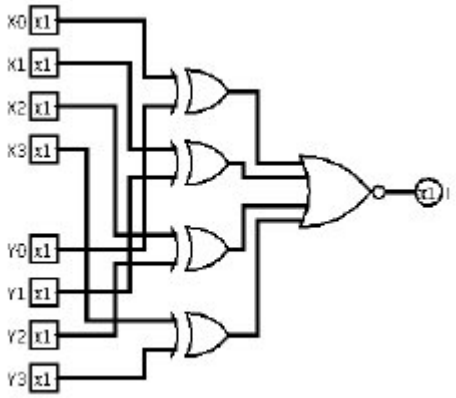
43. *Нацртати логичко коло имплементације кодера 4-2 са приоритетом.*



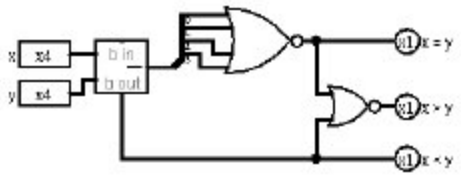
44. *Шта је компаратор? Навести основне врсте компаратора.*

Компаратори су кола која упоређују два податка. Основне врсте су компаратори на једнакост и потпуни компаратори, који враћају који је од два дата улаза већи.

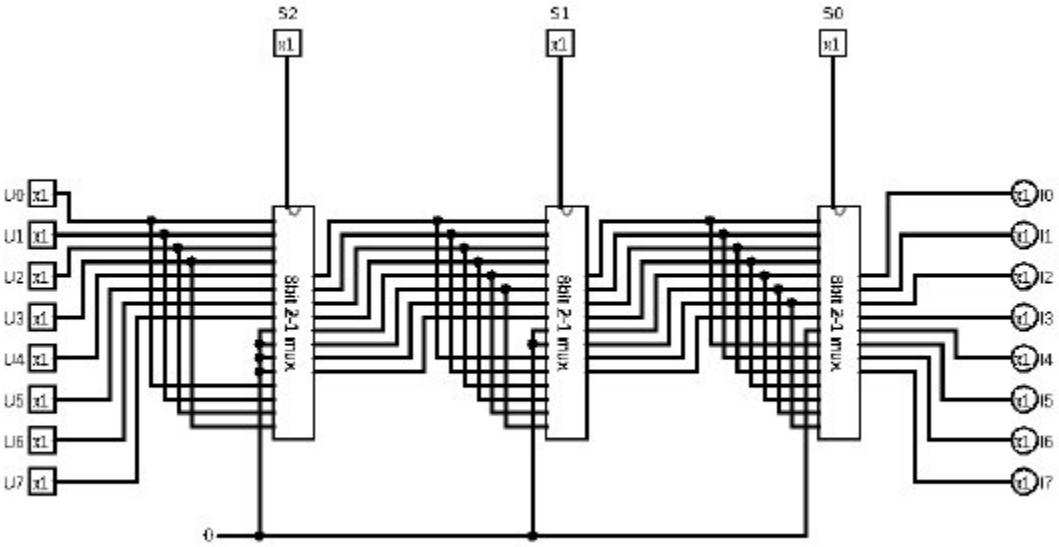
45. *Нацртати логичко коло 4-битног компаратора за поређење на једнакост.*



46. *Нацртати логичко коло 4-битног компаратора за потпуно поређење.*

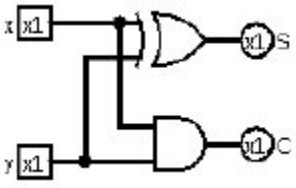


47. *Нацртати логичко коло 8-битног померача.*



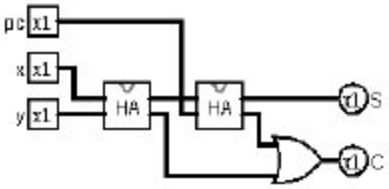
48. *Нацртати истинитосну таблицу и логичко коло бинарног полусабирача.*

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



49. *Нацртати истинитосну таблицу и логичко коло потпуног сабирача.*

x	y	pc	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

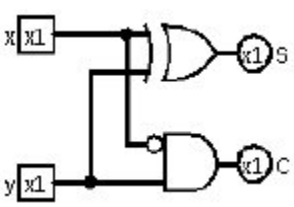


50. *Вишебитни таласасти сабирач. Кашњење.*

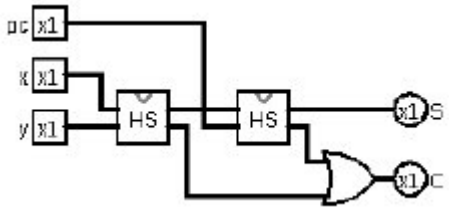
    Вишебитни таласасти сабирач се добија уланчавањем одговарајућег броја потпуних сабирача, тако да i-ти потпуни сабирач рачуна збир i-тих битова, а њихов пренос прослеђује i+1-вом потпуном сабирачу. Пренос последњег потпуног сабирача је пренос целокупног вишебитног сабирача. Кашњење оваквог сабирача је n * 2delta.

51. *Нацртати истинитосну таблицу и логичко коло бинарног полуодузимача.*

x	y	S	C
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



52. *Нацртати истинитосну таблицу и логичко коло потпуног одузимача.*



53. *Вишебитни таласасти одузимач. Кашњење.*

    Вишебитни таласасти одузимач се добија уланчавањем одговарајућег броја потпуних одузимача, тако да i-ти потпуни одузимач рачуна разлику i-тих битова, а њихов пренос прослеђује i+1-вом потпуном одузимачу. Пренос последњег потпуног одузимача је пренос целокупног вишебитног одузимача. Кашњење оваквог одузимача је n * 2delta.

54. *Објаснити укратко принцип рада сабирача са рачунањем преноса унапред.*

Код сабирача са рачунањем преноса унапред се рачуна, само на основу улазних вредности, да ли долази до преноса на сваком биту. То се остварује уз помоћ посебног логичког кола које се назива јединица за рачунање преноса унапред (LCU).

55. *Шта код сабирача са рачунањем преноса унапред означавају вредности P_i и G_i и по којим се формулама рачунају?*

    Вредност G_i означава да се на i-том биту генерише пренос, што се дешава само ако су оба бита 1, па важи $G_i = x_i y_i$. С друге стране, вредност P_i означава да се на i-том биту преноси пренос са неког од претходних битова и рачуна се као $P_i = x_i \oplus y_i$.

56. *Навести формуле по којима LCU јединица рачуна преносе C_i као и групне P и G вредности.*

    Рачуна се по формули $C_i = G_i + P_i C_{i-1}$. Тако је, на пример, $C_1 = G_1 + P_1 G_0 + P_1 P_0 pc$. Групна P вредност се рачуна као $P = P_0 P_1 ... P_{n-1}$ за n-битни сабирач, док се групна G вредност рачуна као $G = G_{n-1} + G_{n-2} P_{n-1} + ... + G_0 P_1 ... P_{n-1}$ за n-битни сабирач.

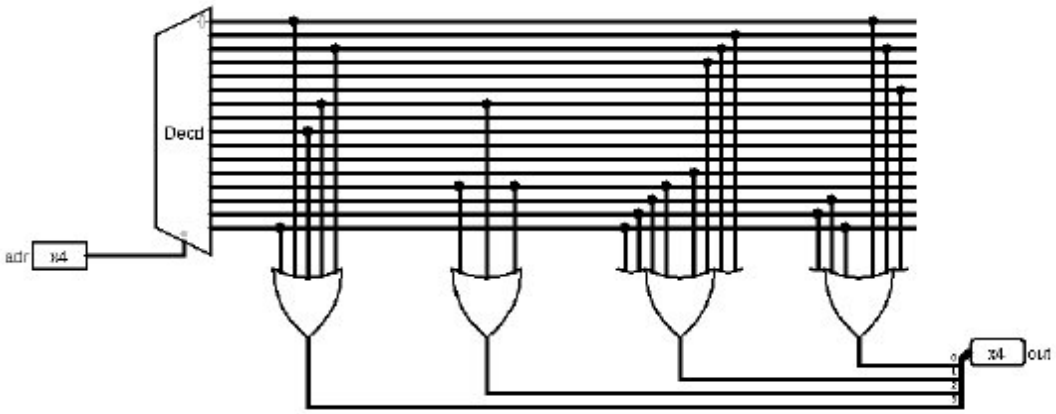
57. *Навести пример имплементације ALU јединице.*

    ALU јединица у себи може садржати различита комбинаторна кола, која служе за рачунање вредности неких израза, а која се на крају повезују на мултиплексер који на додатном улазу има податак о томе која се операција захтева од ALU.

58. *Шта је програмибилни низ логичких елемената (PLA)? Навести пример.*

    Желимо да имплементирамо произвољну логикчу функцију. Међутим, она ће обично у себи имати више нула него јединица, тј. имаће мали број елементарних конјукција у савршеној ДНФ. Стога правимо прво И матрицу, где повезујемо оне улазе који су нам потребни за елементарне конјункције, а потом и ИЛИ матрицу у којој ће бити повезани одговарајући излази из И матрице тако да добијемо наше логичке функције.

59. *Како се помоћу комбинаторних мрежа имплементира неизмењива меморија (ROM)? Пример таблице и одговарајуће имплементације.*



60. *Шта је секвенцијално коло? По чему се секвенцијална кола разликују од комбинаторних кола?*

    Секвенцијална кола су логичка кола која, за разлику од комбинаторних кола чије излазне вредности зависе само од тренутног улаза, дају излаз који зависи од тренутног улаза и претходног стања, чиме се постиже памћење стања.

61. *Нацртати концептуални дијаграм секвенцијалног кола и објаснити основни принцип рада.*

    Секвенцијално коло се садржи од неког комбинаторног кола G које за улаз X и стање S даје стање S, тј. важи $G(X, S) = S$. Повратном спрегом се ово стање враћа на улаз кола G. Секвенцијално коло као крајњи излаз има $F(S)$. За стање кажемо да је стабилно ако је оно фиксна тачка функције G за неко фиксирано X.

62. *Шта је нестабилност секвенцијалног кола, а шта недетерминистичност? Шта је метастабилност?*

    Нестабилност секвенцијалног кола представља осцилација међу неуспешним покушајима тражења стабилног стања. Недетерминистичност представља појаву да секвенцијално коло из једног нестабилног стања може прећи у више стабилних стања у зависности од неких случајних околности. Метастабилност представља стабилизацију секвенцијалног кола на потенцијалу који не представља исправну логичку вредност.

63. *Шта је функција (таблица) преласка секвенцијалног кола? Навести пример.*

    Нека је X произвољна вредност на улазу и S стабилно стање секвенцијалног кола за тај улаз. Ако у неком тренутку улаз добије нову вредност X', тада коло прелази у ново стабилно стање S' при чему је то стање једнозначно означено претходним стањем и новим улазом, тј. постоји пресликавање T такво да је $S' = T(X', S)$. Пресликавање T се назива функција (таблица) преласка секвенцијалног кола.

64. *Објаснити разлику између синхроних и асинхроних секвенцијалних кола.*

    Код асинхроних кола промена се може десити у било ком тренутку, док је са друге стране код синхроних кола временски локализована осетљивост на улазне вредности.

65. *Објаснити улогу часовника. На који начин часовник омогућава синхронизацију секвенцијалних кола?*

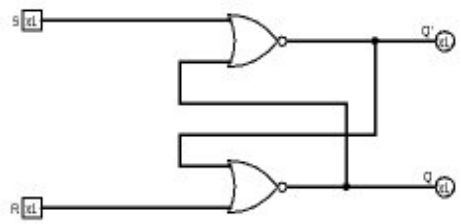
    Часовник је сигнал који наизменично у једнаком ритму мења своју вредност са 0 на 1 и обратно. Сада се секвенцијална кола могу имплементирати тако да поред осталих својих улаза имају и додатни улаз за сигнал часовника. Притом, имплементацијом се обез-

бећује да се стања секвенцијалног кола могу мењати само у тачно одређеним тренуцима, типично на узлазним или силазним рубовима часовника.

66. *Елементи циклуса часовника. Типови часовника. Фреквенција часовника.*

Прелазак часовника са 0 на 1 се назива узлазним рубом часовника, док је прелазак са 1 на 0 силазни руб часовника. Период током ког је сигнал 1 назива се позитивни део циклуса, док је период током ког је сигнал 0 негативни део циклуса. Часовници код који позитивни и негативни део трају исто називају се симетрични часовници, док се они код којих то није случај називају асиметричним. Фреквенција часовника представља број циклуса у једној секунди.

67. *Шта је SR реза? Нацртати имплементацију, таблицу преласка, логички симбол и објаснити понашање.*

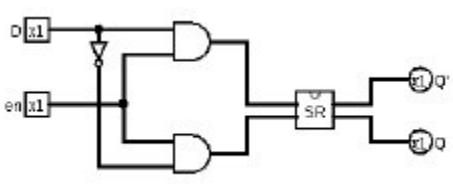


SR реза је секвенцијално коло које се састоји од два НИЛИ кола, има два улаза, S и R, као и два излаза Q и Q', који и представљају стање. По повратној спрези важи: $Q = R \downarrow Q\prime$ и $Q' = S \downarrow Q$. Постоје два стабилна стања која ни за које вредности улаза не воде у нестабилност: $\left(Q,Q'\right) = (0, 1)$ и $\left(Q,Q'\right) = (1, 0)$. Ова стања се одржавају за $(S,R) = (0, 0)$. За $(S,R) = (0, 1)$ коло се стабилизује у стању $\left(Q,Q'\right) = (0, 1)$, а за $(S,R) = (1, 0)$ коло се стабилизује у стању $\left(Q,Q'\right) = (1, 0)$. Како су излазни битови комплементари у стабилним стањима, суштински овиме памтимо један излазни бит, нека је то Q. Дакле, за $(S,R) = (0, 0)$ стање се одржава, за $(S,R) = (0, 1)$ уписује се 0, а за $(S,R) = (1, 0)$ уписује се 1.

S	R	Q	Q^{sled}
0	0	0	0
0	0	1	1
0	1	-	0
1	0	-	1
1	1	-	?

68. *Шта је D реза? Нацртати имплементацију, таблицу преласка, логички симбол и објаснити понашање.*

D реза је секвенцијално коло које се састоји од SR резе и два И кола. Има два улаза, D и e. Када је e 0 стање се одржава. Када је e 1 стање се мења и уписује се вредност D.

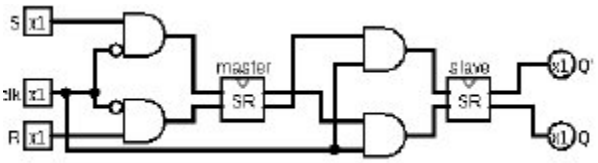


D	e	Q	Q^{sled}
-	0	0	0
-	0	1	1
0	1	-	0
1	1	-	1

69. *Која је основна разлика између резе и флип-флопа?*

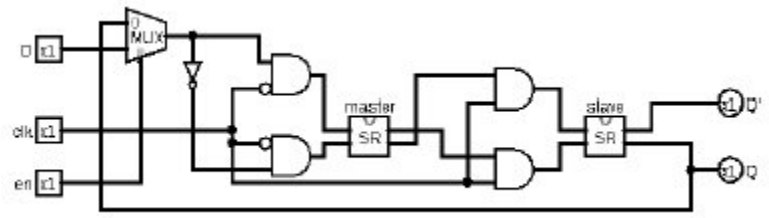
Резе су асинхрона секвенцијална кола, док су флип-флопови синхрона секвенцијална кола и њихово стање је могуће променити само при надоласку одговарајућег руба часовника.

70. *Нацртати имплементацију master-slave SR флип-флопа и објаснити понашање.*



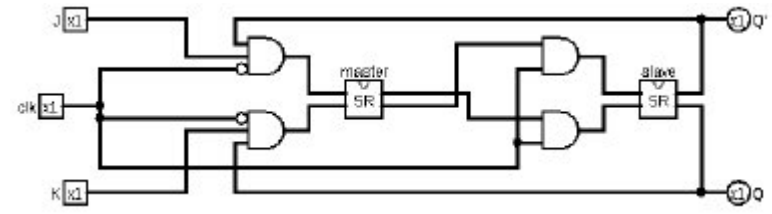
Имамо две серијски повезане SR резе, где је прва „master“, а друга „slave“. Q излаз master резе је повезан на S улаз slave резе, док је Q’ излаз master резе повезан на R улаз slave резе. Тиме се постиже да се вредност master резе уписује у slave резу. Улази master резе су кроз конјункције повезани са инвертованим сигналом часовника, чиме се master отвара током негативног дела циклуса часовника. С друге стране, улази у slave резу су преко конјункција повезани са сигналом часовника, па је ова реза затворена за време негативног дела циклуса часовника, тј. не долази до промена. На узлазном рубу часовника master реза се затвара, док се slave реза отвара и у њу се уписује промењена вредност. Дакле, вредност у флип-флопу се може променити само на узлазном рубу часовника.

71. *Нацртати имплементацију master-slave D флип-флопа и објаснити понашање.*



Имплементација је иста као и код SR флип-флопа, уз додатни мултиплексер са улазима D и вредности флип-флопа и констролним сигналом који означава да ли је дозвољена промена вредности флип-флопа. Ако није, мултиплексер пропушта вредност флип-флопа, иначе пропушта вредност улаза D.

72. *Нацртати имплементацију master-slave JK флип-флопа и објаснити понашање.*

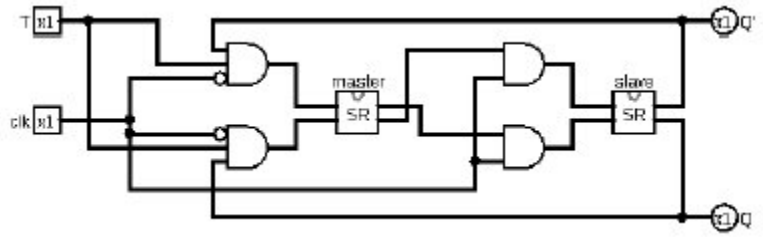


Идеја JK флип-флопа је да се попуни семантика SR флип-флопа. Улаз J има исту улогу као S, а K као R. Дакле, за комбинацију $(J,K) = (0,0)$ одржава се вредност, за $(J,K) = (0,1)$ се поставља на 0, за $(J,K) = (1,0)$ се поставља на 1, а за $(J,K) = (1,1)$ се дефинише да се инвертује тренутни садржај. Имплементира се слично као и SR флип-флоп, с тим што се излаз Q враћа на конјункцију код улаза K, а излаз Q’

се враћа на конјункцију код улаза J. Тиме се спречава пролазак двеју јединица и постиже се инвертовање ако су на улазима обе јединице.

73. *Нацртати имплементацију master-slave T флип-флопа и објаснити понашање.*

    T флип-флоп је суштински JK флип-флоп са два иста бита на улазима. Овиме се постиже да се садржај флип-флопа или чува или инвертује.

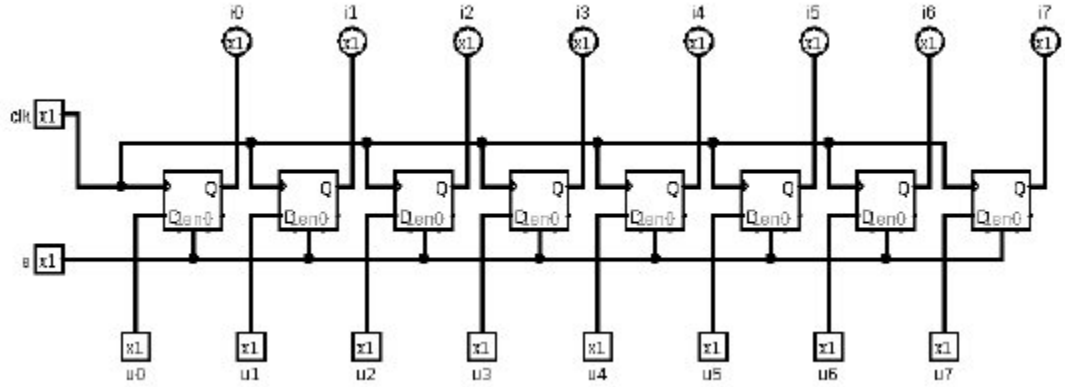


74. *Објаснити проблем „хватања јединице“ код master-slave SR и JK флип-флопова. На који начин се овај проблем може решити?*

    Претпоставимо да нам флип-флоп реагује на силазни руб часовника. Уколико се током позитивног дела циклуса на S/J улазу случајно појавила краткотрајна јединица, она ће бити запамћена приликом силазног руба часовника, иако тада није нужно јединица на том улазу. Овај проблем може да се реши по угледу на D флип-флоп, убацивањем мултиплексера, који пропушта одговарајући излаз за одговарајућу комбинацију улаза. Тиме се постиже да је у master рези све време излаз са мултиплексера, што решава овај проблем.

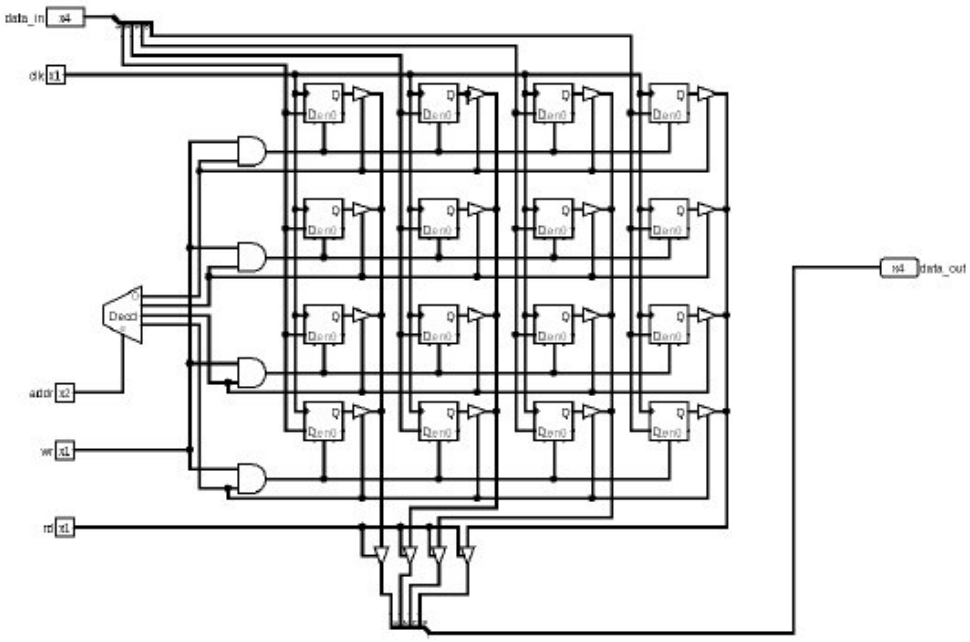
75. *Шта је регистар и како се имплементира? Нацртати пример.*

    Регистар дужине n је коло које чува један n-битни бинарни број. Регистри се обично праве са D флип-флоповима.



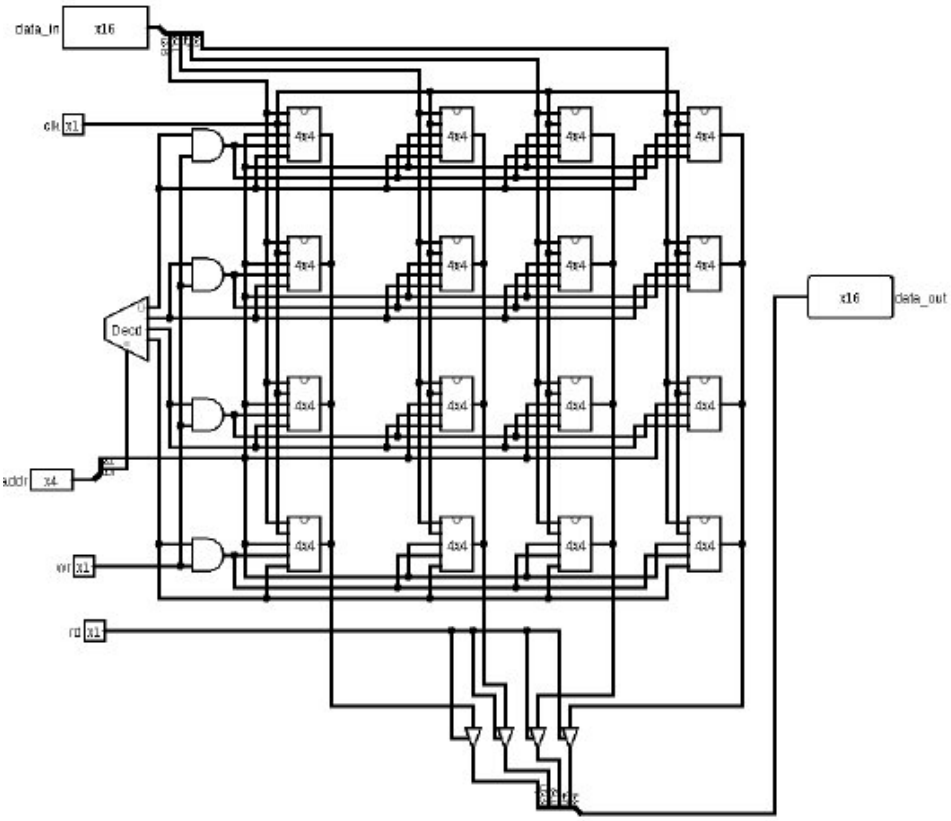
76. *Статичка меморија. Пример реализације меморије 4×4.*

    Статичка меморија се састоји од неколико регистара исте дужине.



77. *На примеру објаснити принцип конструкције већих меморија помоћу мањих.*

Користи се исти принцип као за конструкцију мањих меморија, само што се у матрицу уместо флип-флопова стављају мање меморије.



78. *Ефикасна реализација меморијске ћелије код статичких меморија.*

    Можемо у свакој колони, уместо да имамо у сваком флип-флопу по две резе да поставимо једну заједничку master резу, а да остале буду њој подређене. Тиме постижемо да уместо 2mn реза имамо mn+n+2, што је скоро дупло мање реза.

79. *Објаснити принцип рада меморијске ћелије код динамичких меморија.*

    Код динамичких меморија ћелија се састоји од једног транзистора и једног кондензатора. Сачувана вредност представљена је тиме да ли је кондензатор напуњен. Када желимо да упишемо нешто у ћелију на гејт транзистора доводимо 1. Ако је на битској линији било 1, кондензатор ће се напунити, иначе ће се испразнити. Када желимо да прочитамо ћелију опет отварамо транзистор, а на битску линију доводимо међувредност. Ако је кондензатор био напуњен, сада ће се испразнити, а напон на битској линији ће се мало повећати. Посебно електронско коло ће га појачати до 1, а том приликом ће се кондензатор опет напунити. Ако је кондензатор био празан, напон на битској линији ће пасти, па ће посебно коло спустити напон на 0, чиме ће се опет испразнити кондензатор. Такође, кондензатор се временом сам празни, па је потребно периодично допунити га.

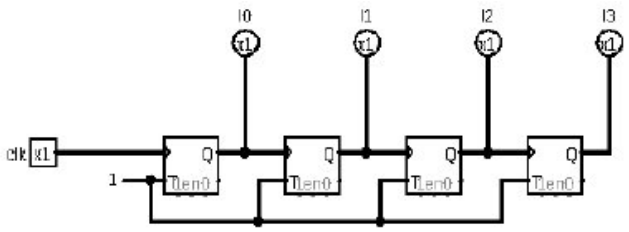
80. *Предности и недостаци динамичких меморија у односу на статичке.*

    Основна мана динамичких меморија је та што је пуњење и пражњење кондензатора споро, па су динамичке меморије значајно спорије од статичких. С друге стране, због тога што у себи имају много мање компоненти, динамичке меморије су значајно јефтиније, па су последично динамичке меморије значајно веће од статичких.

81. *Шта је померачки регистар и где се обично користи?*

    Померачки регистар је регистар који подржава операцију померања свог садржаја за једно место у лево или у десно. Типична примена оваквих регистара је у алгоритмима множења као што је Бутов алгоритам. Друга примена је у конверзији између серијског и паралелног трансфера.

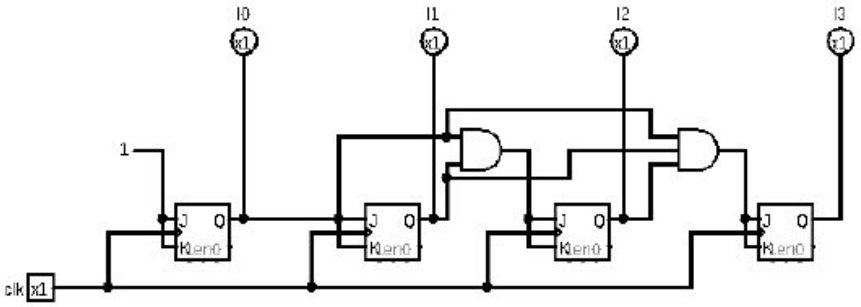
82. *Асинхрони бинарни бројач. Нацртати шему и објаснити принцип рада. Који је основни недостатак асинхроних бројача?*



Асинхрони бинарни бројач се имплементира помоћу n T флип-флопова, који су подешени да реагују на силазни руб, од којих је само онај који представља бит најмање тежине повезан на часовник, док је сваки следећи

уместо на часовник повезан на излаз од претходног флип-флопа. Тиме се управо постиже бинарно бројање. Основни недостатак асинхроних бинарних бројача је велико кашњење које имају.

83. *Синхрони бинарни бројач. Нацртати шему и објаснити принцип рада.*



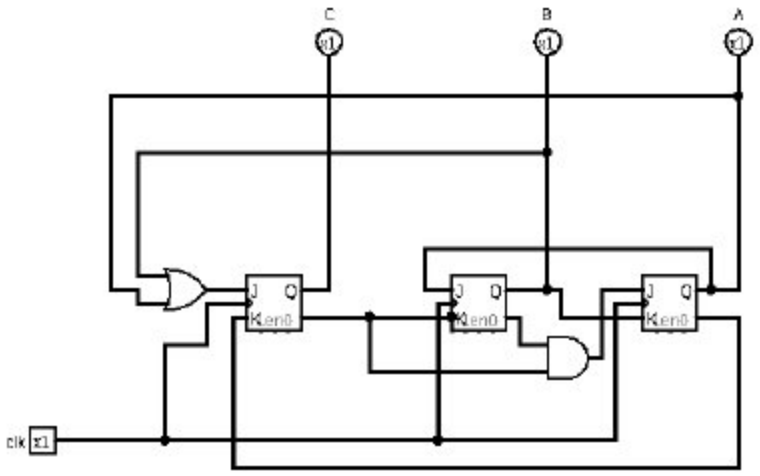
Код синхроних бинарних бројача имамо n ЈК флип-флопова који реагују на узлазни руб часовника. На улазе првог флип-флопа се стављају јединице, а на улазе свих осталих конјункција излазних вредности претходних флип-флопова. Овиме се опет тачно имитира бинарно бројање. Иако имају сложенију имплементацију, имају знатно мање кашњење у односу на асинхроне бројаче.

84. *Дизајн бројача са произвољним редоследом стања. Пример.*

Q	Q^{sled}	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

Посматра се таблица ексцитације ЈК флип-флопа, која представља инвез таблице преласка за ЈК флип-флоп. Сада кодирамо стања бинарним бројевима и правимо таблицу ексцитације бројача. Након тога формирамо логичке функције за све J и K.

A	B	C	A^{sled}	B^{sled}	C^{sled}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	1	0	0	1	–	0	–	0	–
0	0	1	0	0	0	0	–	0	–	–	1
0	1	0	0	0	1	0	–	–	1	1	–
0	1	1	0	1	0	0	–	–	0	–	1
1	0	0	1	1	1	–	0	1	–	1	–
1	0	1	–	–	–	–	–	–	–	–	–
1	1	0	–	–	–	–	–	–	–	–	–
1	1	1	0	1	1	–	1	–	0	–	0

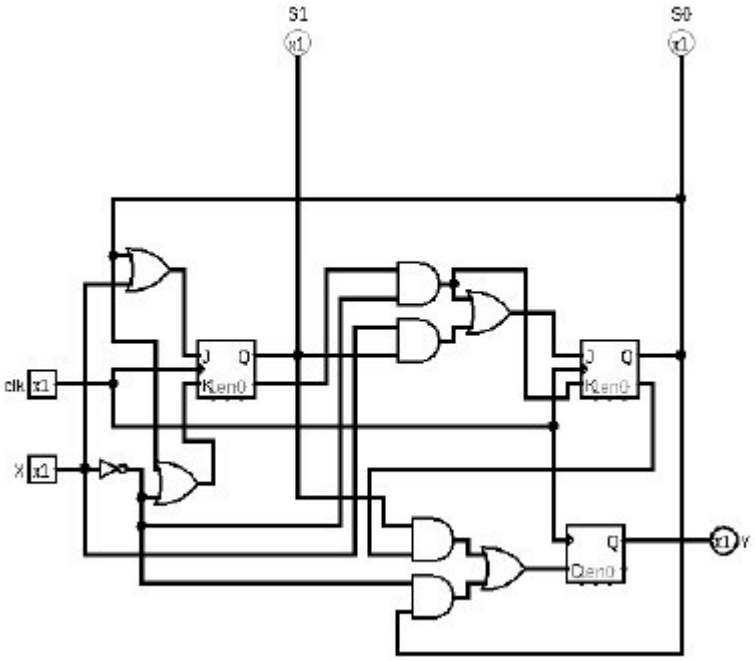


85. *Коначни аутомати и трансдуктори као модел синхроних секвенцијалних кола. Дизајн коначних аутомата. Пример.*

Коначни аутомат је додатно уопштење бројача са произвољним редоследом стања, јер нема фиксирана стања нити њихов редослед, већ може бити променљив. Дизајн коначних аутомата је сличан дизајну бројача са произвољним редоследом. На основу таблице преласка аутомата правимо таблицу ексцитације аутомата, потом се праве одговарајуће логичке функције.

Q	X	Q^{sled}	Y
0	0	1	0
0	1	2	0
1	0	2	1
1	1	3	0
2	0	0	1
2	1	3	1
3	0	1	1
3	1	1	0

S_1	S_0	X	S_1^{sled}	S_0^{sled}	Y	J_1	K_1	J_0	K_0
0	0	0	0	1	0	0	–	1	–
0	0	1	1	0	0	1	–	0	–
0	1	0	1	0	1	1	–	–	1
0	1	1	1	1	0	1	–	–	0
1	0	0	0	0	1	–	1	0	–
1	0	1	1	1	1	–	0	1	–
1	1	0	0	1	1	–	1	–	0
1	1	1	0	1	0	–	1	–	0



86. *Укратко објаснити основни принцип дизајна контролне јединице као коначног аутомата.*

    Контролна јединица на улазу има неке регистре у зависности од којих, као и од претходног стања, треба да одреди ново стање као и да на излаз стави одређене контролне битове. Овај опис управо одговара коначном аутомату.

87. *Навести пример описа неког алгоритма у форми коначног аутомата (само таблица преласка, без реализације самог аутомата).*

88. *Шта је архитектура, а шта организација рачунара?*

    Архитектуру рачунара представљају сви они аспекти дизајна рачунара који занимају програмера који пише програм на том рачунару. Пре свега укључује архитектуру скупа инструкција, али и из којих функционалних јединица се састоји рачунар, како се оне понашају и како се повезују.
    Организација рачунара представља начин имплементације свих функционалних јединица са свим особинама задатим кроз архитектуру рачунара.

89. *Шта обухвата ISA (архитектура скупа инструкција)?*

    Архитектура скупа инструкција обухвата скуп регистара, скуп инструкција, формате инструкција, начине адресирања операнада, величину меморијског простора (физичког и виртуелног), режиме рада, систем прекида и сл.

90. *Шта су троадресни процесори? Пример инструкција и кода. Карактеристике.*

    Троадресни процесори су они који подржавају инструкције са три операнда, где су одредишни и изворишни операнди одвојени. Пример: x = a*b + c*d:
    mul s, a, b
    mul t, c, d
    add x, s, t

91. *Шта су двоадресни процесори? Пример инструкција и кода. Карактеристике.*

    Код двоадресних процесора инструкције имају највише два операнда, где је први операнд уједно и одредишни. Пример: x = a*b + c*d:
    mov t, a
    mul t, b
    mov s, c
    mul s, d
    mov x, t
    add x, s

92. *Шта су једноадресни процесори? Пример инструкција и кода. Карактеристике.*

    Код једноадресних процесора инструкције имају највише један операнд, постоји регистар који се назива акумулатор. Пример: x = a*b + c*d:
    load a
    mul b
    store t
    load c
    mul d
    add t
    store x

93. *Шта су нулоадресни процесори? Пример инструкција и кода. Карактеристике.*

    Код нулоадресних процесора инструкције немају операнде, овакви рачунари се још зову и стек машине. Пример: x = a*b + c*d:
    push a
    push b
    mul
    push c
    push d
    mul
    add
    pop x

94. *Објаснити однос перформанси процесора и броја адреса.*

    Што више адреса има, инструкције су моћније, па је потребан мањи број инструкција. Међутим, инструкције су дуже, па их је теже декодирати, тј. захтевнија је имплементација процесора.

95. *Шта је архитектура load/store? Објаснити.*

    Код load/store архитектуре све операције се извршавају искључиво над регистрима, док постоје посебне инструкције load и store које служе за приступ меморији и које су једине које могу имати меморијске операнде.

96. *Карактеристике CISC архитектура.*

    CISC архитектуре могу садржати и доста сложене операције за које је потребно неколико циклуса да се изведу помоћу оних које су имплементиране у ALU јединици. Такође, за исте инструкције се могу користити различите врсте операнада, што усложњава декодирање. Овакве архитектуре подржавају и већи број формата инструкција и већи број начина адресирања. Карактеристично је да се користе подаци директно из меморије.

97. *Карактеристике RISC архитектура.*

    RISC архитектуре имају веома једноставне операције, углавном само сабирање, одузимање и евентуално множење, као и битовске операције. Имају униформни формат инструкције као и једноставан начин адресирања. Обично се имплементирају као load/store архитектуре, па у раду са основним операцијама могу само да се користе непосредни и регистарски операнди.

98. *Објаснити однос архитектура CISC и RISC.*

    Због доступности сложених операција код CISC архитектура, програми на оваквим архитектурама су обично краћи од програма на RISC архитектури, а такође се могу и директно користити меморијски операнди. Међутим, имплементација процесора са RISC архитектуром је једноставнија, па је и извршавање операција на овим архитектурама брже него на CISC архитектурама.

99. *Структура и формат машинске инструкције.*

    Машинска инструкција је низ битова који се може поделити на операциони код, који кодира која се операција врши наведеном инструкцијом, и операнде, који кодирају над којим подацима се инструкције врше. Формат машинске инструкције одређује начин кодирања операционог кода и операнада за одређену инструкцију.

100. Врсте операнада машинске инструкције.

Операнди се деле на регистарске, меморијске и непосредне. Код регистарских операнада, податак се налази у неком од регистара процесора. Код меморијских операнада, податак се налази на некој адреси у оперативној меморији. Код непосредних операнада, податак је саставни део машинске инструкције.

101. Објаснити директно адресирање меморијских операнада.

Код директног адресирања меморијских операнада задаје се апсолутна адреса у оперативној меморији на којој се налази податак.

102. Објаснити индиректно адресирање меморијских операнада.

Код индиректног адресирања меморијских операнада, адреса у оперативној меморији је записана у неком од регистара процесора, па се у инструкцији даје код тог регистра.

103. Објаснити индексно адресирање меморијских операнада.

Код индексног адресирања меморијских операнада у машинској инструкцији се задају кодови два регистра, чије се вредности сабирају и тиме се добија адреса у оперативној меморији. Уместо првог регистра дозвољено је навести константу, а други регистар је дозвољено помножити неком целобројном константом.

104. Објаснити релативно адресирање меморијских операнада.

Код релативног адресирања меморијских операнада инструкција садржи релативну адресу операнда у односу на адресу тренутне инструкције.

105. Објаснити начине адресирања на x86-64 архитектури.

На x86-64 архитектури постоје непосредно, индиректно регистарско, релативно регистарско, као и индиректно и индексно меморијско адресирање.

106. Објаснити начине адресирања на ARM архитектури.

На ARM архитектури постоје непосредно, индиректно регистарско, релативно регистарско, као и индиректно и индексно меморијско адресирање.

107. Инструкције трансфера. Функција и пример употребе. Примери: x86-64 и ARM архитектура.

Инструкције транфсера копирају вредност са једне локације на другу. На x86-64 архитектури постоји инструкција mov која вредност другог операнда (ако је непосредан),

или вредност на локацији другог операнда уписује у локацију првог операнда. Не могу се користити два меморијска операнда. Постоји и инструкција lea за пренос адресе другог операнда. На ARM архитектури инструкција mov се користи за непосредне и регистарске операнде, инструкција ldr за упис у регистар из меморије, а инструкција str за упи у меморију из регистра.

108. Аритметичко-логичке инструкције. Функција и пример употребе. Примери: x86-64 и ARM архитектура.

Аритметичко-логичке инструкције врше сабирање, одузимање, множење, битовске операције и сл. Постоје инструкције add, sub, mul, and, or итд.

109. Инструкције безусловног скока. Функција и пример употребе. Примери: x86-64 и ARM архитектура.

Инструкција безусловног скока је инструкција за контролу тока која мења вредност програмском бројачу, у коме се чува адреса наредне инструкције. На x86-64 архитектури постоји инструкција jmp, а на ARM архитектури инструкција b.

110. Флегови процесора (O, S, Z, C). Када се постављају и чему служе?

Флегови процесора су контролни битови у процесору који се постављају након одређених операција. Флег O означава да је дошло до прекорачења, флег S означава знак резултата, флег Z означава да ли је резултат нула, а флег C означава да ли је дошло до преноса.

111. Инструкције поређења и њихова улога у реализацији условних скокова. Примери: x86-64 и ARM архитектура.

Инструкције поређења cmp се обично имплементирају одузимањем без промене вредности операнада, а том приликом се ажурирају флегови који говоре о томе који је био резултат поређења, што се може применити у условним скоковима, који представљају грањање програма, тј. до скока долази ако је задовољен неки услов, најчешће нека комбинација флегова.

112. Инструкције условног скока. Функција и пример употребе. Примери: x86-64 и ARM архитектура.

Условни скокови представљају грањање програма, тј. до скока долази ако је задовољен неки услов, најчешће нека комбинација флегова. На x86-64 архитектури постоје инструкције je, jne, jl, jg, jb, ja итд, док на ARM архитектури постоје инструкције beq, bgt, blt, bge, bne итд.

113. Коју комбинацију флегова тестира инструкција jl, а коју jb на x86-64 архитектури?

Инструкција jl проверава S флег, а jb проверава O флег.

114. Објаснити позивање процедура и враћање из њих коришћењем стека за чување повратне адресе. Предности и мане.

Приликом позивања функције адреса повратка се ставља на стек, а на крају се скида са стека и ставља у програмски бројач. Предност ове методе је што је ово општији приступ јер не морамо да бринемо о повратној адреси, међутим мана је та што је ово спорији приступ.

115. Објаснити позивање процедура и враћање из њих коришћењем регистра за чување повратне адресе. Предности и мане.

Приликом позивања функције адреса повратка се ставља у један регистар, чија се вредност на крају функције пребацује у програмски бројач. Предност ове методе је то што је бржи приступ, али мана је та што бисмо, у случају позивања нове функције, морали да сачувамо адресу повратка на сигурно место, најчешће стек.

116. Објаснити пренос аргумената процедуре коришћењем стека. Предности и мане.

Пре позива функције се аргументи сместе на стек и на тај начин се преносе, јер се у неком од регистара чува адреса врха стека. Предност ове методе је то што немамо ограничење броја аргумената, али је мана то што је спора.

117. Објаснити пренос аргумената процедура коришћењем регистара процесора. Предности и мане.

Пре позива функције се аргументи сместе у одређене регистре и на тај начин се преносе. Предност је та што је бржи начин, али мана је то што захтева велик број регистара.

118. На који начин позвана функција може вратити вредност позивајућој функцији?

Вредност се може вратити преко стека, или преко неког од регистара.

119. Објаснити позивање функција на x86-64 архитектури. Како се преноси адреса повратка, аргументи, као и повратна вредност?

На x86-64 архитектури се адреса повратка преноси преко стека, она се ставља на стек приликом инструкције call за позив функције. Првих 6 аргумената се преноси преко ре-

гистара rdi, rsi, rdx, rcx, r8, r9, а наредни се преносе преко стека, у обрнутом редоследу. Повратна вредност се преноси преко регистра rax.

120. Објаснити позивање функција на ARM архитектури. Како се преноси адреса повратка, аргументи, као и повратна вредност?

На ARM архитектури адреса повратка се преноси преко регистра lr, прва 4 аргумента се преносе преко регистара r0-r3, а остали преко стека у обрнутом редоследу. Повратна вредност се преноси преко регистра r0.

121. Које су основне компоненте процесора? Објаснити их.

Аритметичко-логичка јединица (ALU), служи за обављање основних аритметичко-логичких операција.
Регистри процесора који служе за привремено складиштење података унутар процесора.
Контролна јединица која служи за управљање преносом података и извршавањем операција над њима у ALU.
Интерне магистрале, које служе за повезивање компоненти процесора.

122. Шта је ALU јединица и чему служи?

ALU јединица је комбинаторно коло које служи за основне аритметичко-логичке операције као што су сабирање, одузимање, множење, дељење и битовске операције. Има два улаза за податке као и улаз за то која се операција над њима врши.

123. Шта су регистри опште намене и чему служе?

Регистри опште намене су део архитектуре и видљиви су програмеру, тј. могу се користити као операнди у инструкцијама. Служе за брз приступ подацима приликом извршавања програма и вршење операција над њима.

124. Чему служи инструкциони регистар (IR)?

Инструкциони регистар садржи операциони код и остале делове инструкције који су значајни за рад контролне јединице и њено даље одлучивање шта ће се радити. Садржај овог регистра се добија након декодирања машинске инструкције, а овај регистар се спроводи на један од улаза контролне јединице.

125. Чему служи програмски бројач (PC)?

У програмском бројачу се чува адреса тренутне инструкције која се извршава. По завршетку извршавања инструкције, овај регистар се аутоматски увећава за величину текуће инструкције.

126. Чему служи статусни регистар (PSW)?

Статусни регистар садржи флегове који описују резултат последње инструкције. Вредност овог регистра се израчунава у ALU јединици, а спроводи се на улаз контролне јединице, јер може бити значајан за извршавање наредних инструкција.

127. Чему служи регистар меморијских адреса (MAR)?

Регистар меморијских адреса је повезан на адресни део меморијске магистрале, стога се у овај регистар уписује адреса у оперативној меморији на којој се врши читање или писање.

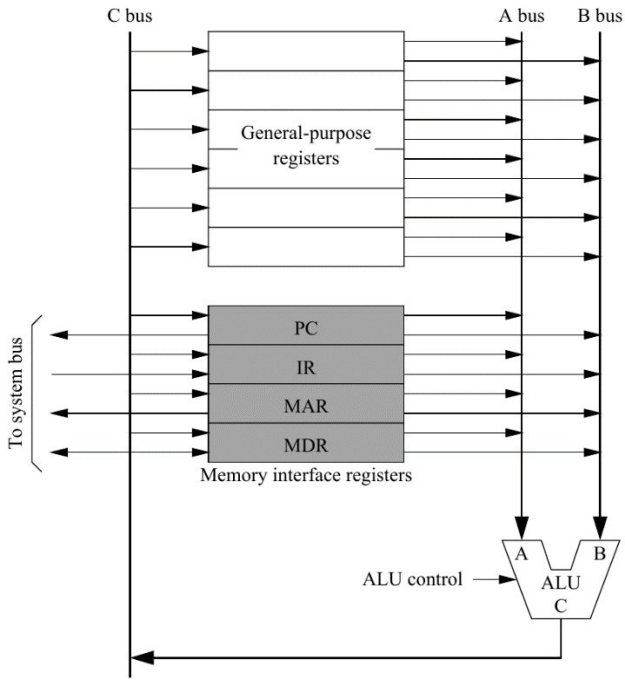
128. Чему служи регистар меморијских података (MDR)?

Регистар меморијских података је повезан на део за податке меморијске магистрале, стога се у овај регистар уписује податак који се жели уписати у меморију, тј. у њега ће бити уписан податак са прочитане адресе из оперативне меморије.

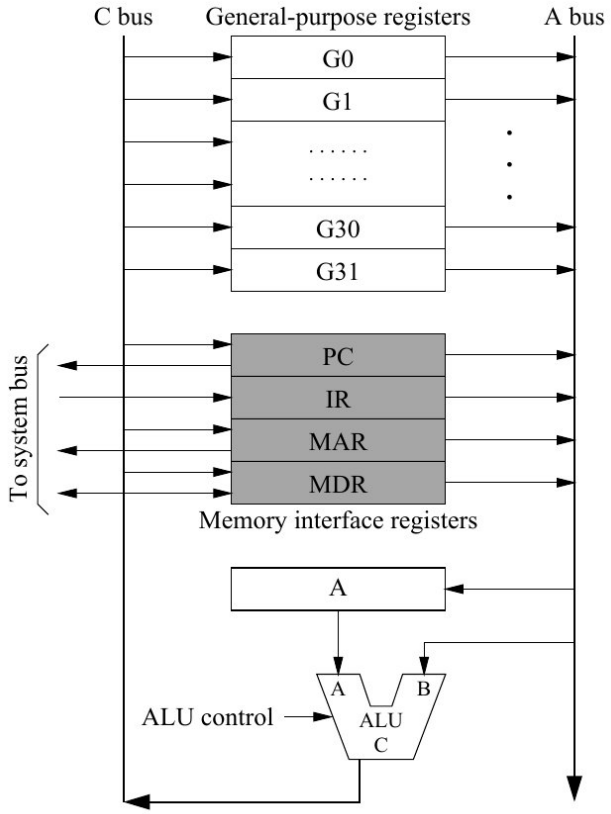
129. Шта је путања података и из чега се састоји?

Путања података је склоп који се састоји од регистара, ALU јединице и интерних магистрала којима се крећу подаци. Улога путање података је да омогући извршавање елементарних операција које хардвер директно подржава, као што су трансфер података између регистара и аритметичко-логичке операције које су имплементиране у ALU јединици.

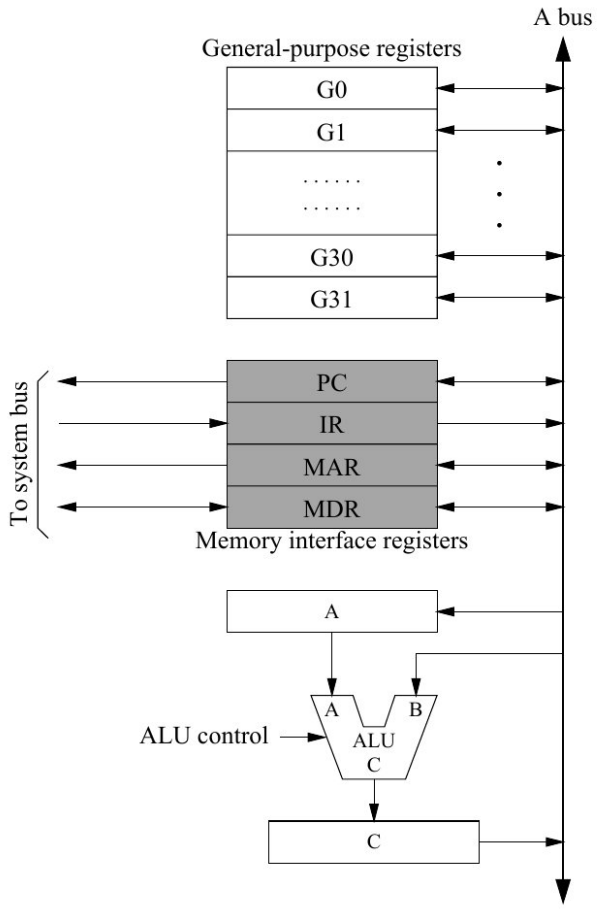
130. Нацртати уопштену схему путање података са три интерне магистрале. Пример извршавања операције.



131. Нацртати уопштену схему путање података са две интерне магистрале. Пример извршавања операције.



132. Нацртати уопштену схему путање података са једном интерном магистралом. Пример извршавања операције.



133. *Шта је контролна јединица? Шта је улаз, а шта излаз из контролне јединице?*

Контролна јединица врши контролу тока података на путањи података. Улаз у контролну јединицу представљају инструкциони регистар и статусни регистри, док на излазу има контролне сигнале којима управља компонентама у оквиру путање података.

134. *Описати основне фазе при извршавању инструкција процесора.*

Дохватање инструкције (Fetch) – инструкција се учитава из меморије.
Декодирање инструкције (Decode) – контролна јединица сазнаје за инструкцију и одређује шта треба да ради.
Извршавање инструкције (Execute).
Упис резултата (Write) – резултат се уписује на одговарајућу локацију.
Обрада прекида (Interrupt) – проверава се да ли има сигнала за прекид.

135. *Објаснити фазу дохватања инструкције.*

Прво се у MAR поставља вредност регистра PC, у коме се чува адреса инструкције коју треба дохватити. Потом се преко меморијске магистрале врши трансфер, а инструкција се нађе у регистру MDR, чиме је инструкција стигла у процесор.

136. Објаснити фазу декодирања инструкције.

Инструкција се пребацује у IR, одакле иде на улаз контролне јединице. У фази декодирања контролна јединица утврђује коју инструкцију треба извршити, а самим тим и које кораке треба предузети.

137. Објаснити фазу извршавања инструкције.

Уколико инструкција садржи меморијске операнде, у овој фази се они дохватају из меморије, на исти начин као што се дохвата и инструкција. У случају аритметичко-логичке операције, вредности се стављају на улазу ALU, а резултат се привремено памти у неком регистру. У случају трансфера између регистара, вредност изворишног регистра се пропушта кроз ALU без операције, а чува се у одредишни регистар. Уколико је трансфер из меморије, након што се вредност дохвати у MDR, пребацује се одредишни регистар. Уколико је трансфер у меморију, вредност изворишног регистра се пребацује у MDR. У случају скокова, уз евентуалну контролу одређених флегова, одређена вредност се уписује у PC. Ако није била задовољена тражена комбинација флегова, не ради се ништа.

138. На које начине се може реализовати контролна јединица? Поређење.

Постоји тврдо ожичена имплементација и микропрограмирана имплементација. Тврдо ожичена иммплементација представља коначни аутомат. Њена предност је брзина, али је мана то што нема флексибилности, тј. свака промена у архитектури тражи потпуно другачију контролну јединицу. С друге стране, микропрограмска имплементација своди имплементацију контролне јединице на микропрограмирање. Мана овог приступа је то што је спорији, али је предност његова флексибилност, где је потребно само променити садржај неких меморијских локација да би се добије другачије функционалности.

139. Објаснити тврдо ожичену (хардверску) имплементацију контролне јединице.

У тврдо ожиченој имплементацији, контролна јединица се имплементира кроз секвенцијално коло које функционише као коначни аутомат. На улазу у контролну јединицу налазе се IR и PSW регистри, док је на излазу скуп контролних сигнала који се шаљу путањи података, оперативној меморији и осталим уређајима. Стања коначног аутомата одговарају фазама извршавања инструкције. Могу постојати и међуфазе код неких захтевнијих операција.

140. Објаснити микропрограмску (софтверску) имплементацију контролне јединице.

У микрипрограмској имплементацији, контролна јединица садржи меморију која се назива контролна меморија. Свака адресибилна локација у овој меморији садржи једну

микроинструкцију. Сада свакој инструкцији одговара један микропрограм, а такође постоји и микропрограм за дохватање из меморије и декодирање.

141. Шта је микроинструкција? Структура микроинструкције.

Садржај сваке адресибилне локације представља једну микроинструкцију. Свака микроинструкција се садржи од контролних сигнала и адресних битова.

142. Шта је микропрограм? Објаснити начин извршавања микропрограма.

Микропрограм представља низ микроинструкција којима се имплементира нека инструкција. Свака микроинструкција се извршава у једном циклусу током кога контролни сигнали из те микроинструкције управљају компонентама рачунарског система. Адресни битови из тренутне микроинструкције се комбинују са садржајем IR и PSW, чиме се добија адреса наредне микроинструкције.

143. Објаснити хоризонтални формат микроинструкција процесора.

Код хоризонталног формата микроинструкција процесора сви контролни битови постоје као појединачни битови микроинструкције и могу се директно усмерити ка одговарајућим компонентама процесора.

144. Објаснити вертикални формат микроинструкција процесора.

Код вертикалног формата микроинструкција процесора идеја је да се препознају неке групе контролних битова међу којима само један буде упаљен и да се коришћењем декодера смањи величина микроинструкције.

145. Карактеристике меморија.

Адресивост – меморија се може посматрати као низ меморијских локација. Ако свака меморијска локација има адресу помоћу које јој можемо приступити, кажемо да је меморија адресива. Ако су адресиве меморијске локације већих величина, кажемо за меморију да је полуадресива.
Врста приступа – означава како се приступа одређеним меморијским локацијама.
Капацитет – количина података која се може сместити у меморији.
Брзина – изражава се кроз кашњење (латенцију), тј. време које протекне од иницијализације трансфера до његовог стварног почетка, као и кроз брзину трансфера, тј. колико података се може пренети у једној секунди.
Постојаност – означава да ли се садржај меморије одржава или губи при губитку напона.
Променљивост – означава да ли је могуће изменити садржај меморије.

146. Навести могуће начине приступа меморији.

Постоје произвољан, секвенцијалан, директан и асоцијативан приступ меморији.

147. Објаснити секвенцијалан приступ меморији.

Меморијским локацијама се може приступати искључиво редом. Овај приступ је био карактеристичан за магнетне траке.

148. Објаснити директан приступ меморији.

Могуће је приступити неким меморијским локацијама, а потом се њиховим околинама приступа секвенцијално. Овај приступ је карактеристичан за масовна складишта као што су HDD, SSD, Flash меморије, CD/DVD итд.

149. Објаснити произвољан приступ меморији.

Могуће је приступити било којој меморијској локацији у приближно константном времену. Овај приступ је карактеристичан за оперативну меморију.

150. Објаснити асоцијативни приступ меморији.

Податку се не приступа преко адресе, већ преко одређене маске која се потом претражује у меморији. Овај приступ је карактеристичан за кеш меморије.

151. Шта је капацитет меморије и у којим јединицама се изражава?

Капацитет меморије представља количину података који се могу сместити у меморију и изражава се у KB, MB, GB и TB.

152. Какве меморије могу бити с обзиром на трајност (постојаност) записа? Примери.

Меморије могу бити постојане, тј. њихов садржај се не губи приликом губитка напона, или непостојане, код којих се губитком напона губи и садржај. Примери непостојаних меморија су регистри процесора, оперативна меморија и кеш меморија. Примери постојаних меморија су HDD, SSD, CD/DVD и Flash.

153. Какве меморије могу бити с обзиром на променљивост њиховог садржаја? Примери.

Меморије могу бити променљиве и непроменљиве. Непроменљива меморија је ROM, променљиве су оперативна меморија, регистри процесора, HDD, SSD итд.

154. Како се изражава брзина меморије? Који фактори највише утичу на брзину меморије?

Брзина меморије се обично изражава у MHz. На брзину меморије највише утичу кашњење (латенција), тј. време које протекне од иницијализације трансфера до његовог стварног почетка, као и брзина трансфера, тј. колико података се може пренети у једној секунди.

155. Објаснити хијерархију меморија.

Што је меморија ближа процесору то је мања и бржа. Меморије се могу поделити на унутрашње и спољашње. Унутрашње меморије су брже, али су углавном непостојане. Спољашње меморије су спорије, немају произвољан приступ, али су постојане, те се користе као медијуми за масовно чување података.
Меморијска хијерархија: регистри процесора → кеш меморија → оперативна меморија (RAM) → ROM → HDD/SSD → CD/DVD/Flash.

156. Шта је ROM? Какве врсте постоје? Где се користи?

ROM је постојана непроменљива меморија са произвољним приступом која у себи садржи програме ниског нивоа који се користе за покретање рачунара. Најчешће се имплементира као логичко коло. Поред ROM-а, постоје и PROM (Programmable ROM), EPROM (Erasable Programmable ROM) и EEPROM (Electrically Erasable Programmable ROM).

157. Шта је RAM? Какве врсте постоје?

RAM (Random Access Memory) је непостојана меморија са произвољним приступом. Постоје статички RAM (SRAM) и динамички RAM (DRAM).

158. Шта је статички RAM и које су његове основне карактеристике? Где се користи?

SRAM је меморија која се имплементира помоћу секвенцијалних кола као што су резе и флип-флопови, самим тим му не треба освежавање да би памтио податке. SRAM је изузетно брз, променљив, али није постојан и изузенто је скуп за израду, због чега су меморије израђене кроз SRAM обично мале. Користи се за регистре процесора и кеш.

159. Шта је динамички RAM и које су његове основне карактеристике? Где се користи?

DRAM је меморија која се имплементира помоћу кондензатора које контролишу транзистори. Због тога што кондензатори временом губе наелектрисање потребно је периодично допуњавати их. DRAM је због тога спорији. Променљив је и није постојан. С дру-

ге стране, релативно је јефтин за израду, данас се меморије имплементиране као DRAM мере у гигабајтима. Користи се за оперативну меморију.

160. Шта су испреплетане меморије? Објаснити.

Испреплетане меморије су једна од техника смањења кашњења код приступу суседним адресама у DRAM меморијама. Имплементира се тако што се меморија подели на мање целине – банке, где нижи битови адресе представљају индекс банке којој приступамо, а виши битови адресе индекс елемента у банци. Овим се постиже да суседне банке (у којима су суседне адресе) одмах читају одговарајуће податке (они деле више битове), чиме им се суштински преклапају кашњења.

161. Које врсте пресликавања меморијских адреса разликујемо? Објаснити.

Постоје пуно и делимично пресликавање меморијских адреса. Код пуног пресликавања сваком меморијском модулу одговара тачно једна комбинација виших битова адресе. С друге стране, код делимичног пресликавања меморијских адреса истом модулу може одговарати више комбинација виших битова адресе.

162. Објаснити пуно пресликавање меморијских адреса.

Код пуног пресликавања сваком меморијском модулу одговара тачно једна комбинација виших битова адресе.

163. Објаснити делимично пресликавање меморијских адреса.

Код делимичног пресликавања меморијских адреса истом модулу може одговарати више комбинација виших битова адресе.

164. Објаснити поравнавање података (меморија).

Оперативна меморија обезбеђује приступ сваком бајту, међутим, меморијска магистрала је шира од тога, па се онда преносе меморијски блокови. Стога се обично тражи да прва адреса у блоку који се преноси буде дељива са неким степеном двојке.

165. Навести основне врсте спољашњих меморија и навести њихове карактеристике.

Магнетне траке – имају секвенцијалан приступ и споре су. Данас су у потпуности превазиђене.
Хард дискова – меморије са намагнетисаним плочама и механичким деловима. Имају директан приступ. Релативно су јефтини и имају велики капацитет, међутим, доста су спорији од свих унутрашњих меморија.
Оптички дискови – имају директан приступ, мање су брзине од хард дискова.

Flash и SSD – базирани су на истој технологији као и EEPROM, али имају директан приступ, па су доста већег капацитета.

166. Објаснити намену и основни принцип рада кеша.

Кеш меморија представља релативно малу количину меморије која се имплементира као SRAM, а по хијерархији се налази између регистара процесора и оперативне меморије. Глана улога му је ублажи разлику у брзинама процесора и оперативне меморије тиме што ће неки подаци бити пребачени у кеш, што може значајно да убрза извршавање програма. Главни принцип рада кеша је принцип локалности.

167. Објаснити принцип локалности, шта је просторна а шта временска локалност? Примери.

Принцип просторне локалности – ако је неки податак био коришћен, велика је вероватноћа да ће и наредни податак бити коришћен. Ово важи како за инструкције тако и за податке.
Принцип временске локалности – ако је неки податак био скоро употребљен, велика је вероватноћа да ће опет бити употребљен. С друге стране, ако неки податак дуго није био употребљен, вероватно ни неће бити употребљен.

168. На који начин кеш користи принципе просторне и временске локалности?

Принцип просторне локалности се користи тако што се цела околина неког податка копира из оперативне меморије у кеш. Принцип временске локалности се користи тако што се скоро коришћени подаци задржавају у кешу, а они који дуго нису коришћени се мењају.

169. Објаснити читање кеша у случају поготка.

У случају поготка податак који је тражен у оперативној меморији је пронађен у кешу, у ком случају се податак у процесор уписује из кеша.

170. Објаснити читање кеша у случају промашаја.

У случају промашаја податак који је тражен у оперативној меморији се не налази у кешу, па се он чита из оперативне меморије, а он се, заједно са околином, убацује у кеш.

171. Објаснити писање кеша у случају промашаја.

У случају промашаја се одговарајући податак са околином пребацује у кеш, где се ажурира. Код политике write-through се ажурира и податак у оперативној меморији, док се код политике write-back ажурирање оперативне меморије оставља за касније.

172. Објаснити писање кеша у случају поготка.

У случају поготка ажурира се податак у кешу. Код политике write-through се ажурира и податак у оперативној меморији, док се код политике write-back ажурирање оперативне меморије оставља за касније.

173. Шта је пресликавање адреса кеша и које врсте пресликавања постоје?

Пресликавање адреса кеша је начин доделе блоковима података из оперативне меморије који се пребацују у кеш адресу кеш линије у којој ће се налазити. Постоје директно, асоцијативно и сет-асоцијативно пресликавање.

174. Објаснити непосредно пресликавање адреса кеша и дати пример.

Код директног/непосредног пресликавања сваки блок меморије може се сместити у само једну кеш-линију. На пример, замислимо да је величина кеша 1MB=2^{20}B, а да је величина сваке кеш-линије 32B. Дакле, имамо 2^{15} кеш-линија, свакој можемо доделити јединствени 15-битни број. За проналазак конкретног податка унутар кеш-линије користимо померај дужине 5 битова. Ако имамо 32-битне адресе, сада ће нам најнижих 5 битова адресе представљати померај, наредних 15 битова број кеш-линије, а преосталих 12 битова ће представљати тзв. таг. Блокови меморије се смештају у кеш-линију која одговара њиховој адреси. У свакој кеш-линији се, поред, у нашем примеру 32B података, налази и тзв. валидни бит, који означава да ли су уопште неки подаци смештени у ту кеш-линију, као и таг, који је у нашем примеру величине 12 битова. Када се проверава да ли је неки податак у кешу, одмах се може закључити коју кеш-линију треба гледати. Прво се проверава валидни бит, ако је он 0, значи да је у питању промашај. У супротном се проверава таг. Ако он одговара траженој адреси, то је погодак, иначе промашај. Дакле, то што сваком податку одговара тачно једна кеш-линија значајно олакшава тражење података у кешу, међутим, може се десити да ће често морати да се мењају одређене кеш-линије.

175. Објаснити асоцијативно пресликавање адреса кеша и дати пример.

Код асоцијативног пресликавања сваки блок може да се смести у било коју кеш-линију. Тако би, по пређашњим претпоставкама, нижих 5 битова адресе означавало померај, који би говорили о томе где се конкретан податак налази у кеш-линији, док би осталих 27 битова представљали таг. Приликом тражења податка у кешу, потребно је упоредити тагове свих кеш-линија са траженим, што значајно отежава имплементацију. Ипак, овај приступ даје теоријски најбољу искоришћеност кеша, јер не морамо да избацујемо неку коришћену линију ако има места у кешу.

176. Објаснити скуп-асоцијативно пресликавање адреса кеша и дати пример.

Код сет-асоцијативног пресликавања кеш-линије су подељене у скупове од по 2, 4, 8 или 16 линија. Сваки блок се може сместити у само један од скупова, а унутар скупа у било коју линију. Претпоставимо да су скупови од по 4 кеш-линије. Опет најнижих 5 битова адресе означава померај. Наредних 13 битова означавају број скупа, а преосталих 14 битова представља таг. Када се тражи податак у кешу, потребно је проверити све тагове унутар одговарајућег скупа.

177. Шта су и чему служе политике замењивања кеша? Набројати их.

Политике замењивања кеша служе за одређивање кеш-линије која треба да буде избачена да би се ослободио простор за нови блок који треба убацити у кеш. Користе се LRU, pseudo-LRU, FIFO и LFU.

178. Објаснити политику замењивања најдуже некоришћене линије кеша (LRU). Добре и лоше стране.

Код LRU се избацује она линија која је најдуже некоришћена. Како се ова метода поклапа са принципом временске локалности, она даје најбоље резултате, међутим, захтевна је за имплементацију на великом броју линија због превеликог броја могућих стања.

179. Објаснити политику замењивања псеудо-најдуже некоришћене линије кеша (pseudo-LRU). Добре и лоше стране

Код pseudo-LRU се кеш-линије деле на половине, где се памти у којој од половина је био најскорије коришћена линија, па се поступак наставља за половину где није најскорије коришћена линија. Ова политика је једноставнија за имплементацију, али није толико прецизна.

180. Објаснити FIFO политику замењивања линије кеша.

Код FIFO (First In First Out) се избацује линија која је прва ушла у кеш. Релативно лако се имплементира, помоћу једне кружне листе.

181. Које политике писања кеша постоје и у чему се разликују?

Write-through – када дође до измене у кешу, промена се уписује и у оперативну меморију.
Write-back – када дође до измене у кешу, промена се не уписује одмах у оперативну меморију, већ постоји тзв. „прљави бит“ који означава да ли је долазило до промене. Ако „прљави бит“ показује да је било промене, онда ће се промена преписати у оперативну меморију приликом избацивања линије из кеша.

182. Објаснити политику писања кеша са пропуштањем (write-through). Добре и лоше стране.

Када дође до измене у кешу, промена се уписује и у оперативну меморију. Добра страна је то што су подаци у оперативној меморији ажурни, као и то што је политика једноставна за имплементацију. Лоша страна је то што је спорија, због честих приступа оперативној меморији.

183. *Објаснити политику писања кеша са преписивањем (write-back). Добре и лоше стране.*

Када дође до измене у кешу, промена се не уписује одмах у оперативну меморију, већ постоји тзв. „прљави бит“ који означава да ли је долазило до промене. Ако „прљави бит“ показује да је било промене, онда ће се промена преписати у оперативну меморију приликом избацивања линије из кеша. Добра страна је то што ова политика додатно штеди време, али је компликована за имплементацију.

184. *Раздвојени и унификовани кеш. Поређење.*

Код раздвојеног кеша постоје одвојене кеш меморије за инструкције и за податке. Код унификованог кеша подаци и инструкције се чувају у истој кеш меморији. Предности раздвојеног кеша су те што се на различит начин може примењивати принцип локалности за инструкције и податке. Такође, код инструкција нема промена, па нема ни политика писања, па је имплементација једноставнија. Међутим, може се десити да један од ова два раздвојена кеша буде релативно непопуњен, док би из другог можда било потребно и избацивати линије.

185. *Објаснити архитектуре вишестепеног кеша и начин њиховог функционисања.*

Како би се додатно попунила празнина у меморијској хијерархији између регистара процесора и оперативне меморије, додају се додатни нивои кеша, који постају све већи и све спорији. Предвиђено је да најкоришћенији подаци буду у кешу најнижег нивоа, а како им опада употреба да се спуштају низ хијерархију. Приликом претраге, подаци се прво траже у кешу најнижег нивоа и даље редом. Када се искористи податак из кеша који није најнижег нивоа, он се пребацује тамо.
Код инклузивне организације сви подаци из кеш меморија нижег нивоа преписују се и у кеш меморије вишег нивоа, где је могуће да меморије вишег нивоа имају веће линије, док то није случај код ексклузивне организације вишестепеног кеша, где све кеш меморије имају линије исте величине.
На модерним рачунарима обично постоје 3 нивоа кеша (L1, L2, L3). L1 и L2 припадају појединачним језгрима, док L3 припада свим језгрима процесора. Само је L1 раздвојен, обично је величине 32KB+32KB са 8-сет-асоцијативним пресликавањем. L2 је величине око 256KB са 4-сет-асоцијативним пресликавањем. L3 је величине 2 до 4 мегабајта са 8 или 16-сет-асоцијативним пресликавањем.

186. Објаснити однос величине кеша и перформанси.

Већи кеш је по правилу спорији, међутим, показује се да повећавање кеша до неке границе убрзава систем, а након те границе даља повећања не убрзавају систем. То се може објаснити тиме што се код одређене величине кеша вероватноћа поготка толико приближи 100% да даље увећавање није могуће.

187. Објаснити однос величине линије кеша и перформанси.

Код малих линија кеша се не користи довољно просторна локалност, док се код великих линија у кеш довлачи превише података који се испостављају непотребним. Оптималном величином кеш-линије се сматра 32 бајта.

188. Објаснити однос асоцијативности и перформанси.

Повећавањем степена асоцијативности расте и шанса за погодак у кешу, стога су потпуно асоцијативне кеш меморије најбоље. Међутим, повећава се значајно и сложеност имплементације, па се стога најчешће користе сет-асоцијативна пресликавања, са што је већим скуповима могуће.

189. Шта је магистрала и чему служи?

Магистрала је скуп линија које повезују два или више уређаја и којима се преносе подаци и инструкције.

190. Како се остварује дељење магистрале? На који начин се спречава колизија сигнала? Објаснити.

Да бисмо повезали више уређаја на магистралу, потребно је спречити истовремено пуштање сигнала од стране различитих уређаја. Ово се обично постиже коришћењем бафера са три стања, где се помоћу контролног сигнала одређује који уређај може да пушта сигнал.

191. Шта је трансакција, а шта операција магистрале? Шта је протокол магистрале?

Трансакција на магистрали је било који заокружен скуп акција. Свака трансакција се састоји од неколико операција. У сваком тренутку на магистрали се може обављати највише једна трансакција. Протокол магистрале представља скуп правила према којима се одвија трансакција између нека два уређаја.

192. Шта су серијске, а шта паралелне магистрале? Поређење.

Серијске магистрале се састоје од једне линије преко које се информације преносе бит по бит. Паралелне магистрале се састоје од више линија, а информације се преносе реч по реч.

Две главне мане паралелних магистрала су искривљеност, појава да се не пренесу сви битови истовремено, што значајно отежава читање података, као и интерференција, која настаје као последица честих промена вредности и због које може доћи до промене вредности на линијама, а што резултује погрешним преносом. Серијске магистрале немају ове мане, а имају додатну предност да се могу обично реализовати као двосмерне додавањем још једне линије. Стога се данас готово свугде примењују серијске магистрале, осим на меморијској магистрали, која повезује процесор и оперативну меморију, а која је паралелна магистрала.

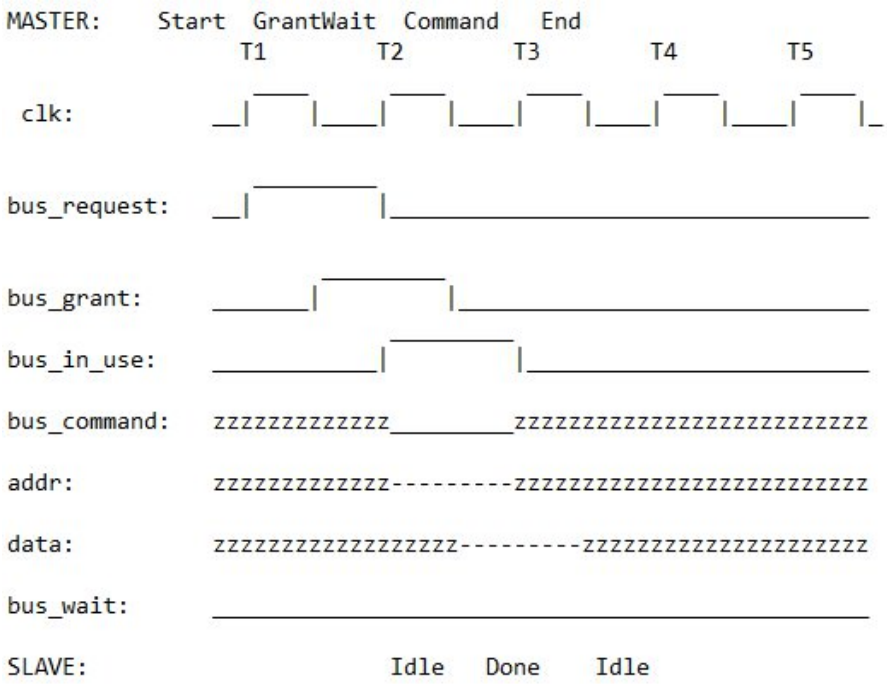
193. *Која је разлика између мултиплексираних и раздвојених магистрала? Поређење.*

Паралелна магистрала може бити мултиплексирана, што значи да се подаци и адресе преносе истим линијама, или раздвојена, што значи да постоје одвојене линије за податке и одвојене линије за адресе. Мултиплексиране магистрале су једноставније, али су и спорије, јер је потребно више циклуса за обављање трансакције.

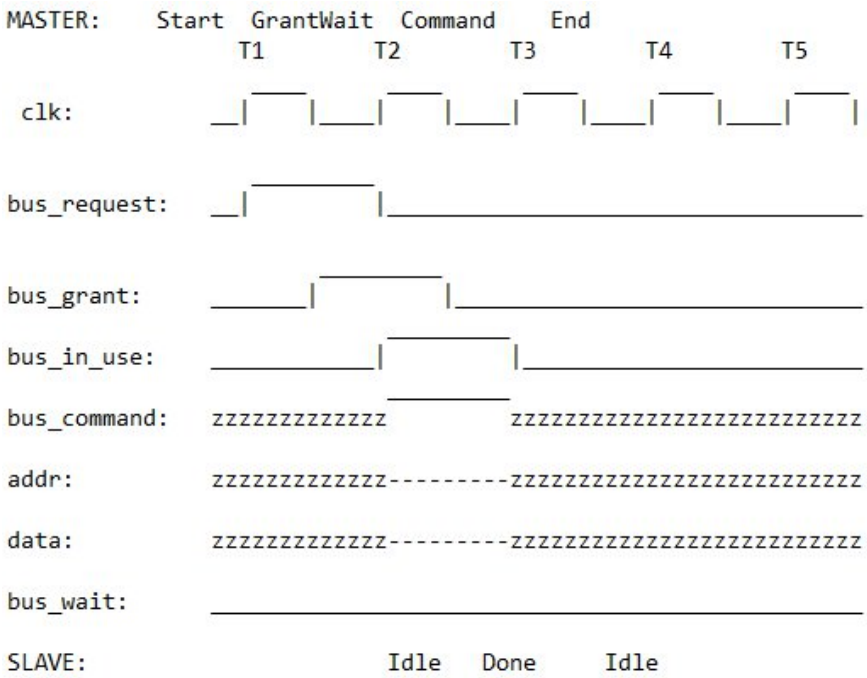
194. *Шта је ширина магистрале?*

Ширина паралелне магистрале је број линија, тј. битова у магистрали. Ширина магистрале одређује границу за вредности адреса и података које можемо да користимо.

195. *Објаснити и представити временским дијаграмом извршавање операције читања у случају синхроне магистрале.*

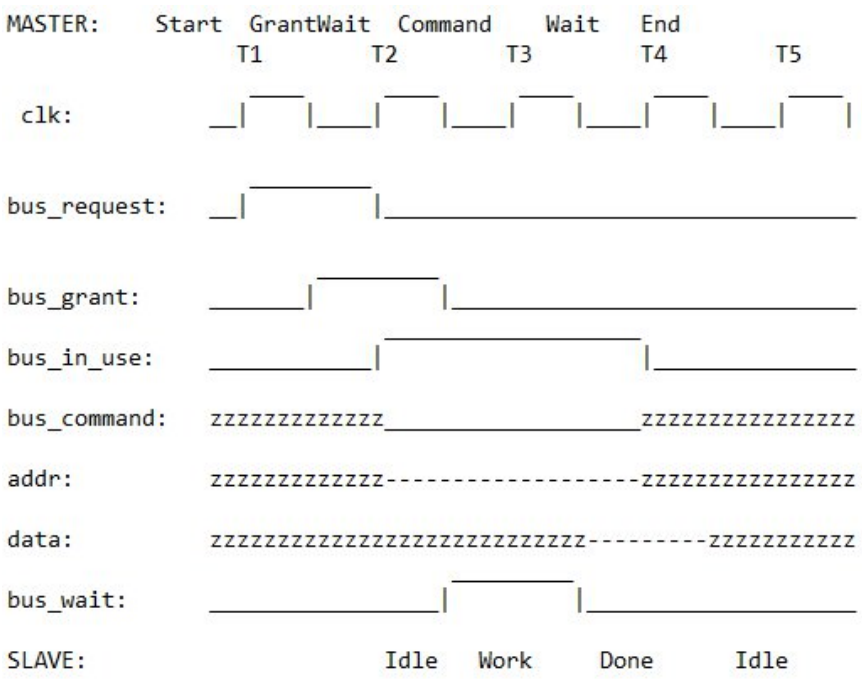


196. *Објаснити и представитби временским дијаграмом извршавање операције писања у случају синхроне магистрале.*



197. *Шта је стање чекања? Када се и како употребљава? Објаснити операцију читања са стањем чекања.*

Процесор је значајно бржи од свих осталих уређаја, па је потребно дати му сигнал да не ради ништа док други уређај, нпр. меморија, није завршио са претходним захтевом.



198. Шта је „преношење блокова података“? Када се и за шта употребљава?

„Преношење блокова података“ подразумева да се једном сложеном операцијом преноси више сложених речи. Обично је ово брже од преношења речи кроз више одвојених операција, због кашњења меморије. Овако се често попуњава кеш.

199. Шта је „read-modify-write“ трансакција и за шта се употребљава?

Овај тип операције омогућава атомичко читање и писање у меморију. Током трајања ове операције нико не може приступати магистрали, што обезбеђује атомичност. Ова операција се обично користи приликом имплементације оперативних система за реализацију ексклузивног приступа.

200. Како се синхронизује рад на асинхроној магистрали? Објаснити сигнале и ток активности (четворофазно руковање).

Рад на асинхроној магистрали се синхронизује помоћу протокола попут „четворофазног руковања“. Пошто асинхроне магистрале немају заједнички часовник, уређаји реагују одмах на пристигле сигнале. Код четворофазног руковања уводе се два контролна сигнала MSYN и SSYN. На почетку master уређај пали MSYN сигнал и поставља податке на магистралу, чиме јавља slave уређају да почиње трансакцију. Slave обавља трансакцију и пали SSYN сигнал да обавести master уређај да је трансакција готова. Потом master гаси MSYN, а онда и slave гаси SSYN и враћа се у полазно стање.

201. Предности и мане асинхроних магистрала у односу на синхроне.

У теорији су асинхроне магистрале брже, међутим, када се имплементира синхрона магистрала између само два уређаја, могуће је подесити такт синхроне магистрале тако да пренос буде бржи него у случају асинхроне. Поред тога, асинхроне магистрале је знатно теже имплементирати. Због свега тога, данас се углавном користе синхроне магистрале.

202. Шта је арбитража магистрале? Објаснити разлику између централизоване и дистрибуиране арбитраже.

Арбитража представља решавање конфликата приликом истовремених захтева за приступ магистрали од стране више уређаја. Код цетрализоване арбитраже постоји једна компонента (арбитар) која разрешава све конфликте. С друге стране, код дистрибуиране арбитраже уређаји који истовремено захтевају приступ магистрали комуницирају међу собом и тако одређују коме ће припасти магистрала.

203. Набројати и укратко објаснити политике додељивања магистрале.

Политика фиксираних приоритета – сви уређаји имају унапред додељене приоритете, на основу којих се и врши арбитража.
Политика ротирајућих приоритета – приоритети уређаја нису унапред фиксирани, већ се мењају кроз време. На пример, може се направити политика где уређај који најдуже чека на магистралу има највећи приоритет или да, на пример, уређај који је управо користио магистралу добије најмањи приоритет.
Хибридне политике – подразумевају комбинације претходне две.

204. Навести и укратко објаснити политике ослобађања магистрале.

Политике без преузимања – уређају који је добио приступ магистрали се он не може одузети, тј. магистрала се ослобађа тек када уређај који има приступ то дозволи. Постоје две варијанте ове политике, једна је да се магистрала ослобађа одмах по завршетку трансакције, а друга је да се магистрала ослобађа по завршетку трансакције само ако неки други уређај тражи приступ магистрали.
Политике са преузимањем – могуће је одузети приступ магистрали уређају ако је приступ тражио уређај вишег приоритета.

205. Објаснити детаљно механизам уланчавања код централизоване арбитраже.

Код механизма уланчавања захтев за приступ представља дисјункцију појединачних захтева за приступ уређаја. Другим речима, арбитар види да је неки од уређаја тражио приступ, али не зна и који. Потом, арбитар шаље сигнал за приступ који пролази редом кроз све уређаје. Ако уређај није тражио приступ, он прослеђује сигнал следећем у низу. Ако уређај јесте тражио приступ, када до њега стигне сигнал он приступа магистрали и престаје да прослеђује сигнал за приступ другим уређајима. Ова метода је једноставна за имплементацију, али подржава само политику фиксних приоритета.

206. Објаснити детаљно механизам независних захтева код централизоване арбитраже.

Сваки уређај је повезан са арбитром и шаље захтев за приступ директно арбитру. Такође, арбитар шаље сигнал за приступ тачно оном уређају којем је приступ одобрен. Овај механизам омогућава сложеније политике доделе, али је захтевнији за имплементацију.

207. Електричне карактеристике серијских магистрала.

Код серијских магистрала теоријски је потребна само једна жица код које нулти напон означава 0, а позитиван напон (нпр. +5V) означава 1. Међутим, како обично нису сви уређаји повезани на исту масу, потребно би било додати жицу која би била повезана на масу, да би се тиме обезбедило да уређаји очитавају напон у односу на исти нулти потенцијал. Међутим, оваква имплементација је подложна грешкама услед електромагнетне индукције из околине, па је чешће наредно решење. Постоје две жице D+ и D-. Логичко 1 се означава позитивним напоном на првој, а негативним на другој, док је за логичку 0 обрнуто. Другим речима, позитивна разлика потенцијала означава 1, а негативна 0.

208. Навести најчешће начине кодирања битова код серијског преноса.

Повратак на нулу (RZ) – након сваког пренетог бита се разлика потенцијала враћа на нулу, чиме се обезбеђује исправна синхронизација. Мана овог приступа је што значајно успорава пренос.
Без повратка на нулу (NRZ) – нема паузе између два бита, међутим, у овом приступу је лако изгубити синхронизацију, поготово код дужег низа истих битова. Ови проблеми се отклањају неким од следећих кодирања: NRZI, 8b/10b, 64b/66b, 128b/130b, 128b/132b.

209. Која је основна предност, а која мана NRZ кодирања у односу на RZ кодирање?

Предност је већа фреквенција, што значи и већа брзина преноса. Мана је потенцијални губитак синхронизације са часовником предајника уколико неко време не дође до промене напона, што се дешава код дужих низова истих битова.

210. Објаснити NRZI кодирање.

Код NRZI кодирања, ако је наредни бит требало да буде 0, шаље се инвертовани претходни бит, док ако је наредни бит требало да буде 1, шаље се исти бит као претходни. Овим се одмах решава проблем дугих низова 0, док се проблем дугих низова 1 може решити додатно тако што се након сваких 5 или 6 јединица у оригиналној поруци уметне 0, која ће променити напон.

211. Укратко објаснити 8b/10b кодирање. Који је разлог за коришћење овог начина кодирања?

Код 8b/10b кодирања се низ од 8 битова кодира низом од 10 битова који су изабрани тако да релативно често долази до промене напона, као и да се постигне DC-баланс, тј. да у поруци буде приближно једнак број 0 и 1. Мана овог кодирања је велик број додатних битова, због чега се данас чешће користе кодирања као што су 64b/66b, 128b/130b, 128b/132b, а која су направљена на сличној идеји.

212. Навести најчешће коришћене паралелне магистрале и њихове најважније карактеристике.

ISA – раздвојена паралелна магистрала са 16-битном магистралом података и 24-битном меморијском магистралом. Повезивала је све уређаје. Фреквенција јој је била до 8,33MHz, а брзина преноса око 8MB/s. Била је актуелна 80-их.
PCI – мултиплексирана паралелна 64-битна магистрала која је повезивала процесор са периферијским уређајима. Фреквенција јој је била 66MHz, а брзина преноса 528MB/s. Њеном појавом се мења архитектура тако да више не постоји једна магистрала која повезује све уређаје, већ више различитих магистрала које су међусобно повезане у мостовима.
Меморијска магистрала – раздвојена паралелна 64-битна магистрала која повезује процесор и оперативну меморију. Подржава два трансфера по циклусу, а са DDR4 меморијом брзина трансфера је око 25,6GB/s.

213. Навести најчешће коришћене серијске магистрале и њихове најважније карактеристике.

PCI Express – двосмерна серијска магистрала која служи за повезивање процесора са периферијским уређајима (графичка карта, звучна карта, мрежна карта итд.). Омогућава директну конекцију између свака два уређаја.
SATA – серијска магистрала која служи за повезивање са дисковима. Омогућава директну конекцију између диска и SATA контролера, који се обично налази у јужном мосту.
USB – служи за повезивање са екстерним уређајима. Нуди различите врсте трансфера у зависности од врсте уређаја. Још једна значајна особина је „plug and play“, тј. могућност да USB контролер аутоматски конфигурише уређај приликом повезивања.

214. Шта је систем прекида и која му је улога?

Систем прекида је механизам измене контроле тока програма. Основна улога му је да омогући процесору да реагује на догађаје који захтевају хитну обраду. Програм бива прекинут, при чему се памти његово стање, а наставља да се извршава ако је то могуће након решавања прекида.

215. Навести и укратко објаснити врсте прекида.

Хардверски прекиди – изазивају их уређаји повезани са процесором који шаљу посебне сигнале за прекид. Деле се на маскирајуће и немаскирајуће.
Софтверски прекиди – изазивају се посебним инструкцијама процесора
Изузеци – изазивају их грешке приликом извршавања инструкција

216. Објаснити хардверске прекиде. Шта су маскирајући, а шта немаскирајући прекиди?

Хардверски прекиди настају довођењем сигнала за прекид на један или више физичких прикључака на процесору. Зову се још и асинхрони прекиди. Могу се десити у било ком тренутку. Деле се на маскирајуће и немаскирајуће. Маскирајуће прекиде процесор може да игнорише неко време, док немаскирајуће мора одмах да решава.

217. Објаснити софтверске прекиде. Која је типична улога софтверских прекида?

Софтверски прекиди настају посебним инструкцијама процесора. На Intel архитектури постоји интрукција int. Зову се још и синхрони прекиди. Дешавају се у унапред предвиђеним тренуцима. Обично се користе за реализацију системских позива, за коришћење функционалности за које је потребан привилегован режим, а у који се са корисничке стране најчешће може ући само изазивањем прекида.

218. Шта су изузеци (у контексту система прекида) и чему служе?

Изузеци су прекиди који настају као последица грешака у извршавању инструкција. Служе да се прекине извршавање програма са грешком и да се, ако је могуће, грешка отклони.

219. Шта је вектор прекида? Шта је дескриптор прекида? Где се налази табела дескриптора прекида?

Вектор прекида представља редни број типа прекида који је настао. У меморији постоји табела дескриптора прекида, чије свако поље представља неки дескриптор прекида, који садржи информације о процедури за решавање прекида. Сваком вектору прекида одговара један дескриптор прекида.

220. Објаснити детаљно начин позивања руковаоца прекидом у случају векторских прекида.

221. Објаснити компоненте и рад контролора прекида PIC 8259.

Контролор прекида PIC 8259 могао је да се повеже са 8 улазних уређаја који би изазивали прекиде. Имао је 8-битни улаз/излаз преко којег је био повезан на магистралу пода-

така и преко којих је слао вектор прекида процесору. Такође, имао је и 3 8-битна регистра помоћу којих је памтио који су уређаји тражили захтев, који захтеви још нису решени, а такође и да маскира неке захтеве као и да спречи изузетке са неких уређаја.

222. Шта су улазно/излазни уређаји?

Улазно/излазни уређаји су уређаји који омогућавају комуникацију рачунара и корисника, тј. омогућавају унос података у рачунар и испис података.

223. Шта су улазно/излазни контролери и која је њихова функција?

У/И контролери су компоненте које врше комуникацију ниског нивоа са У/И уређајима и тиме омогућавају унификован интерфејс за комуникацију процесора и У/И уређаја, дајући процесору на располагање регистре за рад са У/И уређајем.

224. Шта подразумева употреба У/И уређаја путем меморијског мапирања?

Код меморијског мапирања У/И уређаја регистри за одговарајући У/И уређај које обезбеђује У/И контролер се смештају у адресни простор процесора, тј. процесор приступа њима на исти начин као и оперативној меморији. Последично, користе се исте инструкције за приступ регистрима У/И контролера и насумичној меморијској локацији.

225. Шта подразумева употреба У/И уређаја путем изолованог улаза и излаза?

Код изолованог улаза и излаза регистри У/И контролера су имали одвојени адресни процесор, те се њима приступало помоћу посебних инструкција.

226. Објаснити технику програмираног У/И.

Код програмираног У/И процесор је посредник у комуникацији са У/И уређајем. Процесор у бесконачној петљи проверава статус У/И уређаја, све док се не обаве потребне операције.

227. Објаснити технику У/И вођеног прекидима.

Код У/И вођеног прекидима процесор није све време заузет проверавајући статус У/И уређаја, већ У/И контролер даје захтев за прекид када се операција заврши и када процесор одлучује шта се даље дешава.

228. Објаснити директан приступ меморији (DMA). Контролер DMA. Кораци при реализацији DMA приступа.

Код директног приступа меморији, У/И уређај не комуницира са процесором, већ са DMA контролером. Када У/И уређај захтева читање или писање у меморију, процесор

даје DMA контролеру податке о томе који уређај захтева приступ, адреси у меморији, величини података, као и да ли је реч о читању или писању. Након тога, контролер тражи приступ магистрали на коју се ставља дата адреса. Потом контролер шаље статусне сигнале У/И уређају и меморији све док пренос траје. Када се пренос заврши, контролер шаље сигнал за прекид процесору, чиме га обавештава да је пренос завршен.

229. Шта је виртуелна меморија и због чега се користи?

Виртуелна меморија представља виртуелни сопствени адресни простор програма фиксиране величине. Програм види виртуелну меморију као једну нефрагментирану целину и сматра да је сам у њој. Виртуелна меморија не користи само оперативну меморију, већ и спољашња складишта, у која се стављају делови програма који се неко време не користе, а враћају се у оперативну меморију по потреби.
Користи се јер програм може имати већу виртуелну меморију него што има физичке меморије, не долази до преклапања програма и грешака узрокованих тиме, суседне адресе у виртуелној меморији не морају бити суседне и у физичкој, па ово отклања проблем фрагментације, а такође, омогућава употребу апсолутних адреса, јер програм сматра да је цела меморија његова.

230. Објаснити концепт страница виртуелне меморије. Шта су странице, а шта оквири страница?

Виртуелни адресни простор је подељен на странице једнаке величине. Слично, физичка меморија је подељена на оквире.

231. Објаснити пресликавање адреса виртуелне меморије. Пример.

Адресе у виртуелној меморији се пресликавају на адресе у физичкој меморији уз помоћ табеле страница. Табела страница се чува у физичкој меморији. За сваку страницу постоји податак у табели који се назива PTE (Page Table Entry) ставка. Свака ова ставка садржи индекс оквира у којем се страница налази, или пак информацију да се не налази у оперативној меморији. Када се приступа податку који је у страници која није у оперативној меморији долази до прекида приликом ког се одговарајућа страница смешта у оперативну меморију, а захтев за приступ податку се понавља, те се наставља извршавање програма.

232. Пресликавање адреса на више нивоа. Због чега се користи? Пример.

Табеле страница могу да буду толико велике да је немогуће издвојити толико узастопне меморије. Овај проблем се решава пресликавањем адреса на више нивоа. Табелу страница ћемо да поделимо на више табела, а које ће међусобно, на сличан начин, бити повезане у табели директоријум. Приликом превођења, највиши битови одговарају индексу табеле у директоријуму, а наредних неколико индексу странице у посматраној табели.

Овиме добијамо да се у оперативној меморији мора налазити само значајно мања табела директоријум, а касније се табеле страница могу смештати у произвољне оквире. Недостатак је у ефикасности, међутим и њега је могуће ублажити.

233. Шта су и када се користе политике замене страница?

Политике замене страница су алгоритми који се користе да би се страница у оперативној меморији заменила страницом која је била на диску, а примењују се када се оперативна меморија попуни, па није могуће алоцирати оквир за неку страницу коју желимо да читамо.

234. Навести и укратко објаснити најчешће политике замене странице.

FIFO – избацује се она страница која је прва ушла у оперативну меморију.
Second Chance – избацује се она страница која је најдуже у меморији, а није била коришћена.
LFU – избацује се она страница која је најмање пута коришћена.
Псеудо LRU – избацује се она страница која најдуже није била коришћена.

235. Објаснити значај величине странице виртуелне меморије и навести примере.

Повећавањем величина страница смањује се број страница, што представља мање оптерећење за оперативну меморију, а такође се ефикасније користи диск. Међутим, код већих страница већа је вероватноћа да ће доћи до интерне фрагментације, тј. да ће неке странице бити полу-празне, а такође се смањује и прецизност читања.

236. Шта су табеле, а шта директоријуми страница виртуелне меморије? Објаснити.

Табеле страница виртуелне меморије служе за пресликавање адреса из виртуелне меморије у оперативну меморију. Директоријуми су сличне мапе које служе за пресликавање табела страница виртуелне меморије у оперативну меморију. Можемо имати више нивоа директоријума, што ће допринети уштедама у меморији.

237. Шта садрже ставке у табели страница виртуелне меморије? Објаснити.

Ставке у таблици страница виртуелне меморије садрже податак о томе у ком оквиру оперативне меморије се налази тражена страница, или пак информацију да се не налази у оперативној меморији. Поред тога, ставке у таблици страница могу садржати и додатне информације о томе како се користе подаци у наведеној таблици и још неке додатне информације као што је на пример „прљави бит“ који служи као информација да ли је потребно преписати страницу на диску приликом избацивања из оперативне меморије.

238. Шта је бафер таблице страница виртуелне меморије (TLB) и чему служи?

TLB је посебан кеш, који се обично имплементира као асоцијативан или сет-асоцијативан раздвојени кеш у којем се чувају ставке из табела и директоријума страница виртуелне меморије. Служи да се не би сваки пут морало ићи до оперативне меморије по информације и локацији одређене странице виртуелне меморије, чиме се постиже значајно убрзање.

239. Објаснити принцип преклапања инструкција у модерним процесорима.

Уобичајено је да се у процесору обрађује само једна инструкција. Међутим, како се различите фазе обраде одвијају у различитим јединицама, могуће је паралелизовати процес тако да се више инструкција обрађује истовремено, свака у некој одређеној функционалној јединици процесора, где је и једина која се обрађује.

240. Који су основни узроци заустављања „покретне траке“ код преклапања инструкција у савременим процесорима? Које се технике користе за решавање оваквих проблема?

За неке инструкције је потребно више од једног циклуса да се изврше. За неке је потребно више од једног циклуса да се дохвати из меморије, зато што су дуже од осталих или се не налазе у кешу. Декодирање захтевних инструкција може трајати више од једног циклуса, а такође и упис у меморију.
Проблем дохватања меморије се решава техником претходног дохватања, тј. да процесор има бафер у који се смешта више наредних инструкција.
Такође, ако процесор утврди да неке инструкције нису међусобно зависне, може да одлучи да примени технику извршавања ван редоследа, тј. да их изврши различитим редоследом од задатог.
Такође, могуће је да две различите инструкције које се истовремено извршавају траже исте ресурсе. Могуће је и да инструкција као параметар има излаз претходне инструкције, услед чега ће доћи до застоја. То се решава такозваним премошћавањем, тј. резултати инструкција постају познати и пре него што се упишу на одговарајуће место.

241. Објаснити технику извршавања ван редоследа.

Уколико се утврди да неке инструкције не зависе једна од друге, а утврди се да је нека захтевнија од осталих и узроковала би застој, нпр. због дохватања из меморије, инструкције се могу извршити тако да буду најпогоднije по преклапање.

242. Објаснити технике предикције гранања.

Идеја је да када се из меморије дохвати инструкција скока да се хардверски унапред одреди да ли до скока долази или не. Могуће је одредити да понашање сваки пут буде идентично (статички) или да се посматра неколико претходних скокова и да се на онову

тога одлучује (динамички). Показује се да динамичко одлучивање на основу претходне 3 инструкције у 90% случајева да тачну предикцију.

243. Шта су суперскаларни процесори?

Суперскаларни процесори су они код којих постоји већи број одређених функционалних јединица, обично ALU, па онда у процесору може да се истовремено обрађује више од једне инструкције.