Одговори на испитна питања из УОАР-а

Никола Маринковић

29. јун 2026.

1 Логичке функције и логички изрази

1 Написати истинитосне таблице основних логичких везника (НЕ, И, ИЛИ)

x	$\bar{x}$
0	1
1	0

Табела 1: Комплемент (Негација) (НЕ)

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

Табела 2: Дисјункција (ИЛИ)

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

Табела 3: Конјукција (И)

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

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

Табела 4: Шеферова функција (НИ)

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

Табела 5: Пирсова (Лукашиевичева) функција (НИЛИ)

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

Табела 6: Ексклузивна дисјункција (ЕИЛИ)

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

$$x \oplus y = x \cdot \bar{y} + \bar{x} \cdot y \tag{1}$$

4 Навести основне законе алгебре логике.

Аксиоме:

- $(x \cdot y) \cdot z = x \cdot (y \cdot z),\ (x + y) + z = x + (y + z)$ (асоцијативност)
- $x \cdot y = y \cdot x,\ x + y = y + x$ (комутативност)
- $x \cdot (y + z) = x \cdot y + x \cdot z,\ x + y \cdot z = (x + y) \cdot (x + z)$ (дистрибутивност)

- $x + 0 = x,\ x \cdot 1 = x$ (неутрални елемент)
- $x + \bar{x} = 1,\ x \cdot \bar{x} = 0$ (комплементарност)

Важни закони Булове алгебре:

- $x \cdot x = x,\ x + x = x$ (закони идемпотенције)
- $x \cdot 0 = 0,\ x + 1 = 1$ (закони нуле и јединице)
- $x \cdot (x + y) = x,\ x + x \cdot y = x$ (закони апсорпције)
- $\bar{\bar{x}} = x$ (закон двојне негације)
- $\overline{x + y} = \bar{x} \cdot \bar{y},\ \overline{x \cdot y} = \bar{x} + \bar{y}$ (де-Морганови закони)

Коментар: Принцип дуалности (можемо да променимо знаке и добијемо један закон из другог)

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

Коментар: Алгебра логике: двоелементна Булова алгебра

1. Имплементација уређаја који имају два стабилна стања је релативно једноставна, што омогућава прављење логичких кола (уређаји који израчунавају логичке изразе у савременој електронској технологији) на релативно једноставан, јефтин и поуздан начин.
2. Стандардне аритметичке операције над бинарним бројевима је једноставно описати на језику алгебре логике (бинарна аритметика помоћу логичких кола)

6 Шта значи да су два логичка израза еквивалентна?

Два логичка израза су еквивалентна ако имају једнаке вредности у свакој валуацији.

Коментар: Валуација је функција која свим променљивама придружује неку логичку вредност.

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

Литерал је логички израз који је или логичка променљива или негација логичке променљиве (пример: $x, \bar{y}, z$).

Елементарна конјукција је израз који се састоји из конјукције литерала (пример: $x\bar{y}z\bar{u}\bar{v}$).

ДНФ је израз који се састоји из дисјункције елементарних конјукција (пример: $x\bar{y}z + \bar{x}\bar{y}\bar{z} + xy\bar{z}$)

За елементарну конјукцију кажемо да је савршена (у односу на коначни скуп променљива) ако садржи тачно по један литерал за сваку променљиву из скупа (пример: за скуп $P = \{x, y, z\}$ једна савршена елементарна конјукција била би $x\bar{y}z$, а не би била $\bar{x}y$)

ДНФ је савршена ако се састоји из дисјункције савршених елементарних конјукција.

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

Замени појављивање речи конјукција и дисјункција из задатка 7.

9 Укратко описати поступак за свођење логичког израза на ДНФ.

e представља произвољан подизраз израза који се трансформише.

1. Елиминисање примене логичких везника над логичким константама (0 и 1).

$$\bar{0} = 1,\ \bar{1} = 0,\ e \cdot 0 = 0,\ e \cdot 1 = e,\ e + 0 = e,\ e + 1 = 1 \tag{2}$$

2. Учинимо да се негације примењују само на логичке променљиве (закон двојне негације и де-Морганови закони)

$$\bar{\bar{e}} = e,\ \overline{e_1 + e_2} = \bar{e_1} \cdot \bar{e_2},\ \overline{e_1 \cdot e_2} = \bar{e_1} + \bar{e_2} \tag{3}$$

3. Дисјункције се „извлаче“ из конјукција (дистрибутивни закон)

$$e \cdot (e_1 + e_2) = e \cdot e_1 + e \cdot e_2 \tag{4}$$

Коментар: Укратко значи елиминисати све формуле, остављене су овде ради јасноће.

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

Логичка функција реда n је било које пресликавање које свакој n-торци логичких вредности придружује логичку вредност.

$$f : \{0,1\}^n \to \{0,1\} \tag{5}$$

Логичких функција реда n има 2^{2^n} (број комбинација на улазу је 2^n, а на излазу се могу наћи две различите вредности).

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.

- (НЕ, И, ИЛИ)
- (НЕ, И)
- (НЕ, ИЛИ)
- (НИ)
- (НИЛИ)

Коментар: Сваки надскуп потпуног скупа везника је такође потпун скуп везника.

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

$$\begin{aligned} \bar{x} &= \overline{x \cdot x} = x \uparrow x \\ x \cdot y &= \overline{\overline{x \cdot y}} = \overline{\overline{x \cdot y} \cdot \overline{x \cdot y}} = (x \uparrow y) \uparrow (x \uparrow y) \\ x + y &= \overline{\bar{x} \cdot \bar{y}} = \bar{x} \uparrow \bar{y} = (x \uparrow x) \uparrow (y \uparrow y) \end{aligned} \tag{6}$$

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

1. за сваку комбинацију улазних вредности за коју функција има вредност 1 формирамо савршену елементарну конјукцију која је тачна само у тој комбинацији
2. направимо дисјункцију свих тако добијених савршених елементарних конјукција

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

Сложеност логичког израза је број везника који се у њему појављују.

Циљ минимизације је проналажење логичког израза минималне сложености еквивалентног датом изразу.

Значајна је у процесу дизајна логичких кола јер се тиме добија значајна уштеда у процесу производње, као и у потрошњи електричне енергије приликом употребе.

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

Примена одређених логичких закона на ДНФ израз ради смањивања сложености.
K - произвољна конјукција литерала

1. Груписање елементарних конјукција: уколико имамо две елементарне конјукције облика xK и $\bar{x}K$ (садрже супротне литерале по једној променљивој), можемо им смањити сложеност на следећи начин:

$$xK + \bar{x}K = (x + \bar{x}) \cdot K = 1 \cdot K = K \tag{7}$$

2. уколико K можемо груписати на два различита начина са две конјукције K_1 и K_2, примењујемо закон идемпотенције ($K = K + K$)

Горња два правила примењујемо по потреби.
Пример:

$$\begin{aligned} F(x,y,z) &= \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}z \\ &\overset{2}{=} \bar{x}\bar{y}\bar{z} + (\bar{x}\bar{y}z + \bar{x}\bar{y}z + \bar{x}\bar{y}z) + \bar{x}yz + x\bar{y}z \\ &= (\bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z) + (\bar{x}yz + \bar{x}\bar{y}z) + (x\bar{y}z + \bar{x}\bar{y}z) \\ &\overset{1}{=} \bar{x}\bar{y} + \bar{x}z + \bar{y}z \end{aligned} \tag{8}$$

Коментар: Метод алгебарских трансформација можемо применити и на ДНФ израз који није савршен.

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

Карноова мапа је таблица правоугаоног облика чији је укупан број поља једнак 2^n, (n - број променљивих у ДНФ изразу / ред функције). Свако поље таблице одговара једној валуацији. Поља сматрамо суседним уколико се разликују за један симбол, тј. ако су једно поред другог (замислимо да се лева и десна / горња и доња ивица додирују).

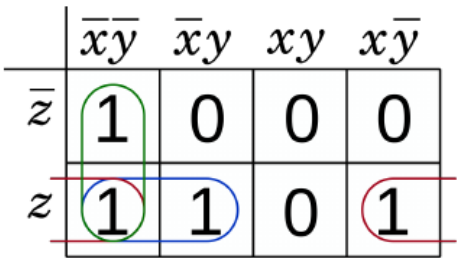
- У поља Карноове мапе се уписују одговарајуће вредности израза који се минимизује (0 или 1).
- Групишемо јединице у мапи, тако да свака јединица буде у бар једној од група. Правила заокруживања:

- Заокружују се само јединице.
- Свака јединица мора бити заокружена макар једном (дозвољено вишеструко заокруживање).
- Заокружујемо искључиво групе од 2^k поља правоугаоног облика.
- Увек се заокружују што веће групе (иако долази до преклапања).
- Ако је се на крају неко заокруживање испоставило као сувишно, избацујемо га.

- Сваком заокруживању одговара једна елементарна конјукција која садржи литерале који су заједнички за сва заокружена поља. (што је заокруживање веће, то их има мање, и конјукције су једноставније)

Пример:

$$F(x,y,z) = \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}z \tag{9}$$



Слика 1: Заокруживање код Карноове мапе

$$\Rightarrow F(x,y,z) = \bar{x}\bar{y} + \bar{y}z + \bar{x}z \tag{10}$$

17 Објаснити методу Квин-Мекласког за минимизацију логичких израза. Пример.

- *Савршене елементарне конјункције* се сортирају растуће по броју неинвертованих литерала и поделе у класе (i-та класа садржи све који имају i неинвертованих литерала).
- Уколико се две конјукције из суседних класа разликују за један литерал, групишу се и преносе у следећи корак.
- Поступак понављамо док не понестане конјукција. (све које нису искоришћене у било ком кораку (прости импликанти) се преносе на следећу фазу алгоритма)

•

- Формирамо табелу простих импликаната (врсте су *савршене елементарне конјукције*, а колоне прости импликанти)
- Означавамо (+) сва поља која имају особину да се прост импликант садржи у *савршеној елементарној конјукцији*.
- Колоне које садрже само један + означавају битне просте импликанте (*савршена елементарна конјукција* зависи само од једног, па се без њега не може)
- Уколико преостану *савршене елементарне конјукције* непокривене битним простим импликантима, тражимо додатне просте импликанте из скупа преосталих који их покривају.

Пример:

$$F(x,y,z) = \bar{x}\bar{y}\bar{z} + \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}z \tag{11}$$

0	$\bar{x}\,\bar{y}\,\bar{z}$
1	$\bar{x}\,\bar{y}z$
2	$\bar{x}yz$
	$x\bar{y}z$

0	$\bar{x}\,\bar{y}\,\bar{z}$	$\bar{x}\,\bar{y}$
1	$\bar{x}\,\bar{y}z$	$\bar{x}z$
		$\bar{y}z$
2	$\bar{x}yz$	
	$x\bar{y}z$	

Слика 2: Прва фаза: одабир простих импликаната

	$\bar{x}\,\bar{y}\,\bar{z}$	$\bar{x}\,\bar{y}z$	$\bar{x}yz$	$x\bar{y}z$
$\bar{x}\,\bar{y}$	$\oplus$	+		
$\bar{x}z$		+	$\oplus$	
$\bar{y}z$		+		$\oplus$

	$\bar{x}\,\bar{y}\,\bar{z}$	$\bar{x}\,\bar{y}z$	$\bar{x}yz$	$x\bar{y}z$
$\bar{x}\,\bar{y}$	$\oplus$	$\boxplus$		
$\bar{x}z$		$\boxplus$	$\oplus$	
$\bar{y}z$		$\boxplus$		$\oplus$

Слика 3: Друга фаза: битни прости импликанти

$$\Rightarrow F(x,y,z) = \bar{x}\bar{y} + \bar{y}z + \bar{x}z \tag{12}$$

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

Небитна вредност ће бити третирана као вредност 1 уколико нам омогућава веће заокруживање.

У супротном, биће третирана као вредност 0.

x	y	z	u	$F(x,y,z,u)$
0	0	0	0	0
0	0	0	1	–
0	0	1	0	0
0	0	1	1	1
0	1	0	0	–
0	1	0	1	1
0	1	1	0	–
0	1	1	1	–
1	0	0	0	–
1	0	0	1	–
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	–
1	1	1	0	1
1	1	1	1	–

Табела 7: Табела истинитости функције $F(x,y,z,u)$ с небитним вредностима

	$\overline{x}\,\overline{y}$	$\overline{x}y$	xy	$x\overline{y}$
$\overline{z}\,\overline{u}$	0	–	1	–
$\overline{z}u$	–	1	–	–
zu	1	–	–	0
$z\overline{u}$	0	–	1	0

Слика 4: Заокруживање небитних вредности

$$\Rightarrow F(x,y,z,u) = \bar{x}u + y \tag{13}$$

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

- У првој фази се небитне вредности третирају као 1 (учествују у груписању) (потенцијално проналазимо веће групе и смањујемо сложеност израза)
- У другој фази се небитне вредности третирају као 0 (не наводе се у табели простих импликаната) (не морамо да их покривамо јер функција у њима не мора да буде једнака 1)

Пример:
Задатак из табеле 7.

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

Слика 5: Прва фаза: прости импликанти (небитне вредности третирамо као 1)

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

Слика 6: Друга фаза: битни прости импликанти (небитне вредности третирамо као 0)

$$\Rightarrow F(x,y,z,u) = \bar{x}u + y \tag{14}$$

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

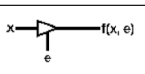
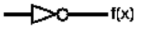
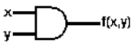
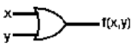
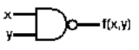
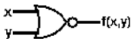
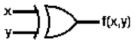
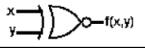
Користи се за проналажење најмањег могућег скупа додатних простих импликаната (најмањи број конјукција).

- Из табеле се бришу врсте које одговарају битним простим импликантима, и колоне које одговарају *савршеним елементарним конјукцијама* које су њима покривене.
- Изводимо нову функцију (Петрикову функцију) где свака заграда представља једну колону, а унутар заграде сабирамо редове (просте импликанте) који покривају ту колону
- Алгебарским трансформацијама поједностављујемо израз и то је наш подскуп додатних простих импликаната

2 Логичка кола

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

Бафер, бафер са три стања,
НЕ, И, ИЛИ, НИ, НИЛИ, ЕИЛИ, НЕИЛИ коло.

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*)	$f(x) = x$	
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*)	$f(x,y) = x \sim y$	

Слика 7: Логичке капије

22 Нацртати симбол и објаснити функцију NMOS транзистора.

Сорс и дрејн представљају два острва N типа која се налазе на подлози P типа. Сорс је повезан на негативан напон, а дрејн на позитиван. Да би дошло до провођења струје, потребно је на гејт довести велики позитиван напон (у односу на сорс).

Чинећи то, под утицајем електричног поља малобројни слободни електрони из P подлоге се гомилају око изолатора и формирају N канал, кроз који теку од сорса ка дрејну.

Транзистор (потпуно) проводи када је напон на гејту логичка јединица. У супротном, не проводи.

Симбол: 8.

Коментар: Ако ти није јасно шта се дешава, отвори скрипту на 49-50 страни.

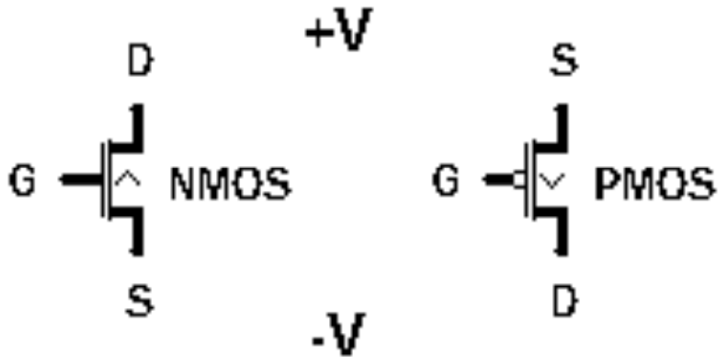
23 Нацртати симбол и објаснити функцију PMOS транзистора.

Сорс и дрејн представљају два острва P типа која се налазе на подлози N типа. Сорс је повезан на **позитиван напон**, а дрејн на **негативан**. Да би дошло до провођења струје, потребно је на гејт довести велики негативан напон (у односу на сорс).

Чинећи то, малобројне шупљине из N супстрата бивају привучене од стране електричног поља гејта и гомилају се испод изолатора и формирају P канал, кроз који се шупљине усмерено крећу од сорса ка дрејну.

Транзистор (потпуно) проводи када је напон на гејту логичка нула. У супротном, не проводи.

Коментар: Ознака PMOS транзистора садржи кружић на гејту и то је разликује од ознаке NMOS транзистора.



Слика 8: Ознаке NMOS и PMOS транзистора

1 Коментар: (не верујем да ће ово питати али је боље знати (страна 48 у скрипти))

Допирање је уградња релативно мале количине атома неког другог елемента различите валентности у кристалну решетку силицијума. (силицијум је четворовалентан елемент)

N-допирани силицијум подразумева допирање петовалентним елементом (нпр. фосфором). Он ће креирати четири ковалентне везе са суседним атомима силицијума, али ће његов преостали, пети валентни електрон, остати неповезан. Стога, требаће му мало енергије да постане слободан. Тиме не настаје шупљина, али ће концентрација слободних електрона бити знатно већа. Зове се N-допирани силицијум јер у њему доминирају негативни слободни носиоци наелектрисања.

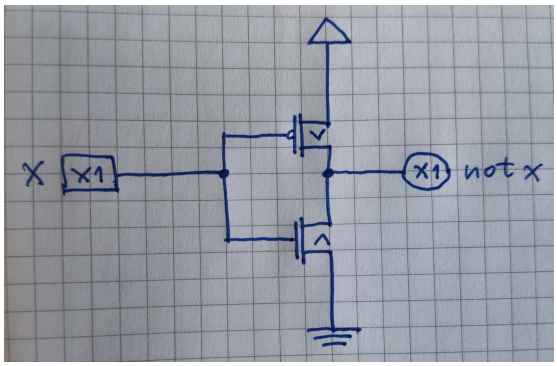
За P-допирани силицијум важи обрнуто, допира се тровалентним елементом (бор), и у њему доминирају позитивни слободни носиоци наелектрисања (већа концентрација шупљина).

MOS транзистори представљају полупроводничку компоненту која има три прикључка: сорс, дрејн и гејт. Уз одговарајуће напоне на сорсу и дрејну, MOS транзистор ради као прекидач - струја може да тече од сорса ка дрејну под условом да се одговарајући напон доведе на гејт.

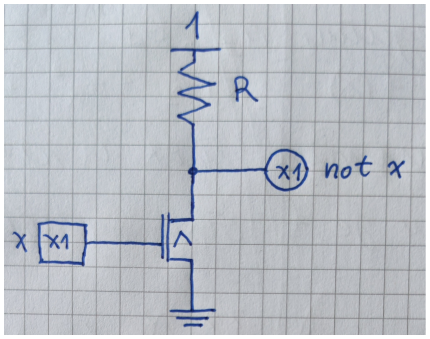
Кроз гејт не протиче струја, јер је изолатором одвојен од остатка транзистора. Његова улога је да својим електричним потенцијалом

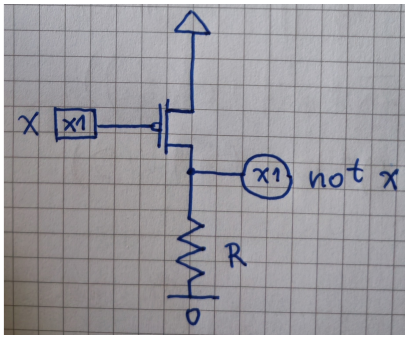
створи одговарајуће електрично поље које ће у транзистору привући носиоце наелектрисања и обезбедити проток струје од сорса ка дрејну.

24 Имплементација НЕ кола у CMOS-у.



Слика 9: НЕ коло у CMOS-у



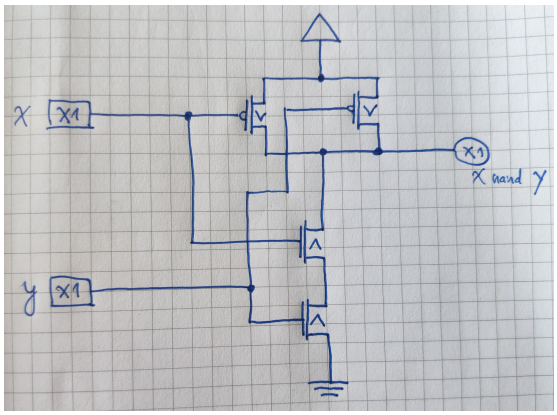


Слика 10: НЕ коло у NMOS-у (лево) и PMOS-у (десно)

25 Имплементација НИ и И кола у CMOS-у.

1 НИ (NAND) ↑

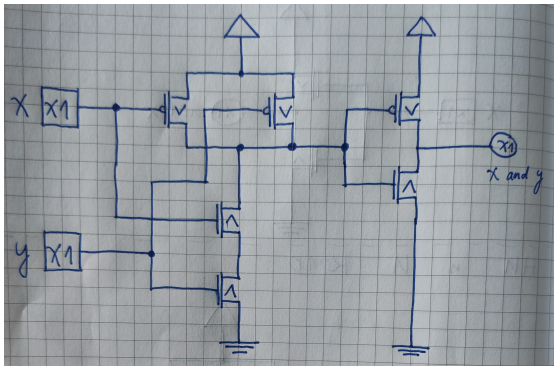
Горња мрежа: два паралелна PMOS транзистора.
Доња мрежа: два редно везана NMOS транзистора.



Слика 11: НИ коло у CMOS-у

2 И (AND) ·

Добија се тако што надовежемо НЕ коло на НИ коло (обрнуто интуицији).

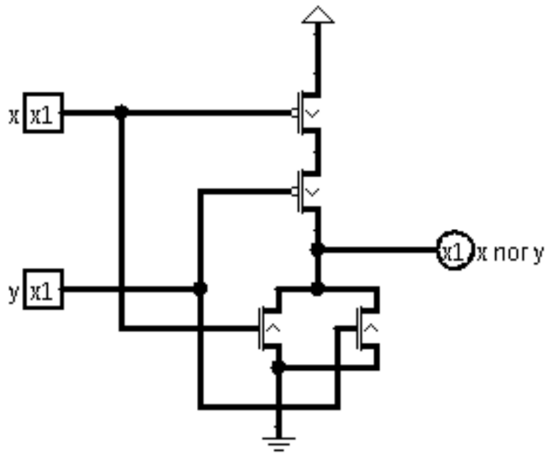


Слика 12: И коло у CMOS-у

26 Имплементација НИЛИ и ИЛИ кола у CMOS-у.

1 НИЛИ (NOR) ↓

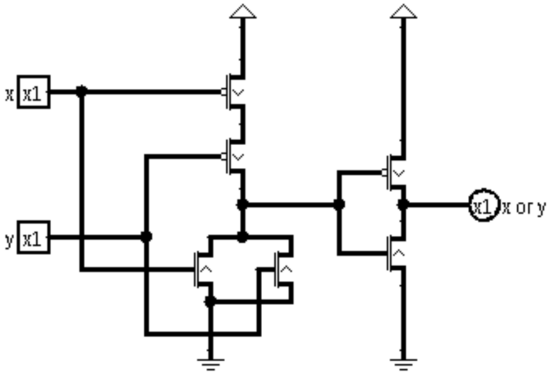
Аналогно НИ колу, само обрнуто:
Горња мрежа: два редно везана PMOS транзистора.
Доња мрежа: два паралелно везана NMOS транзистора.



Слика 13: НИЛИ коло у CMOS-у

2 ИЛИ (OR) +

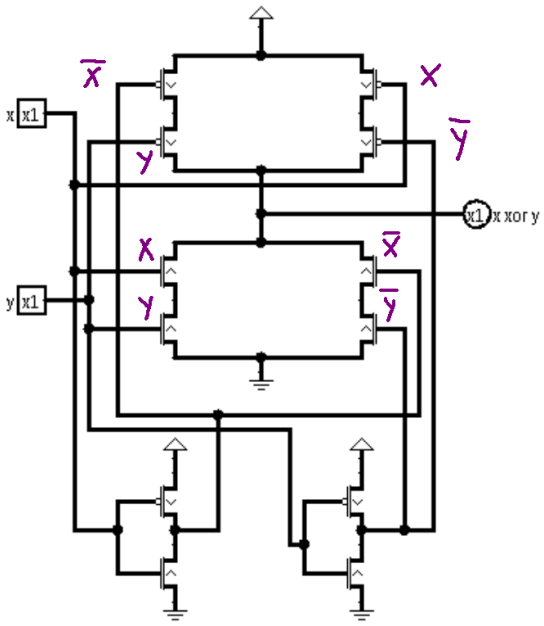
Надовежемо НЕ коло на НИЛИ коло.



Слика 14: ИЛИ коло у CMOS-у

27 Имплементација ЕИЛИ кола у CMOS-у.

Допуни објашњење.



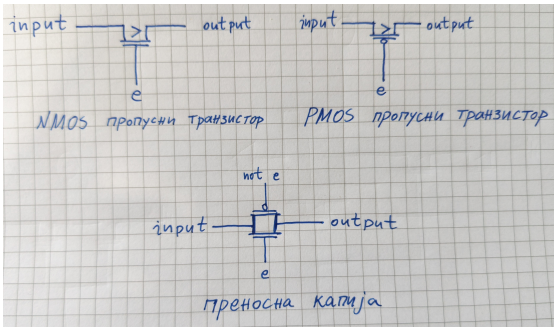
Слика 15: ЕИЛИ коло у CMOS-у

28 Пропусни транзистори и преносне капије. Функција и улога.

Пропусни транзистор контролише пропуштање неког сигнала од једне тачке кола ка другој у зависности од вредности неког другог сигнала.

Преносне капије су CMOS варијанта пропусних транзистора.

Користе се приликом реализације бафера са три стања.



29 Шта је бафер са три стања и чему служи?

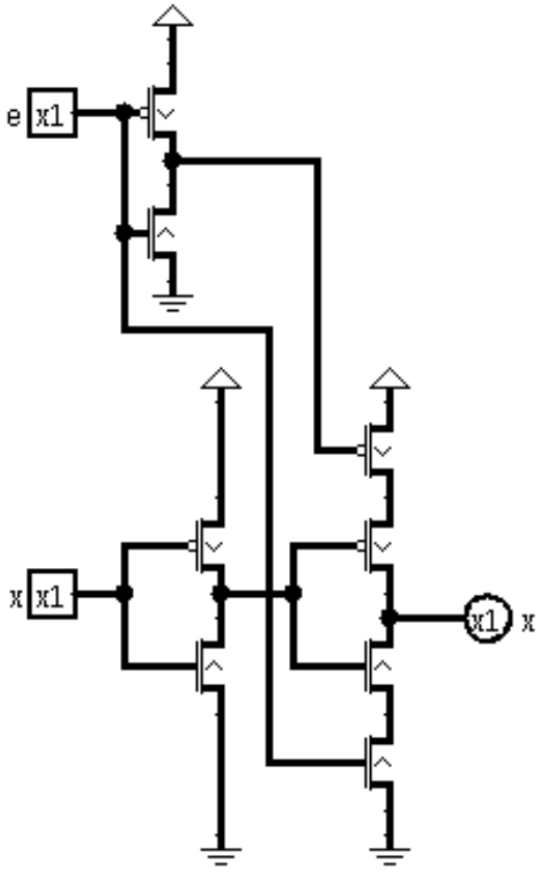
Бафер је коло које реализује идентичку функцију ($f(x) = x$). Са логичке тачке гледишта нема функцију, али се користи за појачавање сигнала.

Бафер са три стања додатно има и контролни улаз e:

- Ако је $e = 1$, понаша се исто као и обичан бафер
- Ако је $e = 0$, на излазу је вредност високе импедансе (Z).

Омогућава произвољно укључивање/искључивање делова кола.

30 Имплементација бафера са три стања у CMOS-у.



Слика 16: CMOS бафер са три стања

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 (тј. уместо да даје вредност 0 или 1, даје Z који третирамо као да кола нема). Ова вредност не утиче на друге вредности у колу. Иако није део алгебре логике, постоји како би се спречиле недефинисане вредности (два кола истовремено враћају различите резултате (кад се саберу добијемо необичне резултате)).

3 Комбинаторна кола

32 Шта је комбинаторно коло?

Логичка кола код којих се вредности израза (у сваком тренутку) могу изразити као логичке функције од вредности улаза (у том истом тренутку).

Вредности које су биле на улазима раније не утичу на тренутну вредности излаза (немају могућност памћења стања).

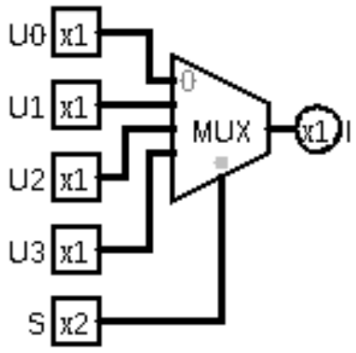
33 Навести најважније врсте основних комбинаторних кола.

- Мултиплексер
- Демултиплексер
- Декодер
- Кодер

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

Комбинаторно коло које омогућава избор једне од више понуђених вредности.

Има 2^k улаза и један излаз, као и додатних k селекционих улаза помоћу којих се врши избор једног од улаза.

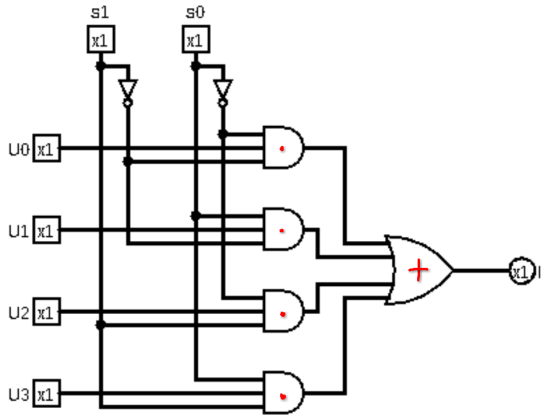


Слика 17: Шематска ознака 4-на-1 мултиплексера

S_1	S_0	Излаз Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Табела 8: Таблица истинитости мултиплексора 4:1

35 Нацртати логичко коло имплементације мултиплексора 4-1.



Слика 18: Имплементација 4-на-1 мултиплексера

36 Како се мултиплексор употребљава за имплементацију логичких функција?

Мултиплексери се могу користити за декомпозицију логичких функција на више функција мањег реда (једноставнијих). То радимо тако што дефинишемо одвојене функције које рачунају резултат првобитне када су неке логичке вредности у њој фиксиране. Потом, прослеђујемо те улазне вредности као селекционе улазе, те мултиплексер зна које право решење треба да врати.

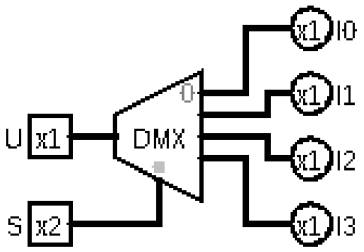
Коментар: У случају функција великог реда, декомпозиција се може вршити на више нивоа (са више мултиплексера).

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

Демултиплексер врши обрнуту функцију од мултиплексера. Има један улаз и 2^k излаза, при чему се улаз преусмерава на тачно један од излаза, у зависности од вредности k-битног селекционог улаза. (селекциони улаз тумачимо као бинарни број чија вредност је индекс излаза)

Користи се када је потребно одредити одредиште вредности која се преноси неком магистралом.

Додај истинитосну таблицу.



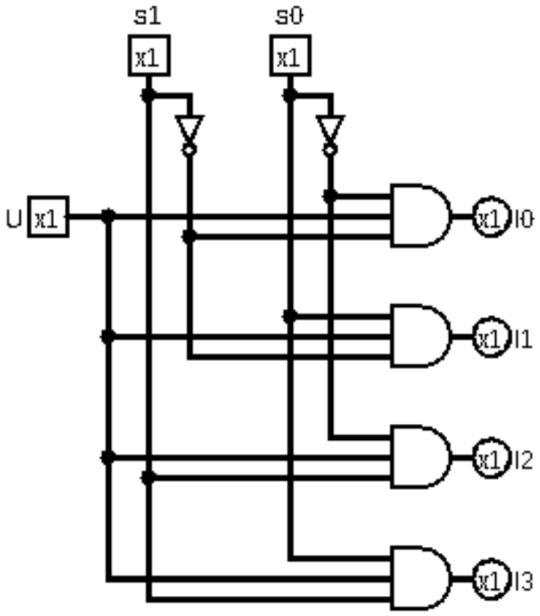
Слика 19: Шематска ознака 1-на-4 демултиплексера

S_1	S_0	Излаз I_0	I_1	I_2	I_3
0	0	U	0	0	0
0	1	0	U	0	0
1	0	0	0	U	0
1	1	0	0	0	U

Табела 9: Таблица истинитости мултиплексора 4:1

38 Нацртати логичко коло имплементације демултиплексора 1-4.

Имплементација која на неактивним излазима даје вредност високе импедансе би користила бафере са три стања.

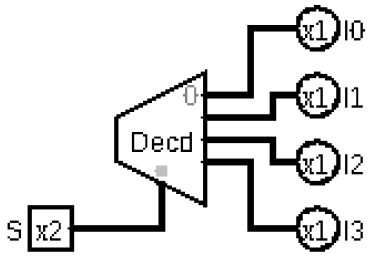


Слика 20: Имплементација 1-на-4 демултиплексера

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

Комбинаторно коло које декодира бинарно записани број и на основу његове вредности активира одговарајући сигнал на излазу. Има k-битни селекциони улаз и 2^k излаза. Улаз представља бинарни број који је индекс излаза.

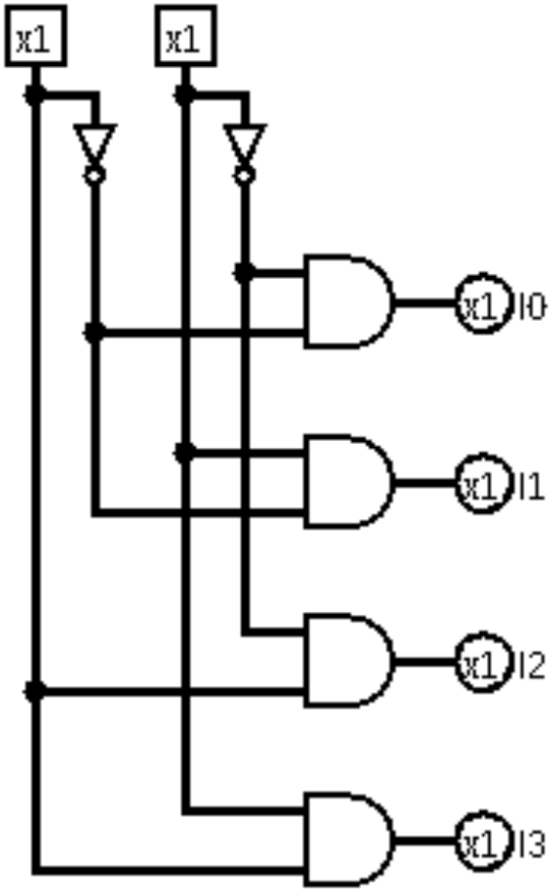
Селектовани улаз има вредност 1, а остали вредност 0.



Слика 21: Шематска ознака 2-на-4 декодера

Додај истинитосну таблицу.

40 Нацртати логичко коло имплементације декодера 2-4.



Слика 22: Имплементација 2-на-4 декодера

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

Кодер (енкодер) је коло које врши обрнуту функцију од декодера. Има 2^k улаза и k излаза (зовемо га 2^k-на-k кодер).

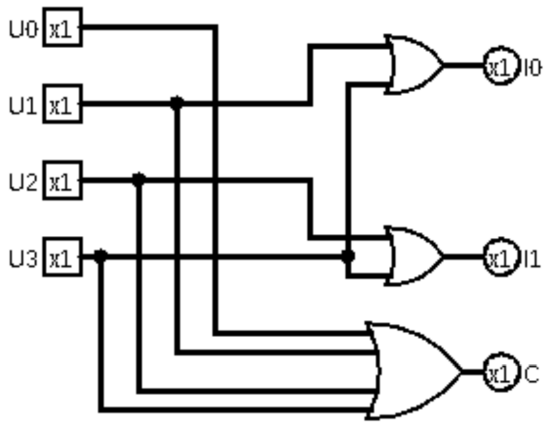
Претпостављамо да је у сваком тренутку један од улаза једнак 1, док су остали 0. Излази декодера могу да се тумаче као бинарни број који представља индекс улаза који је једнак 1.

Уколико знамо да у групи регистара један садржи неку вредност, можемо да га откријемо тако што извршимо конјукцију те вредности

и вредности у регистру над сваким регистром, и проследимо излаз кодеру, који ће нам вратити тачан индекс траженог регистра.

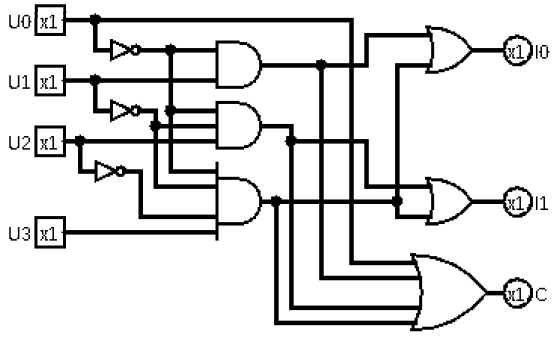
Недостатак овог кодера је што његово понашање није добро дефинисано када више улаза истовремено има вредност 1. Стога, улазима додељујемо приоритет (ако је више њих 1, бирамо онај већег приоритета) и то називамо кодер са приоритетом.

42 Нацртати логичко коло имплементације кодера 4-2.



Слика 23: Имплементација 4-на-2 кодера

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

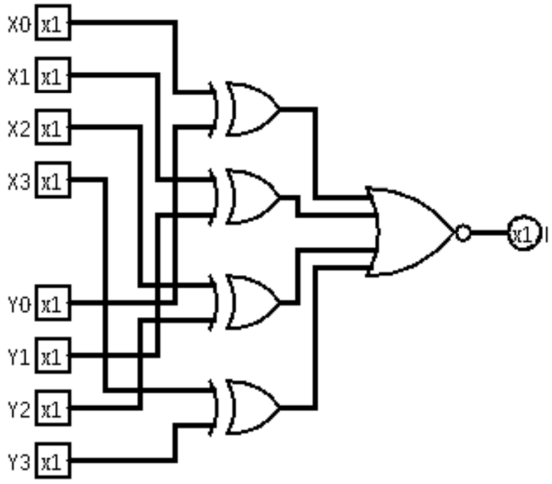


Слика 24: Имплементација 4-на-2 кодера са приоритетом

44 Шта је компаратор? Навести основне врсте компаратора.

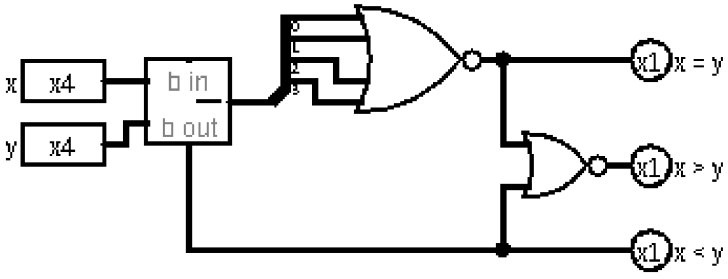
Компаратори су кола која упоређују два податка. Основне врсте су компаратори на једнакост и потпуни компаратори (поредак).

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



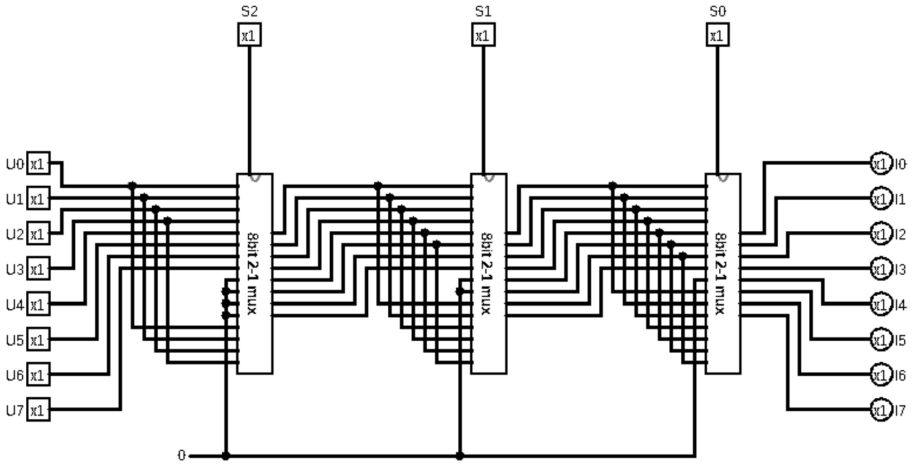
Слика 25: Имплементација 4-битног компаратора на једнакост

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



Слика 26: Имплементација 4-битног потпуног компаратора

47 Нацртати логичко коло 8-битног померача.

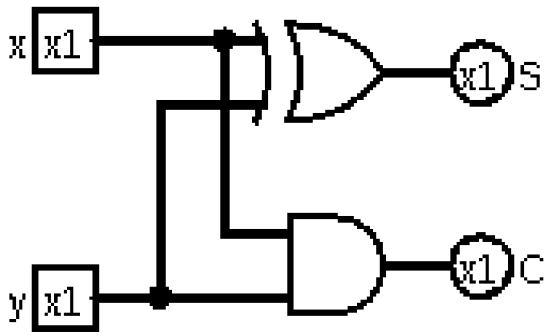


Слика 27: Имплементација 8-битног померача у лево

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

Слика 28: Истинитосна таблица бинарног полусабирача

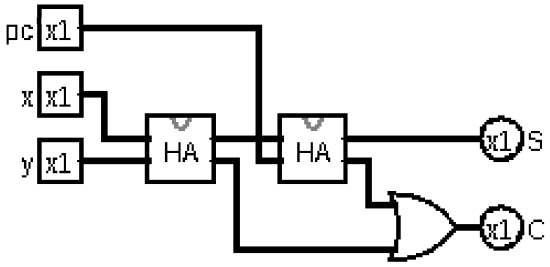


Слика 29: Имплементација бинарног полусабирача

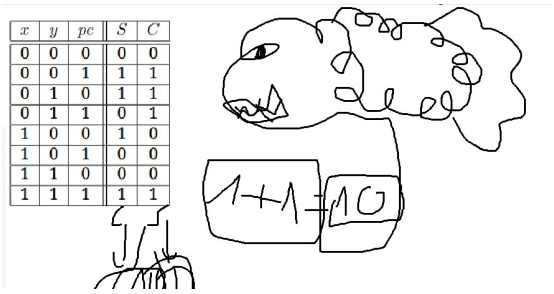
49 Нацртати истинитосну таблицу и логичко коло потпуног сабирача.

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

Табела 10: Истинитосна таблица потпуног сабирача



Слика 30: Имплементација бинарног потпуног сабирача



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

Слика 31: Босански потпуни сабирач

50 Вишебитни таласасти сабирач. Кашњење.

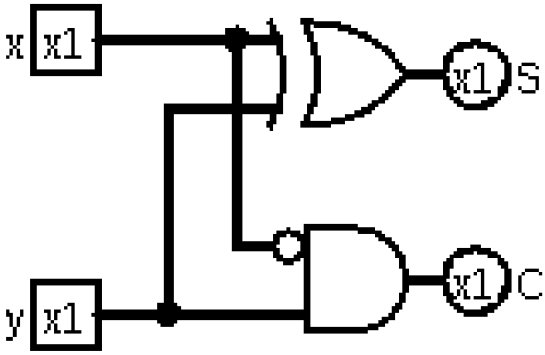
Вишебитни таласасти сабирач (енг. ripple carry adder) се користи за сабирање вишебитних вредности. Добија се уланчавањем одговарајућег броја потпуних сабирача, тако да i-ти потпуни сабирач рачуна збир i-тих битова, а његов пренос прослеђује $i+1$-вом потпуном сабирачу. Пренос последњег је пренос целог вишебитног сабирача.

Један сабирач може да настави сабирање тек када му претходни пошаље свој пренос, те кашњење расте са бројем битова ($n \cdot 2\Delta$).

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

Слика 32: Истинитосна таблица полуодузимача

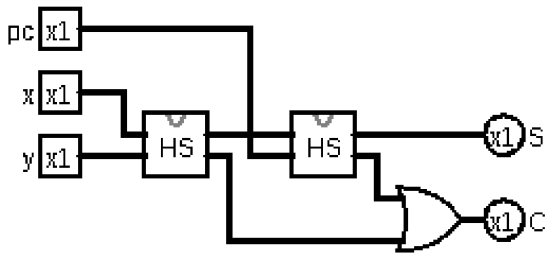


Слика 33: Имплементација бинарног полуодузимача

52 Нацртати истинитосну таблицу и логичко коло потпуног одузимача.

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

Табела 11: Истинитосна таблица потпуног одузимача



Слика 34: Имплементација бинарног потпуног одузимача

53 Вишебитни таласасти одузимач. Кашњење.

Исто као и 50, добија се уланчавањем више потпуних одузимача, тако да i-ти потпуни одузимач рачуна разлику i-тих битова, и њихов пренос прослеђује $i+1$-вом потпуном одузимачу. Пренос последњег потпуног одузимача је пренос целокупног вишебитног одузимача.

И код вишебитног таласастог одузимача сложеност расте са бројем битова ($n \cdot 2\Delta$).

Коментар: И ово можемо заменити потпуним одузимачем са рачунањем преноса унапред .

54 Објаснити укратко принцип рада сабирача са рачунањем преноса унапред.

Уместо да се на пренос чека претходни сабирач, само на основу улазних вредности израчунава се пренос на сваком биту. То се ради углавном помоћу посебног логичког кола званог 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.

- $G_i = x_0 \cdot y_0$ - говори нам да се на i-том биту генерише пренос
- $P_i = x_0 \oplus y_0$ - да ли i пропагира претходни пренос C_{i-1} ка следећем биту

56 Навести формуле по којима CLA јединица рачуна преносе Ci као и групне P и G вредности.

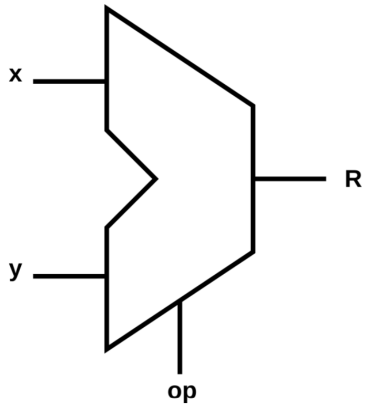
- $C_i = G_i + P_i C_{i-1}$ - пренос

- $G_i = G_{i-1} + G_{i-2}P_{i-1} + G_{i-3}P_{i-2}P_{i-1} + \ldots + G_0P_1 \cdot \ldots \cdot P_{i-1}$
- $P_i = P_0P_1 \ldots P_{i-1}$

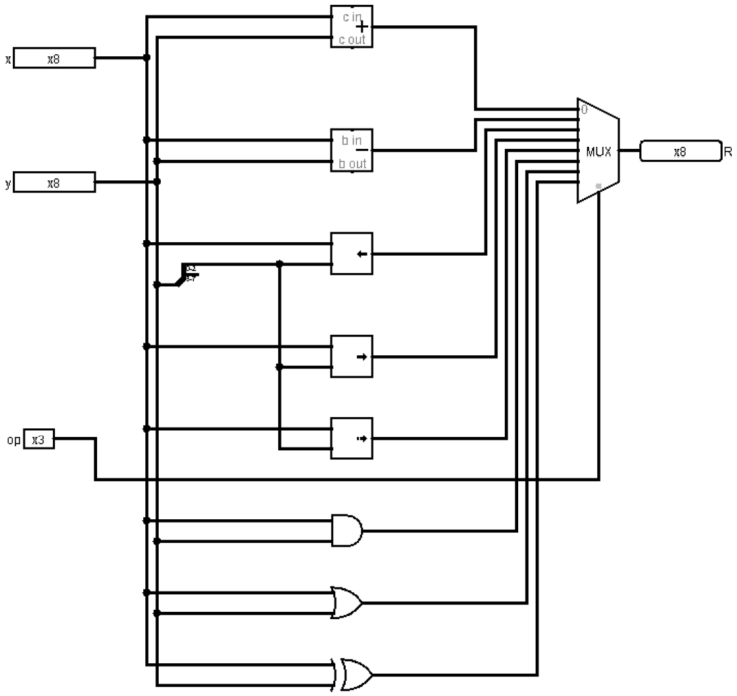
57 Навести пример имплементације ALU јединице.

Аритметичко логичка јединица у себи може да садржи различита комбинаторна кола, а помоћу мултиплексера се на улазу може одређивати која нам је операција потребна.

Коментар: ALU може имати и додатне улазе који могу да утичу на резултат (пример употребе: софтверско уланчавање сабирања)



Слика 35: Симбол за аритметичко логичку јединицу



Слика 36: Имплементација аритметичко логичке јединице

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

Ово користимо уколико желимо да имплементирамо произвољну логичку функцију на чипу без да идемо у фабрику.

Може се замислити као савршени ДНФ који садржи сваку могућу комбинацију улаза, али свака конјукција даје 0 (свака веза сорса и земље има мали осигурач). Тај мали осигурач се може прегорети пуштањем јаче струје, чиме се изазива да конјукција даје резултат 1.

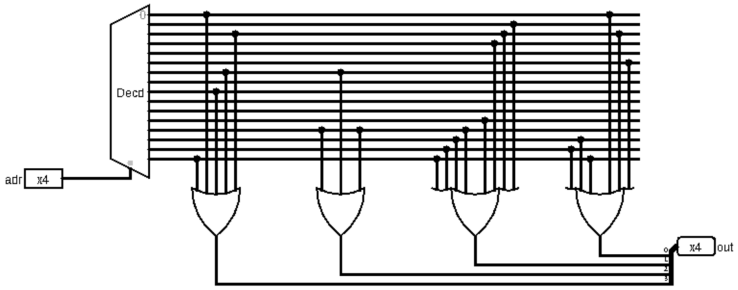
Пример: PROM (програмабилна ROM).

Не знам колико је ово лепо објашњено тако да ево још једног (више на страни 92 у скрипти).

Желимо да имплементирамо произвољну логичку 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.

На декодер се везује улаз (адреса нашег бита), а у зависности од броја 1 додајемо толико дисјункција, које се везују за оне излазне линије које представљају адресу. (ако је вредност 0, дисјункције ће дати 0).



Слика 37: Имплементација ROM меморије преко комбинаторних мрежа

4 Секвенцијална кола

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

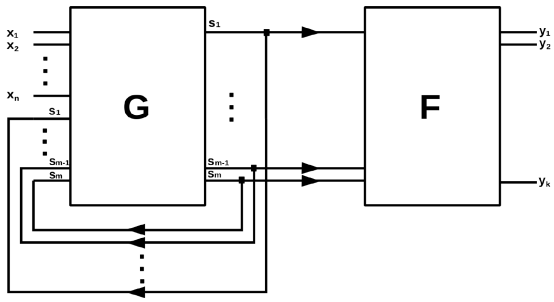
За разлику од комбинаторног кола, код којих вредност излаза зависи искључиво од вредности улаза, излаз секвенцијалних кола зависи од тренутног улаза и претходног стања (поседује памћење стања).

Под стањем логичког кола се подразумева низ $S = (s_1, s_2, \ldots, s_m)$ чије се вредности могу одржавати унутар самог кола. Вредност стања се може мењати променом вредности на улазу, али његова нова вредност зависи и од старог стања (повратна спрега).

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

Састоји се од неког комбинаторног кола G које за улаз X и стање S даје стање S ($G(X, S) = S$ (стабилно стање)). Повратном спрегом се стање враћа на улаз кола G.

Секвенцијално коло као крајњи излаз има $F(S)$ (где је F нека логичка функција). За стање кажемо да је стабилно ако је оно фиксна тачка функције G за неко фиксирано X.



Слика 38: Концептуална шема секвенцијалног кола

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

Ако S није фиксна тачка функције G за дато X (важи да $G(X,S) \neq S$), тада ће стање почети да се мења. Резултат:

- Стабилно стање: коло може стићи у неку фиксну тачку (стабилно стање) S', тј. почеће да важи $G(X,S') = S'$.
- Нестабилност секвенцијалног кола: коло осцилује између различитих стања, неуспешно тражећи стабилно стање
- Недетерминистичност: Коло може отићи у стабилно стање S', али и у неко друго стабилно стање S'' (непредвидљиво).
- Метастабилност: Коло се стабилизује у „међустању“ (нпр. на вредности $+2.5V$, што није исправна логичка вредност)

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

Нека је X произвољна вредност на улазу и S такво да је $S = G(X,S)$ (тј. S је стабилно стање за улаз X). Ако у неком тренутку улаз добије нову вредност X', тада коло прелази у ново стабилно стање S' (тј. важи $S' = G(X',S')$), при чему се то ново стање S' може детерминистички одредити претходним стањем и новим улазом.

$$S' = T(X',S), T - \text{фиксирана векторска логичка функција} \qquad (15)$$

За кола која имају овакво понашање кажемо да су добро дефинисана и стабилна, а T је функција (таблица) преласка кола G.

Коментар: Понекад описано својство неће важити за све вредности X. Тада се за стабилан рад кола мора обезбедити да на улазе долазе

само дозвољене вредности. Функција преласка неће бити тотална, већ само парцијално дефинисана.

Коментар: Излаз Y_n у тренутку t_n зависи од секвенце свих претходних улаза $X_0, X_1, \ldots, X_n$ (не само од X_n као код комбинаторних кола). Одатле потиче назив секвенцијална кола.

64 Објаснити разлику између синхроних и асинхроних секвенцијалних кола.

Код асинхроних кола, промена стања може наступити у било ком тренутку (време је континуална величина), а код синхроних кола, до промена може доћи само током одређених јасно дефинисаних временских периода (време је дискретна величина).

Синхрона кола се лакше дизајнирају да раде поуздано, али су знатно спорија (јер им је брзина одређена ферквенцијом која мора да буде иста за сва кола у систему).

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

Часовник (clock) представља јединствени синхронизациони сигнал који се придодаје свим секвенцијалним колима у систему. То је сигнал који наизменично у једнаком ритму мења вредност са 0 на 1 (узлазни руб) и обрнуто (силазни руб).

Користи се како би се вршила имплицитна синхронизација кола. Секвенцијална кола сада имају и додатни улаз за часовник. Имплементацијом се обезбеђује да се стање таквог секвенцијалног кола може мењати само током једног руба часовника.

66 Елементи циклуса часовника. Типови часовника. Фреквенција часовника.

Прелазак часовника са 0 на 1 се зове узлазни руб (узлазна ивица), док се прелазак са 1 на 0 зове силазни руб (силазна ивица) часовника. Временски период између две узлазне (силазне) ивице се назива циклус часовника. Време трајања јединице је позитивни део циклуса, а време трајања нуле негативни део циклуса. Они могу трајати једнако (симетрични часовници), али не морају (асиметрични часовници). Број циклуса у секунди назива се ферквенција часовника. Часовник се обично генерише помоћу кварцног осцилатора. Улаз часовника обележава се са clk.

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

Коментар: Реза је коло које има могућност чувања једнобитног стања. Улази резе омогућавају постављање и одржавање вредности стања, док излази омогућавају очитавање тренутног стања кола.

SR реза састоји се из два НИЛИ ($\downarrow$) кола. Има два једнобитна улаза R и S, стање се састоји из пара битова (Q, Q'), а излази су једнаки битовима стања. Важи релација повратне спреге:

$$\begin{aligned} Q &= R \downarrow Q' \\ Q' &= S \downarrow Q \end{aligned} \tag{16}$$

Постоје два стабилна стања $((Q^{sled}, Q'^{sled}) = (Q, Q'))$ која ни за које вредности улаза не воде у нестабилност.

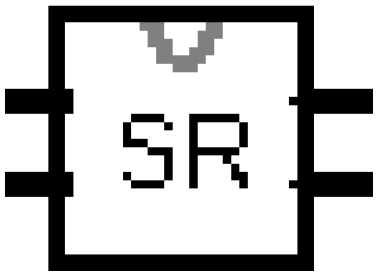
- $(S, R) = (0, 1)$ имамо само једно стабилно стање $(Q, Q') = (0, 1)$, док остала воде у ово стање (с кашњењем)
- $(S, R) = (1, 0) ditto \Rightarrow (Q, Q') = (1, 0) ditto$
- $(S, R) = (0, 0)$ имамо два стабилна стања $(Q, Q') = (0, 1)$ и $(Q, Q') = (1, 0)$ (остала воде у циклус (нестабилност) (у физичком свету ће екстерни фактори променити стање једног и довести до стабилности (недетерминистичност)))
- Ако је улаз $(S, R) = (1, 1)$, добићемо $(Q, Q') = (0, 0)$, што ће довести до циклуса када променимо вредност улаза. Због тога, овај улаз морамо учинити недопустивим.

Закључујемо да ово коло чува један бит информације, јер је Q' увек комплемент бита Q.

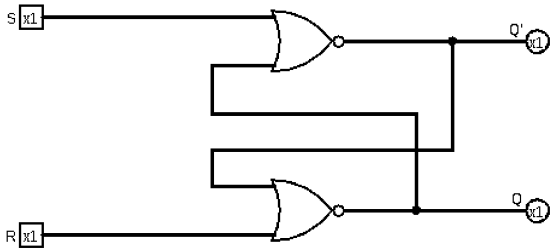
- $(S, R) = (0, 0)$ одржава запамћену вредност Q
- $(S, R) = (1, 0)$ поставља бит Q на 1 (постављање (сетовање) бита)
- $(S, R) = (0, 1)$ поставља бит Q на 0 (искључивање (ресетовање бита))

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

Табела 12: Таблица преласка SR резе



Слика 39: Симбол SR резе



Слика 40: Имплементација SR резе

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

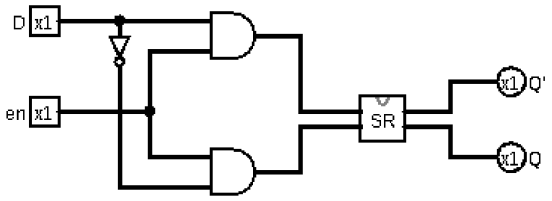
Један од начина да се реши проблем недопустивог улаза код SR резе је укидање другог улаза. Уместо њега, прослеђујемо негацију првог и преименујемо га у D.

Иако смо на тај начин избегли две јединица на улазима SR резе, изгубили смо могућност две нуле (симултано и моћ памћења). Због тога, уводимо додатан улаз e (enable).

Улаз D резе (data) је улаз на који доводимо вредност једнобитног податка који желимо да сачувамо у колу.

Када је e = 1, тада D реза ради у транспарентном режиму (свака промена вредности на улазу D се преноси на излаз Q). За e = 0, излаз Q задржава вредност која је претходно сачувана и не реагује на промене улаза D.

Не могу да нађем симбол.



Слика 41: Имплементација D резе

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

Табела 13: Таблица преласка D резе

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

Синхрони аналог резе је флип-флоп.

За разлику од резе, код које се промене стања догађају асинхроно, флип-флоп има додатни *clk* улаз за часовник. Промене се могу догодити само у тренутку наиласка одговарајућег руба часовника.

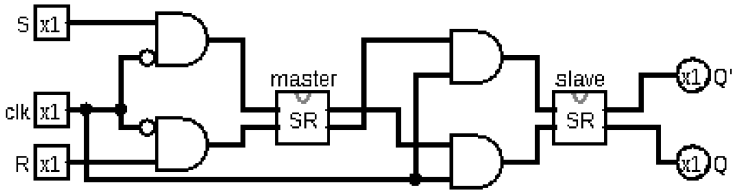
Коментар: Флип-флопови могу се имплементирати тако да реагују на било који од два руба часовника (чак и оба).

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

Када се на часовнику очита одговарајући руб, коло на основу вредности S и R одлучује да ли ће променити стање и на који начин.

Имплементација се састоји из две SR резе (лева је главна (господар) а десна подређена реза (слуга)). Q излаз master резе је повезан на S излаз slave резе, док је Q' излаз master резе је повезан на R излаз slave резе. Тиме се постиже да се вредност master резе уписује у slave резу.

Улази master резе су кроз конјукцију повезани са инвертованим сигналом часовника (мастер се отвара током негативног циклуса) (обрнуто за slave). Дакле, вредност у флип-флопу може се променити само на улазном рубу часовника.

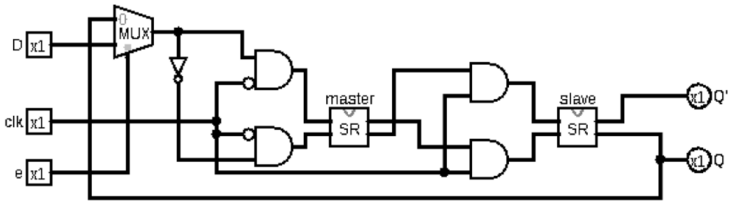


Слика 42: Имплементација SR флип-флопа

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

Имплементација је идентична као и код SR флип-флопа, уз додатак мултиплексер са улазима D и вредности флип-флопа, и контролним сигналом који означава да ли је дозвољена промена вредности флип-флопа.

Ако није, мултиплексер пропушта вредности флип-флопа, а ако јесте, пропушта вредност D .



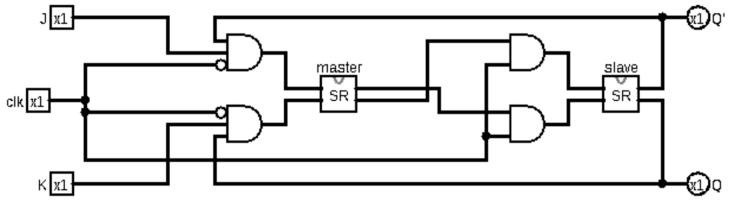
Слика 43: Имплементација D флип-флопа

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

Идеја JK флип-флопа је да се попуни семантика (додефинише) SR флип-флоп.

- $(J, K) = (0, 0), (J, K) = (0, 1), (J, K) = (1, 0)$ имају исту семантику као и код SR флип-флопа.
- $(J, K) = (1, 1)$ има улогу да инвертује вредност сачувану у флип-флопу.

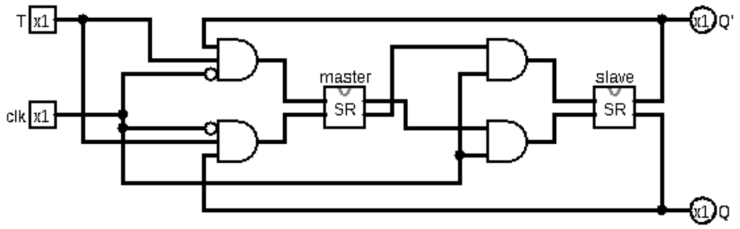
Имплементира се слично као и SR флип-флоп, једина разлика су две додатне спреге које се са излаза Q и Q' враћају назад у конјукције испред главне резе (са K и J). Тиме се спречава пролазак двеју јединица и у том случају се остварује инвертовање.



Слика 44: Имплементација JK флип-флопа

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

T флип флоп је JK флип флоп чији су улази спојени на један улаз (као и код JK флип-флопа, 1 (1, 1) означава инвертовање).



Слика 45: Имплементација T флип-флопа

Коментар: Асинхроне верзије JK и T кола (као резе) биле би нестабилне. Приметимо да се синхроност ових кола може искористити за наизменично мењање вредности (бројачи)

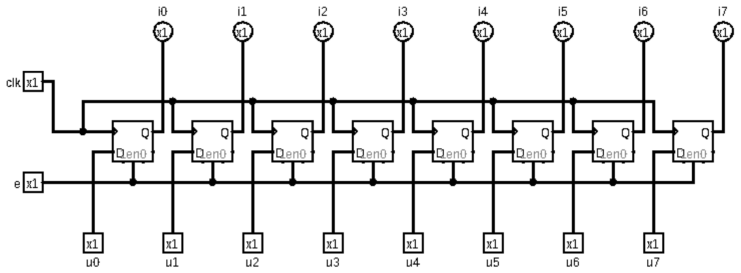
74 Објаснити проблем „хватања јединице” (1s catching problem) код master-slave RS и JK флип-флопова. На који начин се овај проблем може решити?

Претпоставимо да флип-флоп реагује на силазни руб часовника. Ако 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 (који је стабилан), 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. Основне операције са регистрима су читање и упис. Регистри се обично праве од D флип-флопова.



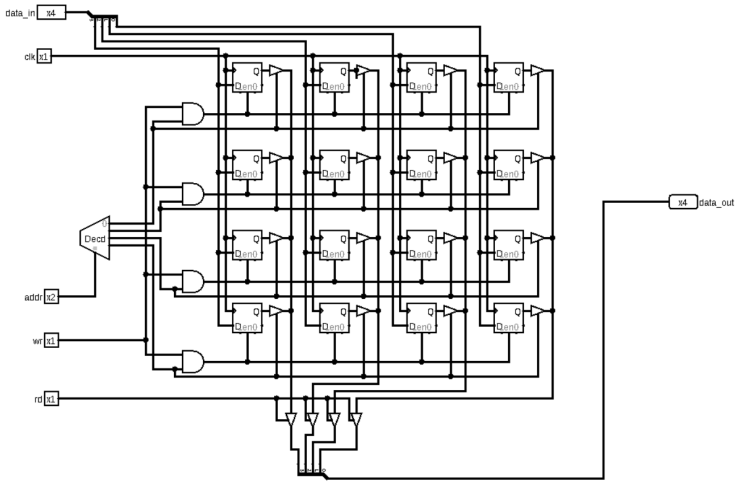
Слика 46: Имплементација 8-битног регистра

76 Статичка меморија. Пример реализације синхроне статичке меморије 4×4 помоћу флип-флопова.

Статичке меморије за чување појединачних битова користе резе (секвенцијална кола која могу имати два стабилна стања). Синхроне меморије су синхронизоване часовником.

Меморија на слици је структурно организована као матрица флип-флопова димензије $n \times m$ где свака врста представља једну меморијску локацију. Адреса се чита помоћу декодера, при чему свака излазна линија декодера активира одговарајућу врсту матрице.

- У случају операције уписа, излазна линија декодера се користи да активира e улазе свих флип-флопова у тој врсти матрице. Притом, e улази се активирају само ако је $wr = 1$, што је постигнуто додатним конјукцијама за сваку врсту матрице.
- Читање се контролише баферима са три стања који се налазе на излазима флип-флопова (свака излазна линија декодера активира те бафере у одговарајућој врсти, те се управо те вредности шаљу на излаз). Додатни бафери са три стања при дну шеме омогућавају да се вредности прослеђују на излаз само када је $rd = 1$.

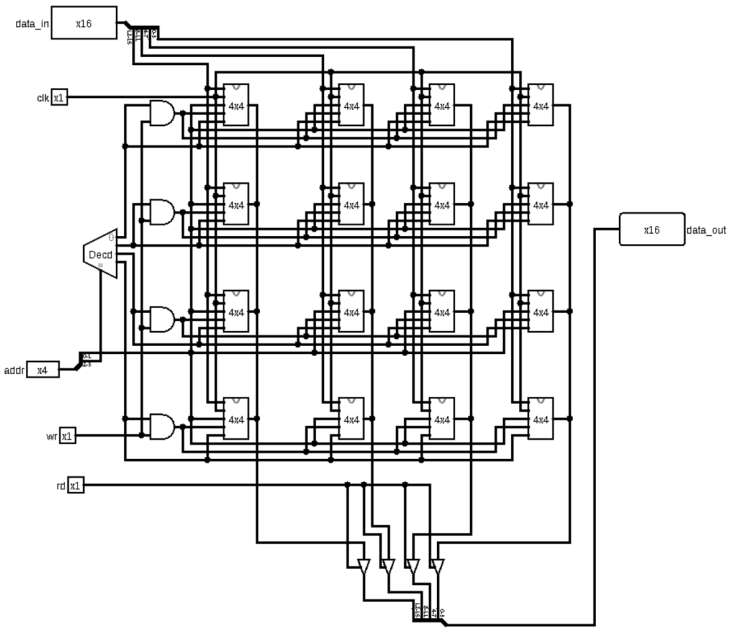


Слика 47: Имплементација синхроне статичке меморије 4×4

77 На примеру објаснити принцип конструкције већих меморија помоћу мањих.

Исти принцип као и код конструкције мањих меморија (76), само што се у матрицу уместо флип-флопова стављају мање меморије.

Можемо слагати и више оваквих меморија да добијемо још веће меморије.



Слика 48: Синхрона меморија 16×16 реализована помоћу меморија 4×4

78 Ефикасна реализација меморијске ћелије код статичких меморија.

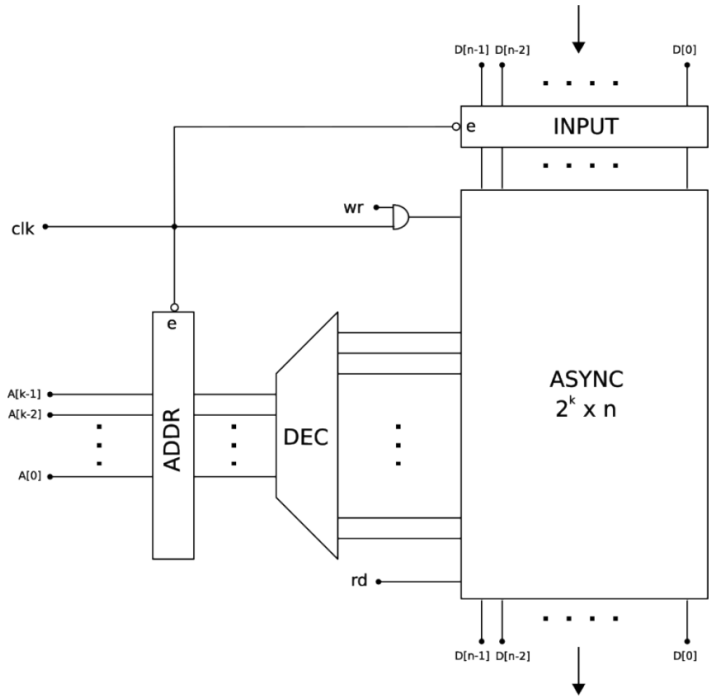
Меморија $m \times n$ је у оригиналној имплементацији садржала mn флип-флопова. С обзиром да се сваки флип-флоп састоји од две резе у господар-слуга организацији, имам $2mn$ реза.

Ово можемо оптимизовати тако што цела колона дели заједничку master резу, док се ћелије састоје само од slave реза. На овај начин смо смањили број реза са $2mn$ на $mn + n + 2$ (однос тежи ка 2 за веће матрице).

Коментар: Примећујемо да смо ову синхрону меморију добили тако што смо на асинхрону додали синхронизациону логику.

79 Објаснити како се од асинхроних меморија могу конструисати синхроне меморије и нацртати одговарајући блок дијаграм.

Најпре направимо матрицу асинхроне меморије која се састоји из асинхроних меморијских ћелија (по могућности оптимизованих), а затим је синхронизујемо додавањем синхронизационе логике.



Слика 49: Блок шема синхроне меморије од асинхроне

80 Објаснити принцип рада меморијске ћелије код динамичких меморија.

Код динамичких меморија, ћелија се састоји од једног транзистора и једног кондензатора. Вредност бита је представљена напуњеношћу кондензатора (када је кондензатор напуњен, потенцијал његове горње електроде је висок, те он чува 1, и обрнуто). Транзистор контролише пуњење и пражњење.

Ако желимо да упишемо нешто у ћелију, на гејт транзистора доводимо 1. Ако је на битској линији било 1, транзистор ће се напунити, у супротном ће се испразнити.

Када желимо да прочитамо вредност у ћелији отварамо транзистор, а на битску линију доводимо међувредност (2.5 V). Ако је кондензатор био напуњен, испразниће се и повећаће напон на битској линији. Посебно електронско коло ће напон повећати до 1, чиме ће напунити и кондензатор.

Ако је кондензатор био празан, мало ће се напунити, а напон ће пасти. Посебно коло ће спустити напон битске линије на 0, чиме ће испразнити и кондензатор.

Кондензатор се временом сам празни, па га је потребно допунити

га.

81 Предности и недостатци динамичких меморија у односу на статичке.

- Предност: израда динамичких меморија је много јефтинија, због знатно мањег броја компоненти (уместо 6 транзистора имамо 1 транзистор и један кондензатор). То омогућава израду меморије високог капацитета по ниској цени (гигабајти).
- Мана: статичке меморије су много брже (користе се за израду регистара и кеш меморије) зато што се кондензатори празне, па морамо да их често освежавамо, и током тог периода не можемо да приступамо меморији.

82 Шта је померачки регистар и где се обично користи?

Померачки регистри су регистри који имају могућност померања свог садржаја за једно место у лево или у десно (логичко или аритметичко).

Типична примена оваквих регистара је у имплементација алгоритама множења (Бутов алгоритам).

Друга примена је у конверзији између серијског и паралелног трансфера. Код серијског трансфера, податак се преноси бит по бит. Код паралелног трансфера, сви битови се преносе одједном, путем одговарајућег броја паралелних линија. Можемо учитати први бит, померити регистар у лево, учитати други и понављати поступак, и онда проследити цео податак паралелном линијом (може и обрнуто).

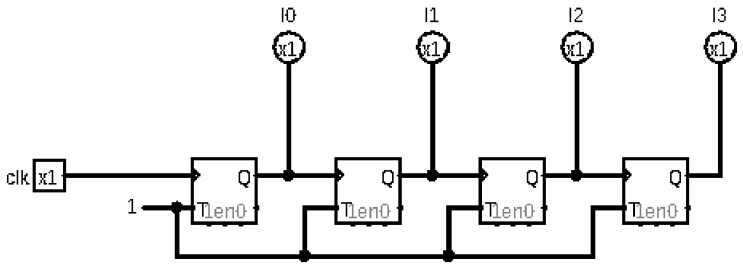
83 Асинхрони бинарни бројач. Нацртати шему и објаснити принцип рада. Који је основни недостатак асинхроних бројача?

Асинхрони бинарни бројач се имплементира помоћу n T флип-флопова, који конструисани тако да реагују на силазни руб часовника, а чији су сви T улази повезани на константну јединицу (то значи да ће при свакој транзицији мењати стање).

Флип-флоп најмање тежине (крајњи десни на слици) је директно повезан на сигнал часовника, а осталима се на улаз за часовник доводи вредност излаза претходног. То значи да ће сваки следећи флип-флоп своје стање мењати при промени вредности претходног флип-флопа (дупло спорије) (то одговара бинарном бројању).

Назив асинхрони бројач потиче од тога што су сви T флип-флопови повезани на различите синхронизационе сигнале (часовнике).

Мана је велико кашњење (до промене вредности долази таласасто, један по један флип-флоп).

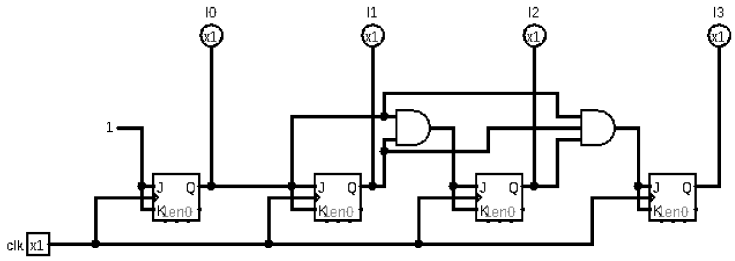


Слика 50: Шема асинхроног бинарног бројача

84 Синхрони бинарни бројач. Нацртати шему и објаснити принцип рада.

Синхрони бинарни бројач има n JK флип-флопова који реагују на **узлазни руб часовника** (сви су повезани на стварни излаз часовника). На улазе првог флип-флопа се стављају јединице, а на улазе свих осталих конјукције излаза претходних. Овако се опет имитира бинарно бројање.

Иако имају сложенију имплементацију, имају знатно мање кашњење (оно зависи само од кашњења конјукција).



Слика 51: Шема синхроног бинарног бројача

85 Дизајн бројача са произвољним редоследом стања. Пример.

Коментар: Бројач са произвољним редоследом стања броји користећи бројеве ван поретка (поредак је другачије дефинисан)

Да бисмо генерисали бројач који броји у неком произвољном, унапред задатом редоследу, можемо приступити на исти начин као и код дизајна обичног бројача, само што ћемо прослеђивати другачије J и K од обичног, који би изазвали излаз броја који је на реду.

То можемо постићи генерисањем таблице ексцитације (инверз таблице преласка за неко секвенцијално коло) , тј таблицу која ће

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

Табела 14: Таблица ексцитације за ЈК флип-флоп

нам рећи које вредности треба да буду на улазу како бисмо добили одговарајући излаз. На пример, инверзијом таблице преласка ЈК флип-флопа (44) добијамо следећу таблицу ексцитације:

Претпоставимо да желимо да направимо бројач који броји у редоследу:

$$0 \to 4 \to 7 \to 3 \to 2 \to 1 \to 0 \to \dots$$

(највећи број 7, довољно је 3 бита, конвертујемо у бинарне бројеве)

$$000 \to 100 \to 111 \to 011 \to 010 \to 001 \to 000 \to \dots \tag{17}$$

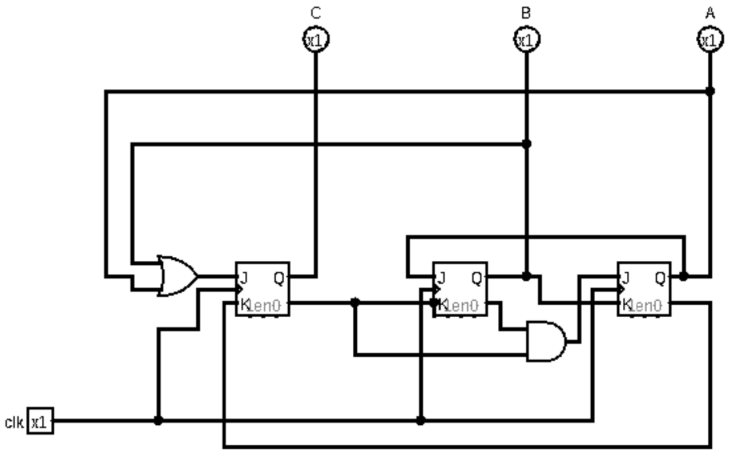
Означимо битове бројача (A, B, C) и запишемо све преласке у облику таблице ексцитације 15. Бројач не мора да прође кроз сва тробитна стања те их означавамо као небитне вредности.

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

Табела 15: Таблица ексцитације за бројач са произвољним редоследом стања

Сваку од колона J_A, K_A, J_B, K_B, J_C, K_C посматрамо као функције од тренутног стања (од колона A, B, C). За те функције одређујемо минимални ДНФ и добијамо:

$$J_A = \overline{B}\,\overline{C} \qquad J_B = A \qquad J_C = B + A$$
$$K_A = B \qquad K_B = \overline{C} \qquad K_C = \overline{A}$$



Слика 52: Шема бројача са произвољним редоследом стања дефинисаног 17

86 Коначни аутомати као модел синхроних секвенцијалних кола. Дизајн коначних аутомата. Пример.

Коначни аутомат је коло којем, за разлику од бројача са произвољним редоследом стања (где су стања унапред дефинисана), на улазу можемо наредити на које стање да пређе у следећем кораку. Притом, може имати и излаз чија се вредност генерише приликом сваког преласка из стања у стање, а која зависи од претходног стања и вредности улаза у тренутку преласка на ново стање (коначни транскдуктор).

Коначни аутомат представља најопштији модел синхроног секвенцијалног кола. Произвољни аутомат има своје улазе, стање и излазе, као и таблицу преласка, коју можемо дефинисати потпуно произвољно у складу са нашим потребама.

Посматрајмо аутомат који има 4 стања (0, 1, 2, 3) као и један једнобитни улаз X и излаз Y. Аутомат је дефинисан таблицом преласка 16.

На основу таблице преласка дефинишемо таблицу ексцитације (17). Она изгледа слично као за бројач са произвољним редоследом

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

Табела 16: Пример таблице преласка коначног аутомата

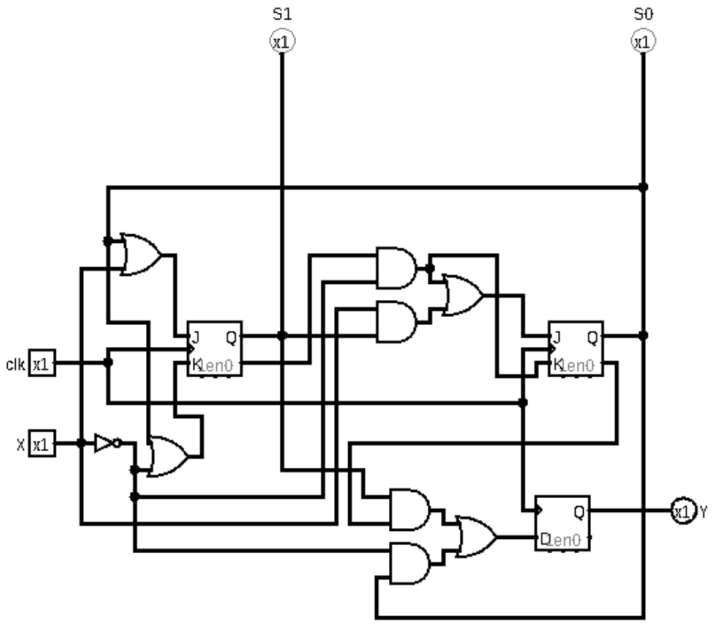
S_1	S_0	X	S_1^{sled}	S_0^{sled}	Y	J_1	K_1	J_0	K_0
0	0	0	0	1	0	0	–	1	–
0	0	1	1	0	0	1	–	0	–
0	1	0	1	0	1	1	–	–	1
0	1	1	1	1	0	1	–	–	0
1	0	0	0	0	1	–	1	0	–
1	0	1	1	1	1	–	0	1	–
1	1	0	0	1	1	–	1	–	0
1	1	1	0	1	0	–	1	–	0

Табела 17: Пример таблице ексцитације коначног аутомата

стања, али има две додатне колоне: улаз X и излаз Y (у случају вишебитних улаза и излаза, за сваки бит би имали по једну додатну колону).

Упарујемо сва могућа стања са свим могућим вредностима на улазу, те ће број врста матрице бити једнак $2^{(j+j)}$.

Колоне J_1, K_1 и J_0 се попуњавају на основу таблица ексцитације ЈК флип-флопа (на основу старог стања S_1, S_0 и новог стања S_1^{sled}, S_2^{sled}). Имплементација аутомата дата је на слици 53.



Слика 53: Имплементација једног коначног аутомата

87 Укратко објаснити основни принцип дизајна контролне јединице као коначног аутомата.

Контролна јединица је одговорна за управљање током (ток) програма. У својој основној варијанти, она је коначни аутомат чије стање означава корак програма који се тренутно извршава.

На њен улаз се доводе информације које утичу на ток програма (нпр. статус претходно извршене операције), док се на излазу налазе контролни сигнали који управљају током (ток) програма у том кораку (одређују изворište, одредиште и операцију која ће бити примењена над подацима).

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

Пример се налази на 16.

5 Принцип рада рачунара

89 Објаснити разлику између рачунара са фиксираним и ускладиштеним програмом.

- Рачунари са фиксираним програмом извршавају само један унапред задати програм (одређен таблицом преласка коначног аутомата у контролној јединици рачунара и једини начин да буде промењен је да се контролна јединица дизајнира од почетка)
- Рачунари са ускладиштеним програмом репрезентују програм као ниску бинарних бројева (машински код) који представљају елементарне рачунске операције које подржава ALU (машинске инструкције). Задатак таквог рачунара је да узима инструкције и учитава их у меморију, утврђује њихово значење (декодира) и изврши њихов ефекат (операцију).

  Скуп свих машинских инструкција које рачунар разуме назива се машински језик.

Уколико имамо само једну меморију у којој чувамо програм и податке, тада такве рачунаре називамо рачунарима Фон-Нојмановог типа.

Ако постоје одвојени меморијски простори за програм и за податке, то су рачунари Харвардског типа.

(подразумевамо да су сви рачунари у наставку Фон-Нојмановог типа)

90 Објаснити улогу контролне јединице код рачунара са фиксираним и код рачунара са ускладиштеним програмом.

- Код рачунара са фиксираним програмом, аутомат контролне јединице реализује специфичан алгоритам којим решавамо неки проблем
- Код рачунара са ускладиштеним програмом, аутомат контролне јединице имплементира алгоритам интерпретације машинског језика (декодира и извршава ефекат машинских инструкција)

Коментар: Рачунар са ускладиштеним програмом можемо посматрати као рачунар са фиксираним програмом ком је програм заправо „интерпретација“ једног рачунара са фиксираним програмом.

6 Централни процесор

91 Шта је архитектура, а шта организација рачунара?

Архитектуру рачунара представљају сви они делови дизајна рачунара који занимају програмера који пише програм на том рачунару. Укључује архитектуру скупа инструкција, али и из којих функционалних јединица се састоји рачунар, како се оне понашају и како су повезане.

Организација рачунара представља начин имплементације свих функционалних јединица са свим особинама задатим кроз архитектуру рачунара.

Једна иста архитектура може бити имплементирана на много начина.

92 Шта обухвата ISA (архитектура скупа инструкција)?

Скуп свих својстава централног процесора која су од значаја за ефективно коришћење процесора и његову комуникацију са окружењем (величина речи процесора, скуп регистара опште намене, скуп инструкција, формати инструкција, врста операнада и начини њиховог адресирања, режими рада процесора, нивои привилегија извршавања инструкција, механизми управљања меморијом, величина адресног простора (виртуелног и стварног) и слично).

У суштини све што је потребно програмеру да зна како би могао да пише програме на машинском (и асемблерском) језику тог процесора.

93 Како делимо архитектуре према броју операнада инструкција?

- Једноадресни рачунар - нема регистре опште намене, већ само специјалан регистар „акумулатор“ који садржи податак који се тренутно обрађује
- Двоадресни рачунар - наводе се оба операнда операције (сабирање), а један од њих је и одредишни (пример: Intel x86)
- Троадресни рачунар - наводимо одредишни регистар, и два операнда (пример: ARM)
- Нулоадресне архитектуре - стек машине (операнди се стављају на стек, инструкција врши скидање са стека и потом резултат ставља на стек)

94 На примеру објаснити разлике у перформансама између троадресних, двоадресних и једноадресних архитектура.

Употреба инструкција са већим бројем операнада има тендендију да смањи број меморијских трансфера, али чини инструкције дужим и тежим за превођење.

Пример: рачунање израза $Y = (A + B) \cdot (C - D)$.

- Троадресна архитектура:
  - ADD R1, A, B
  - SUB R2, C, D
  - MUL Y, R1, R2

  Иако је програм најкраћи по броју линија, инструкције су најдуже па се најдуже декодирају.

- Двоадресна архитектура:
  - MOV R1, A
  - ADD R1, B
  - MOV R2, C
  - SUB R2, D
  - MUL R1, R2
  - MOV Y, R1

  Инструкције су краће, али их има више (компромис)

- Једноадресна архитектура:
  - LOAD A
  - ADD B
  - STORE T
  - LOAD C
  - SUB D
  - MUL T
  - STORE Y

  Инструкције су најкраће, али их има највише.

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

Инструкције не могу имати ниједан меморијски операнд, већ искључиво раде са регистрима. Како бисмо ствари учитавали из меморије, користимо LOAD и STORE инструкције које учитавају меморију у регистар или чувају податке из регистра.

96 Карактеристике CISC архитектура.

У CISC (complex instruction set computer) архитектуре типично су уграђене функције вишег нивоа, за које је понекад потребно више циклуса да се изврше помоћу операција имплементираних у ALU јединици. То је учињено како би се смањио број инструкција и консеквентно број приступа меморији.

Осим тога, за ове архитектуре типично је да њихове инструкције могу користити најразличитије врсте операнада (регистри разних величина, адресе у меморији), што усложњава декодирање, али смањује број инструкција.

97 Карактеристике RISC архитектура.

Основна карактеристика (RISC) архитектура је да аритметичко-логичке инструкције могу да се користе само са регистарским и непосредним (константним) операндима. Подаци се најпре морају довући из меморије у регистре, над њима се обавити операција, а потом вратити у меморију (load/store).

Друга основна карактеристика је релативно велики број регистара опште намене (неопходно за ефективнији load/store).

Такође, сложеније операције нису подржане на процесору (синус, логаритам, петља) и морају се имплементирати софтверски. Све инструкције су исте дужине.

С обзиром да захтевају да програмер учита податке и тек онда с њима нешто ради, типично су програми написани за RISC архитектуру доста дужи, те заузимају више меморије и захтевају чешће приступе меморији. Ове мане се неутралишу предвиђањем следеће инструкције (с обзиром да су све исте дужине то је знатно лакше). Инструкције се брже декодирају јер нема пуно могућности. Такође, лакше је имплементирати овакав процесор јер захтева мање комплексних инструкција.

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.

Елаборирај на основу горњих питања.

99 Структура и формат машинске инструкције.

- Операциони код (opcode) - говори нам која операција је у питању (његова величина је максималан број инструкција које архитектура подржава)
- Адресни део (поље операнада) - над којим подацима се операције врше (константе, регистри, меморијске адресе)

Формат може бити:

- према броју адреса (максималан број операнада): троадресни, двоадресни, једноадресни и нулоадресни
- према дужини: инструкције фиксне дужине (RISC) и инструкције променљиве дужине (CISC)

100 Врсте операнада машинске инструкције.

- регистарски: податак се налази у неком од регистара
- меморијски: податак се налази на некој адреси у меморији
- непосредни: податак је саставни део инструкције (константан)

101 Објаснити директно адресирање меморијских операнада.

Инструкција садржи апсолутну адресу податка у меморији који се користи као операнд инструкције. Користи се за податке чија је адреса позната у фази писања (глобалне променљиве и статичке локалне променљиве).

Мана је што су адресе велике (32 бита) и инструкција због њих расте.

102 Објаснити индиректно адресирање меморијских операнада.

Инструкцији се задаје место на ком се адреса налази, одакле је процесор узима и чита њене податке. Код модерних процесора, то место је регистар, те не губимо на брзини.

Користи се када адреса податка није позната током фазе превођења (локалне променљиве аутоматског животног века, динамичко алоцирани подаци).

103 Објаснити индексно адресирање меморијских операнада.

Задају се два регистра, чије се вредности сабирају и тиме се добија адреса у оперативној меморији. (може и регистар и константа) (понекад се друга вредност множи са неким офсетом (величина податка))

Типично се користи за приступање елементима у низу.

104 Објаснити релативно адресирање меморијских операнада.

Као део инструкције назива се померај у односу на *PC* регистар. Та вредност се сабира са вредности у регистру и добија се апсолутна адреса (лако реферишемо на инструкције које се налазе у „околини“ текуће инструкције).

Корисно је за „кратке скокове“.

Коришћењем релативног адресирања добијени код је „независан од позиције“ у адресном простору процесора (програм се може учитати са било које адресе у меморији и ради исправно без икаквих модификација након учитавања).

105 Објаснити адресирање база + померај и навести пример употребе.

Састоји се од базног регистра и помераја, где се апсолутна адреса добија њиховим сабирањем. Омогућава нам да адресирамо податке који се налазе у околини адресе која је на регистру (системски стек).

106 Инструкције трансфера. Функција и пример употребе.

Копирање података са једне локације на другу.

x86: mov - може да у регистар или меморијску адресу уписану у регистар упише вредност из другог регистра, са друге меморијске адресе или константу (може да ради копирање из процесора директно у улазно-излазне уређаје)

lea - рачуна адресу неког податка и прослеђује је у регистар

ARM64: mov копира вредности у регистар (константе или из регистара), док ldr и str копирају вредност из меморије/регистра у регистар/меморију

107 Аритметичко-логичке инструкције. Функција и пример употребе.

Врше аритметичке и логичке операције над подацима. Постоје једноставније попут сабирања (add), одузимања (sub), логичких и аритметичких шифрова (lsl, lsr, asr), а постоје и компликованије попут множења и дељења које не морају бити имплементиране на свакој архитектури.

Неке од ових инструкција ће се разликовати за сваки тип података над којима вршимо операције (означени, неозначени, цели, децимални бројеви).

108 Инструкције безусловног скока. Функција и пример употребе.

Спада у инструкције контроле тока. Њен операнд је адреса инструкције за коју желимо да се следећа изврши (неке инструкције подржавају и да се адреса налази у регистру). То ради тако што ту адресу уписује у PC регистар.

Ова инструкција увек врши скок (intel: jmp, arm: b)

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

Контролни битови које аритметичко логичка јединица уписује у FLAGS регистар након неких операција. Говори нам више о операцији (да ли је дошло до прекорачења, који је резултат итд.) На основу ових заставица можемо вршити условне скокове.

- O - дошло је до прекорачења
- S - знак резултата
- Z - резултат је нула
- C - дошло је до преноса

Carry - прекорачење максималне **неозначене** вредности, или је позајмица приликом одузимања.

Overflow - прекорачење максималне **означене** вредности.

110 Инструкције поређења и њихова улога у реализацији условних скокова. Пример.

Инструкције поређења врше аритметичке операције (типично одузимање) над регистрима и, уместо да резултат испишу, само ажурирају флегове.

Пример (x86): cmp 2, 3 // врши се одузимање $2-3=-1$, што поставља S флаг на 1, па знамо да је мањи.

111 Инструкције условног скока. Функција и пример употребе.

Условни скок скаче уколико је испуњен неки услов (обично формулисан у вредностима флегова). Углавном постоје одвојена поређења за означене и неозначене бројеве.

x86: je, jne, jl, jg, jb, ja

ARM64: b.eq, b.ne, b.hs, b.ls, b.hi, b.lo, b.ge, b.le, b.gt, b.lt

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

Инструкција jl (да ли је означени број мањи од другог) проверава да ли су S (sign) и O (overflow) флегови различити.

Инструкција jb (да ли је неозначени број мањи од другог) проверава да ли је дошло до преноса (C (carry) флег).

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

Приликом позивања процедуре адреса повратка се ставља на стек, а на крају се скида са стека и ставља у програмски бројач.

Предност је што не морамо да бринемо да ли ће нека инструкција обрисати адресу повратка, а мана је што морамо да приступамо меморији, што је споро.

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

Приликом позивања процедуре адреса повратка се смешта у неки регистар процесора.

Предност је што доста брже приступамо овом регистру него меморији, а мана је што, ако пожелимо да позовемо неку функцију, ту стару адресу повратка морамо да чувамо на неком сигурном месту (најчешће стеку) како је друга функција не би изменила (јер и она мора да чува адресу повратка за себе).

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

Аргументи које желимо да проследимо позваној процедури се смештају на врх стека (неки регистар увек показује на врх стека).

Предност је што није ограничено колико аргумената можемо да проследимо, а мана је што је приступање стеку споро.

116 Објаснити пренос аргумената процедуре коришћењем регистара процесора. Предности и мане.

Пре позива процедуре, аргументе које желимо да јој пошаљемо смештамо у неке регистре.

Предност је што је приступ регистрима брз, а мана је што имамо ограничен број регистара.

117 На који начин позвана функција може вратити вредност позивајућој функцији?

Функција своју повратну вредност може вратити у неком од регистара, или да је смести на стек.

118 Које су основне компоненте процесора? Објаснити их.

Централни процесор се састоји од путање података и контролне јединице - (управља преносом података).

Делови путање података су:

- ALU јединица
- регистри процесора
- интерне магистрале (линије које их повезују)

Наведене компоненте чине минималну организациону целину и присутне су у сваком процесору (модерни процесори имају многе друге компоненте, попут меморијске јединице, меморијског контролера, контролера магистрале и тако даље)

119 Шта је ALU јединица и чему служи?

Аритметичко-логичка јединица је комбинаторно коло које служи за обављање основних аритметичко-логичких операција над подацима (сабирање, одузимање, битовске операције).

Има два улаза за податке, излаз и излаз за додатне информације попут преноса, прекорачења, промене знака итд.

120 Шта су регистри опште намене и чему служе?

Регистри опште намене су део архитектуре процесора (видљиви програмеру) и могу се користити у инструкцијама као регистарски операнди.

Коментар: Поред њих, сваки процесор садржи и одређене регистре специјалне намене који служе за имплменетацију интерних функција процесора. Најчешће нису део архитектуре (уз ретке изузетке), па их није могуће користити у инструкцијама као операнде.

121 Чему служи инструкциони регистар (IR)?

Инструкциони регистар у себи чува декодирану инструкцију: операциони код и остале компоненте инструкције релевантне за рад контролне јединице. Вредност овог регистра се доставља на улаз контролне јединице, како би онда могла да зна која се инструкција извршава и могла да, сходно томе, одреди наредне кораке у свом алгоритму.

Улаз овог регистра је додатно повезан са магистралом података меморијске магистрале. На овај начин је омогућемо да се током дохватања инструкције она учита у IR регистар.

Са друге стране, вредност овог се може пустити и на интерне магистрале, како би се пренела у други регистар. Ово може бити потребно у случају да инструкција садржи апсолутну или релативну адресу меморијског операнда - тада је ту адресу потребно пребацити у MAR регистар или је пропустити кроз ALU ради израчунавања апсолутне адресе операнда.

122 Чему служи програмски бројач (PC)?

Програмски бројач садржи адресу инструкције коју следећу треба извршити. Вредност се увећава након сваке инструкције тако да показује на следећу инструкцију у програму.

Обично није доступан директно, већ се баратање врши посредно преко инструкције скока.

123 Чему служи статусни регистар (PSW)?

Битови статусног регистра разматрају се засебно и представљају *заставице* или *флегове*. Они се постављају од стране ALU након извршене операције и служе да квалитативно опишу резултат.

На неким архитектурама се вредност флегова поставља након сваке инструкције која користи ALU ($x86\_64$), док се на другим вредност флегова поставља само након операције поређења (ARM64), док остале то не раде, осим ако им није назначено.

Вредност PSW се прослеђује на улаз контролне јединице, што омогућава условно извршавање инструкција (типично за условни скок).

На неким архитектурама овај регистар може садржати и контролне флегове, чије се вредности постављају посебним инструкцијама, чиме се контролише будуће понашање процесора (режим рада, начин контроле меморије, понашање појединачних инструкција итд).

124 Чему служи регистар меморијских адреса (MAR)?

Адресни регистар меморије служи да се у њему формира меморијских операнада које треба учитати из меморије или их уписати у меморију (због тога је (осим са интерним магистралама) директно повезан на адресну магистралу меморијске магистрале).

На овај начин, израчуната адреса се може директно слати у адресни улаз меморије у циљу дохватања и чувања меморијских операнда.

125 Чему служи регистар меморијских података (MDR)?

Улога регистра меморије за податке је да се у њега смешта операнд који је управо дохваћен из меморије, као и да се у њему припрема податак који треба уписати у меморију као нову вредност меморијског операнда.

Меморијски операнди се обично приликом дохватања уписују у неки регистар опште намене. Слично, приликом уписа у меморијске операнде, обично у њих копирамо вредност неког регистра опште намене. Било би исувише компликовано направити директну везу регистара и магистрале података. Због тога, MDR регистар постоји.

С обзиром да CISC архитектуре аритметичке инструкције могу имати меморијске операнде, подаци са те адресе се учитавају у MDR и он се користи као аргумент ALU јединице.

И улаз и излаз MDR је повезан са меморијском магистралом.

126 Шта је путања података (енгл. *datapath*) и из чега се састоји?

Састоји се од регистара, ALU јединице и интерних магистрала којима се крећу подаци. Улога путање података је да омогући извршавање елементарних операција над неким улазом.

127 Нацртати уопштену шему путање података са три интерне магистрале. Пример извршавања операције.

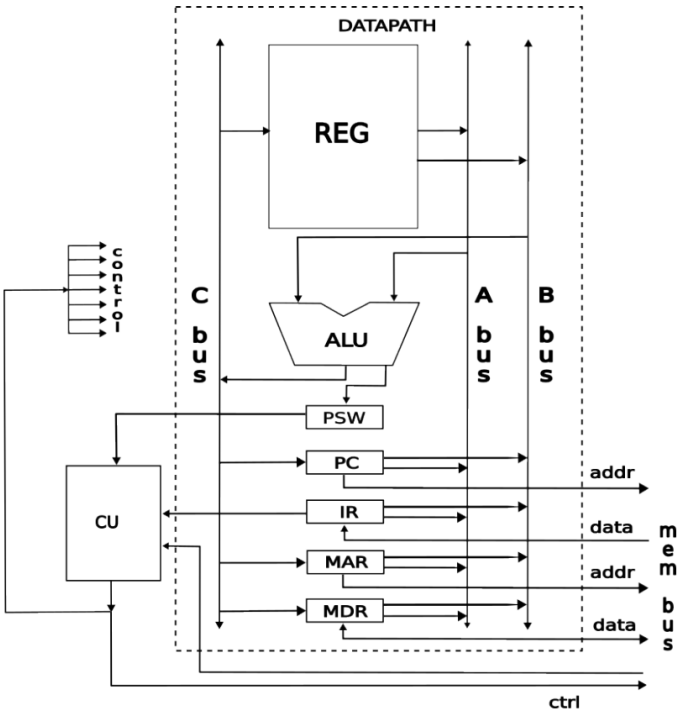
Две магистрале за читање (*A bus* и *B bus*) и једна за писање (*C bus*).

Ефикасније извршавање операција на путањи података са већим бројем интерних магистрала има цену која се огледа у сложенијој имплементацији. У овом случају, потребна су нам три декодера (два за читање и један за упис) с обзиром да је у истом циклусу могуће реферисати на три регистра опште намене истовремено.

Операција $R_2 = R_0 + R_1$ (један циклус)

- $R_0[add]R_1 \leftarrow_C R_2$

при чему се подразумева да се први операнд ALU јединице преноси путем магистрале A, а други путем магистрале B .



Слика 54: Процесор са путањом података са три интерне магистрале

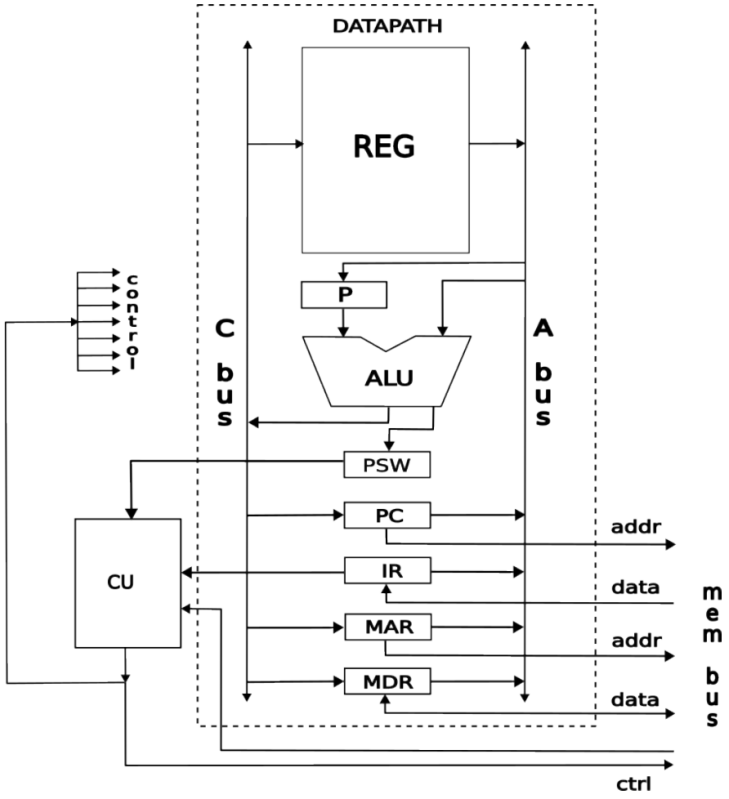
128 Нацртати уопштену шему путање података са две интерне магистрале. Пример извршавања операције.

Већи број интерних магистрала омогућава ефикаснију реализацију операција у оквиру путање података, јер је могуће вршити више трансфера у сваком циклусу.

На примеру структуре путање података са слике видимо две магистрале - за читање (*A bus*) и за писање (*C bus*). Магистралом за читање вредност из било ког регистра доводимо на улаз ALU, док магистралом за упис излаз ALU јединице можемо пренети у било који регистар.

Операција $R_2 = R_0 + R_1$ (два циклуса) ($\rightarrow_A$ означава трансфер путем магистрале A:

- $R_1 \rightarrow_A P$
- $R_0[add]P \rightarrow_C R_2$



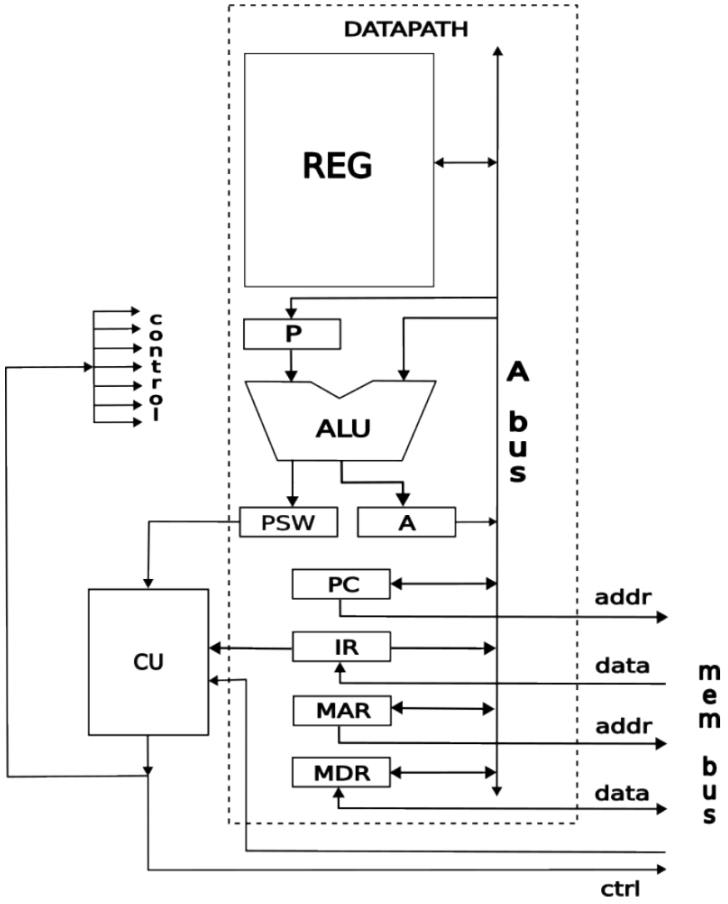
Слика 55: Процесор са путањом података са две интерне магистрале

129 Нацртати уопштену шему путање података са једном интерном магистралом. Пример извршавања операције.

Основна карактеристика овакве структуре путање података је једноставност. Недостатак је у већем броју циклуса који је потребан да би се обавила нека операција.

Операција $R_2 = R_0 + R_1$ (три циклуса):

- $R_1 \rightarrow P$
- $R_0[add]P \rightarrow A$
- $A \rightarrow R_2$



Слика 56: Процесор са путањом података са једном интерном магистралом

130 Шта је контролна јединица? Шта је улаз, а шта излаз из контролне јединице?

Контролна јединица је коначни аутомат који имплементира алгоритам интерпретације машинског језика користећи могућности путање података

процесора.

Улаз у контролну јединицу су инструкциони регистар и статусни регистри, а на излазу су контролни сигнали којима се управља компонентама у оквиру путање података.

131 Описати основне фазе при извршавању инструкција процесора.

- Дохватање инструкције (Fetch)
- Декодирање инструкције (Decode)
- Извршавање инструкције (Execute)

132 Објаснити фазу дохватања инструкције.

Учитавање инструкције из оперативне меморије у процесор. Инструкција се путем магистрале података из меморије дохвата у регистар IR, одакле постаје доступна контролној јединици која је даље тумачи и извршава.

Након тога, врши се увећање регистра PC како би он показивао на следећу инструкцију (константно за ARM или у зависности од дужине инструкције за x86).

Како је меморија знатно спорија од процесора, често није у стању да одговори на захтев у истом циклусу. Због тога контролна јединица често мора да чека већи број циклуса док не добије сигнал из меморије да је садржај допремљен. То се назива циклуси чекања (wait cycle).

Други разлог због кога дохватање инструкције може захтевати више циклуса је што су неке инструкције дугачке (CISC), садрже више битова него што је ширина магистрале података, те се не могу пренети преко ње у само једном трансферу (дохватају се део по део).

133 Објаснити фазу декодирања инструкције.

Врши се тумачење битова инструкције у IR регистру и одређивање наредног корака у њеном извршавању.

Из операционог кода инструкције контролна јединица закључује о којој се инструкцији ради, који је њен формат, колико има операнада и које су врсте (уколико није учитана у целости врши се учитавање остатка).

На основу врсте операнда, закључује се како до њих доћи (ако су меморијски шаљу се у MAR регистар, ако је индексно сабирају се база и индекс и шаљу MAR регистру).

Декодирање може бити једноставно, дугачко један циклус уколико инструкције имају једноставан формат (ARM), али и комплексно које траје пар циклуса (x86).

134 Објаснити фазу извршавања инструкције.

Извршавање операције која одговара декодираној инструкцији. Може зависити само од инструкције, али и од флегова из PSW регистра.

Уколико инструкција има меморијски операнд, он се дохвата у овој фази из меморије (вредност из меморије са адресе у MAR регистру се пребацује путем магистрале у MDR).

У случају аритметичко логичке операције (или трансфера), регистри који садрже податке (вредност из меморије смо преселили у MDR) се шаљу на улаз ALU и њихов резултат се уписује у регистар мету. (код поређења се врши алтерација флегова)

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

- Тврдо ожичена имплементација (hardwired) - коначни аутомат имплементира се у виду секвенцијалних кола
- Микропрограмирана имплементација (microprogrammed) - алгоритам интерпретације се реализује помоћу посебног микрокода који се чува у контролној меморији

Тврдо ожичена имплементација је бржа (ARM64), али свака измена захтева редизајнирање целог кола.

Микропрограмирана имплементација је знатно спорија, али олакшава уношење измена у алгоритам чак и када процесор напусти фабрику.

136 Објаснити тврдо ожичену (хардверску) имплементацију контролне јединице.

Контролна јединица се имплементира као секвенцијално коло које функционише као коначни аутомат. С обзиром да се интерпретација машинског кода дешава на стварном хардверу, брза је, али уколико је начињена грешка, тешко се мења. Због тога је практична код једноставних архитектура где нема пуно простора за прављење грешака приликом дизајна (RISC).

137 Објаснити микропрограмску (софтверску) имплементацију контролне јединице.

Алгоритам интерпретације машинског кода реализује се као микропрограм који се чува у посебној меморији унутар контролне јединице (контролна меморија (може бити непроменљива и променљива)). Свака адресибилна локација унутар ове меморије садржи једну микроинструкцију која

описује један корак у извршавању машинске инструкције (извршава се у једном циклусу).

Свакој инструкцији одговара један микропрограм састављен од микроинструкција.

138 Шта је микроинструкција? Структура микроинструкције.

Микроинструкција се налази унутар контролне меморије и описује један корак у извршавању машинске инструкције (извршава се током једног циклуса).

Састоји се из два дела:

- контролни сигнали - приликом извршавања микроинструкције се постављају на излаз контролне јединице и управљају радом компоненти путање података (током тог циклуса)
- адресни битови - учествују у одређивању адресе следеће микроинструкције која треба да буде извршена

139 Шта је микропрограм? Објаснити начин извршавања микропрограма.

Микропрограм је низ микроинструкција којима се имплементира нека инструкција.

Прво се на контролну јединицу шаље адреса микроинструкције којом инструкција почиње, она се проналази, ставља у микроинструкцијски регистар (током силазног руба часовника). На излазу се вредност регистра дели на контролне битове и адресу следеће инструкције.

Контролни битови се шаљу на излаз контролне јединице како би се извршила микроинструкција (током узлазног руба часовника), а адреса следеће се прослеђује назад контролној јединици како би се овај поступак поновио.

Може постојати посебан микропрограм који врши дохватање и декодирање микроинструкције.

140 Објаснити хоризонтални формат микроинструкција процесора.

У хоризонталном формату се сви контролни сигнали представљају посебним битовима и у неизмењеном облику се прослеђују на излаз контролне јединице. Такав приступ је лако разумљив и једноставан, али му је мана што је такав формат веома дугачак, с обзиром на огроман број контролних сигнала у модерним процесорима.

141 Објаснити вертикални формат микроинструкција процесора.

У вертикалном формату се типичне комбинације битова контролних сигнала компактније кодирају, те је микроинструкција краћа. Овакве микроинструкције се морају декодирати на излазу, како би се добили одговарајући „распаковани“ контролни сигнали.

7 Оперативна меморија

142 Из којих делова се састоји меморијска магистрала и која је њихова улога?

Улога меморијске магистрале (паралелна магистрала) је повезивање централног процесора и оперативне меморије.

Састоји се од:

- магистрале података - преноси машинске инструкције из меморије ка процесору, и податке (операнде) у оба смера
- адресна магистрала - преноси адресе од процесора до меморије
- контролна магистрала - састоји се од скупа контролних сигнала помоћу којих процесор и меморија размењују команде и информације о статусу

143 Објаснити операцију читања податка путем меморијске магистрале без циклуса чекања.

У овом сценарију, меморија је довољно брза да одговори на захтев процесора унутар основног временског оквира, па процесор нема „празан ход“ током ког не извршава ништа већ само чека спору компоненту.

На почетку првог такта, процесор укључује en сигнал (меморији ставља до знања да треба реагује на сигнале из магистрале). Сигнал rd/wr одређује да ли је у питању операција читања (0) или писања (1). Потом, процесор на адресну магистралу поставља вредност адресе у меморији са које жели да врши читање (сигнал addr) (ако вредности нема, на адресној магистрали је вредност високе импедансе).

Меморија очитава вредности на силазном рубу часовника, и у складу са захтевом прочитати вредност са дате адресе и поставити је на магистралу података (сигнал data).

На следећем узлазном рубу, процесор ће очитати вредност са магистрале и сачувати је у одговарајућем регистру (MDR или IR), и деактивираће своје сигнале. Меморија ће на следећем силазном рубу уочити да су сигнали склоњени те ће и она склонити вредност са магистрале и поставити data на вредност високе импедансе.

По овом протоколу, читава операција читања захтева два циклуса на магистрали.

Коментар: Описана меморија може да преноси податке само током једног руба часовника. Модерне меморије пружају могућност оба (double data rate или ddr).

144 Објаснити операцију читања података путем меморијске магистрале са циклусом чекања.

С обзиром да је меморија знатно спорија од процесора, неће стићи да заврши своје операције током само једног циклуса, те ће морати да некако сигнализира процесору да сачека. То се постиже помоћу сигнала *wait* (1 значи да ће процесор чекати, тј. бити у стању „празног хода“, током ког не ради ништа све док се вредности сигнала не постави на 0).

Осим тога, идентично је као и операција читања података без чекања 143.

145 Објаснити операцију писања податка путем меморијске магистрале без циклуса чекања.

Одговор је измена 143.

146 Објаснити операцију писања података путем меморијске магистрале са циклусом чекања.

Одговор је измена 144.

147 Објаснити операцију читања блока података путем меморијске магистрале.

Уколико желимо да истовремено читамо блок података, потребно је увести додатни сигнал *burst* који ће меморији назначити да треба да проследи блок података (довољно велики да попуни ширину меморијске магистрале (64 бита)).

148 Објаснити ефикасну организацију оперативне меморије засновану на двостепеном адресирању (адреса врсте + адреса колоне)? Које су предности овог приступа?

Уместо да, као што смо до сада, чувамо низ ћелија (бајтова) којима приступамо коришћењем демултиплексера који одговара броју могућих адреса, ћелије организујемо у матрице једнаке дужине и ширине.

Како бисмо знали где треба да идемо, користимо један демултиплексер који на основу горње половине адресе одређује у којој се врсти налази адреса коју тражимо, и по демултиплексер за сваку врсту који исто ради са доњим битовима адресе да нам одреди тачну позицију.

Овакав приступ има више предности:

- просторно је ефикасније расподелити меморијске ћелије у матрицу јер онда имамо краћа растојања између ћелија и тиме убрзавамо претрагу
- уколико желимо да прочитамо један блок података, они се налазе ближи један другом, те је доста брже

149 Објаснити вишебајтни пренос података. Шта је меморијска реч?

С обзиром да је дужина модерних меморијских магистрали 64 бита уместо 8, бајтови се из меморије преносе у пакетима од по 8. Како бисмо тај поступак убрзали, градимо меморијске блокове краће дужине (8 или 16 битова), и слажемо их тако да добијемо ширину од 64 бита (то се зове меморијски ранг). Рангови (дакле уместо индивидуалних бајтова) се организују у врсте и колоне (онако како је описано да се меморија гради).

Приликом читања из меморије, добављамо цео блок који садржи адресу бајта који нам треба (процесор шаље адресу без последња три бита), и у процесору одбацујемо непотребне делове. Овај поступак знатно убрзава читање јер се адресе брже декодирају, а и често нам не треба само један бајт из меморије, већ више њих (види алајновање).

Последица је да се подаци из процесора увек шаљу у пакетима од по 8 бајтова, који се зову меморијска реч (тј. део меморије који процесор може да прочита, упише или обради током једног циклуса).

150 Објаснити принципе меморијског поравнања података у оперативној меморији.

С обзиром да су типови података које типично користимо вишебајтни, уколико их само држимо у меморији без неке претходне обраде, може доћи до тога да се унутар једне речи налази само део податка који нам треба.

Због тога, користимо поравнање, правило које налаже да се први бајт податка одређене величине сме налазити само на адреси која је дељива са његовом величином (нпр. int се може налазити на адресама 0, 4, 8, …јер ће онда цео стати у један меморијски ранг).

Ова правила могу бити наметнута самим хардвером (ако не важи долази до прекида), али и софтвером (у ARM64 ће програм радити спорије ако поравнање није обезбеђено).

151 Укратко објаснити принцип рада испреплетених меморија као и разлог због којих их користимо?

Већу меморију можемо изградити од мањих (које називамо блокови), те би првих пар битова адресе представљали адресу блока, а остатак адресу унутар једног од блокова. Такав приступ омогућава нам да, уместо секвенцијално, адресама приступамо наизменично, унутар различитих блокова.

Међутим, типично су подаци сачувани секвенцијално, као на пример низ, те се у већини случајева налазе у истом блоку. То можемо решити тако што преокренемо ситуацију, последња два бита адресе заправо представљају адресу блока, те је загарантовано да ће се две вредности које су једна поред друге налазити у различитим блоковима.

152 Објаснити читање података са узастопних адреса код испреплетених меморија.

Код испреплетаних меморија, узастопне адресе распоређене су тако да се налазе у различитим блоковима (подмеморијама), те можемо истовремено да трагамо за обе уместо да чекамо да блок заврши (јер се не може истовремено читати из блока из очигледних разлога (read on write)). Једино што нам константно одузима време је иницијално одређивање у којим се блоковима адресе налазе, али је то знатно брже јер блокова има експоненцијално мање него адреса.

153 Објаснити упис података на узастопне адресе код испреплетених меморија.

С обзиром да су блокови у које уписујемо податке раздвојени (јер су подаци узастопни), једино кашњење је одређивање у ком блоку се адреса налази, и смештањем податка у његов бафер (за упис). Следећи податак ће поновити поступак, међутим он одлази у други блок, па га то што се у претходни уписује неће спречити да се симултано упише, што знатно убрзава процес уписа.

Коментар: Проблем који испреплетане меморије решавају је што меморија не сме да дозволи да се на истој адреси истовремено упишу подаци, а тражење адресе у великој меморији је споро, те делимо меморију на мање делове, који су сами по себи меморије, а тражење у коју од таквих меморија се уписује је знатно брже него над целом.

154 Шта је меморијски контролер и које су му основне функције?

С обзиром да начини приступања меморији умеју да се знатно закомпликују, уместо да целу функционалност препуштамо централном

процесору, меморијски контролер је секвенцијално коло чија је улога да то уради уместо њега. Од процесора прима адресе података, и на основу начина организације меморијских кола води рачуна да пронађе што ефикаснији начин да добави или упише податке.

Главне функционалности меморијског контролера су:

- иницијализација и препознавање меморијских модула: контролер утврђује који су слотови попуњени и њихов капацитет
- мапирање меморијских адреса: процесор адресни простор посматра као целину, те меморијски контролер такву адресу мора да мапира у
  - број канала
  - број модула и ранга
  - број банке на чипу
  - адресе врсте
  - адресе колоне

  Постоје разне методе мапирања ових компоненти на смислене адресе.
- Оптимизација приступа меморији - меморијски контролер мора да препозна обрасце попут испреплетаног приступа и FPM режима и ефективно их примени
- Освежавање меморије - с обзиром да користимо динамичке меморије, кондензатори се празне те морамо да их допунимо (то се ради на начин који претерано не омета приступање)

8 Спољашње меморије

155 Која је разлика између унутрашњих и спољашњих меморија?

- унутрашње меморије су директно повезане са централним процесором путем магистрала (док спољашње нису, не припадају рачунару у ужем, Фон-Нојмановом смислу)
- унутрашње немају постојан капацитет (или ако имају су непроменљиве), док спољашње имају
- унутрашње меморије су бајт адресибилне, док код спољашњих можемо да читамо само веће целине (512 бајтова до 4 килобајта)
- унутрашње меморије су знатно брже од спољашњих

156 Набројати бар три врсте унутрашњих и бар три врсте спољашњих меморија.

Унутрашње: регистри, кеш, радна меморија
Спољашње: магнетне траке, оптички дискови, тврди дискови, флеш меморије

157 Које све карактеристике разматрамо приликом упоређивања различитих врста меморија (унутрашњих и спољашњих)?

- адресибилност - која је најмања меморијска јединица која се може адресирати
- начин приступа - у којој мери физичка локација адресибилне јединице утиче на време приступа њој
- измењивост - да ли меморија дозвољава упис
- постојаност - да ли је меморији потребно електрично напајање како би очувала свој садржај
- величина - колико података се може сместити у меморију
- брзина - време потребно за читање или уписивање података у меморију
- цена - колико је скупо произвести меморију у односу на њену величину

158 На који начин делимо меморије на основу њихове адресибилности?

- бајт-адресибилне: сваки бајт има своју адресу и могуће му је приступити директно (RAM и ROM)
- реч-адресибилне: адресибилне јединице су речи (типично величине два, четири или осам бајтова) (регистри опште намене)
- блок-адресибилне: адресибилне јединице су блокови (величине 512B до 4KB) (већина модерних спољашњих меморија) (користимо неку другу меморију да поделимо блок на мање целине)
- неадресибилне: не приступа се путем адресе, већ на неки други начин (кеш меморије)

159 Навести могуће начине приступа меморији.

- непосредни: свим адресибилним јединицама се приступа у приближно истом времену (RAM, ROM, регистри процесора, флеш спољашње меморије)
- секвенцијални: подацима се приступа редом којим су сачувани у меморији (магнетне траке)
- директан: време приступа сектору зависи од његове позиције на диску, али се не одређује секвенцијално (магнетни дискови и дискете)

160 Објаснити секвенцијалан приступ меморији.

Адресибилним јединицама се приступа редом којим су физички смештене у меморију (ако желимо да приступимо n, морамо прво приступити адресама $\ldots, n-2, n-1$).

Овакав приступ карактеристичан је за магнетне траке.

161 Објаснити директан приступ меморији.

Меморије са оваквим приступом (магнетни дискови, дискете, тврди дискови) подељене су на мање целине, секторе, којима је приступ непосредан, и након тога се њихова околина секвенцијално обилази у потрази за адресом која нам је потребна.

162 Објаснити непосредан приступ меморији.

Свим адресибилним јединицама се приступа у приближно једнаком времену, без обзира на њену физичку локацију.

Примери оваквих меморија су RAM (random access memory), ROM, регистри процесора, као и флеш меморије.

163 Шта је капацитет меморије и у којим јединицама се изражава?

Капацитет меморије је количина података које можемо да сместимо у њу. Изражена је у бајтовима, чак и када меморија није бајт-адресибилна, ради лакшег поређења.

Како је бајт релативно мала јединица, у пракси се користе веће:

- килобајт (KB) = 1024 B
- мегабајт (MB) = 1024 KB
- гигабајт (GB) = 1024 MB

- терабајт (TB) = 1024 GB
- петабајт (PB) = 1024 TB

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

- непостојана/волатилна: када електричног напајања нестане, подаци се бришу (RAM, транзистори, кеш меморија)
- постојана/перзистентна: није јој неопходно електрично напајање да би очувала свој садржај (скоро све спољашње меморије, ROM)

165 Какве меморије могу бити с обзиром на измењивост њиховог садржаја? Примери.

- неизмењива: након првобитног дефинисања садржаја допуштају само читање вредности (ROM, CD-ROM, DVD-ROM)
- измењива: допушта читање и упис (измену вредности података) (све остале)

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

Брзина се односи на време потребно за читање и писање података. Обично се изражава у мегахерцима (MHz).

На брзину меморије највише утичу:

- кашњење: време од тренутка задавања команде до тренутка када меморија крене да испоручује податке процесору. Узроковано је самом унутрашњом организацијом меморије.
- брзина трансфера: количина података који се из меморије могу прочитати у јединици времена. Зависи од брзине меморије, као и брзине канала за пренос који се користе.
- време приступа: укупно време које је потребно од тренутка када се захтева приступ податку до тренутка када он постане доступан за читање. Зависи од кашњења, брзине трансфера и много других фактора.

167 Објаснити хијерархију меморија.

Меморије у рачунарском систему су организоване хијерархијски, почев од меморија које су најближе процесору (регистри и кеш меморије), па све до меморија које су најудаљеније од процесора (спољашње меморије).

Меморије које су ближе процесору су веома брзе, али су зато релативно скупе и малих капацитета. Како се удаљавамо од процесора, постају спорије, веће и јефтиније. Такође, постојаност је карактеристика спољашњих меморија. Са друге стране, спољашње меморије често немају могућност адресирања на нивоу бајтова или речи, што је карактеристично за унутрашње.

168 Навести основне врсте спољашњих меморија и навести њихове карактеристике.

1 Магнетне траке

Дугачка трака премазана магнетним материјалом и намотана на котур. Информација се чува интензитетом или поларитетом магнетних поља на некој тачки.

Трака се премотава мотором са једног котура на други, и упис и испис се врши намагнетисаном главом која мења поларитет тачке. Користе секвенцијални приступ и због тога су најспорија форма спољашње меморије. Користиле су се 50. година прошлог века. Подаци уписани на магнетне траке могу дуго да трају без пропадања, а и врло су јефтине (због тога се користе за архивирање велике количине података).

Читачи нису толико јефтини али шта да им радиш.

2 Магнетни дискови

Слично као магнетна трака, само је диск.

Површина магнетног диска је логички подељена на стазе које представљају скуп битова на истом растојању од центра. Стазе су повезане на секторе који су кружни исечци (пица). Избор стаза се врши померањем главе, а избор сектора се врши ротацијом дискова.

Уређаји засновани на магнетним дисковима могу имати већи број дискова, сваки са сопственом главом, који се могу читати симултано (цилиндар).

Временом су класични магнетни дискови замењени дискетама, које су доста мање, па напослетку и тврдим дисковима, који имају знатно већи капацитет.

3 Оптички дискови

За разлику од магнетних дискова, оптички дискови користе принцип одбијања светлости за представу података.

Представља пластични диск који је са доње стране обложен слојем поликарбоната преко ког се премазује рефлектујући слој (алуминијум).

Током израде, на површини се праве мала удубљења, чија се рефлексивност разликује од рефлексивности равних делова (нарезивање). Очитавање се врши помоћу посебног уређаја, читача оптичких дискова, који испаљује ласере и врти диск како би открио податке на њему.

Постоје CD дискови, DVD дискови и Blue-ray дискови (сортирани по капацитету и брзини)

4 Постојане полупроводничке меморије

Немају покретне делове, те нису подложне кваровима.

Заснивају се на MOS транзистору са плутајућим гејтом (структура је слична NMOS транзистору, осим што изнад места где долази до пробоја постоји плутајући гејт, који у себи чува наелектрисање након уписа).

Овакве меморије називају се EEPROM (флеш) меморије. Постоје NAND и NOR флеш.

Агресивно детаљисање у скрипти на 313. страни, суштина је горе.

9 Кеш меморија

169 Објаснити намену и основни принцип рада кеша.

Кеш меморија представља малу количину меморије (SRAM), по хијерархији између регистара и RAM меморије.

Главна улога је да смањи разлику између брзине процесора и оперативне меморије тако што у себи чува копије најчешће коришћених података, што значајно убрзава рад програма.

Главни принцип рада кеша је принцип локалности.

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

Принцип временске локалности: ако је програму у неком тренутку потребан неки податак (или инструкција), врло је вероватно да ће му исти податак бити поново потребан у блиској будућности.

Принцип просторне локалности: ако је програму у неком тренутку потребан неки податак (или инструкција), врло је вероватно да ће му у блиској будућности бити потребни и подаци из његове околине у меморији.

171 На који начин кеш користи принципе просторне и временске локалности?

Временска локалност: када се подаци искористе, они ће бити у кешу све док он не буде принуђен да их избрише (када дођу нови подаци). Када му понестане простора, кеш брише оне податке којима се није дуго приступало, а оставља оне којима јесте.

Просторна локалност: у кеш се не учитава један податак, већ читава околина дате адресе у меморији (32 или 64 бајта).

172 Објаснити читање кеша у случају поготка.

Уколико дође до поготка, процесор ће преузети податке из кеша, а главној меморији неће ни приступати. С обзиром на брзину кеш меморије, број циклуса чекања ће бити значајно мањи.

173 Објаснити читање кеша у случају промашаја.

Уколико дође до промашаја, податак се прво копира из меморије у кеш, а затим се прослеђује процесору. Овим се спречава промашај кеша при наредном приступу истом податку (што се очекује у блиској будућности). С обзиром да ћемо морати да приступамо главној меморији, то ће узроковати знатно дуже чекање (cache miss).

174 Објаснити писање кеша у случају промашаја.

Постоје два приступа:

- упис без алокације - податак се ажурира у меморији, без учитавања у кеш
- упис са алокацијом - податак се ажурира у меморији (не мора уколико користимо политику одложеног уписа), и одговарајући кеш блок се учитава у кеш

175 Објаснити писање кеша у случају поготка.

Зависи од политике уписа.

- политика директног уписа - податак се ажурира и у кешу, и у меморији
- политика одложеног уписа - податак се ажурира у кешу, а у меморију се учитава тек након што из кеша бива избачен

176 Објаснити директно мапирање адреса кеша и дати пример.

Коментар: Свака адресибилна локација у кеш меморији представља једну линију кеша (cache line). Она у себи може да чува један кеш блок (64 бајта који почињу на позицији у меморији дељивој са 64.)

Сваки кеш блок може бити учитан у само једну, унапред фиксирану кеш линију чији се редни број може добити на основу редног броја кеш блока. Уколико је линија на коју учитавамо нови кеш блок заузета, из ње се избацују стари подаци и уписују нови.

Предност је јефтина имплементација претраге, упоређујемо тагове само на тим специфичним местима. Мана је што ће већина кеша вероватно остати празна.

Пример недостаје.

177 Објаснити асоцијативно мапирање адреса кеша и дати пример.

Сваки кеш блок може да се смести у било коју кеш-линију. Приликом претраге, потребно је упоредити тагове свих кеш-линија са траженим, што успорава алгоритам и отежава имплементацију. Овај приступ даје најбољу искоришћеност кеша, јер не морамо да избацујемо коришћену линију ако има места у кешу.

Пример недостаје.

178 Објаснити скуп-асоцијативно мапирање адреса кеша и дати пример.

Кеш линије су подељене на скупове од по 2, 4, 8 или 16 линија. Сваки блок има тачно одређен скуп у ком може да се смести, а у скупу може да се налази у било којој линији.

Овај приступ је комбинација асоцијативног и директног.

Пример недостаје.

179 Због чега се врши замена линија кеша? Набројати најзначајније политике замене.

Замена кеша се врши:

- потпуно асоцијативни кеш: када је кеш попуњен
- кеш са директним мапирањем: када је попуњена јединствена кеш линија у коју се мора учитати тражени кеш блок
- скуп асоцијативни кеш: када је попуњен одговарајући скуп придружен датом кеш блоку

Најзначајније политике замене су LRU, псеудо-LRU и политика случајног избора.

Коментар: Политика случајног избора брише једну насумичну линију из кеша. Перформанса је упоредивих са LRU.

180 Објаснити LRU политику замене линије кеша. Добре и лоше стране.

Води се евиденција о томе којој линији најдуже није приступано, и када дође до замене, избацује се најдуже некоришћена.

Предност се огледа у принципу временске локалности, ако линија није дуго коришћена, вероватно је непотребна. Главни проблем је што евиденција о најдуже некоришћеној линији је што је број стања расте веома брзо са бројем кандидата.

181 Објаснити псеудо-LRU политику замене линије кеша. Добре и лоше стране.

Замењује се линија која довољно дуго није коришћена (не инсистирамо да то буде најдуже некоришћена).

Кеш линије се деле на половине, где се памти у којој половини је била најдуже некоришћена линија, те се та половина поново дели на половине.

Ова политика је једноставна за имплементацију, али није довољно прецизна.

182 Које политике писања кеша постоје и у чему се разликују?

Постоје политика директног и одложеног уписа. Допуни.

183 Објаснити политику директног уписа (write-through). Добре и лоше стране.

При сваком упису се вредност података ажурира и у кешу и у меморији.

Предност је једноставност имплементације (не морамо да бринемо о подацима приликом избацивања), Недостатак је што је потребно приступати меморији при свакој операцији уписа.

184 Објаснити политику одложеног уписа (write-back). Добре и лоше стране.

Приликом уписа ажурира се само копија у кешу, док се копија у меморији ажурира тек када кеш линија бива избачена из кеша.

Додатно, можемо водити рачуна о томе да ли је податак у кешу модификован увођењем „прљавог бита“ који нас обавештава о томе.

Предност је што се не приступа главној меморији, већ само кешу. Недостатак је у сложенијој имплементацији операције замене линије кеша (морамо да проверавамо да ли су подаци измењени и, ако јесу, да их упишемо у меморију).

185 Раздвојени и унификовани кеш. Поређење.

Унификовани кеш у себи садржи све податке који се кеширају, и инструкције и податке.

Раздвојени кеш подразумева постојање одвојеног кеша за инструкције и за податке.

Предности раздвојеног кеша

- Оба кеша можемо боље прилагодити обрасцима приступа који се разликују за ове две врсте података.
- Два кеша не морају бити исте величине.
- Кеш за инструкције се може имплементирати једноставније (инструкције се никад не модификују те можемо укинути политике уписа, чување информације о променама итд.)
- Такође знатно се може смањити колизија приликом наизменичног приступа инструкцијама и подацима.

Предности унификованог кеша

- Једноставнија имплементација
- Код раздвојеног кеша може доћи да се један кеш попуни док је други празан, што се не дешава код унификованог

186 Објаснити архитектуре вишестепеног кеша и начин њиховог функционисања.

Брзина кеш меморије опада са њеном величином, те нам велика кеш меморија не доноси значајно убрзање, али се мала кеш меморија брзо напуни, што је чини бесмисленом. Због тога, уводи се више кеш меморија различитих брзина и величина, које су надовезане једна на другу између процесора и главне меморије (нивои кеша).

Кеш једног нивоа тиме третира кеш нивоа испод његовог као сопствену системску меморију, и из њега захтева нове кеш линије, а њему враћа кеш линије које је заменио другим.

187 Објаснити разлику између ексклузивног и инклузивног вишестепеног кеша. Предности и мане.

Инклузивни кеш: ако се неки податак налази у кешу на неком нивоу, налазиће се и у кешевима на свим нижим нивоима. Кад дође до промашаја у кешу вишег нивоа, одговарајућа кеш линија се копира из кеша нижег нивоа и остаје у њему.

Ексклузивни кеш: сва три нивоа су дисјунктни. Када дође до замене линија у кешу вишег нивоа, два кеш нивоа замењују линије које у себи чувају (у нижем се налази она избачена из вишег).

Предност ексклузивног кеша је ефикаснија употреба меморијског простора, међутим, његова имплементација је сложенија јер захтева сложенију логику за интеракцију између различитих нивоа кеша.

Додатна предност инклузивног кеша је што величине кеш линија на различитим нивоима не морају бити исте, те ће нижи нивои садржати ширу околину податка коју ће моћи да проследе кешу вишег нивоа по потреби.

10 Систем прекида

188 Шта је систем прекида и која му је улога?

Систем прекида омогућава да се програм који се извршава привремено прекине, при чему се памти његово стање (контекст) - програмски бројач, флегови, вредности релевантних регистара процесора.

Контрола се преноси на неки други програм у меморији који је задужен за обраду прекида (опслуживање догађаја због којих је прекид настао). Када се прекид адекватно обради, контрола се враћа претходном програму (прекид је невидљив за прекинути програм).

Основна улога му је да омогући процесору да реагује на догађаје који захтевају хитну обраду.

189 Навести и укратко објаснити врсте прекида.

- хардверски прекид: изазивају га хардверске компоненте рачунара (маскирајући и немаскирајући)
- софтверски прекид:
- изузетак:

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

Хардверске прекиде (hardware interrupts) изазивају хардверске компоненте рачунара када желе да привуку пажњу процесора, најчешће у циљу обраде неког асинхроног догађаја. Могу се десити у било ком тренутку услед околности које није могуће предвидети (типично догађај који иницира корисник (притисак дугмета)). Зовемо их и асинхрони прекиди.

Маскирајући прекиди допуштају да процесор одлучи да их игнорише (то најчешће регулише неки процесорски флег). То је корисно зато што постоје неке критичне ситуације када је неопходно не реаговати на сигнале које улазно-излазни уређаји шаљу - на пример, док оперативни систем врши неке деликатне операције над сопственим структурама података.

Немаскирајући прекиди се не могу игнорисати ни на који начин и морају бити обрађени када до њих дође (хардверски кварови).

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

Софтверске прекиде изазивају програми који се извршавају, позивом одговарајуће инструкције. Дешавају се када програм то пожели, у тренутку који он одабере (софтверски прекиди се зову и синхроним).

Улога софтверског прекида је да се преда контрола оперативном систему, уз аутоматски прелазак на виши ниво привилегија.

Многим осетљивим ресурсима рачунара (попут улазно-излазних уређаја, спољашњих меморија и слично) није могуће приступити директно, из разлога ефикасности и безбедности. Процесор физички спречава приступ овим ресурсима програмском коду са ниским нивоом привилегија. Уместо тога, кориснички програм помоћу прекида захтева услугу од оперативног система који ће му их пружити на безбедан и ефикасан начин.

У ту сврху, оперативни систем обезбеђује скуп *системских позива*, посебних функција оперативног система које кориснички програми могу позивати и на тај начин захтевати одређене услуге.

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

Изузеци су врста прекида које изазива сам процесор, као реакцију на неку непредвиђену ситуацију током извршавања инструкција програма. Догађају се као последица неисправног кода (дељење нулом, прекорачења, извршавање инструкција за које програм нема привилегија, недостатак ресурса).

Користе се како би оперативни систем покушао да отклони ове грешке или уколико није у могућности, заустави неисправан програм.

193 Објаснити обраду хардверских прекида прозивком.

Идентификација уређаја који је изазвао прекид и обрада прекида.

- контролер уређаја који захтева прекид у свој статусни регистар уписује специјалну вредност (уређај жели прекид)
- контролер шаље сигнал за прекид процесору
- процесор реагује тако што чува контекст текућег програма и извршава *јединствену* функцију за обраду прекида
- функција редом „прозива“ улазно-излазне уређаје тако што очитава статусне регистре њихових контролера
- након што пронађе контролер чији статусни регистар говори да се ради о уређају који захтева прекид, даље комуницира са тим контролером у циљу обраде прекида
- када се обрада прекида заврши, контрола се враћа прекинутом програму

Имамо јединствену функцију коју нам пружа оперативни систем, а која обрађује све врсте прекида (аутоматски се позива кад се прекид догоди). Редослед прозивке се одређује у складу са неком политиком, како би сви уређаји који траже прекид били „прозвани“.

Главна предност је једноставност хардверске подршке, све што нам је потребно је регистар (IFR) који чува адресу јединствене функције за обраду прекида.

С друге стране, механизам прозивке није нарочито ефикасан, нарочито у случају већег броја улазно-излазних уређаја.

194 Објаснити обраду хардверских прекида засновану на векторима прекида.

Доминантан приступ обради хардвера у данашње време (interrupt vectors). Уређајима (или другим догађајима) се придружују цели бројеви (зову се вектори) који их једнозначно идентификују. За сваки вектор постоји посебна функција која се позива само за његов прекид.

С обзиром на то, хардверска подршка је знатно сложенија. Мора постојати регистар који у себи садржи адресу табеле свих дескриптора (функција) за сваки вектор.

- Приликом појаве прекида, процесор активира посебан излазни сигнал којим обавештава уређај да је спреман да обради његов прекид. Уређај реагује тако што прослеђује процесору вектор свог прекида путем меморијске магистрале.
- Процесор помоћу вектора прекида и регистра приступа дескриптору прекида, из њега чита привилегије. Улази у тај ниво привилегија и извршава специфичну функцију за обраду прекида.
- Након што се заврши обрада прекида, функција враћа све како је било пре свог позива, излази из свог нивоа привилегија, и прекинути програм наставља са радом

Иако је компликованији, приступ је значајно ефикаснији јер процесор директно зове функцију која обрађује прекид, и одмах зна који га је уређај изазвао.

Проблем једино настаје када више уређаја истовремено изазове прекид. Алгоритам за њихово бирање мора се имплементирати хардверски.

195 Шта је контролер прекида и чему служи? Објаснити начин обраде прекида у присуству контролера прекида.

Контролер прекида користи се да одабере редослед обраде хардверских прекида. Уместо процесору, улазно-излазни уређаји захтевају прекид од њега, а он одлучује који ће проследити процесору.

Након што сигнали за прекид стигну у контролер, он процесору шаље сигнал за прекид. Када процесор потврди да може да изврши обраду прекида, контролер применом неке политике најприоритетнији уређај и његов вектор прекида шаље процесору.

По завршетку свог рада, функција за обраду прекида обавештава контролер да је обрада прекида успешно завршена. Њему је то сигнал да може да бира следећи уређај, или уколико их нема више, да стане.

196 Објаснити на који начин се хардверски прекиди користе у сврху распоређивања процеса и послова од стране оперативног система.

Политике са преотимањем.

Расподела помоћу хардверских прекида се може реализовати на два начина:

1. Прекиди изазвани корисничким акцијама: свака акција корисника с улазним уређајима изазива хардверски прекид
    - Контрола се принудно одузима од процеса који је извршава и предаје оперативном систему, који контролу преноси процесу за који је корисничка интеракција релевантна.

- Мана је што се контекст мења само док је корисник активан, што запоставља позадинске процесе

2. Периодични прекиди генерисани хардверским тајмером (Дељење времена)
    - Да би се постигао ефикасан вишепрограмски рад (multitasking) и праведно дељење времена (timesharing), хардверски прекиди се вештачки изазивају у равномерним интервалима коришћењем посебног хардверског уређаја (програмабилни интервални тајмер).
    - Интервал између два прекида се зове временски квант
    - Када тајмер генерише прекид, оперативни систем преузима контролу и покреће неки алгоритам за распоређивање процеса (одређивање који ће се процес извршити: кружно (round-robin) или на основу задатих приоритета)
    - Ствара се илузија да се програми извршавају истовремено

197 Објаснити на који начин се софтверски прекиди користе за имплементацију системских позива на оперативним системима.

Иако системски позиви делују као обичне функције, ако бисмо их као такве извршили, радили би на истом нивоу привилегија као и корисник (најнижем).

Потребна реализација остварује се помоћу софтверских прекида. Системски позив се врши тако што се у регистре процесора (или на стек) поставе информације о жељеном системском позиву и његовим параметрима, а затим се изазива софтверски прекид.

Оперативни систем обезбеђује посебну функцију за обраду овог прекида која ће се извршити са високим нивоом привилегија. На основу информација које је корисник доставио, извршиће одговарајућу функцију оперативног система која имплементира жељени системски позив.

11 Улазно излазни уређаји

198 Како делимо улазно-излазне уређаје према смеру преноса података?

- Улазни уређаји: помоћу њих се подаци и програми учитавају у рачунар, тј. подаци теку од уређаја ка процесору и меморији

- Излазни уређаји: помоћу њих се подаци из рачунара могу проследити некоме у спољној средини (кориснику, другом рачунару и сл.), тј. подаци теку од процесора ка уређају

199 Навести најчешће коришћене врсте улазних уређаја на модерним рачунарима.

Тастатура, показивачки уређаји (миш, додирна површина), микрофони и камере.

200 Навести најчешће коришћене врсте излазних уређаја на модерним рачунарима.

Штампачи (ласерски, млазни), екрани (графички кориснички интерфејс), звучници.

201 Навести примере двосмерних улазно-излазних уређаја на модерним рачунарима.

Екрани осетљиви на додир, комуникациони уређаји (мрежни адаптери са жичаном везом, бежични мрежни адаптери, блутут адаптери и слично), спољне меморије (оптички дискови, флеш меморије, тврди дискови, SSD дискови).

202 Шта су улазно-излазни контролери и која је њихова функција?

Врше комуникацију са улазни-излазним уређајима уместо процесора Основне функције:

- да обезбеде управљање улазно-излазним уређајем и комуникацију са њим, узимајући у обзир специфичности самог уређаја (ако уређај прослеђује аналогни сигнал, обезбедити одговарајућу конверзију у дигитални сигнал који процесор разуме) (многи уређаји имају механичке покретне делове, те је потребно контролисати њихов рад)
- да обезбеде релативно једноставан и униформисан интерфејс према процесору, како би сакриле специфичности уређаја од процесора (модел сличан као при комуникацији са оперативном меморијом)

Поједини улазно-излазни уређаји имају интегрисане контролере (мрежни адаптери), док је код неких других контролер одвојена компонента (звучници и микрофони се на рачунар повезују уз помоћ звучне картице, док се монитор повезује помоћу графичке картице).

203 Описати типичан интерфејс улазно-излазног контролера који се користи за комуникацију са процесором.

Улазно-излазни контролер у себи садржи извесни број регистара којима процесор може приступати путем интерфејса контролера. Типично, садржи следеће регистре:

- командни регистар: у овај регистар процесор уписује команде којима се захтева иницијација одговарајуће операције са уређајем. Појединим командама може се вршити подешавање контролера које ће се примењивати у наставку рада
- статусни регистар: у овај регистар сам контролер уписује информације о свом статусу и/или о статусу извршене операције (нпр. да ли је контролер спреман да прихвати следећу команду, да ли је уређај заузет јер је операција у току, да ли је дошло до грешке и слично)
- регистар података: служи за размену података између процесора и контролера. (излазна операција - процесор уписује, улазна операција, контролер уписује а процесор чита). Може их бити више и неки од њих могу имати специфичне намене

204 Објаснити меморијски мапирани улаз/излаз.

Меморијски мапирани улаз/излаз подразумева уградњу регистара улазно-излазних контролера у адресни простор процесора (регистирма придружујемо из адресног процесора). Процесор ће моћи да чита и уписује у те регистре користећи исте инструкције као и за меморијске трансфере.

Једино што се прескаче јесте кеш меморија, јер сам улазно-излазни уређај може да мења вредност својих регистара, те нам је немогуће закључити да ли се у кешу налазе исти подаци као и у регистрима.

205 Објаснити изоловани улаз/излаз.

Процесор има два адресна простора, један за у који се мапира радна меморија, и други у који се мапирају регистри улазно-излазних контролера. Простори не морају бити исте величине (меморијски је обично знатно већи) и архитектура обезбеђује посебан скуп инструкција трансфера за приступ улазно-излазном адресном простору.

Пример архитектуре: x86

206 Објаснити технику програмираног улаза/излаза.

У случају синхроних улазно-излазних операција (програм бира да до њих дође), најједноставнији начин комуникације између процесора

и улазно-излазних контролера се састоји у томе да процесор, након издавања команде, чека у петљи док не добије одговор.

Процесор најпре проверава да ли је уређај слободан, уколико није, сачекаће да постане. Након тога се у командни регистар уписује одговарајућа команда која налаже контролеру да започне операцију. Потом ће процесор чекати да уређај заврши, и кад до тога дође, дохватиће очитане податке и отићи својим путем.

207 Објаснити технику улаза/излаза вођеног прекидима.

Процесор издаје наредбу контролеру да га обавести када изврши задату операцију, и наставља да ради неки други користан посао. Када се ослободи, контролер ће послати сигнал за прекид процесору, што ће га натерати да запамти стање тренутног програма, прочита оно што му је контролер доставио у регистру, смести податак на одговарајуће место у меморији, и настави извршавање оног програма.

208 Објаснити директан приступ меморији (DMA). Контролер DMA. Принцип рада.

Контролер по завршетку операције улазно-излазног уређаја не комуницира с процесором, већ са DMA контролером, који му налаже на који начин да добављене податке смести у радну меморију.

- Када је потребно обавити неку улазно-излазну операцију која подразумева масовни трансфер података, процесор шаље одговарајућу команду улазно-излазном контролеру
- Потом процесор DMA контролеру шаље релевантне информације (идентификатор улазно-излазног контролера, смер трансфера, број речи које је потребно пребацити, и почев од које адресе)
- Процесор наставља да ради неки користан посао, док DMA контролер на себе преузима улогу процесора. Чека да му улазно-излазни контролер пошаље сигнал да је прва реч спремна за читање
- Када му стигне реч, DMA контролер наређује контролеру да податак постави на магистралу података, а меморији шаље сигнал и адресу за упис
- Након што се прва реч пренела, улазно-излазни контролер наставља са читањем следеће речи, над којом се примењује идентичан поступак, све док не понестане података у захтеву
- Када се сви подаци пренесу, DMA шаље сигнал за прекид процесору, чиме га обавештава да су подаци спремни

12 Виртуелна меморија

209 Шта је виртуелна меморија и због чега се користи?

Виртуелна меморија је виртуелни сопствени адресни простор неког програма. Програм види виртуелну меморију као једну целину и сматра да је сам у њој. Виртуелна меморија се може протезати преко више независних делова радне меморије, а чак може користити и спољашња складишта, у која премешта делове програма који се нису дуго користили, а враћа их у оперативну меморију када постану потребни (програм није свестан да је до тога дошло).

Користи се из пар разлога

- сви програми могу заузимати више радне меморије него што систем има на располагању
- како би се избегло преклапање програма (један већ учитан програм стоји тачно на средини и дели меморију на два дела исувише мала да би у њих појединачно наш програм стао)
- програмер не размишља о адресама на којима се налазе остали програми, јер су му доступне само оне којима програм сме да приступа
- програм не може да прочита меморију неких других програма и потенцијално угрози безбедност система

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

Најчешћи вид реализације виртуелне меморије је техника страничења. Виртуелни адресни простор програма и физичка меморија се не посматрају као целина, већ су подељени на мање блокове фиксне и једнаке величине.

Виртуелни адресни простор програма подељен је на странице (код x86 архитектуре, странице су величине 4 KB). Програм користи виртуелне адресе које се логички састоје од броја странице и помераја.

Физичка меморија (и радна меморија и спољашње складише) подељена је на оквире који су исте величине као и странице.

Повезивање странице виртуелне меморије и оквира физичке меморије врши се преко табеле страница (page table) која за сваку виртуелну страницу бележи у ком се физичком оквиру налази.

211 Објаснити пресликавање адреса виртуелне меморије. Пример.

Пресликавање адреса је процес у коме се виртуелна адреса коју програм користи претвара у стварну физичку адресу у радној меморији.

Одвија се хардверски (у процесору) уз помоћ структура које одржава оперативни систем.

Да би се омогућило пресликавање, виртуелна адреса се дели на два дела, број странице (горњи битови) који представља индекс странице на којој се налазимо, и померај (offset) (нижи битови) који одређује тачну локацију бајта унутар странице.

Када се ради пресликавање, број странице се користи као индекс за табелу страница, која нам саопштава у ком оквиру се адреса налази.

Потом се на на битове индекса придодаје померај, и добијена адреса је адреса реалног податка у меморији. Уколико је податак тренутно ван меморије, он се учитава и тек онда се враћа његова адреса.

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

Главни проблем са основном организацијом виртуелне меморије је што табеле страница могу постати изузетно велике и заузети превише корисног простора у радној меморији.

Решење је да, уместо да чувамо табелу странице у непрекидном облику, и њу саму распршимо по меморији у произвољне слободне оквире. Да бисмо знали где се која страница табеле налази, уводимо Директоријум табеле страница, табелу највишег нивоа. Тако, чак и делове табеле страница можемо чувати на диску уколико дуго нису коришћени.

Пример: у случају двостепеног пресликавања адреса на архитектури x86, виртуелна адреса се логички дели на:

- број директоријума (10 битова) - индекс у директоријуму табеле страница
- број странице (10 битова) - индекс унутар одабране странице табеле страница
- померај (12 битова) - тачна позиција бајта унутар странице

Напомена: и директоријум можемо поделити на странице, и изградити директоријум директоријума (x86_64).

213 Шта су и када се користе политике замене страница?

Политика замене страница подсећа на замену кеш линија, с тим да, када се радна меморија препуни, по некој политици избацујемо једну страницу и чувамо је у спољној меморији (најчешће по некој варијанти LRU стратегије).

214 Објаснити значај величине странице виртуелне меморије и навести примере.

Величина странице виртуелне меморије знатно утиче на перформансе, искоришћеност меморије и ефикасност рада са диском.

Увођење и употреба већих страница има своје предности и мане:

- Предност:
- мање табеле страница - укупан број страница од којих се састоји адресни простор је мањи, па су и табеле страница мање
- ефикасније коришћење диска - спољни дискови углавном боље раде када већој количини података приступамо одједном уместо у мањим целинама
- Мана:
- велика фрагментација: с обзиром да се цела страница прослеђује програму, већа је шанса да она не буде у потпуности искоришћена, и ако то применимо на све странице пола меморије бива празно
- мања прецизност: када је програму потребна адреса, програм са њом учитава и огромну страницу, коју можда нема потребе да учитава

215 Шта су таблице, а шта директоријуми страница виртуелне меморије? Објаснити.

Таблица страница виртуелне меморије је табела која омогућава превођење виртуелних адреса у физичке адресе.

Директоријуми страница виртуелне меморије су табеле највишег нивоа која садржи позицију оквира у којем се налази део таблице страница виртуелне меморије. Користи се зато што чување целе таблице у меморији захтева превише простора који може бити искоришћен за нешто боље.

216 Шта садрже ставке у таблици страница виртуелне меморије? Објаснити.

- број оквира физичке меморије - уколико је виртуелна страница учитана у меморију, ова ставка садржи тачан редни број оквира у ком се налази Ако страница није у меморији, ставка садржи информацију о њеном одсуству.
- контролни битови:

- битови права приступа: одређује дозвољене операције над страницом (да ли је дозвољена за читање/писање)
- бит коришћења: да ли је страница била коришћена од тренутка када је учитана у физичку меморију (помаже нам приликом замене страница)
- прљави бит: саопштава нам да ли је садржај странице мењан (помаже приликом замене страница)

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

Како се табеле и директоријуми страница налазе у радној меморији, да бисмо извршили превођење виртуелних у физичке адресе, потребно је да одемо у меморију по одговарајуће ставке из табеле. С обзиром да је приступ меморији спор, неопходно је користити кеширање.

TLB је специјални кеш у процесору који се користи у ове сврхе (мали, веома брз, обично имплементиран као раздвојен кеш). У случају мањих бафера таблице користимо потпуно асоцијативно мапирање, док се генерално користи скуп-асоцијативно мапирање.

13 Магистрале

218 Која је основна разлика између серијских и паралелних магистрала?

Серијске магистрале се састоје из једне линије преко које се подаци преносе бит по бит, док се паралелне састоје из више линија преко којих се подаци преносе реч по реч.

219 Које су предности серијских магистрала у односу на паралелне?

Приликом преношења речи код паралелних магистрала, не стижу сви битови на одредиште истовремено. То је зато што је немогуће направити n линија исте дужине и проводљивости. Временске разлике између пристизања бита су веома мале, али при великим ферквенцијама преноса долазе до изражаја и отежавају пријемном уређају да исправно очита вредности са магистрале.

При великим ферквенцијама долази до веома брзог смењивања јединица и нула (напонских нивоа) на линијама магистрале, што доводи до електромагнетне индукције, која може да утиче на сигнале са различитих паралелних линија и мењају их (интерференција).

220 Навести примере паралелних магистрала.

PCI, PATA, екстерни паралелни порт, меморијска магистрала.

221 Навести примере серијских магистрала.

PCI-EXPRESS, SATA, USB .

14 Завршна реч

Хвала свима који су помогли при реализације ове скрипте.
Посебно хвала Мини Немет која је нашла грешку у питању 131 :)
За крај види слику 31.