9 SINTEZA SINHRONIZIRANIH SEKVENČNIH VEZIJ

9. 1 Sinteza iz besednega opisa

Zgled: Detektor sekvence.

Sinhronski sekvenčni avtomat z dvema vhodnima in dvema izhodnima stanjema mora dati na izhodu stanje 1 vsakič, ko je na vhodu detektirana sekvenca 0101, v vseh ostalih primerih je izhodno stanje 0.

To pomeni, da vhodni sekvenci 010101 pripada izhodna sekvenca v časovnem zaporedju 000101.

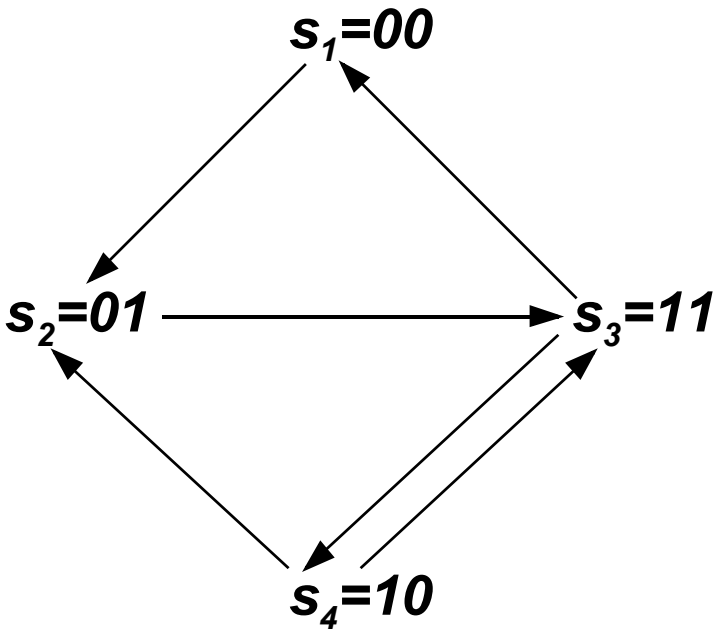
Sintezo bomo opravili na osnovi Mealyjevega modela.

Zgradimo tabelo prehajanja stanj, ki s formalnim jezikom opisuje delovanje avtomata.

S	$\Delta^1 S, Z$	
	$x_1 = 0$	$x_2 = 1$
s_1	s_2, 0	s_1, 0
s_2	s_2, 0	s_3, 0
s_3	s_4, 0	s_1, 0
s_4	s_2, 0	s_3, 1

Tabela prehajanja stanj dobi sedaj takšno obliko:

y_1y_2	$\Delta^1y_1\ \Delta^1y_2$		z	
	x = 0	x = 1	x = 0	x = 1
00	01	00	0	0
01	01	11	0	0
11	10	00	0	0
10	01	11	0	1



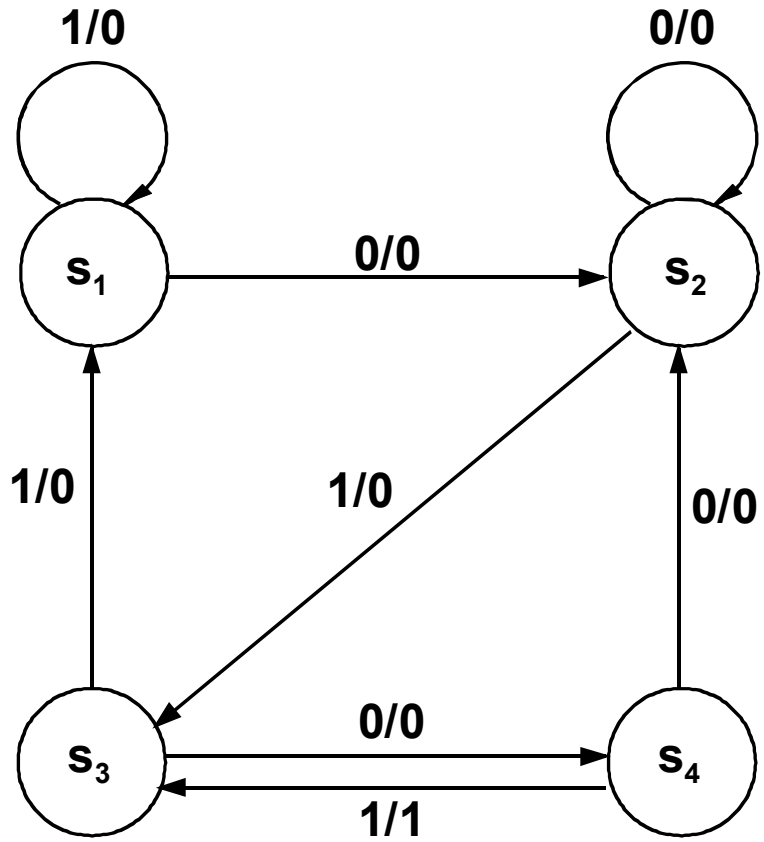
Za realizacijo izberemo tokrat D spominski celici.

y_1y_2	$\Delta^1y_1\ \Delta^1y_2$		z	
	x = 0	x = 1	x = 0	x = 1
00	01	00	0	0
01	01	11	0	0
11	10	00	0	0
10	01	11	0	1

$$\Delta^1y_1 = \bar{x}y_1y_2 + x\bar{y}_1y_2 + xy_1\bar{y}_2$$

$$\Delta^1y_2 = y_1\bar{y}_2 + \bar{x}y_1\bar{y}_2 + \bar{y}_1y_2$$

$$z = xy_1\bar{y}_2$$



Zamenjajmo sedaj kodni besedi pri stanjih $\mathbf{s_3}$ in $\mathbf{s_4}$. In poiščimo novo tabelo prehajanja stanj in izhodno tabelo.

y_1y_2	$\Delta^1y_1\ \Delta^1y_2$		z	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
00	01	00	0	0
01	01	10	0	0
10	11	00	0	0
11	01	10	0	1

$$\Delta^1y_1 = \bar{x}y_1\bar{y}_2 + xy_2$$

$$\Delta^1y_2 = \bar{x}$$

$$z = xy_1y_2$$

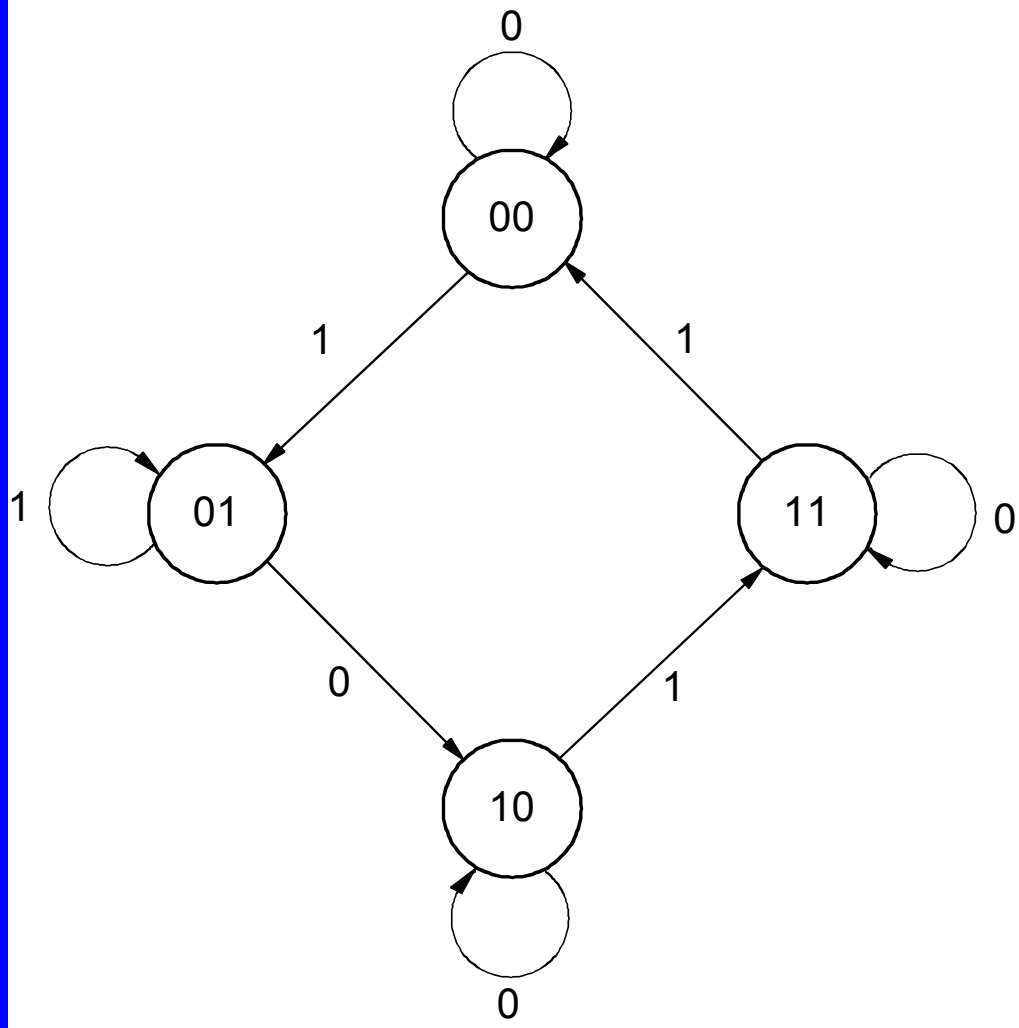
Vidimo torej, da kodiranje notranjih stanj avtomata ni enoznačen proces.

S spretnim kodiranjem lahko torej dosežemo velik prihranek vezja.

9. 2 Sinteza vezja iz diagrama stanj

Zgled:

Konstruirati želimo sekvenčno vezje s spodnjim diagramom stanj. Odločili smo se za uporabo JK spominskih celic.

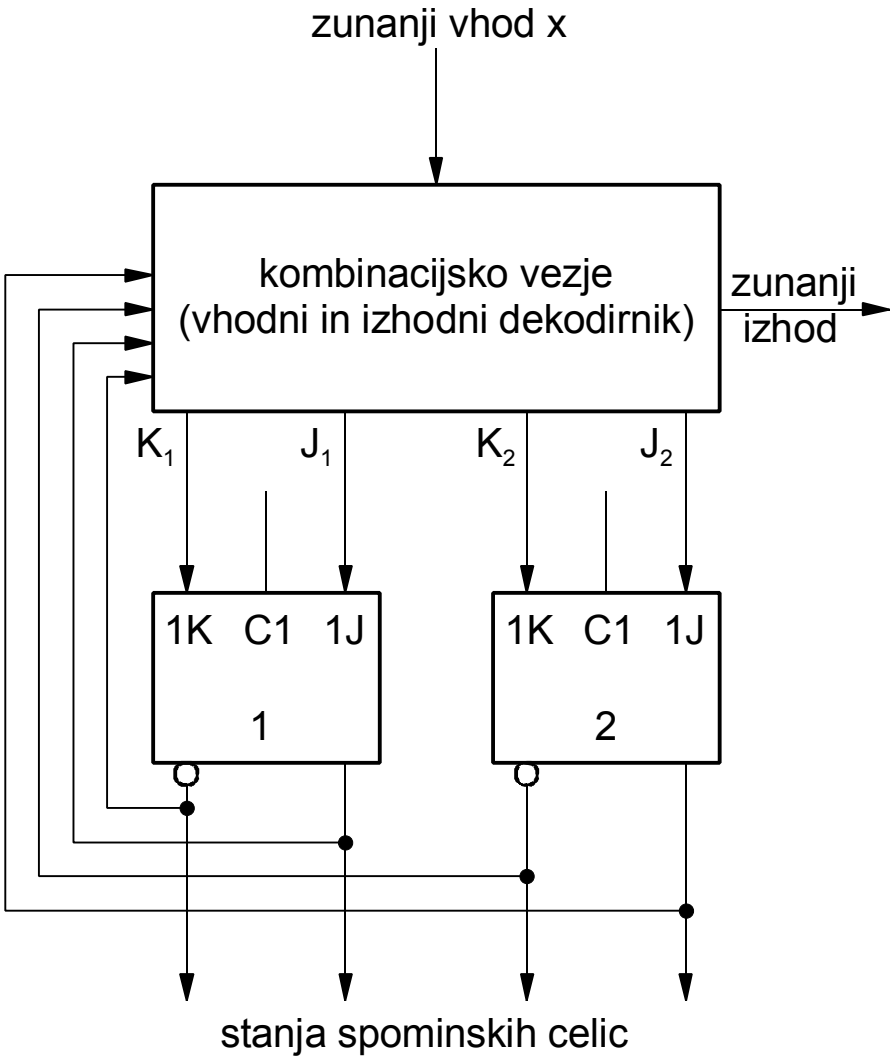


Sed. stanje		Naslednje stanje			
		Sed. x = 0		Sed. x = 1	
y_1	y_2	$\Delta^1 y_1$	$\Delta^1 y_2$	$\Delta^1 y_1$	$\Delta^1 y_2$
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

Sed. stanje		Naslednje stanje			
		Sed. x = 0		Sed. x = 1	
y_1	y_2	$\Delta^1 y_1$	$\Delta^1 y_2$	$\Delta^1 y_1$	$\Delta^1 y_2$
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

Vhodi v odločit. vezje			Naslednje		Izhodi iz odločitvenega vezja			
Sed. stanje		vhod	stanje		Vhodi v spominske celice			
y_1	y_2	x	$\Delta^1 y_1$	$\Delta^1 y_2$	J_1	K_1	J_2	K_2
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Sekvenčno vezje mora - glede na splošne oblike konstrukcije sekvenčnih vezij - imeti strukturo, kakršno vidimo na naslednji sliki:



Vhodi v kombinacijsko vezje so definirani s sedanjim stanjem in vhodnimi stolpci.

Izhodi iz kombinacijskega vezja pa so definirani v stolpcih za vhode k spominskim celicam.

Iz vzbujalne tabele moremo torej dobiti izjavnostno tabelo za kombinacijsko vezje.

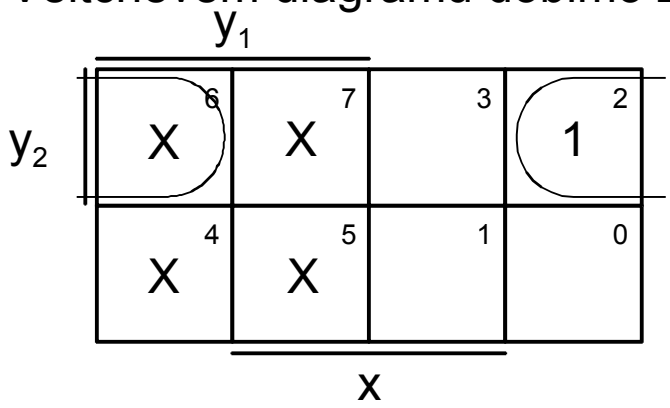
Vhodne spremenljivke so y_1, y_2, x, izhodi pa K_1, J_1, K_2, J_2.

Iz vzbujalne tabele sledi:

y_1	y_2	x	J_1	K_1	J_2	K_2
0	**0**	**0**	**0**	**X**	**0**	**X**
0	**0**	**1**	**0**	**X**	**1**	**X**
0	**1**	**0**	**1**	**X**	**X**	**1**
0	**1**	**1**	**0**	**X**	**X**	**0**
1	**0**	**0**	**X**	**0**	**0**	**X**
1	**0**	**1**	**X**	**0**	**1**	**X**
1	**1**	**0**	**X**	**0**	**X**	**0**
1	**1**	**1**	**X**	**1**	**X**	**1**

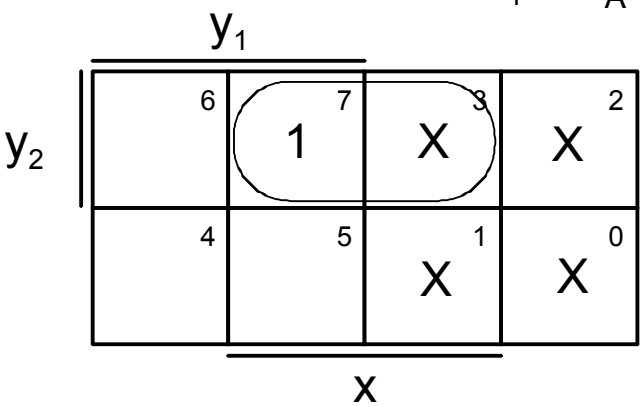
$J_1 = \bar{A}B\bar{x}$ + štiri neopredelejene kombinacije

V Veitchevem diagramu dobimo zato:

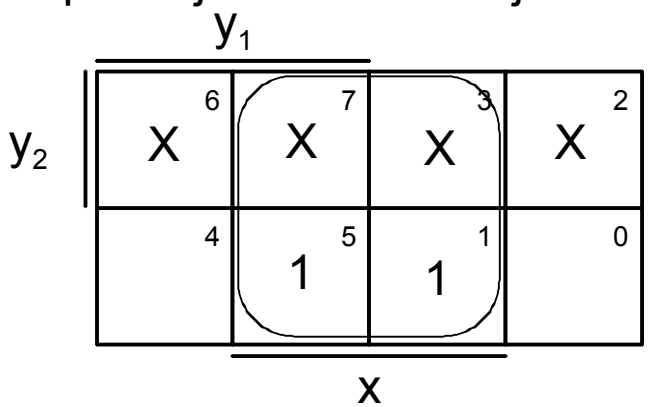


Poenostavljeno je torej: $J_1 = y_2\bar{x}$

Podobno dobimo za K_1 $K_A = ABx +$ štiri neopredeljene kombinacije



$K_1 = y_2x$

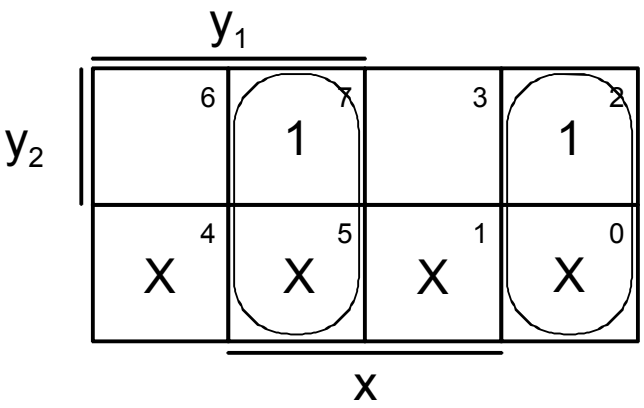


$J_2 = x$

Za spominsko celico 2 pa:

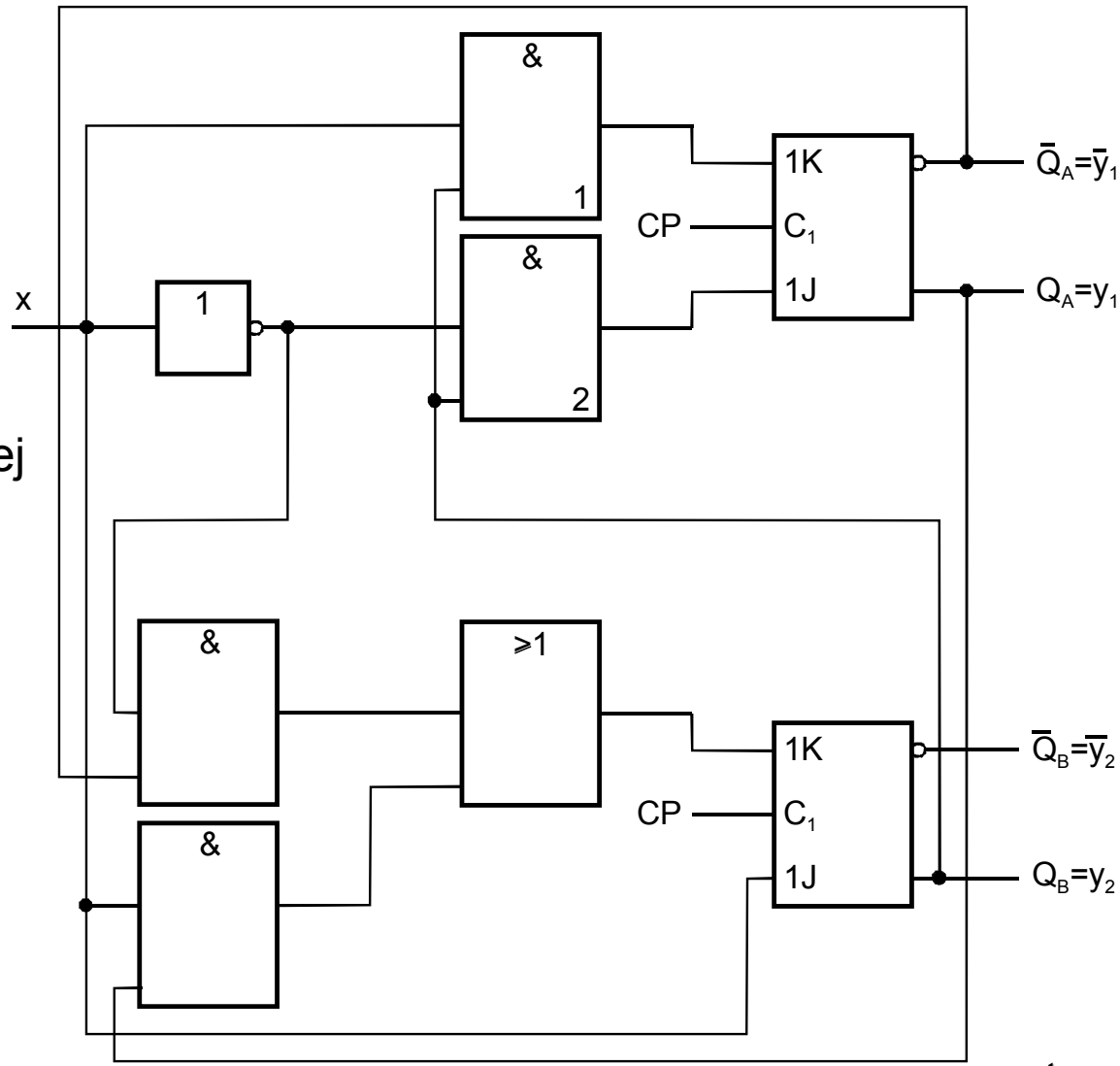
$J_2 = \bar{y}_1\bar{y}_2x + y_1\bar{y}_1x +$ štiri neopredeljene kombinacije

$K_2 = \bar{y}_1 y_2 \bar{x} + y_1 y_2 x +$ štiri neopredeljene kombinacije



$K_2 = y_1 x + \bar{y}_1 \bar{x}$

Celotno sekvenčno vezje je torej takole:



9. 3. Sinteza vezja iz časovnega diagrama

Konstruirajte vezje z spominsko celico Q in dvema vhodoma x_1 in x_2 tako, da bo realiziran spodnji časovni diagram.

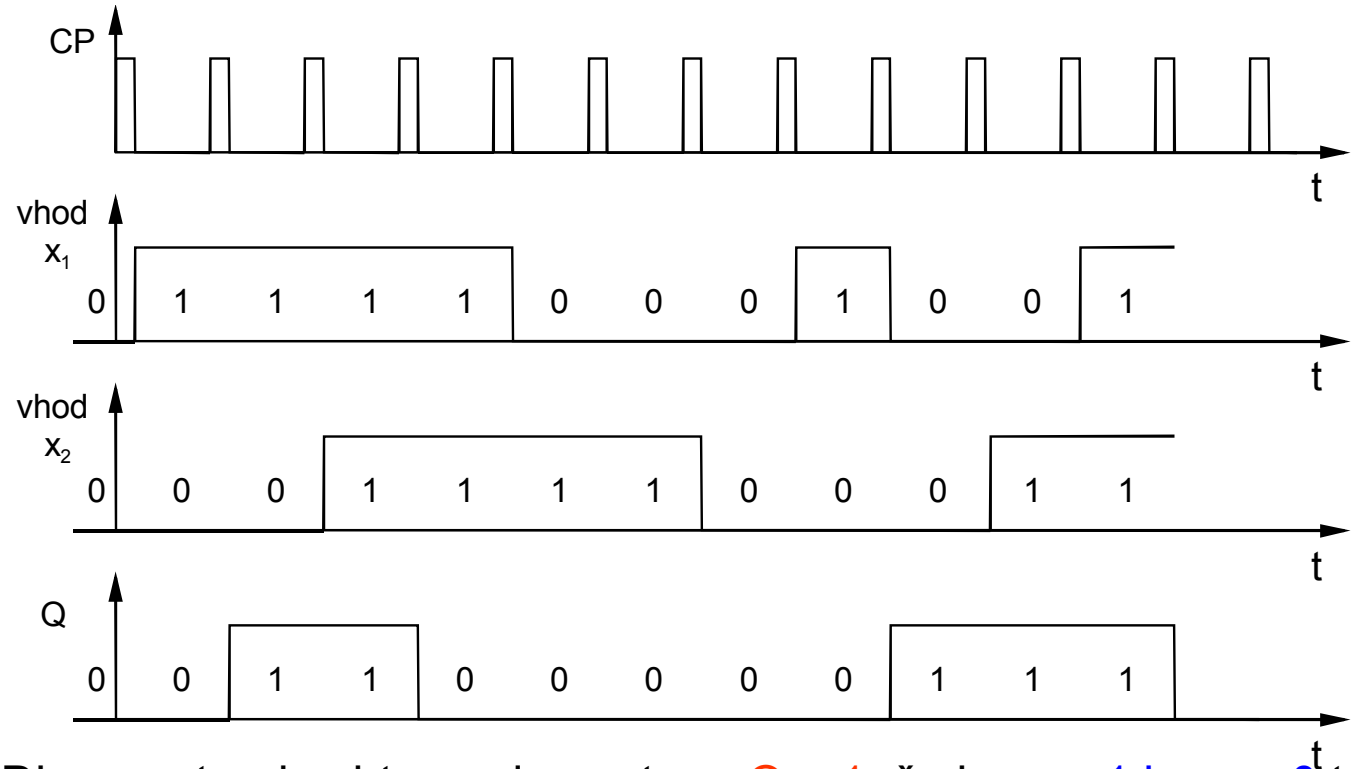


Diagram torej zahteva, da postane Q = 1, če je $x_1 = 1$ in $x_2 = 0$ ter Q = 0, če je $x_1 = 1$ in $x_2 = 1$ pri ostalih vhodnih kombinacijah $x_1 = 0$, $x_2 = 0$ ter $x_1 = 0$, $x_2 = 1$ pa naj ostane stanje nespremenjeno.

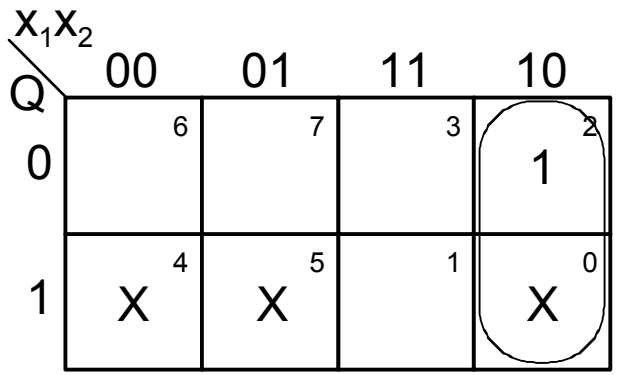
Diagram torej zahteva, da postane Q = 1, če je $x_1 = 1$ in $x_2 = 0$ ter Q = 0, če je $x_1 = 1$ in $x_2 = 1$ pri ostalih vhodnih kombinacijah $x_1 = 0$, $x_2 = 0$ ter $x_1 = 0$, $x_2 = 1$ pa naj ostane stanje nespremenjeno.

Sed. stanje	Sedanja vhoda		Nasl. stanje	Vhodi k spom. celicam	
Q	**x_1**	**x_2**	**$\Delta^1 Q$**	**S**	**R**
0	0	0	0	0	X
0	0	1	0	0	X
0	1	0	1	1	0
0	1	1	0	0	X
1	0	0	1	X	0
1	0	1	1	X	0
1	1	0	1	X	0
1	1	1	0	0	1

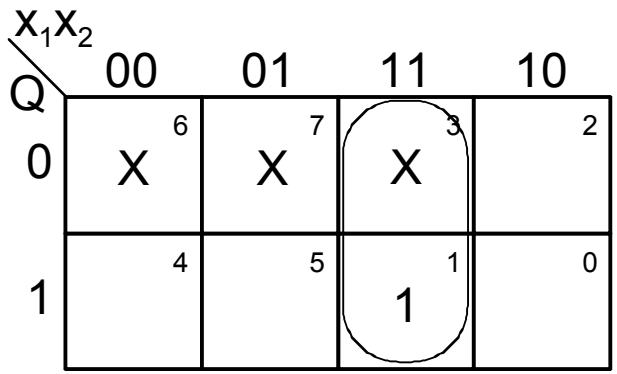
$$S = x_1 \overline{x}_2$$

$$R = x_1 x_2$$

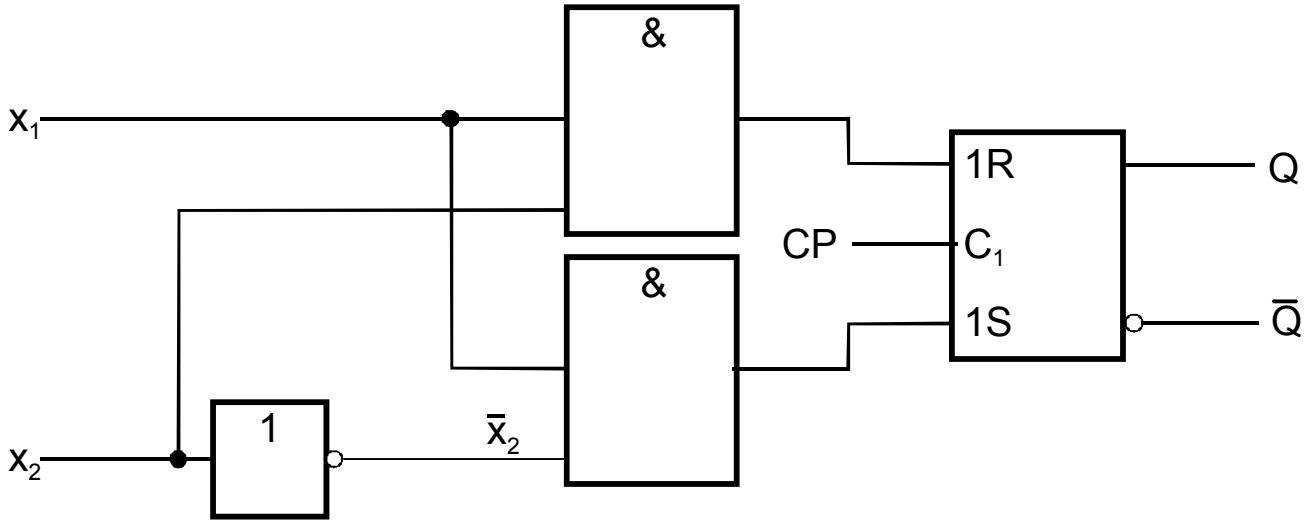
Funkcija S je:



Funkcija R pa:



In končno je vezje takole:



Sinteza nepopolno opredeljenega avtomata

Zgled:

Sekvenčno vezje z dvema spominskima celicama in enim vhodom mora na dajati naslednje sekvence. Če je vhod = 1, naj dajeta celici na svojih izhodih sekvenco 00 01 10.

Če je vhod = 0, naj se ponavlja sekvenca 11, 10, 01.

Reši to nalogo kot popolno opredeljen in kot nepopolno opredeljen avtomat

9. 4 Števci

0, 1, 2, ... , m, 0, 1, 2,...

$m \leq 2^n - 1$

Delilniki frekvence: 1 vhod in 1izhod – izhod se spremeni vsakih d vhodnih impulzov

Števec, ki šteje 0, 1, 2,..., m, je mogoče zgraditi z delilnikom z (m+1)

Delilnik frekvence z "d" je mogoče zgraditi s poljubnim števcem, ki šteje vsaj od 0, 1, 2,.., d -1, 0, 1, 2,...

9. 4. 1. Razdelitev števcev

- navadni ali enostavni števci (0, 1, 2,...., m, 0, 1, 2,.....) $m \leq 2^n - 1$
- posebni ali specialni števci

Vrste navadnih števcev:

binarni števci z različnimi kodami

dekadni števci

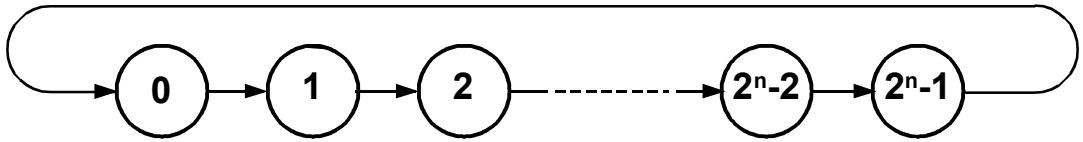
števci po modulu m

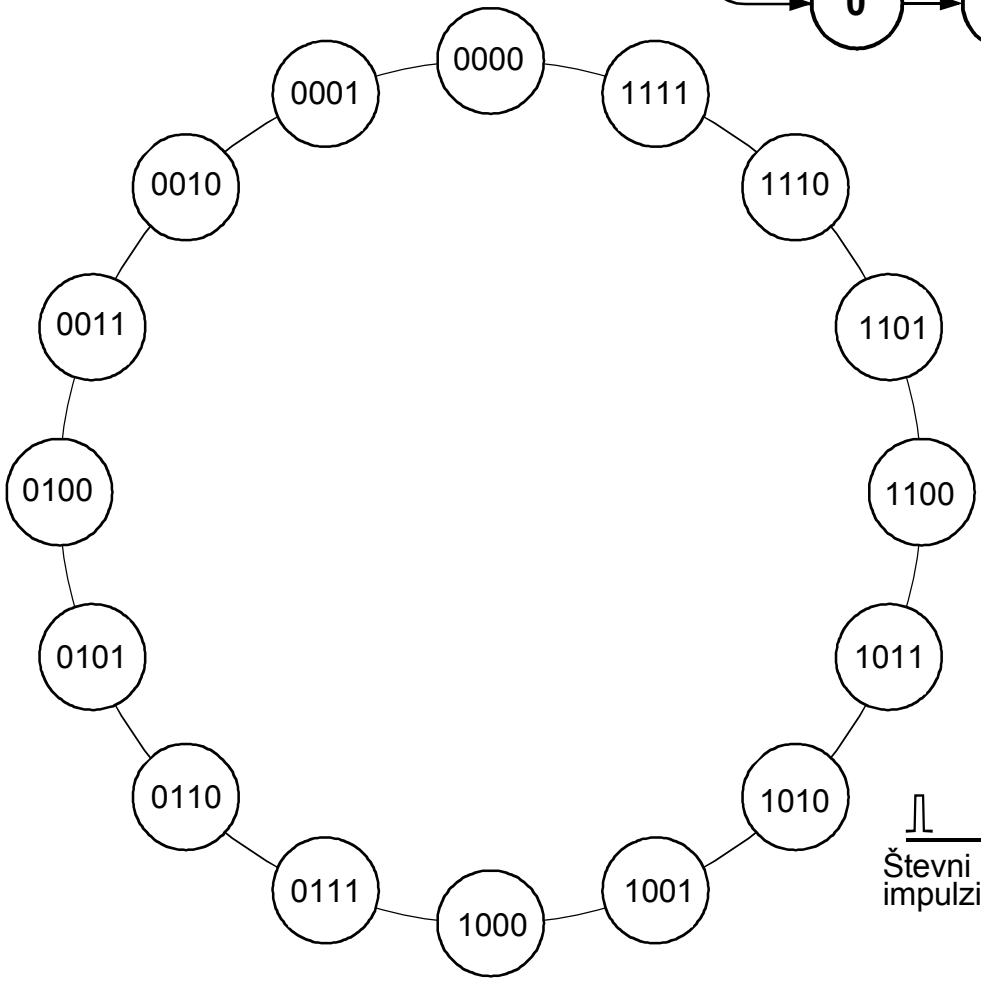
delilniki s številom "d"

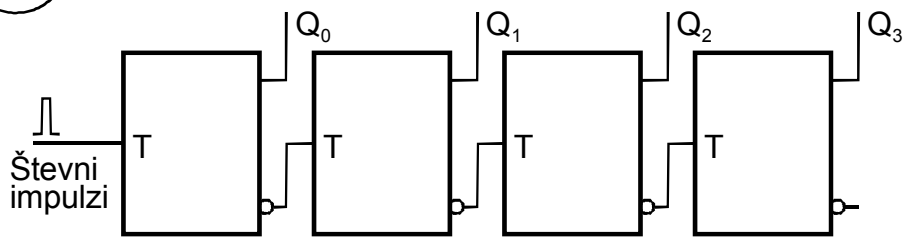
Posebni ali specialni števci:

- dvosmerni števci
- krožni števci
- števci s prepletanjem
- številno dekodirna vezja

9. 4. 2 Binarni števci

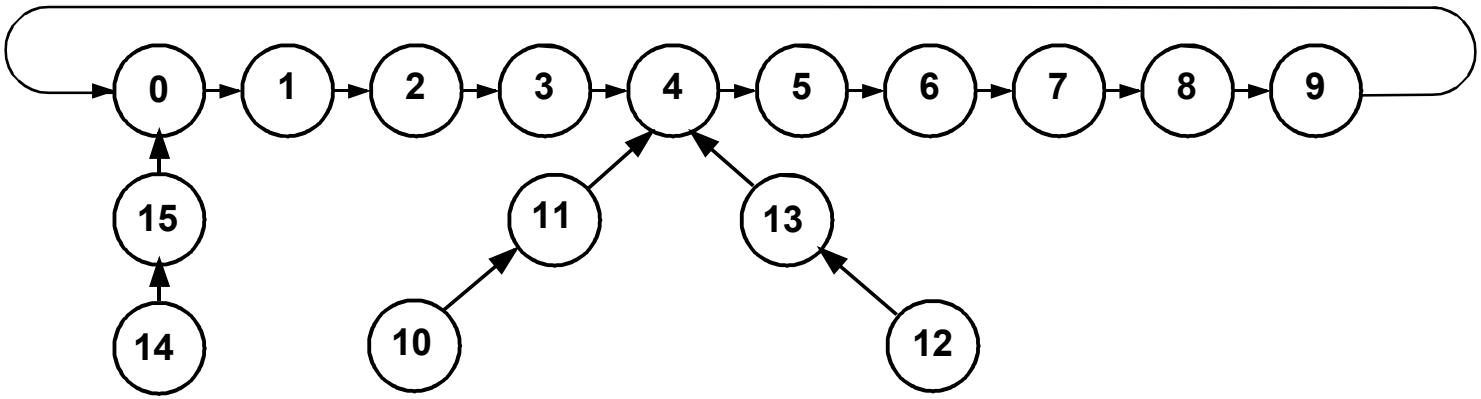






9. 4. 3 Dekadni števci

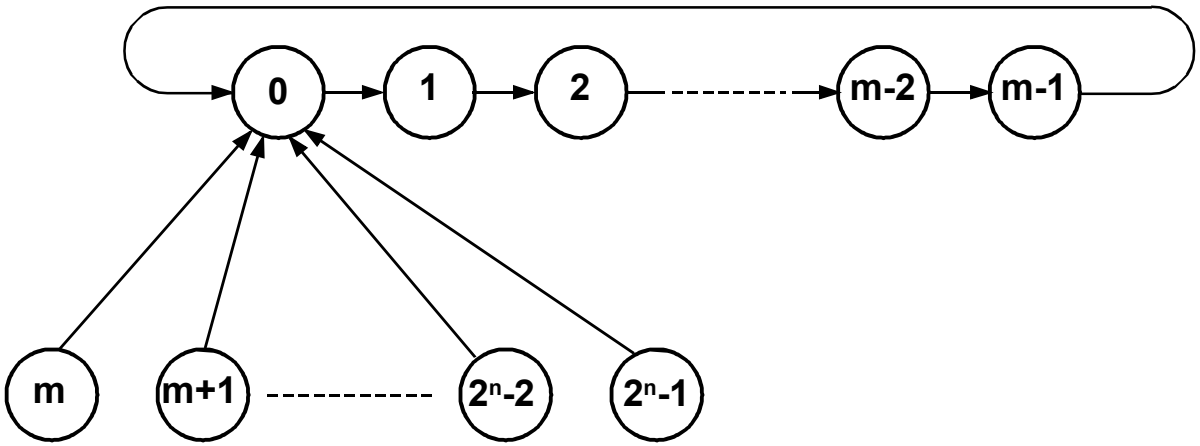
Dekadni Johnsonov števec



9. 4. 4 Števci po modulu "m"

0, 1, 2, ..., m, 0, 1, 2, ...

$m \leq 2^n - 1$



9. 4. 5 Krožni števci

9. 4. 5. 1 Standardni krožni števci

Standardni krožni števec potrebuje “n” spominskih celic za štetje po modulu “n+1”

Primer standardnega krožnega števca:

Q_1	Q_2	Q_3	Q_4
0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

ali

Q_1	Q_2	Q_3	Q_4
0	0	0	0
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0
0	0	0	0

9. 4. 5. 2 Krožni števci s prepletanjem

Krožni števec s prepletanjem pa potrebuje “n/2” spominskih celic za štetje po modulu “n”

Primer števca s prepletanjem

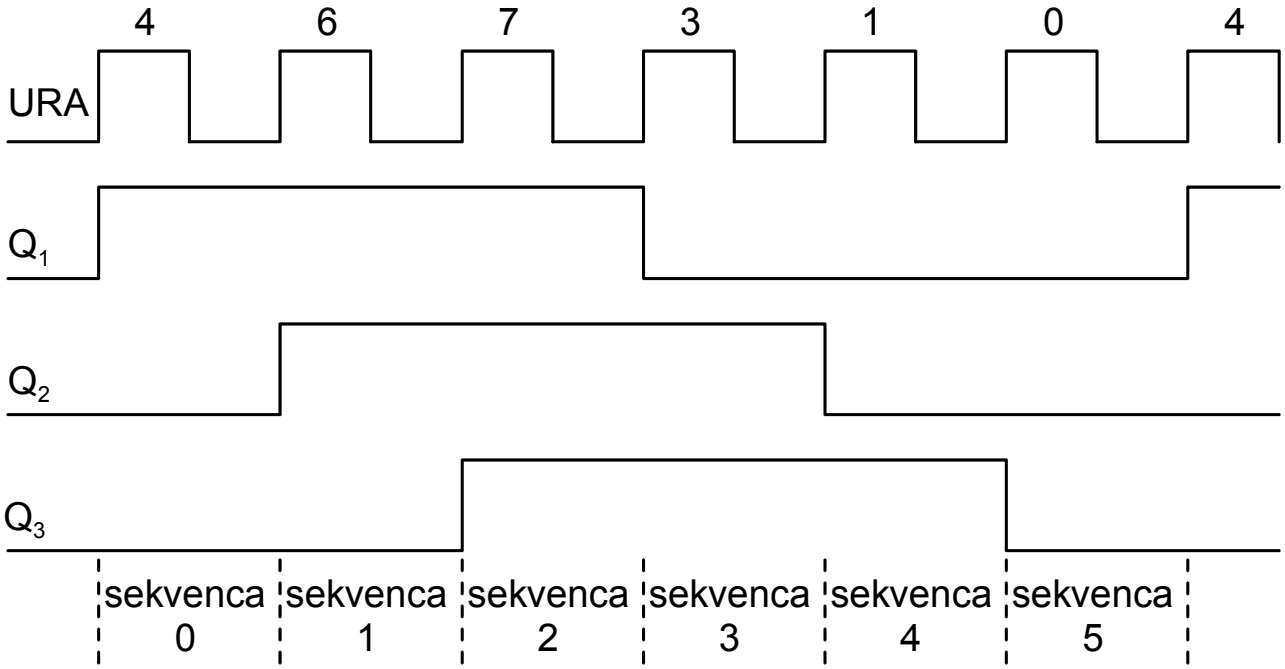
Q_1	Q_2	Q_3
0	0	0
1	0	0
1	1	0
1	1	1
0	1	1
0	0	1
0	0	0

Q_1	Q_2	Q_3
0	0	0
0	0	1
0	1	1
1	1	1
1	1	0
1	0	0
0	0	0

Potrebujemo torej n/2 spominskih celic za štetje po modulu n

Zanka nastane z invertiranjem enega od krajnih bitov (LSB ali MSB) in premikom na drugi konec

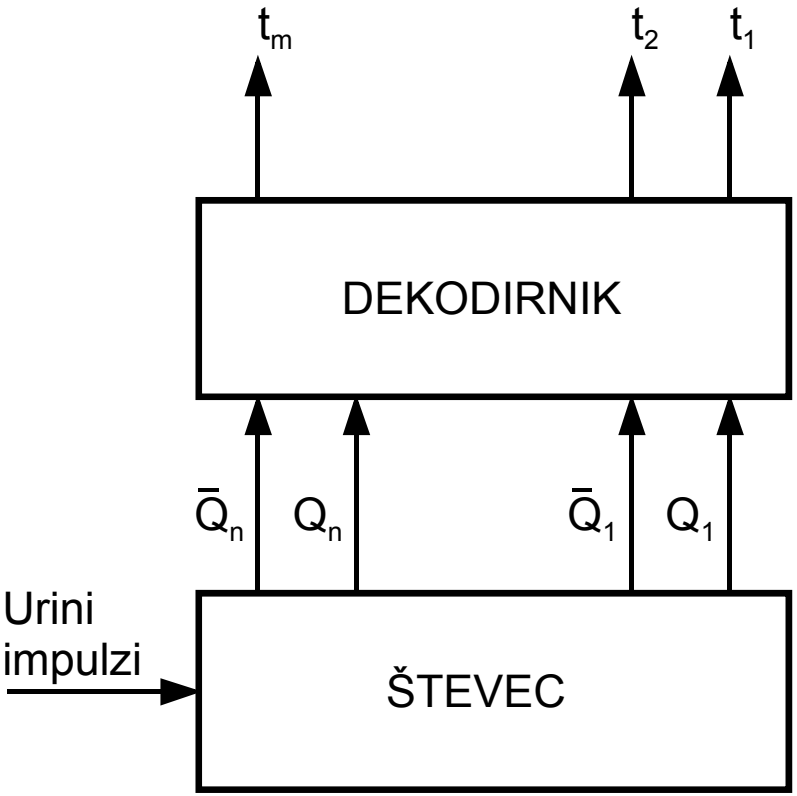
Časovni diagram števca s prepletanjem

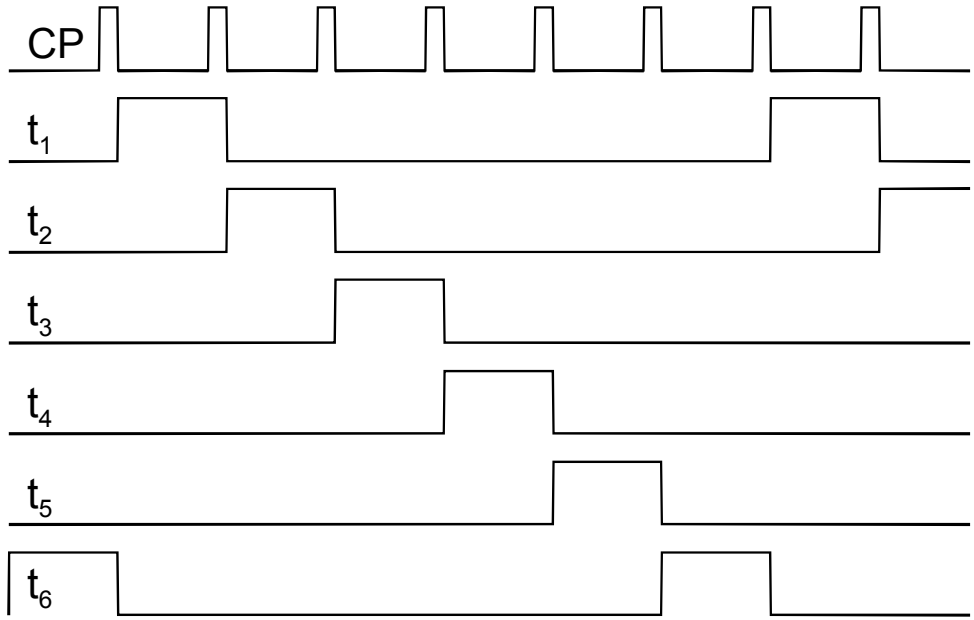


Če uporabimo 3/8 dekodirnik lahko dobimo šest ločenih časovnih impulzov.

Za to vrsto vezij uporabljamo skupno ime “števno dekodirna vezja”.

9. 4. 6 Števno dekodirna vezja





9. 5 Registri

Binarno informacijo shranjujemo v urejenih skupinah spominskih celic, ki jim pravimo registri.

Skupina "n" spominskih celic (pravimo ji n-bitni register) je torej zmožna shraniti "n" bitov.
Operacije, ki jih izvaja nek register (vpis, brisanje, pomik), določajo zunanji kontrolni signali.

Po obsežnosti vezja, ki tvori tak register, so to vezja srednje stopnje integracije (MSI).

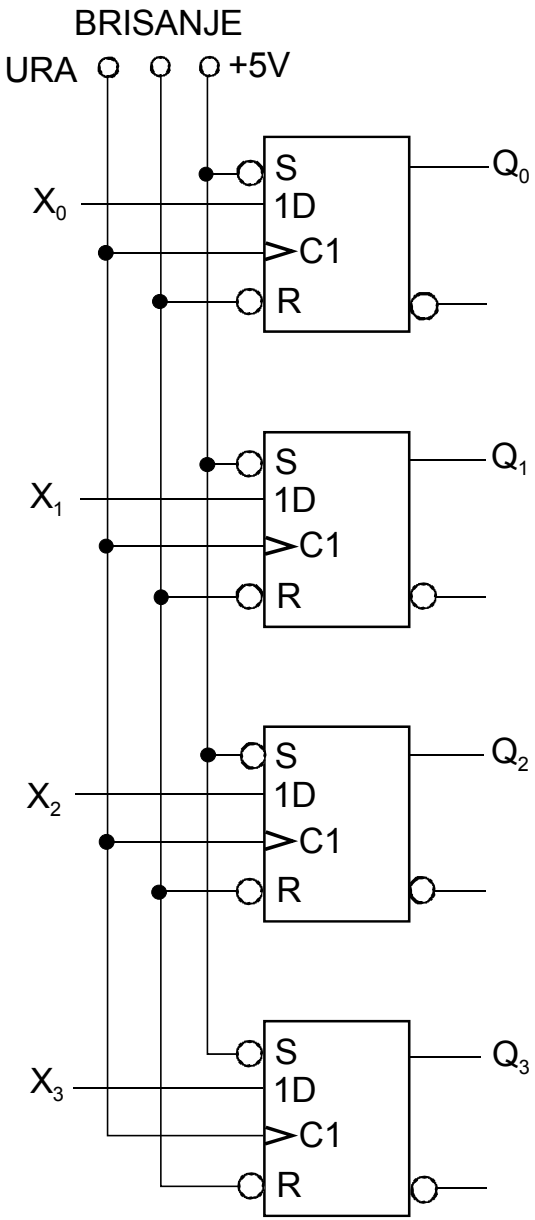
9. 5. 1 Pomnilniški registri

$$Q_i = \overline{R}(\Delta^{-1}x_i), \quad \text{oziroma} \quad \Delta^{1}Q_i = \overline{R}x_i; \quad i = 1,2,...,n$$

Lahko pa uporabimo kombinacijo z R-S spominsko celico:

$$Q_i = \overline{R}(x_i + \Delta^{-1}Q_i); \qquad i = 1,2,...,n$$

Zgled:



9. 5. 2 Univerzalni premikalni register

- paralelni zapis
- serijski premik desno in
- serijski pomik levo

a) P a r a l e l n i z a p i s

Če je $P = 1$, $S_R = 0$, $S_L = 0$

se podatki vhodov x_0–x_3 prenesejo v A_0–A_3

b) P o m i k d e s n o

Če postavimo $S_R = 1$, $P = 0$, $S_L = 0$,

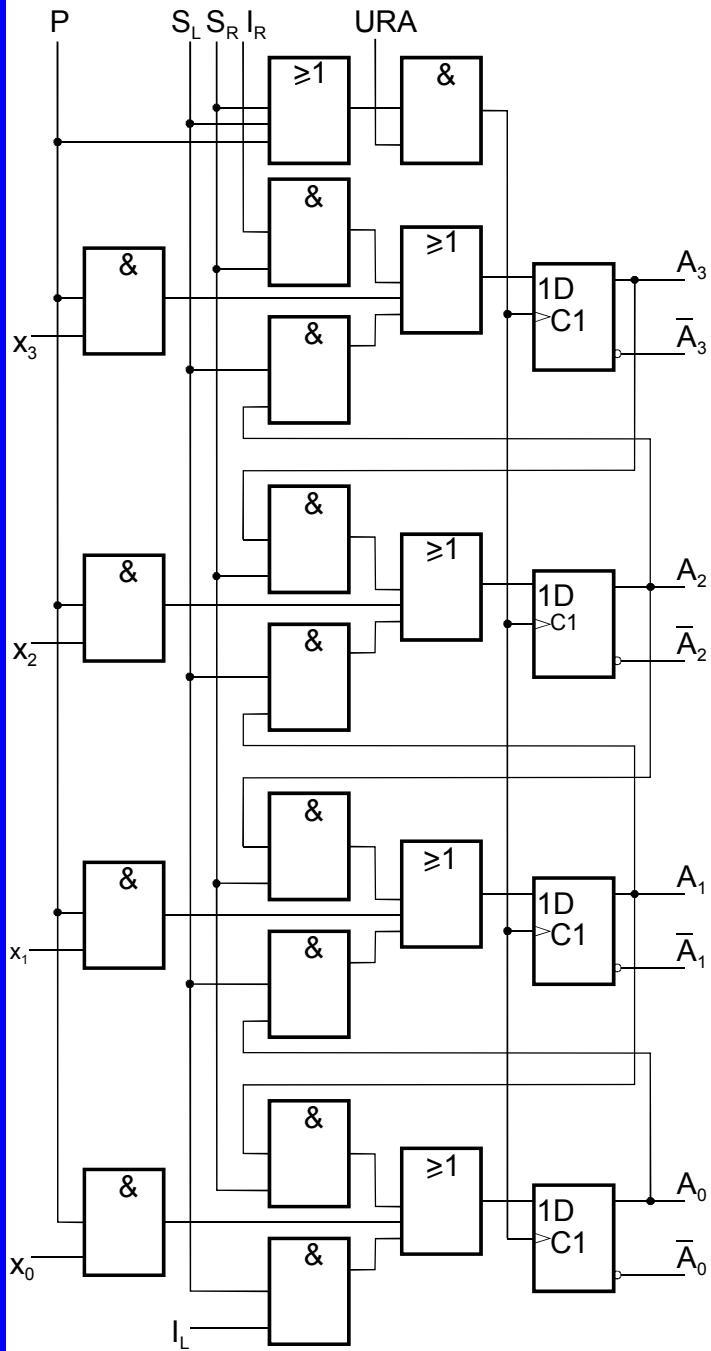
se podatki v registru pomikajo proti desni

c) P o m i k l e v o

Če je $S_L = 1$, $P = 0$, $S_R = 0$,

gredo podatki proti levi.

Simbolični diagram:



$$D_{A_0} = I_L S_L + x_0 P + A_1 S_R$$

$$D_{A_1} = A_0 S_L + x_1 P + A_2 S_R$$

$$D_{A_2} = A_1 S_L + x_2 P + A_3 S_R$$

$$D_{A_3} = A_2 S_L + x_3 P + I_R S_R$$

Tabela prehajanja stanj univerzalnega registra pri izvajanju operacije pomika v levo:

$S_L=1$, $S_R=0$, $P=0$

Predpostavka: vhod I_L se v času izvajanja operacije ne spreminja

Sedanje stanje				Naslednje stanje $I_L=1$				Naslednje stanje $I_L=0$			
A_3	A_2	A_1	A_0	A_3	A_2	A_1	A_0	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0	1	0	0
0	0	1	1	0	1	1	1	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0	0
0	1	0	1	1	0	1	1	1	0	1	0
0	1	1	0	1	1	0	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1	0	0	0	0
1	0	0	1	0	0	1	1	0	0	1	0
1	0	1	0	0	1	0	1	0	1	0	0
1	0	1	1	0	1	1	1	0	1	1	0
1	1	0	0	1	0	0	1	1	0	0	0
1	1	0	1	1	0	1	1	1	0	1	0
1	1	1	0	1	1	0	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	1	0

Števila bitov takšnih registrov so v praksi zelo različna.

Poglejmo vhodne funkcije poljubnega »n« - bitnega univerzalnega registra.

Na “i”- tem mestu več bitnega registra bo imel vhod D naslednjo vhodno funkcijo:

$$D_i = Px_i + S_R Q_{i+1} + S_L Q_{i-1}; \quad PS_R S_L = 0$$

Robni funkciji pa sta naslednji:

$$D_{n-1} = Px_{n-1} + S_R I_R + S_L Q_{n-2}; \quad PS_R S_L = 0$$

$$D_0 = Px_0 + S_R Q_1 + S_L I_L; \qquad PS_R S_L = 0$$

Zato lahko zapišemo enačbo “i” - tega izhoda v naslednjih oblikah:

$$\Delta^1 Q_i = Px_i + S_R Q_{i+1} + S_L Q_{i-1}$$

oziroma za sedanje stanje:

$$Q_i = \Delta^{-1}(Px_i) + \Delta^{-1}(S_R Q_{i+1}) + \Delta^{-1}(S_L Q_{i-1})$$

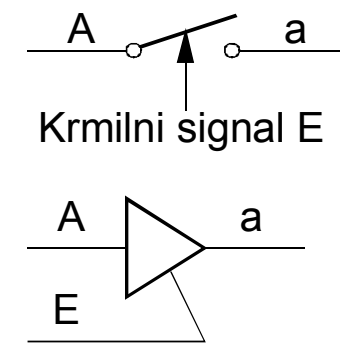
Krmiljenje premika v tem registru zahteva:

Paralelni zapis: P=1, S_R=0, S_L=0

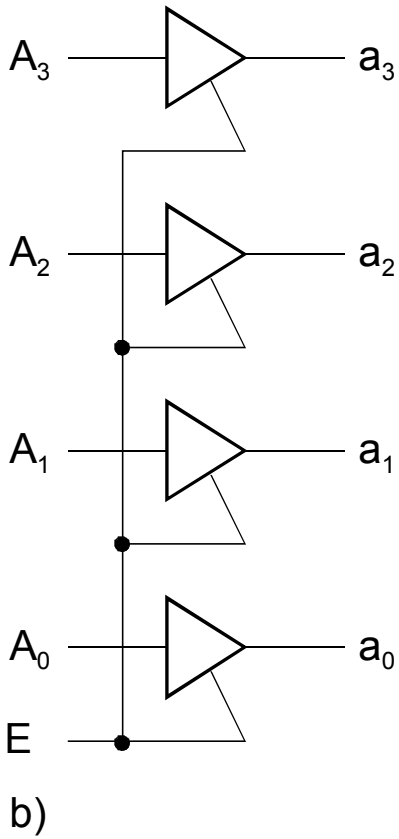
Pomik v desno: S_R=1, S_L==0, P=0

Pomik v levo: S_L=1, S_R=0, P=0

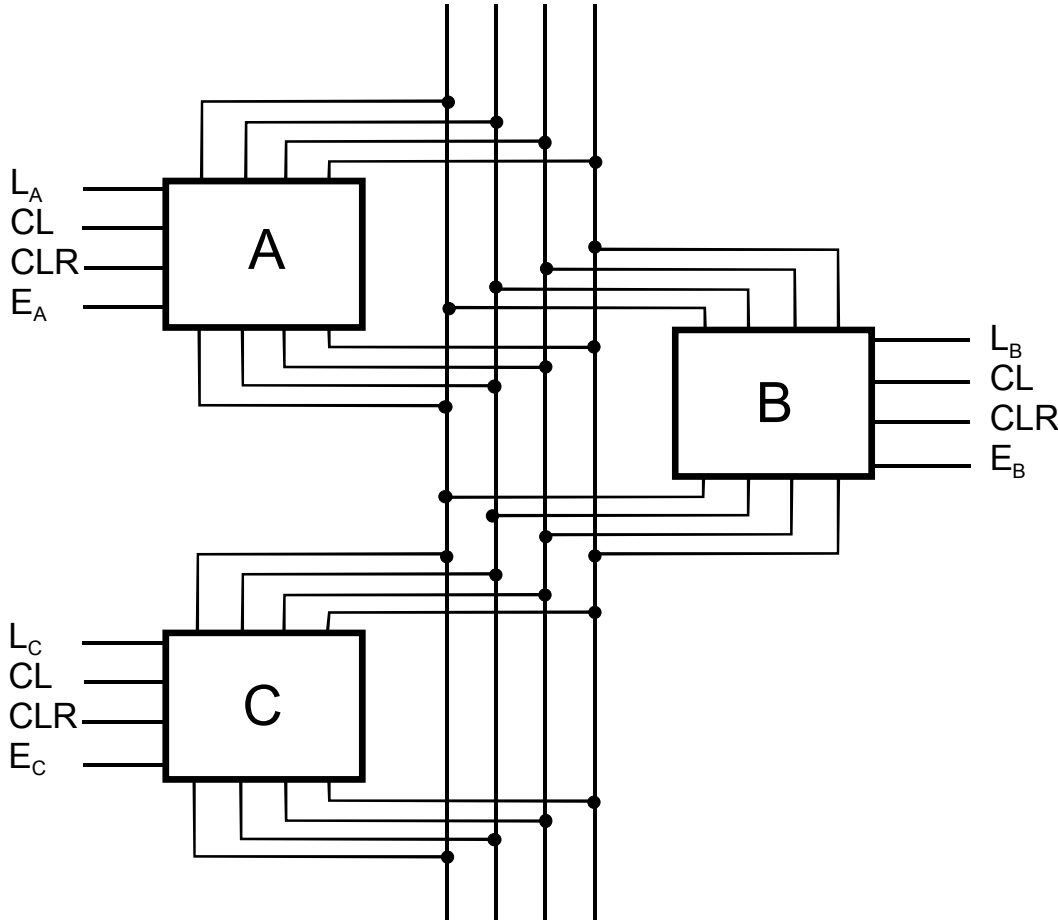
9. 5. 3 Stikalo treh stanj in povezovanje registrov



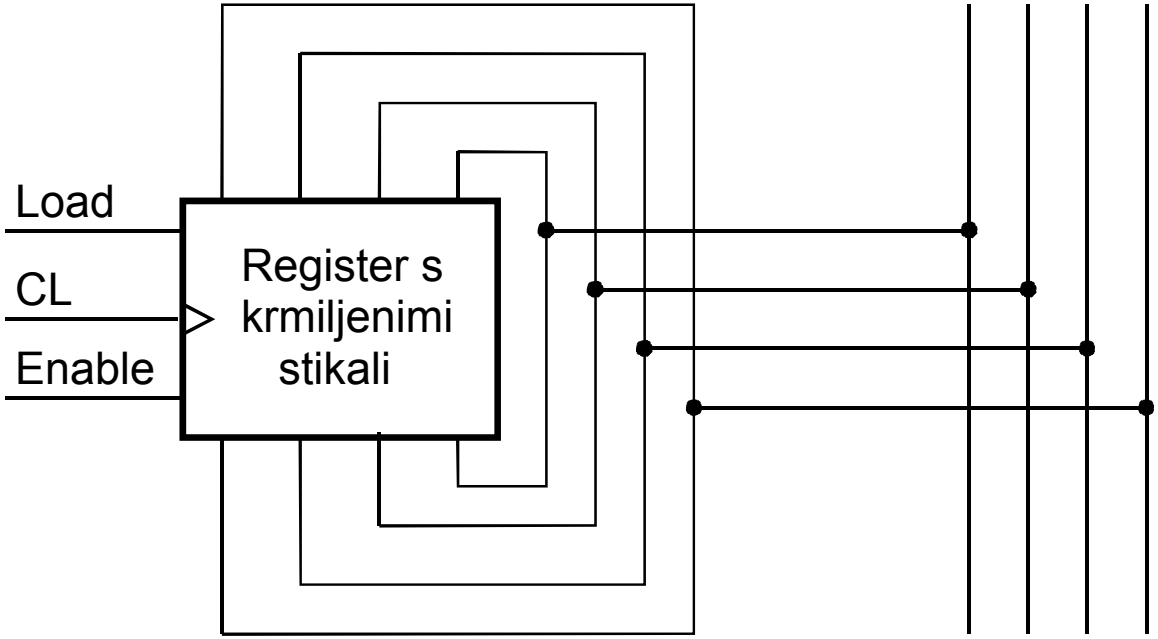
a)

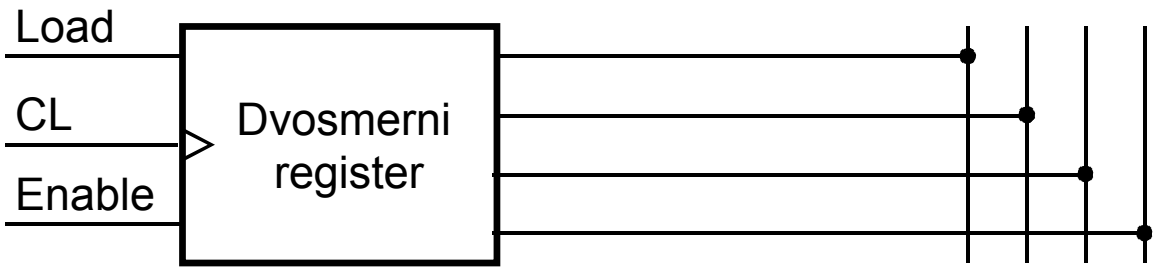


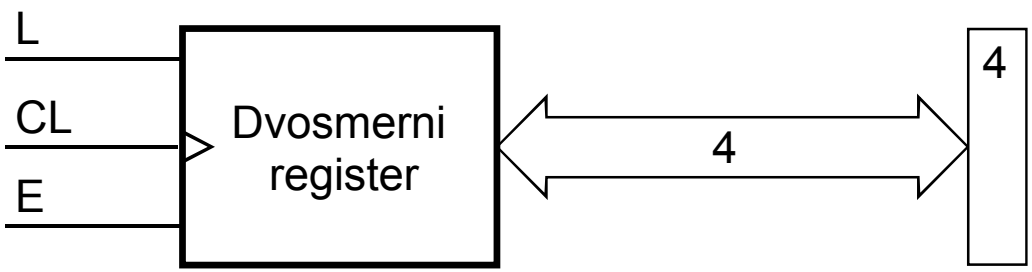
b)



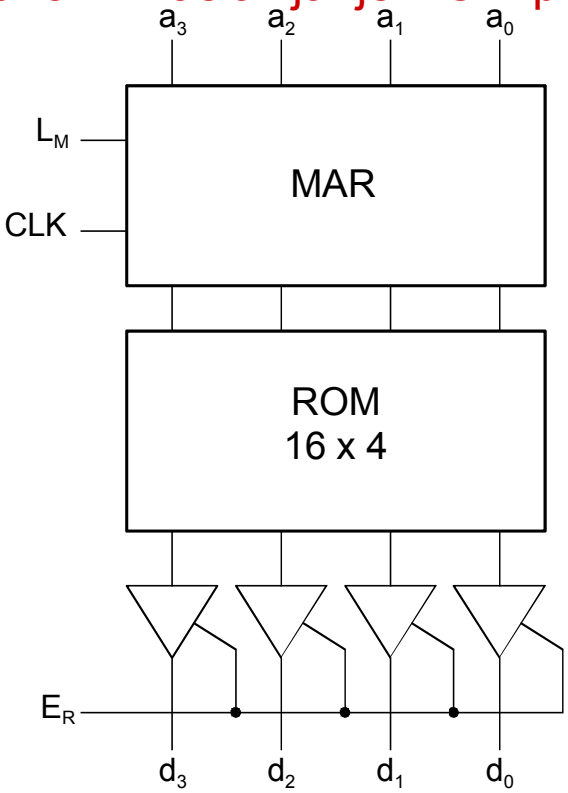
Dvosmerni registri:







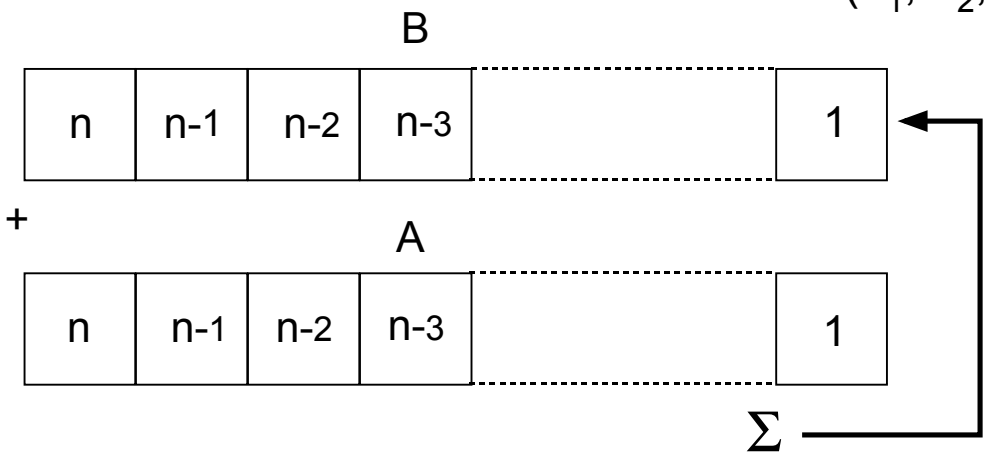
9. 5. 4 Naslavljanje ROM pomnilnika preko adresnega registra



9. 6 Sekvenčna aritmetična vezja

9. 6. 1 Seštevalniki

$A = (a_1, a_2, a_3, \ldots\ldots, a_n)$; $B = (b_1, b_2, b_3, \ldots\ldots b_n)$



Iščemo $a_i \oplus b_i \oplus c_i$

Pravilnostna tabela

a_i	b_i	c_i	$\Delta^1 q_i$	c_{i+1}
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

$$\Delta^1 q_i = \Delta^1 b_i$$

$$\Delta^1 b_i = \overline{a_i}\,\overline{b_i} c_i + \overline{a_i} b_i \overline{c_i} + a_i \overline{b_i}\,\overline{c_i} + a_i b_i c_i$$

$$c_i = a_{i-1} b_{i-1} + a_{i-1} c_{i-1} + b_{i-1} c_{i-1}$$

$$c_1 = 0$$

Primerjava $\Delta^1 b_i$ s splošno pomnilno enačbo nam da:

$$g_{i1} = \overline{a_i}\,\overline{c_i} + a_i c_i = a_i \equiv c_i$$

$$g_{i2} = \overline{a_i} c_i + a_i \overline{c_i} = a_i \oplus c_i$$

Za osnovno pomnilno celico izberimo D celico.

$$\Delta^1 q_i = g_{i1} q_i + g_{i2} \overline{q_i}$$

$$D_i = (\overline{a_i \oplus c_i}) b_i + (a_i \oplus c_i) \overline{b_i}$$

$$c_1 = 0$$

$$D_1 = \overline{a_1} b_1 + a_1 \overline{b_1}$$

$$c_2 = a_1 b_1$$

$$D_2 = (\overline{a_2 \oplus a_1 b_1}) b_2 + (a_2 \oplus a_1 b_1) \overline{b_2}$$

$$c_3 = a_2 b_2 + (a_2 + b_2) a_1 b_1$$

$$D_3 = \overline{\{a_3 \oplus [a_2 b_2 + (a_2 + b_2) a_1 b_1]\}} b_3 + \{a_3 \oplus [a_2 b_2 + (a_2 + b_2) a_1 b_1]\} \overline{b_3}$$

9. 6. 2 Primerjalniki

Primer serijske primerjave vsebine registra

	A= a_n, a_{n-1},.....a_0 B= b_n, b_{n-1},.....b_0
Vsebina A večja	√ √
Vsebina B večja	√
Vsebini enaki	√

Vsebini se pomikata tako, da se na mestu "n" najprej pojavi najmanj pomemben bit in po n korakih najbolj pomemben bit.

Primerjavo izvajamo na mestu "n", rezultat pa dobimo na mestu n+1.

Za izvajanje serijske primerjave potrebujemo tri spremenljivke a_n, b_n in P.
P je posebno mesto s pomenom:

P = 0 – izvajanje primerjave

P = 1 – podajanje rezultata na mestu n+1.

q_a	q_b	Pomen	f_1	f_2
0	0	A = B	0	0
0	1	A < B	0	1
1	0	A > B	1	0
1	1	X	1	1

Funkciji odvzamemo, ko je P=1

Začetno stanje q_a, q_b naj bo: $q_a = q_b = 0$

q_a	q_b	P	a_n	b_n	$\Delta^1 q_a$	$\Delta^1 q_b$	f_1	f_2
0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	1	1	1
0	0	0	1	0	1	0	1	1
0	0	0	1	1	0	0	1	1
0	0	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	1

q_a	q_b	P	a_n	b_n	$\Delta^1 q_a$	$\Delta^1 q_b$	f_1	f_2
0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	1	1	1
0	0	0	1	0	1	0	1	1
0	0	0	1	1	0	0	1	1
0	0	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	1
0	0	1	1	0	0	0	1	0
0	0	1	1	1	0	0	0	0
0	1	0	0	0	0	1	1	1

Ko je P = 0 sta funkciji f_1 in f_2 vedno 1.

$$\Delta^1 q_a = q_a \overline{P}(a_n + \overline{b}) + \overline{q}_a \overline{P} a_n \overline{b_n}$$

$$\Delta^1 q_b = q_b \overline{P}(\overline{a_n} + b_n) + \overline{q_b} \overline{P} \overline{a_n} b_n$$

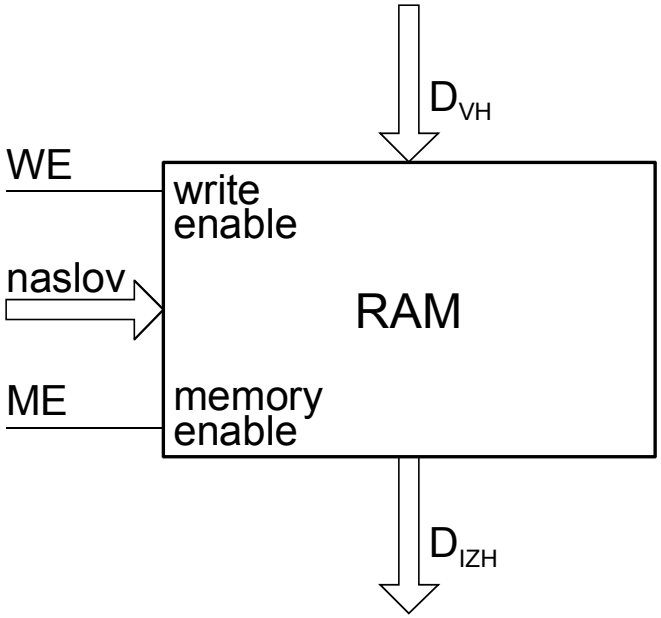
$$f_1 = \overline{P} + a_n \overline{b_n} + q_a \overline{b_n} + q_a a_n$$

$$f_2 = \overline{P} + \overline{a_n} b_n + q_b b_n + q_b \overline{a_n}$$

9. 7. Pomnilniška sekvenčna vezja

9. 7. 1 Pomnilnik z naključnim dostopom – RAM pomnilnik

Glavna značilnost RAM-a je, da lahko vanj električno vpišemo binarne besede na ustrezne naslove.



ME	WE	Operacija	Izhod	
0	0	zadrži	lebdeč	
0	1	zadrži	lebdeč	
1	0	beri	zvezan	naslovljena beseda na izhodu
1	1	piši	lebdeč	naslovljena beseda zapisana

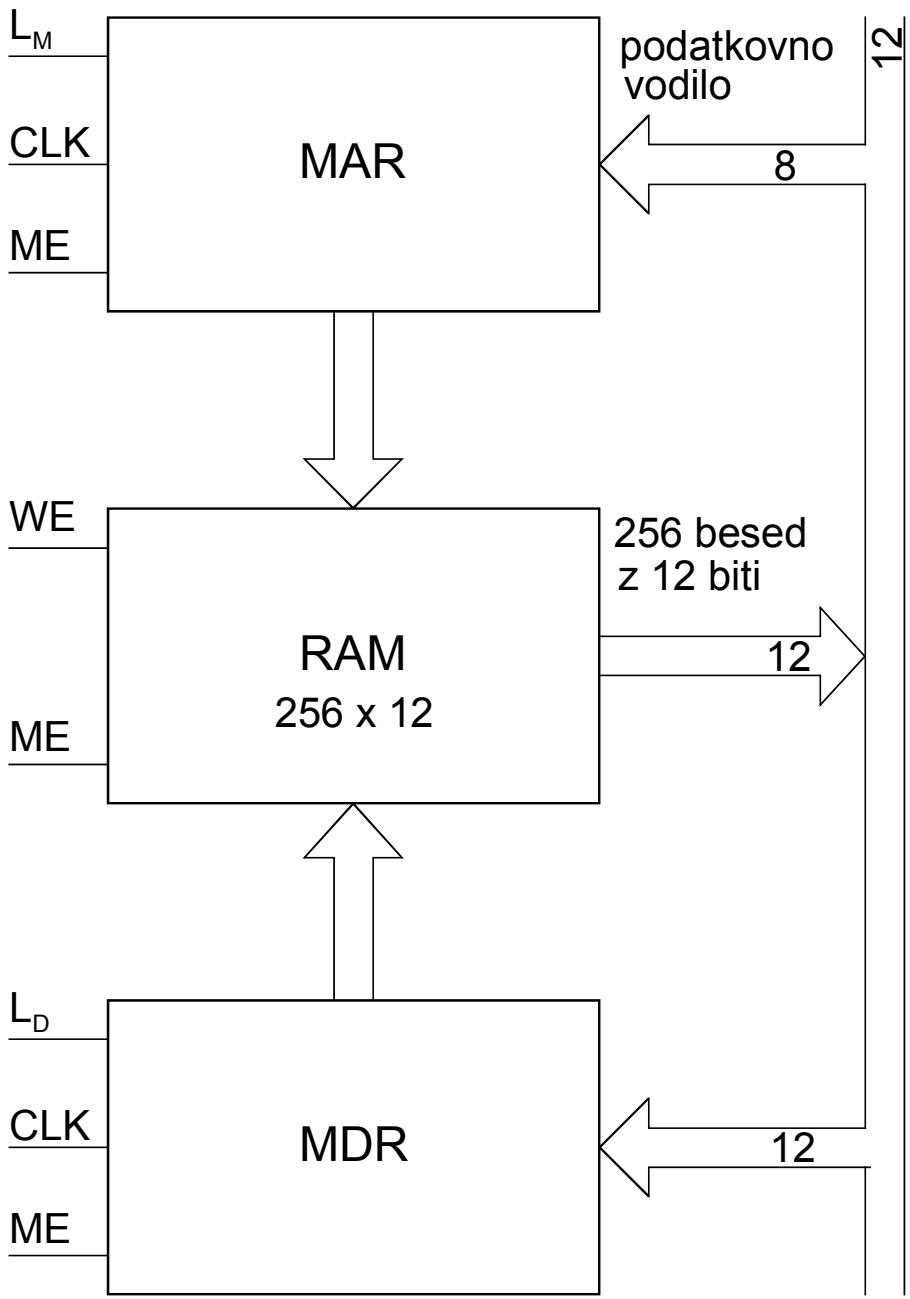
Spremljajoči registri:

- spominsko adresni register MAR
- podatkovni register MDR

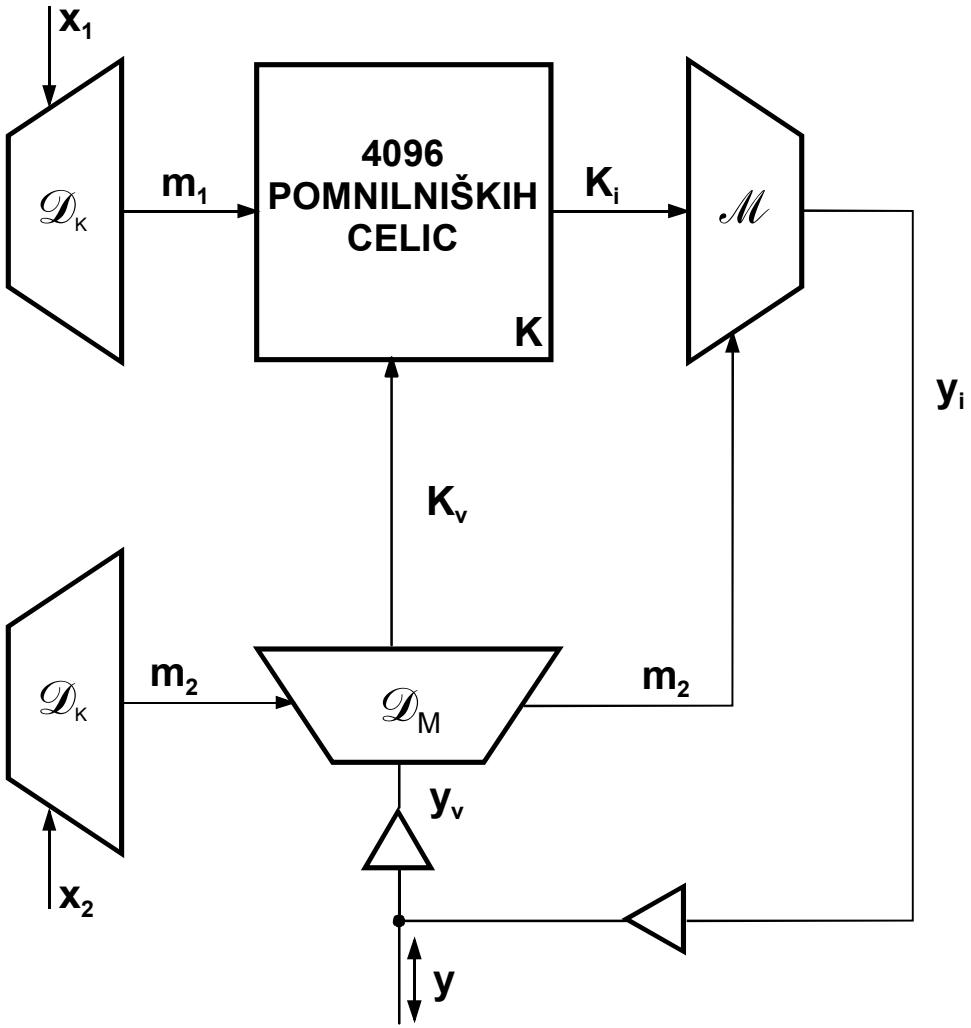
Register MDR je potreben zato, da prepreči težave pri prehodnih pojavih ob vpisovanju podatkovnih besed.

Tako je npr. MDR napolnjen, še preden postane WE=1, prav tako pa ostane podatkovna beseda še v MDR, ko pade WE ponovno na nič.

Polnjenje MDR je možno le, če je $L_D = 1$.



Za podrobnejše razumevanje funkcije tega pomnilnika si oglejmo še njegove osnovne gradnike.



Vpis:

$$\mathbf{K}_V = \mathbf{m}_2^T \Sigma \,\&\, \mathbf{y}_V = \mathbf{m}_1^T \Sigma \,\&\, \mathbf{y}$$

Branje pomnilnika:

$$\mathbf{y}_i = \mathbf{m}_2 \Sigma \,\&\, \mathbf{K}_i = \mathbf{m}_2 \Sigma \,\&\, (\mathbf{m}_1^T \Sigma \,\&\, \mathbf{y})$$

Pri tem je:

y – podatkovni vektor v pomnilniku

$\mathbf{y}_i$ – izhodni podatkovni vektor

$\mathbf{y}_v$ – vhodni podatkovni vektor

$\mathbf{K}_v$ – vhodna matrika

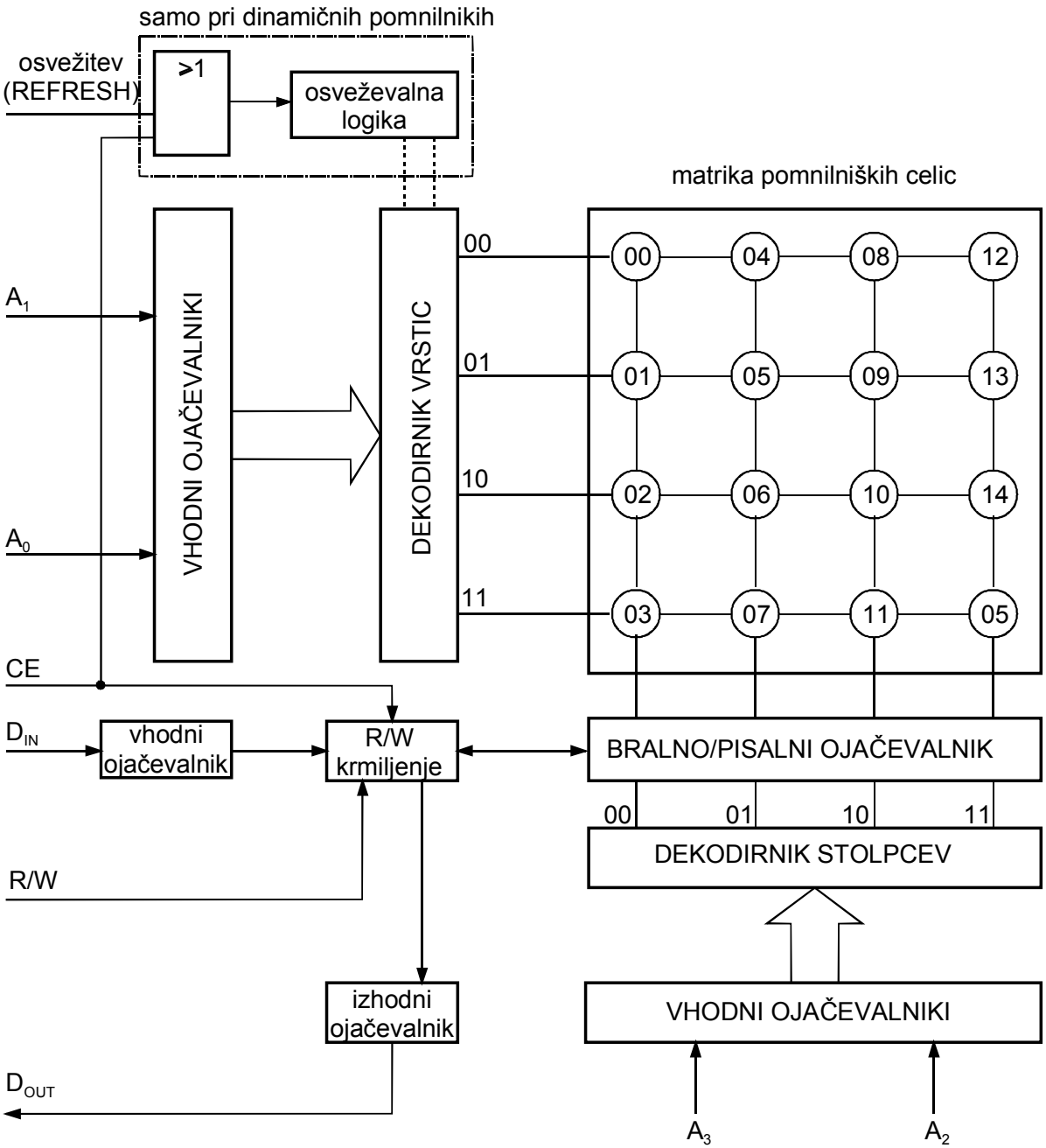
$\mathbf{K}_i$ – izhodna matrika

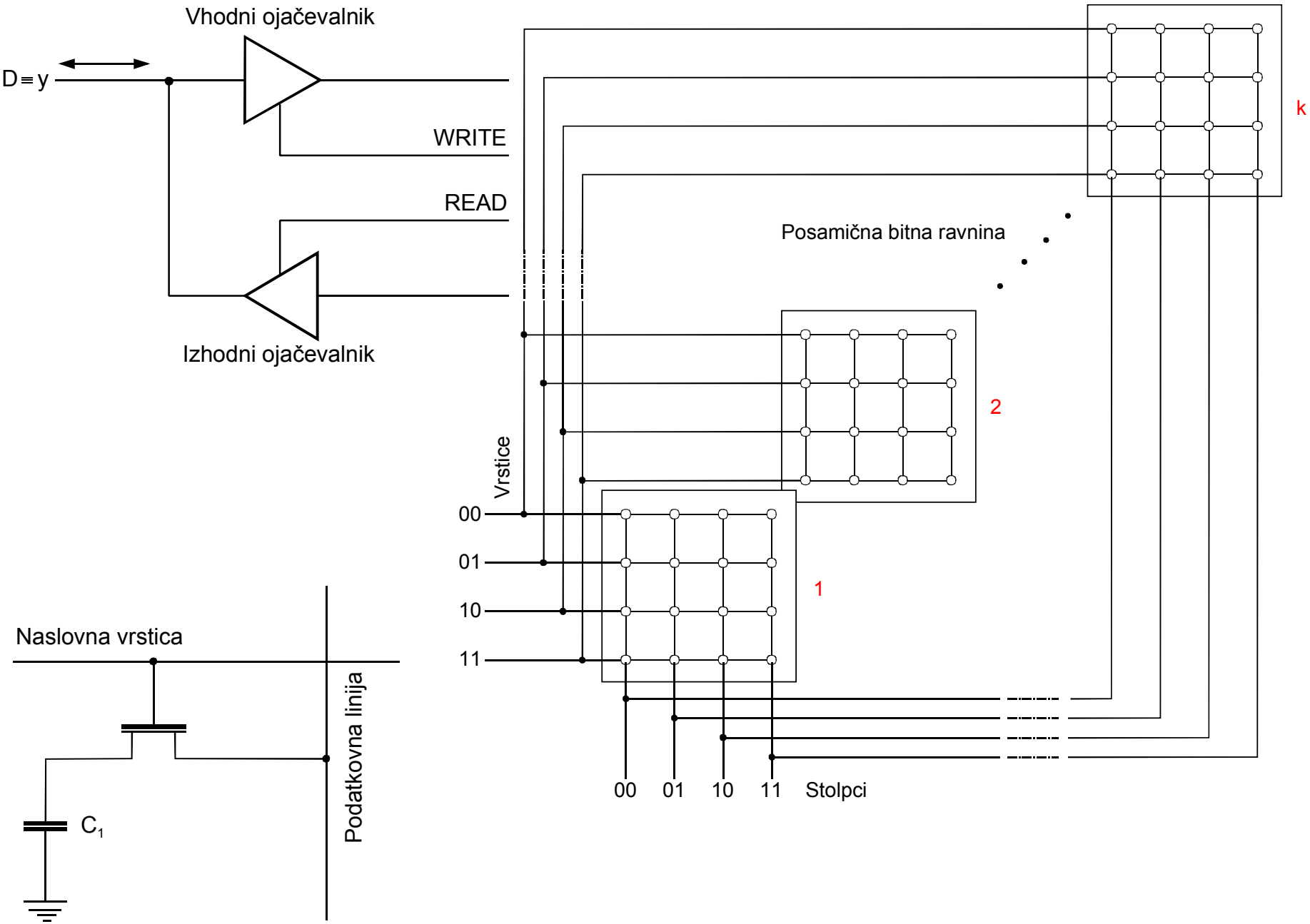
$\mathbf{x} = [\mathbf{x}_1 , \mathbf{x}_2]$;

$\mathbf{m} = [\mathbf{m}_1 , \mathbf{m}_2]$

$$\mathbf{K} = \mathbf{m}^T \Sigma \,\&\, \mathbf{y}$$

$$\mathbf{y} = \mathbf{m} \Sigma \,\&\, \mathbf{K}$$


samo pri dinamičnih pomnilnikih
osvežitev
(REFRESH)
>1
osveževalna
logika
matrika pomnilniških celic
VHODNI OJAČEVALNIKI
DEKODIRNIK VRSTIC
A1
A0
00
01
10
11
00 04 08 12
01 05 09 13
02 06 10 14
03 07 11 05
CE
DIN
vhodni
ojačevalnik
R/W
krmiljenje
BRALNO/PISALNI OJAČEVALNIK
00 01 10 11
DEKODIRNIK STOLPCEV
R/W
izhodni
ojačevalnik
VHODNI OJAČEVALNIKI
DOUT
A3
A2


Vhodni ojačevalnik
D ≡ y
WRITE
READ
Izhodni ojačevalnik
Posamična bitna ravnina
k
2
1
Vrstice
00
01
10
11
00 01 10 11 Stolpci
Naslovna vrstica
Podatkovna linija
C1

9. 7. 2 Asociativni pomnilnik – CAM (Content Addressable Memory)

CAM pomnilnik je razširjena verzija RAM pomnilnika.

Naloga teh pomnilnikov je pregledovanje vsebin podatkov v bazi, ki zadoščajo postavljenemu kriteriju.

Izvajanje takšne operacije je sočasno na vseh lokacijah pomnilnika.

Da lahko iskalni niz primerjamo z vsemi besedami pomnilnika potrebujemo naslednje operacije:

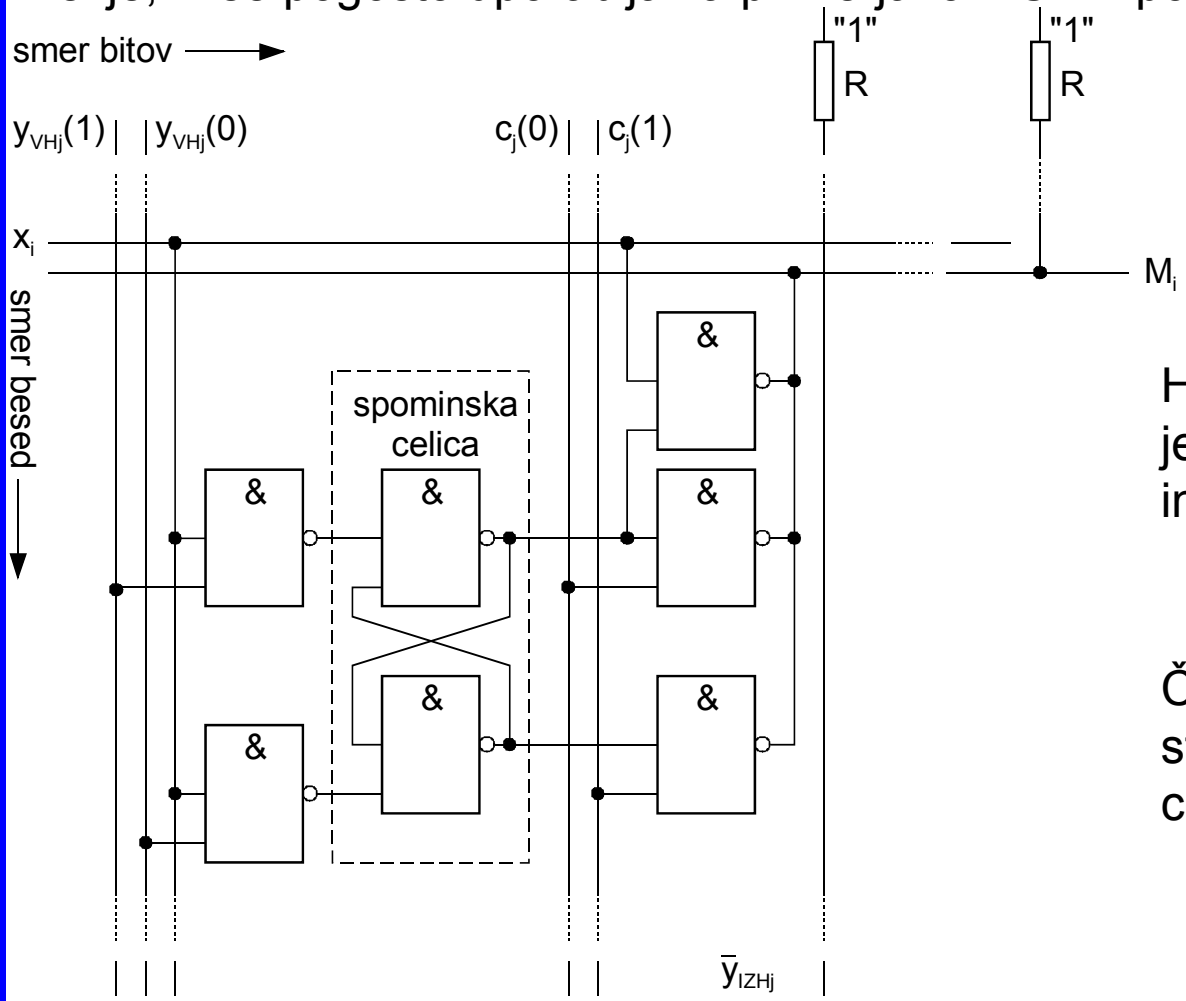
1. pošiljanje iskalnega niza na vse lokacije v pomnilniku
2. posamično primerjavo iskalnega niza z vsako shranjeno besedo v pomnilniku

3 zbiranje informacije katere besede ustrezajo kriteriju v iskalnem nizu.

V elektronski izvedbi je edina učinkovita metoda za pošiljanje iskalnega niza uporaba paralelnih signalnih linij, na katere so priključene vse spominske lokacije.

Signali teh linij se primerjajo s stanjem pomnilnika v posebnem vgrajenem vezju.

Vezje, ki se pogosto uporablja za primerjavo v CAM pomnilnikih je prikazano na sliki:



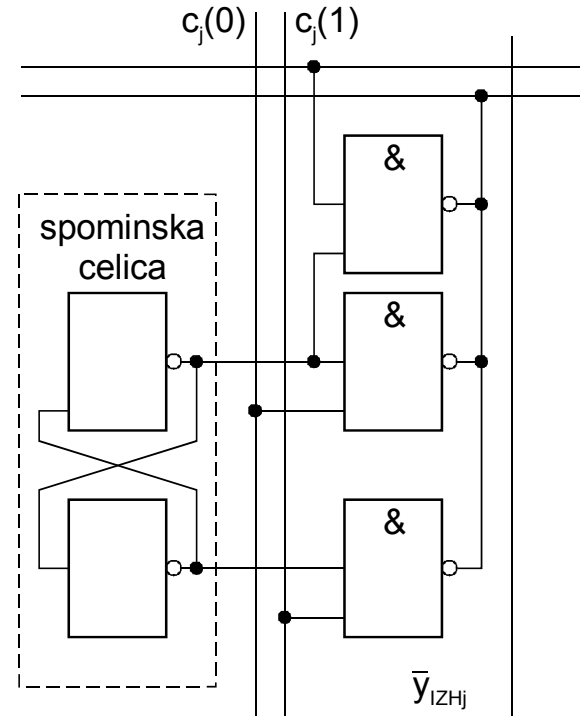
Horizontalna linija s signalom x_i je naslovna linija za vpisovanje in branje podatkov.

Če je x_i=1, potem y_{IZHj} podaja stanje naslovljene spominske celice (njeno negirano vrednost)

Dodatna prednost oziroma zmožnost tega vezja je ta, da lahko realiziramo OR funkcijo velikega števila spremenljivk (več tisoč).

Paralelna vezava izhodov na liniji M in y_{IZH} daje že znano operacijo »WIRED OR«.

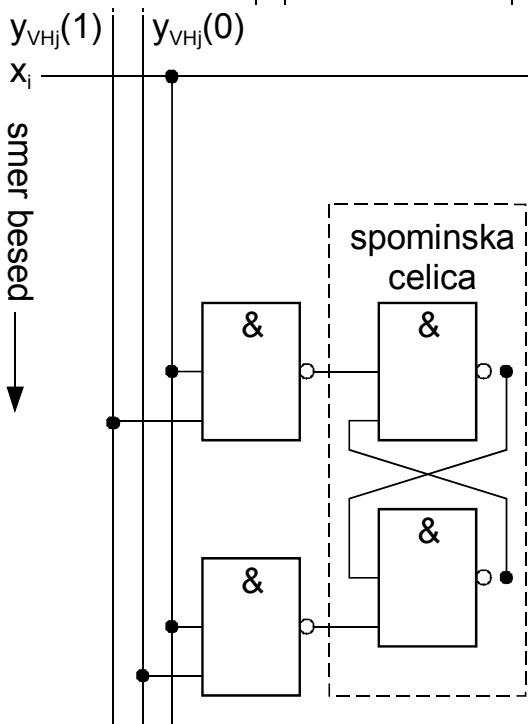
Dvojne krmilne linije $e_j(0)$ in $e_j(1)$ omogočajo iskanje in maskiranje:



Če želimo ugotoviti ali je na tem mestu v pomnilniku »0« damo signalu $e_j(0)$ vrednost 1, signalu $e_j(1)$ pa vrednost 0

Ko želimo mesto maskirati morata biti $e_j(0)=0$ in $e_j(1)=0$.

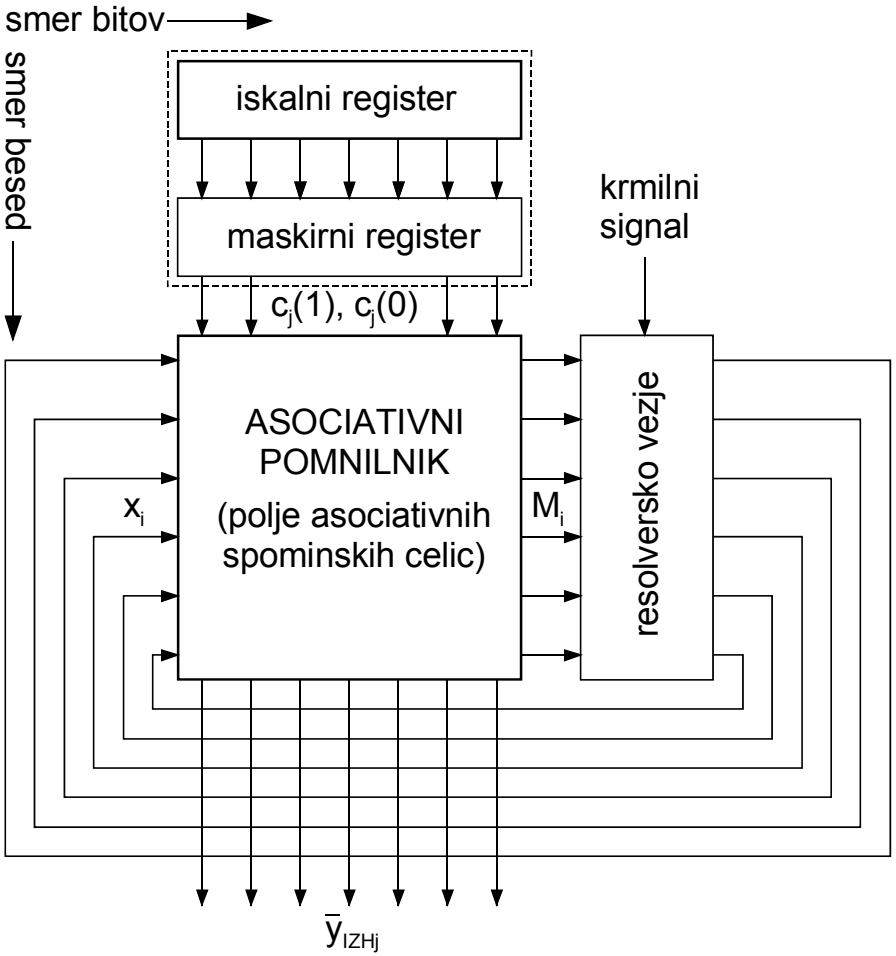
Rezultat teh operacij je $M_i = 1$ če in samo če se iskane vrednosti nahajajo v spominski celici.



Za vpis v spominsko celico služijo linije y_{VH}, ki so skupne vsem besedam v spominu.

Beseda spomina je izbrana z adresno linijo in če želimo vpisati enico mora biti $y_{VHj}(0) = 0$ in $y_{VHj}(1) = 1$

In obratno za vpis ničle mora biti $y_{VHj}(0) = 1$ in $y_{VHj}(1) = 0$.



Centralni blok predstavlja polje asociativnih spominskih celic, kakršne so bile predstavljene.

Iskalni niz je shranjen v iskalnem registru, maskirna beseda pa v maskirnem registru.

Vrednosti signalov $e_j(0)$ in $e_j(1)$ se generirajo iz delov vsebin v iskalnem in maskirnem registru.

Resolversko vezje poskrbi za to, da se v časovnem sosledju na njegovem izhodu pojavijo vse besede, ki ustrezajo danemu kriteriju.

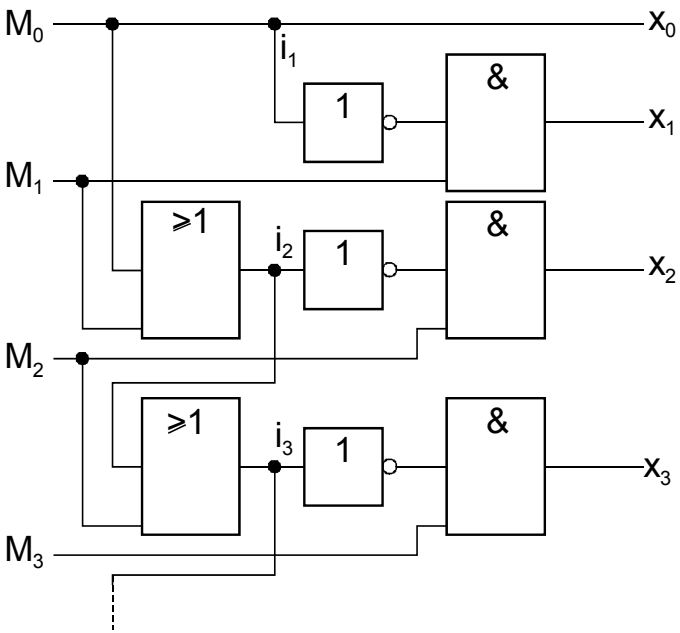
To vezje ima enako število vhodov in izhodov

S pomočjo krmilne logike je mogoče izbrati eno od aktivnih linij in jo posredovati izhodu.

Takšna operacija se potem krožno ponavlja za vse aktivne linije.

Izhod resolverja je hkrati naslov x_i CAM pomnilnika in vsebina i-te besede se pojavi na izhodu pomnilnika v obliki komplementa: $\bar{\mathbf{y}}_{IZH} = (\bar{y}_{IZH1},\ldots,\bar{y}_{IZHn})$

Prioritetna logika resolverja



M_i signali, ki ustrezajo posameznim izbranim besedam, so izhodi zunanjih bistabilnih vezij, ki so imela vpisano enico preko vhodnih M linij .

Osnovni princip delovanja je zadržati – inhibirati vse nižje ležeče signale, tako da je sočasno aktiven samo en izhod.

Ko je odgovarjajoča beseda prebrana se bistabilno vezje resetira na«0« nakar se lahko pojavi novi izhodni signal nekje na nižjem mestu, kjer je prva enica.

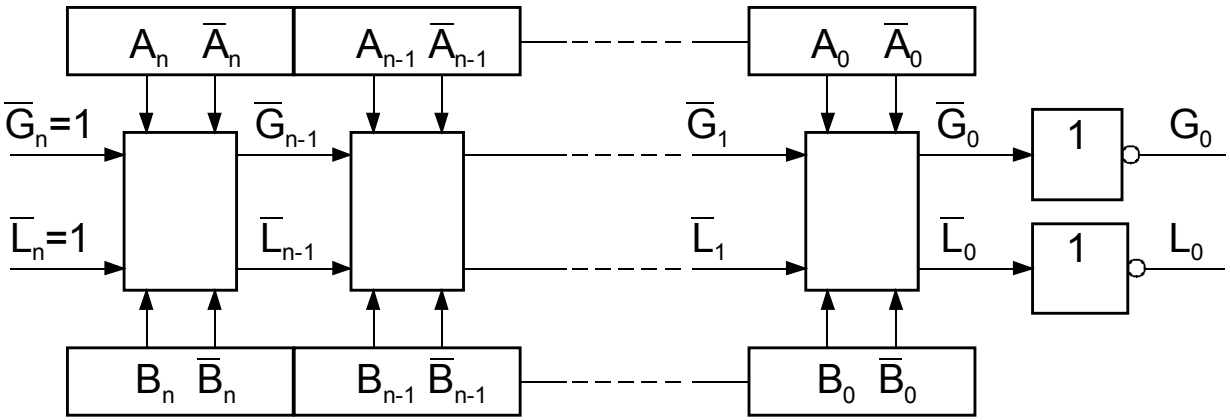
Pomanjkljivost: generiranje signalov v kaskadi.

10ns zakasnitev posamezne operacije

Za samo 1000 besed potrebuje ta pomnilnik čas 10 mikrosekund, da doseže stacionarno stanje.

Paralelna primerjava vrednosti

Pomembna operacija v teh pomnilnikih je iskanje besed, katerih numerična vrednost je ali večja ali manjša od vrednosti, ki jo ima iskalni niz



Register A vsebuje iskalni niz; zaradi preprostosti in s tem boljše preglednosti, ni vključen tudi maskirni register.

Register B naj predstavlja na primer eno od lokacij v spominu.

Med vsakim parom bitov (A_i, B_i) se nahaja identično iterativno logično vezje, ki posreduje rezultate primerjave svojemu sosedu vzdolž G in L linij.

Iterativna primerjava

Privzemimo hipotetično, da je proces primerjave startal skrajno levo v obeh registrih (numerično pri bitu z najvišjo vrednostjo) in se nato nadaljeval do i-tega mesta.

Če lahko tvorimo takšna signala G_i in L_i ki bosta enolično ločila med tremi možnostmi:

1. A > B
2. A < B
3. nobena izmed gornjih relacij ni potrjena

Potem lahko na osnovi vrednosti A_i, B_i, G_i in L_i logično določimo in posredujemo podobno informacijo (i - 1) mestu s signaloma G_{i-1} in L_{i-1}

Samo po sebi je namreč razumljivo, da če se dve števili ne ujemata na vseh mestih levo od i-tega je eno izmed njiju enolično večje od drugega; ne glede na to, kakšne vrednosti zavzamejo biti z nižjimi vrednostmi.

Preklopni funkcij, ki lahko opravita iterativno operacijo primerjanja sta:

$$G_{i-1} = G_i + (A_i \ \overline{B}_i \ \overline{L}i)$$

za vse i

$$L_{i-1} = L_i + (\ \overline{A}_i \ B_i \ \overline{G}_i)$$

z začetnima vrednostma $G_n = L_n = 0$.

Če je $A_n = 1$ in $B_n = 0$ je očitno, da je A večji od B.

Iz enačb: $G_{i-1} = G_i + (A_i \ \overline{B}_i \ \overline{L}i)$ in $L_{i-1} = L_i + (\ \overline{A}_i \ B_i \ \overline{G}_i)$ dobimo za vse $i < n$

$G_i = 1$ in $L_i = 0$.

Na drugi strani pa če je $A_n = 0$ in $B_n = 1$ je za vse $i < n$ $G_i = 0$ in $L_i = 1$.

Toda, če je $A_n = B_n$ je $G_{n-1} = L_{n-1} = 0$ in primerjava vrednosti se mora izvršiti na osnovi bitov z nižjo vrednostjo.

V tem primeru se enaka odločitev izvrši na bitih A_{n-1} in B_{n-1} in proces steče do skrajno desne pozicije, kjer se končno pojavita signala G_0 in L_0.

V primeru, da je $G_0 = 1$ in $L_0 = 0$, je A večji od B; če pa se pojavi obraten rezultat

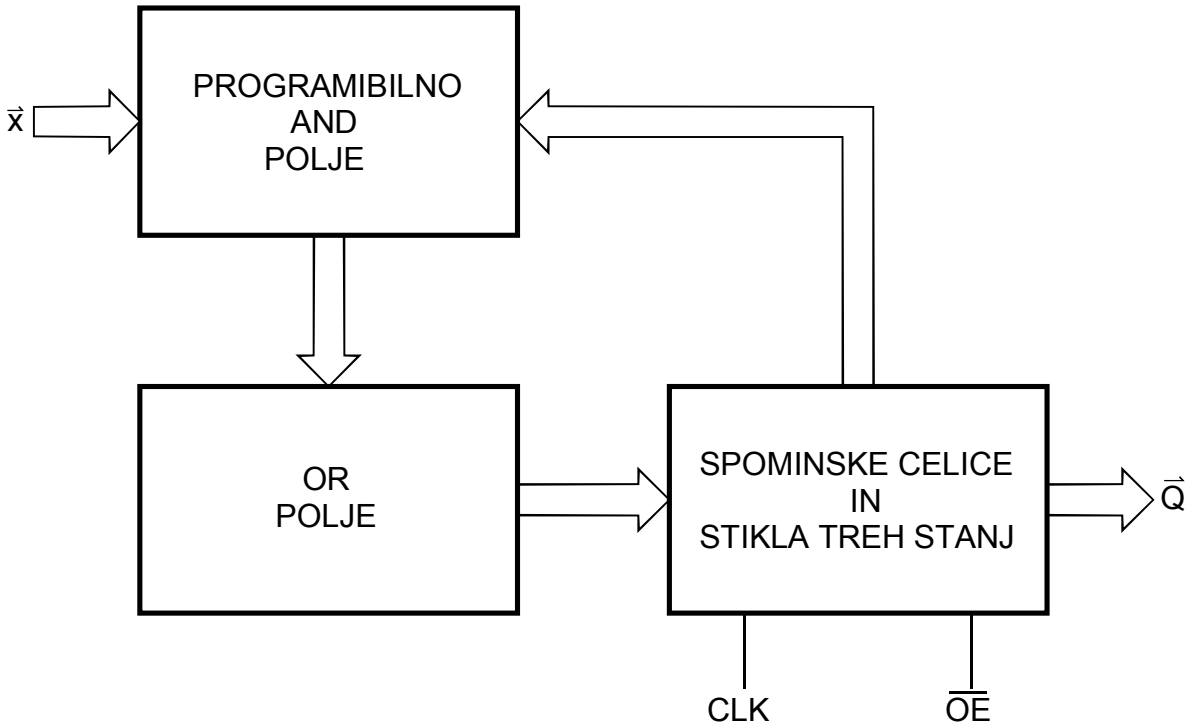
$G_0 = 0$ in $L_0 = 1$ je seveda B večji od A.

V primeru enakega rezultata $G_0 = L_0 = 0$ sta števili seveda enaki.

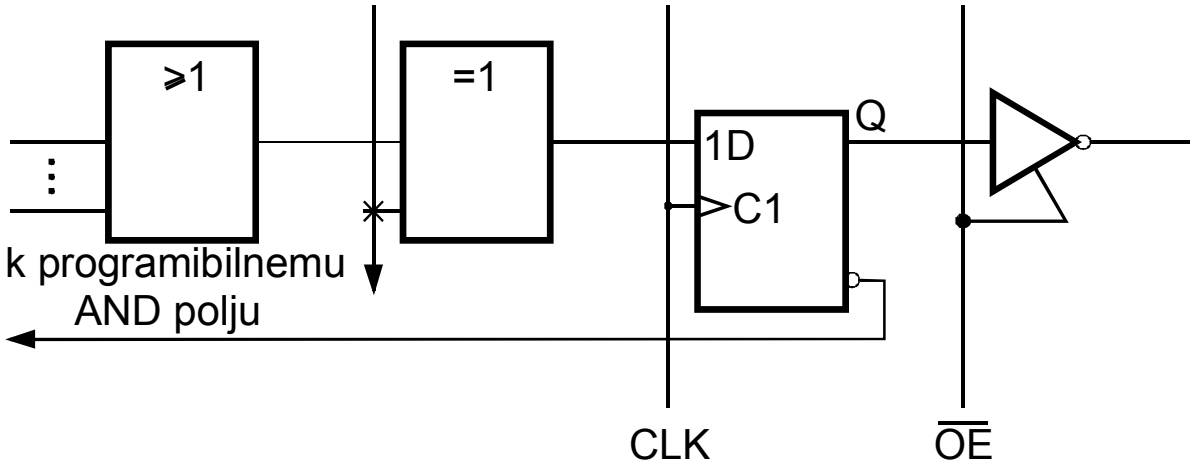
9. 8 Programibilna sekvenčna vezja

9. 8. 1 PAL arhitektura z dodatnimi spominskimi celicami

- programibilni vhodno/izhodni priključki
- programibilna polariteta izhodnih signalov
- vgrajeni XOR elementi med AND-OR poljem in spominskimi celicami.



Programiranje izhodne polaritete



programibilni vhod je normalno priključen na maso, zato XOR vezje deluje le kot bafer.

Izhodna polariteta je zaradi invertiranja v stikalu treh stanj obrnjena.

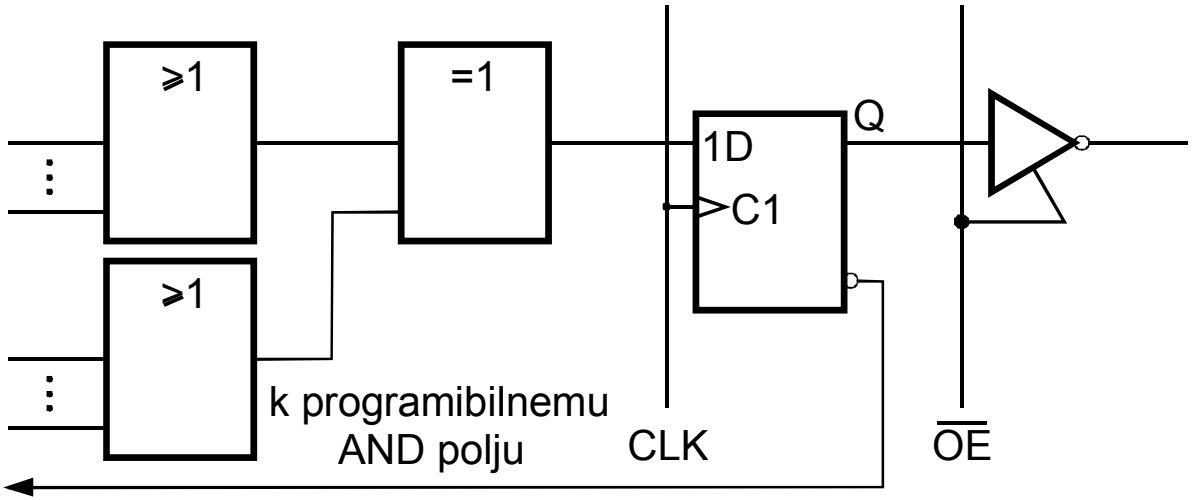
Če pa je drugi vhod v XOR sprogramiran, kar pomeni, da je zveza z maso prekinjena, je ta vhod v zraku in XOR vezje opravlja invertiranje vhodnega signala.

Izhod iz stikala treh stanj pa je v tem primeru neinvertiran.

Programibilna izhodna polariteta ni ugodna samo navzven, temveč tudi za načrtovanje znotraj vezja – enostavnejša dopolnilna funkcija.

V takem primeru je smiselno preveriti, če nima morda negirana funkcija manjše število konjunkcij, tako da to ne predstavlja več omejitve glede števila vhodov v OR vezje; izhod pa potem invertirati s XOR vezjem.

Poleg PAL vezij s programibilno izhodno polariteto obstajajo tudi vezja, ki imajo sicer na istem mestu vgrajene XOR elemente, vendar vse vhode teh elementov fiksno vezane na izhode OR vezij, tako da omogočajo funkcijsko zvezo: AND-OR-XOR.

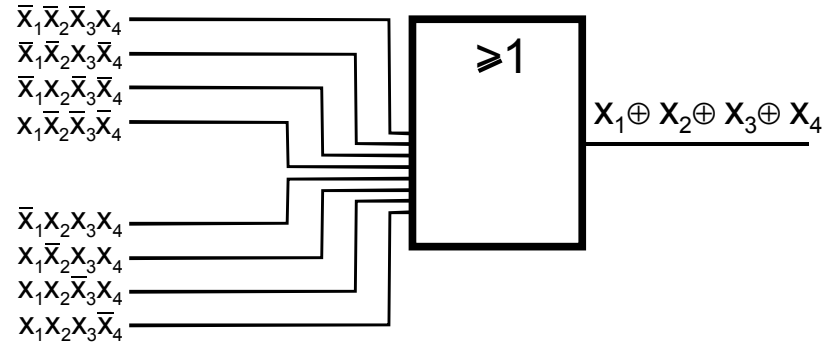


Prednost, ki jo prinaša arhitektura z vgrajenim XOR vezjem, si oglejmo na primeru realizacije XOR operacije med štirimi neodvisnimi spremenljivkami:

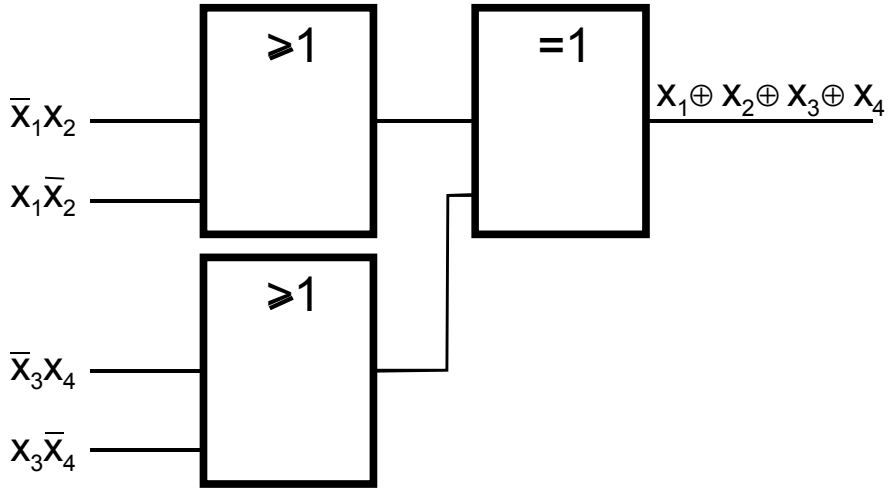
$x_1 \oplus x_2 \oplus x_3 \oplus x_4$

Pri realizaciji z navadnim PAL vezjem bi potrebovali osemvhodni OR element.

Če pa uporabimo AND-OR-XOR izvedbo, pa zadostujeta dva dvovhodna OR elementa.



a)



b)

9. 8. 2 Programibilna vezja z makro celicami

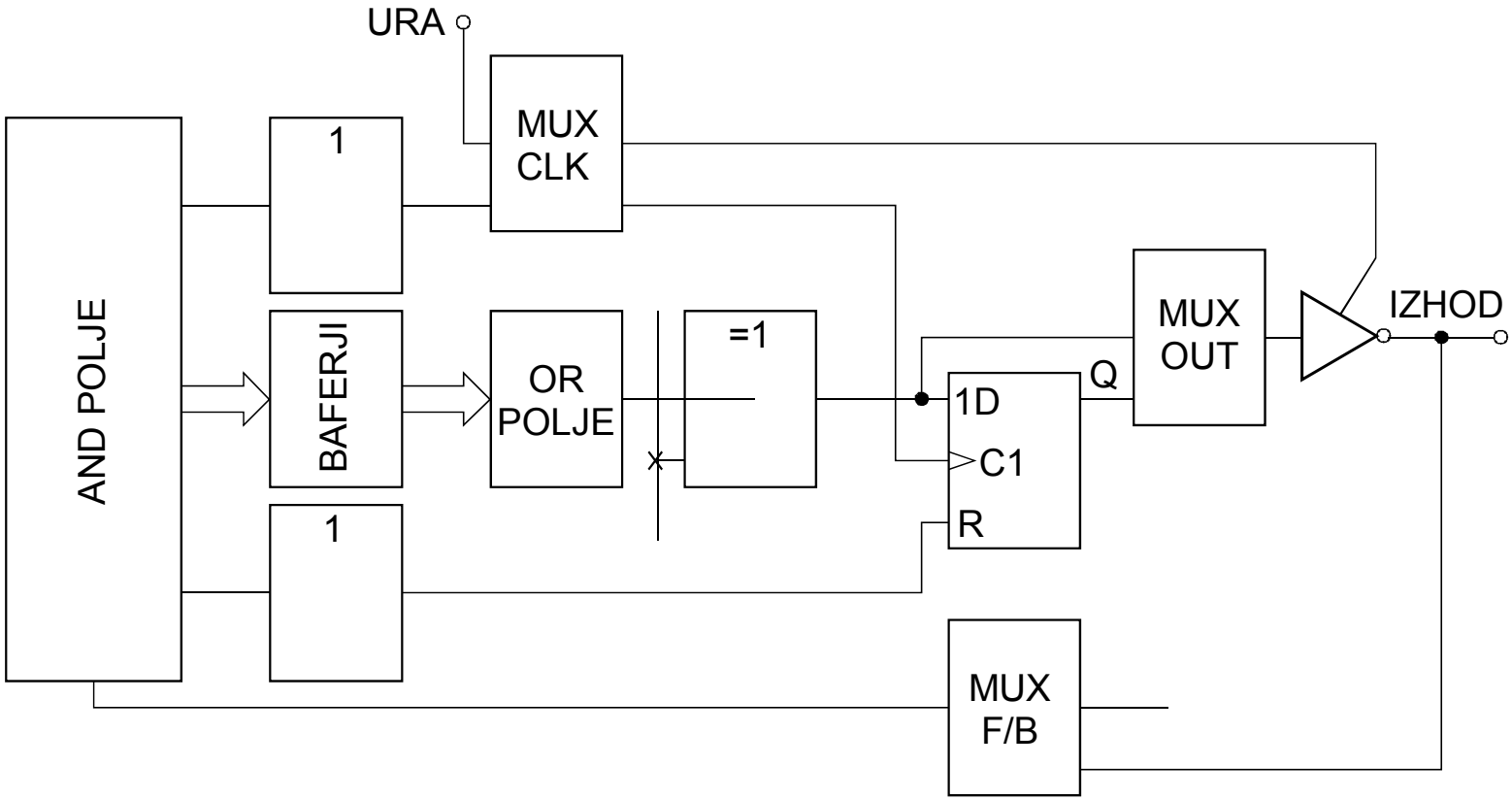
Pri teh vezjih je, za razliko od prejšnjih, ki so najbolj pogosto narejena v TTL tehnologiji, uporabljena CMOS tehnologija EEPROM vezij.

V tej tehnologiji je zelo enostavno realizirati vezja multipleksorjev, ti pa so poleg spominskih celic glavni funkcionalni sklopi makro celic.

Uporaba EEPROM tehnologije in vključitev multipleksorjev pa omogoča tem vezjem dve zelo važni prednosti:

Najprej je zaradi tehnološke izvedbe možno enostavno preprogramiranje in večja kompleksnost, ker odpade okenček za brisanje.

Makro celica firme ALTERA



Multipleksorji omogočajo neodvisno programiranje izhodnih linij, povratnih zvez in urinih impulzov za vsako celico posebej.

Selekcijski vhodi multipleksorjev so krmiljeni z izhodi EPROM vezja.

Izhodni multipleksor določa, ali bo izhod sekvenčen ali kombinacijski

Multipleksor v povratni zvezi odloča o tem, ali bo vhod v AND polje priključen na izhod spominske celice ali pa na zunanji signal.

Na ta način lahko programiramo zunanje priključke tako, da so bodisi vhodi ali pa izhodi zopet za vsako celico posebej

Nekatere izvedbe imajo celo v vsaki celici po dve povratni zvezi.

S tem lahko izkoristimo spominsko celico za notranje stanje, pripadajoč priključek pa kot neodvisen vhod, kar bistveno prispeva k izkoriščenosti vezja.

Tretji multipeksor omogoča izbiro urinega impulza med od zunaj dovedenim ali generiranim znotraj makro celice. Ker je ta drugi urin impulz generiran z neko konjunkcijo, je lahko poljubna kombinacija neodvisnih vhodov in dodatnih zunanjih urinih signalov.