[P220]

Увод у архитектуру рачунара

2

Саша Малков

Универзитет у Београду

Математички факултет

2013/2014

[P271]

Увод у архитектуру рачунара

Саша Малков

Тема 3

Минимизација логичких функција (наставак)

Метод Квин-МекКласког

- Карноове мапе нису погодне у случају више од 4 променљиве
  - Изнад 6 постаје веома тешко за мануелну употребу
- Метод Квин МекКласког се назива и *таблични метод*
- Поступак минимизације у три корака:
  - Проналажење простих импликаната
  - Одређивање битних импликаната
  - Укључивање додатних простих импликаната

Корак 1 - Циљ

- Проналажење простих импликаната
  - Циљ
    - Проналажење скупа терма који покрива све случајеве
    - Одговара проналажењу група у Карноовим мапама

Корак 1 - Поступак

- Проналажење простих импликаната
  - Конструише се табела – по ред за сваки дисјункт СНДФ
    - Редови се уреде у растућем редоследу према броју променљивих које нису комплементиране
    - Редови се групишу према броју комплементираних променљивих
  - Сваки терм се пореди са термима из претходне групе
    - “Упарени” су ако се разликују само по стању једне променљиве
    - Упарени се означе (✓)
    - Терм без те променљиве се додаје у нову табелу
  - Процес се понавља за све групе терма у табели
  - Поступак се понавља на новој табели све док има терма који могу да се упаре
  - Сви преостали неозначени терми (из свих табела) су “прости импликанти”

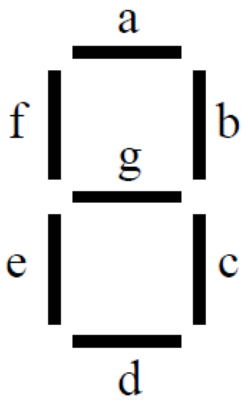
Корак 2

- Одређивање битних импликаната
  - Циљ
    - Препознавање импликаната који се морају појавити у сваком минимизованом облику
  - Поступак
    - Прави се табела у којој врсте одговарају простим импликантима а колоне дисјунктима полазне СДНФ
    - Ако је прости импликант део дисјункта, пресек врсте и колоне се означава са X
    - Ако колона садржи само једно X, оно се заокружи – то је “битан прости импликант”
    - Ако у врсти постоји заокружен X, око свих незаокружених X у врсти се нацрта квадрат
    - Ако у свакој колони постоји X са кругом или квадратом, поступак је завршен – конјункција битних импликаната је резултат

Корак 3

- Укључивање додатних простих импликаната
  - Циљ
    - Ако битни импликанти нису довољни да покрију резултат, укључује се минималан број додатних простих импликаната
  - Поступак
    - За сваки непокривен дисјункт (колону) се бира по један прост импликант (“додатни прост импликант”)
    - Додатни прости импликанти се бирају тако да њихов број буде минималан

Пример 1



- Минимизација сегмента *d* седмоделног дисплеја без недефинисаних вредности:
  - F = A’B’C’D’ + A’B’CD’ + A’B’CD + A’BC’D + A’BCD’ + AB’C’D’ + AB’C’D

Пример 1

Корак 0
A’B’C’D’
A’B’CD’
A’B’CD
A’BC’D
A’BCD’
AB’C’D’
AB’C’D

Пример 1

Корак 0	Корак 1
A’B’C’D’	A’B’C’D’
A’B’CD’	A’B’CD’
A’B’CD	AB’C’D’
A’BC’D	A’B’CD
A’BCD’	A’BC’D
AB’C’D’	A’BCD’
AB’C’D	AB’C’D

Пример 1

Корак 0	Корак 1		
A’B’C’D’	A’B’C’D’	✓	A’B’D’
A’B’CD’	A’B’CD’	✓	B’C’D’
A’B’CD	AB’C’D’	✓	A’B’C
A’BC’D	A’B’CD	✓	A’CD’
A’BCD’	A’BC’D		AB’C’
AB’C’D’	A’BCD’	✓	
AB’C’D	AB’C’D	✓	

Пример 1

Корак 0	Корак 1		
A’B’C’D’	A’B’C’D’	✓	**A’B’D’**
A’B’CD’	A’B’CD’	✓	**B’C’D’**
A’B’CD	AB’C’D’	✓	**A’B’C**
A’BC’D	A’B’CD	✓	**A’CD’**
A’BCD’	**A’BC’D**		**AB’C’**
AB’C’D’	A’BCD’	✓	
AB’C’D	AB’C’D	✓	

Пример 1

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A’B’D’							
B’C’D’							
A’B’C							
A’CD’							
AB’C’							
A’BC’D							

Пример 1

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A’B’D’	×	×					
B’C’D’	×					×	
A’B’C		×	×				
A’CD’		×			×		
AB’C’						×	×
A’BC’D				×			

Пример 1

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A’B’D’	×	×					
B’C’D’	×					×	
A’B’C		×	⊗				
A’CD’		×			⊗		
AB’C’						×	⊗
A’BC’D				⊗			

Пример 1

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A’B’D’	×	×					
B’C’D’	×					×	
A’B’C		☒	⊗				
A’CD’		☒			⊗		
AB’C’						☒	⊗
A’BC’D				⊗			

Пример 1

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A’B’D’	×	×					
B’C’D’	×					×	
A’B’C		☒	⊗				
A’CD’		☒			⊗		
AB’C’						☒	⊗
A’BC’D				⊗			

Пример 1

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A’B’D’	×	×					
B’C’D’	☑					×	
A’B’C		☒	⊗				
A’CD’		☒			⊗		
AB’C’						☒	⊗
A’BC’D				⊗			

Пример 1

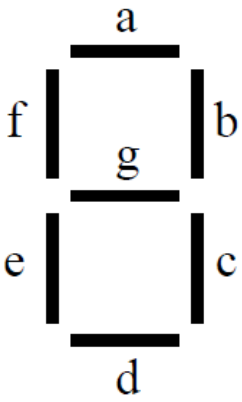
- Решење:
  - F = AB’C’ + A’B’C + A’CD’ + B’C’D’ + A’BC’D

“Небитни” случајеви

- Ако постоје аргументи за које резултат није важан
  - У првом кораку се разметрају као да је у њиховом случају резултат 1
  - У кораку 2 их занемарујемо и не наводимо

Пример 2

- Минимизација сегмента *d* седмоделног дисплеја са недефинисаним вредностима:
  - F = A’B’C’D’ + A’B’CD’ + A’B’CD + A’BC’D + A’BCD’ + AB’C’D’ + AB’C’D
  - Небитне вредности:
    - AB’CD’, AB’CD, ABC’D’, ABC’D, ABCD’, ABCD



Пример 2

Корак 0
A’B’C’D’
A’B’CD’
A’B’CD
A’BC’D
A’BCD’
AB’C’D’
AB’C’D
AB’CD’
AB’CD
ABC’D’
ABC’D
ABCD’
ABCD

Пример 2

Корак 0	Корак 1
A’B’C’D’	A’B’C’D’
A’B’CD’	A’B’CD’
A’B’CD	AB’C’D’
A’BC’D	A’B’CD
A’BCD’	A’BC’D
AB’C’D’	A’BCD’
AB’C’D	AB’C’D
AB’CD’	AB’CD’
AB’CD	ABC’D’
ABC’D’	AB’CD
ABC’D	ABC’D
ABCD’	ABCD’
ABCD	ABCD

Пример 2

Корак 0	Корак 1		Корак 2
A’B’C’D’	A’B’C’D’	✓	A’B’D’
A’B’CD’	A’B’CD’	✓	B’C’D’
A’B’CD	AB’C’D’	✓	A’B’C
A’BC’D	A’B’CD	✓	A’CD’
A’BCD’	A’BC’D	✓	AB’C’
AB’C’D’	A’BCD’	✓	AB’D’
AB’C’D	AB’C’D	✓	AC’D’
AB’CD’	AB’CD’	✓	B’CD’
AB’CD	ABC’D’	✓	AB’C
ABC’D’	AB’CD	✓	B’CD
ABC’D	ABC’D	✓	AB’D
ABCD’	ABCD’	✓	ABC’
ABCD	ABCD	✓	BC’D
			AC’D
			BCD’
			ACD’
			ABD’
			ACD
			ABD
			ABC

Пример 2

Корак 0	Корак 1		Корак 2		Корак 3
A’B’C’D’	A’B’C’D’	✓	A’B’D’	✓	B’D’
A’B’CD’	A’B’CD’	✓	B’C’D’	✓	AB’
A’B’CD	AB’C’D’	✓	A’B’C	✓	AC’
A’BC’D	A’B’CD	✓	A’CD’	✓	CD’
A’BCD’	A’BC’D	✓	AB’C’	✓	AD’
AB’C’D’	A’BCD’	✓	AB’D’	✓	B’C
AB’C’D	AB’C’D	✓	AC’D’	✓	AB
AB’CD’	AB’CD’	✓	B’CD’	✓	AD
AB’CD	ABC’D’	✓	AB’C	✓	AC
ABC’D’	AB’CD	✓	B’CD	✓	
ABC’D	ABC’D	✓	AB’D	✓	
ABCD’	ABCD’	✓	ABC’	✓	
ABCD	ABCD	✓	BC’D		
			AC’D	✓	
			BCD’	✓	
			ACD’	✓	
			ABD’	✓	
			ACD	✓	
			ABD	✓	
			ABC	✓	

Пример 2

Корак 0	Корак 1		Корак 2		Корак 3		Корак 4
A’B’C’D’	A’B’C’D’	✓	A’B’D’	✓	B’D’		A
A’B’CD’	A’B’CD’	✓	B’C’D’	✓	AB’	✓	
A’B’CD	AB’C’D’	✓	A’B’C	✓	AC’	✓	
A’BC’D	A’B’CD	✓	A’CD’	✓	CD’		
A’BCD’	A’BC’D	✓	AB’C’	✓	AD’	✓	
AB’C’D’	A’BCD’	✓	AB’D’	✓	B’C		
AB’C’D	AB’C’D	✓	AC’D’	✓	AB	✓	
AB’CD’	AB’CD’	✓	B’CD’	✓	AD	✓	
AB’CD	ABC’D’	✓	AB’C	✓	AC	✓	
ABC’D’	AB’CD	✓	B’CD	✓			
ABC’D	ABC’D	✓	AB’D	✓			
ABCD’	ABCD’	✓	ABC’	✓			
ABCD	ABCD	✓	BC’D				
			AC’D	✓			
			BCD’	✓			
			ACD’	✓			
			ABD’	✓			
			ACD	✓			
			ABD	✓			
			ABC	✓			

Пример 2

Корак 0	Корак 1		Корак 2		Корак 3		Корак 4
A’B’C’D’	A’B’C’D’	✓	A’B’D’	✓	**B’D’**		**A**
A’B’CD’	A’B’CD’	✓	B’C’D’	✓	AB’	✓	
A’B’CD	AB’C’D’	✓	A’B’C	✓	AC’	✓	
A’BC’D	A’B’CD	✓	A’CD’	✓	**CD’**		
A’BCD’	A’BC’D	✓	AB’C’	✓	AD’	✓	
AB’C’D’	A’BCD’	✓	AB’D’	✓	**B’C**		
AB’C’D	AB’C’D	✓	AC’D’	✓	AB	✓	
AB’CD’	AB’CD’	✓	B’CD’	✓	AD	✓	
AB’CD	ABC’D’	✓	AB’C	✓	AC	✓	
ABC’D’	AB’CD	✓	B’CD	✓			
ABC’D	ABC’D	✓	AB’D	✓			
ABCD’	ABCD’	✓	ABC’	✓			
ABCD	ABCD	✓	**BC’D**				
			AC’D	✓			
			BCD’	✓			
			ACD’	✓			
			ABD’	✓			
			ACD	✓			
			ABD	✓			
			ABC	✓			

Пример 2

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A							
B’D’							
CD’							
B’C							
BC’D							

Пример 2

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A						×	×
B’D’	×	×				×	
CD’		×			×		
B’C		×	×				
BC’D				×			

Пример 2

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A						×	⊗
B’D’	⊗	×				×	
CD’		×			⊗		
B’C		×	⊗				
BC’D				⊗			

Пример 2

	A’B’C’D’	A’B’CD’	A’B’CD	A’BC’D	A’BCD’	AB’C’D’	AB’C’D
A						☒	⊗
B’D’	⊗	☒				☒	
CD’		☒			⊗		
B’C		☒	⊗				
BC’D				⊗			

Пример 2

- Решење:
  - F = A + B’C + B’D’ + CD’ + BC’D

Задаци

- Функција израчунава 1 акко је дати неозначен 4-битни цео број дељив са 4.
  - Минимизовати помоћу Карноових мапа и методом Квин МекКласког.
- Функција израчунава 1 акко је дати неозначен 4-битни цео број у опсегу [5,10].
  - Минимизовати помоћу Карноових мапа и методом Квин МекКласког.
- Функција израчунава 1 акко је дати означен 4-битни цео број у скупу {-5, -4, -2, 2, 4, 5}.
  - Минимизовати помоћу Карноових мапа и методом Квин МекКласког.

[P271]

Увод у архитектуру рачунара

Саша Малков

Тема 4

Комбинаторне мреже

Комбинаторна мрежа

- Логичко коло је *комбинаторна мрежа* ако:
  - логички елементи су међусобно повезани и
  - у сваком тренутку излаз зависи само од вредности улаза у “том истом” тренутку

- Суштина је у одсуству било каквог чувања стања у самој мрежи
- Користи се и назив *комбинаторно коло*
- Сва представљана кола до сада су била комбинаторне мреже

Значај

- Комбинаторне мреже представљају виши ниво апстракције логичких кола
- Омогућавају употребу сложенијих логичких елемената при пројектовању логичких кола
- Њиховом употребом се смањује број потребних елемената при изградњи логичких кола

Начин дефинисања

- У општем случају комбинаторна мрежа се састоји од *n* бинарних улаза и *m* бинарних излаза
- Може се описати помоћу:
  - табеле истинитосних вредности
    - по ред за сваку од 2^n могућих комбинација улазних вредности
    - по колона за сваку улазну и излазну вредност
  - повезаног скупа графичких симбола
  - логичких функција које описују везу улаза и излаза

Врсте комбинаторних мрежа

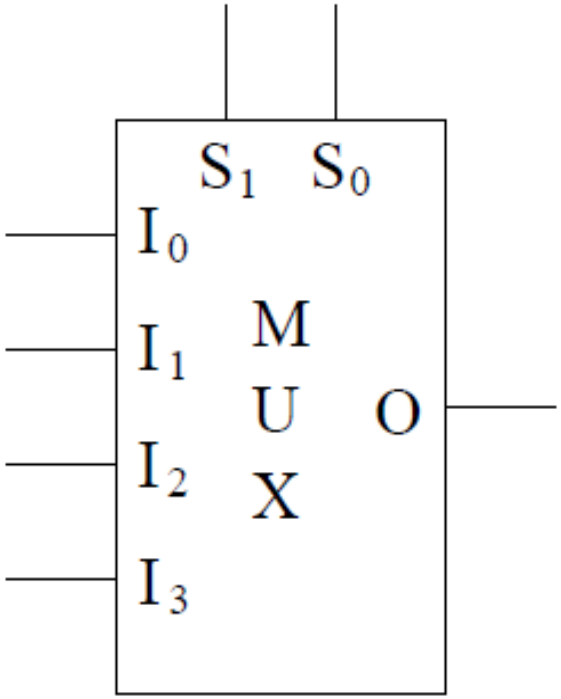
- Неке од најважнијих врста комбинаторних мрежа:
  - Мултиплексори
  - Демултиплексори
  - Декодери
  - Енкодери
  - Компаратори
  - Сабирачи
  - Програмабилни низ логичких елемената

Мултиплексори

- *Мултиплексори* су комбинаторне мреже које имају:
  - 2^n улаза
  - *n селекторских* улаза
  - 1 излаз
- Вредност излаза одговара вредности улаза који је одређен вредношћу селекторских улаза

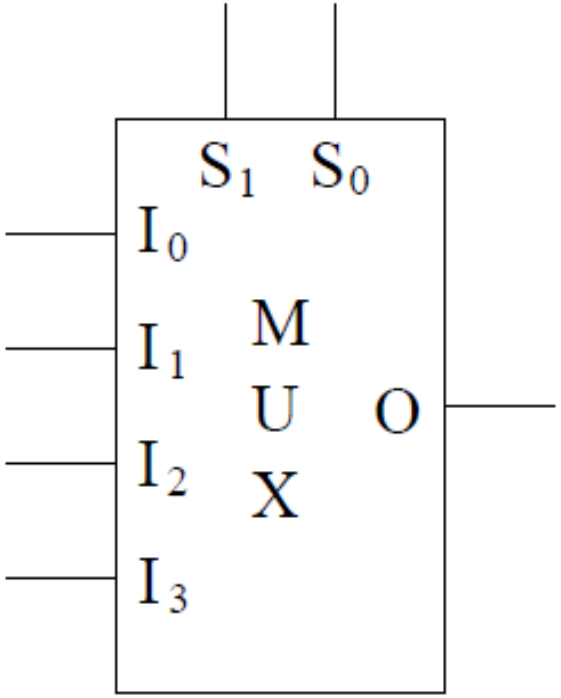
Мултиплексор 4-1

- Графичко представљање и одговарајућа таблица



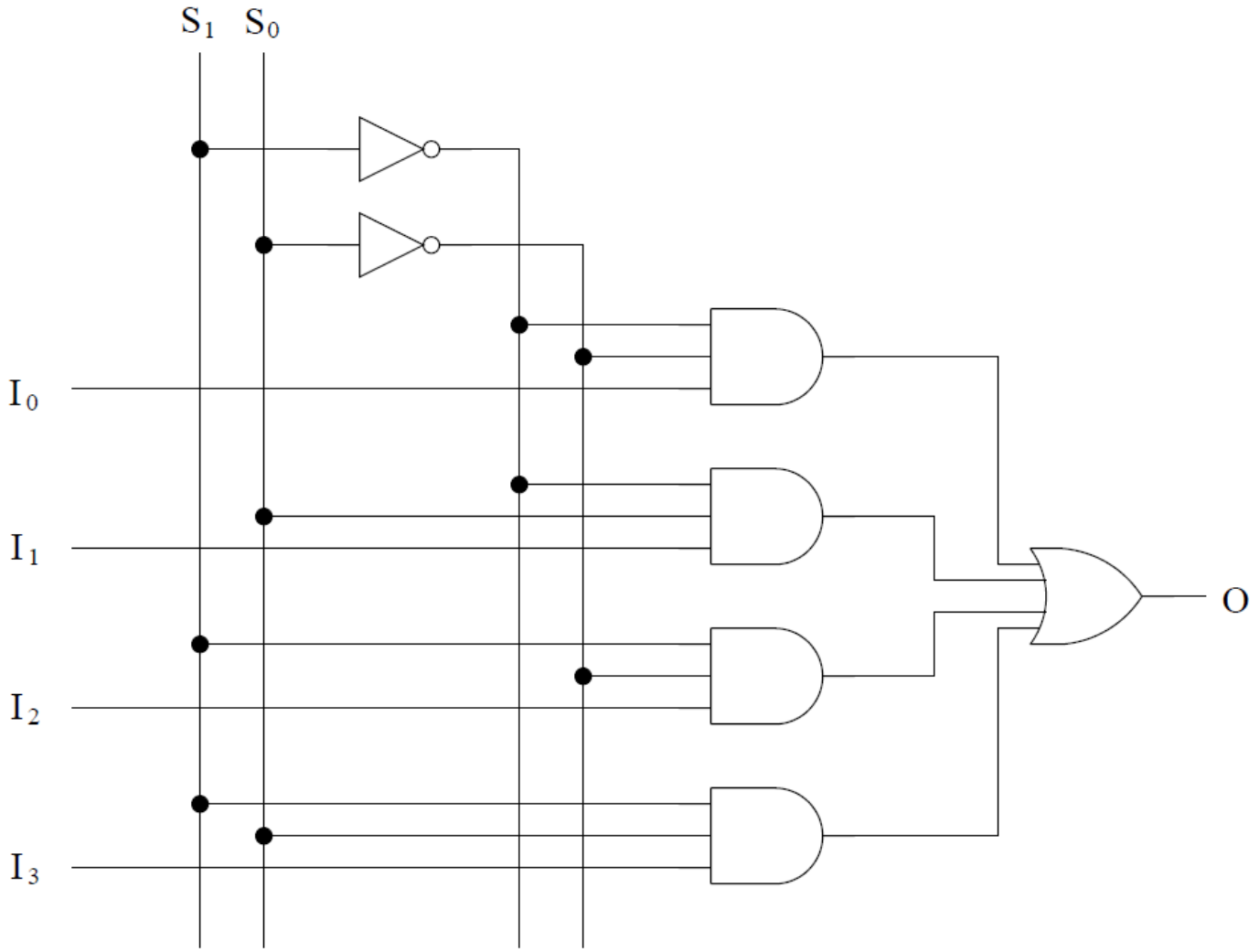
Мултиплексор 4-1

- Графичко представљање и одговарајућа таблица



S_1	S_0	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Имплементација мултиплексора

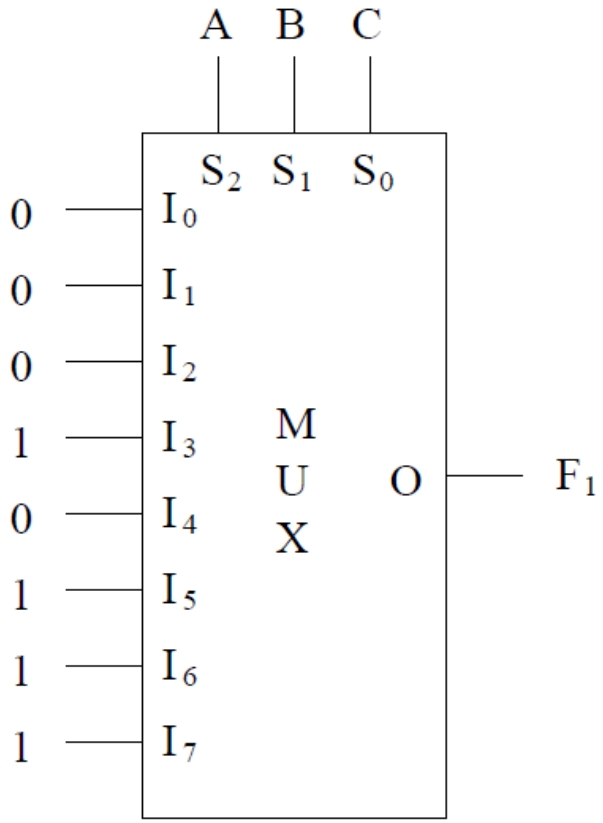


Примена мултиплексора

- Осим основне примене комбиновања више улаза у један излаз (одабирања), бирањем улазних вредности се мултиплексори могу употребљавати за имплементирање функција од селекторских улаза
  - 1. начин: на улазе се доведу константне вредности, тако да одговарају вредностима функције за одговарајуће селекторске улазе
  - 2. начин: процесом редукције се мултиплексором може имплементирати функција са n+1 аргумената
    - (у неким случајевима може и више)

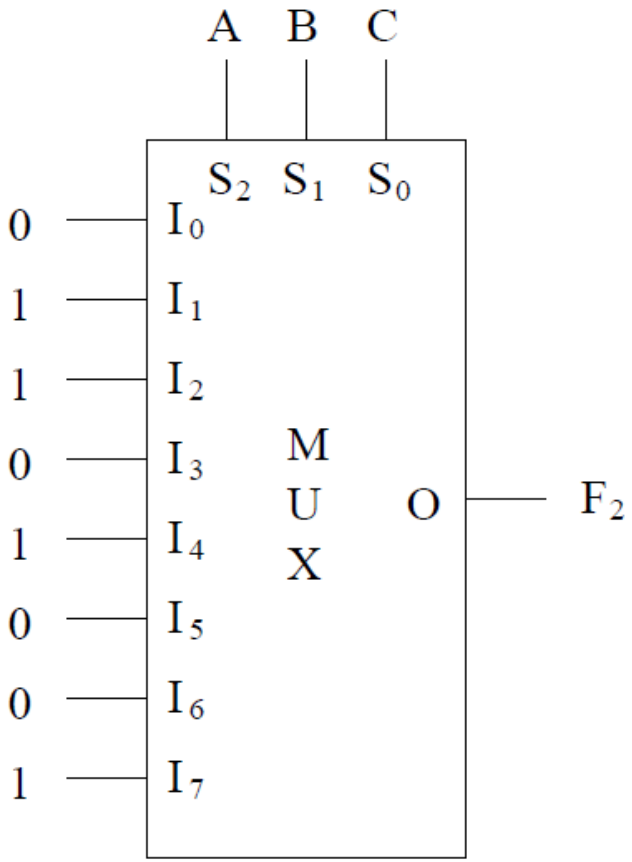
Пример одабирања улаза

- Израчунавање заступљенијег бита на селекторским улазима



Пример одабирања улаза

- Израчунавање парности за селекторске улазе



Редукција

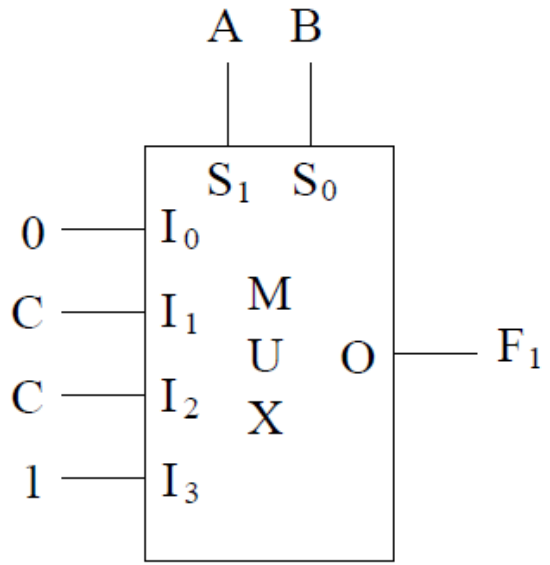
- Идеја редукције је да се један од аргумената преточи у резултат функције:
  - изабере се један аргумент, нпр. A_k
  - препознају случајеви у којима важи $F = A_k$ или $F = A_k$'
  - у осталим случајевима се функција представи тако да не зависи од аргумента A_k

Пример редукције

- Редукција полазне таблице и имплементација мултиплексором:
  - Функција рачуна већински бит

A	B	C	F_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A	B	F_1
0	0	0
0	1	C
1	0	C
1	1	1

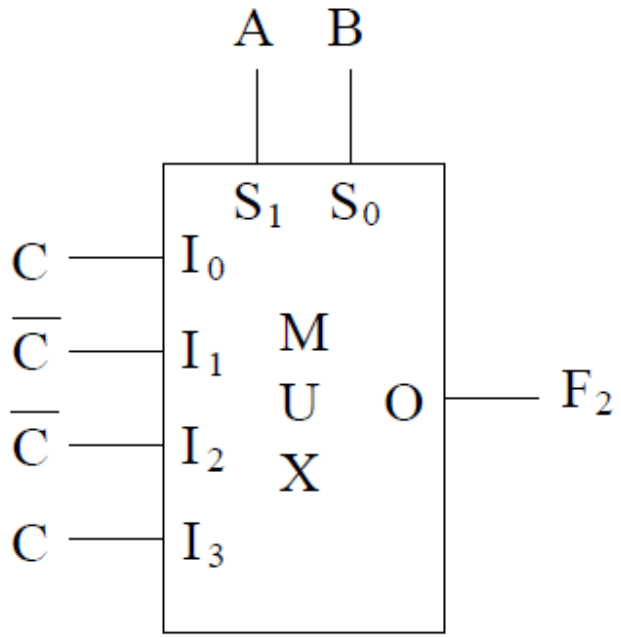


Пример редукције

- Редукција полазне таблице и имплементација мултиплексором:
  - Функција рачуна парност

A	B	C	F_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

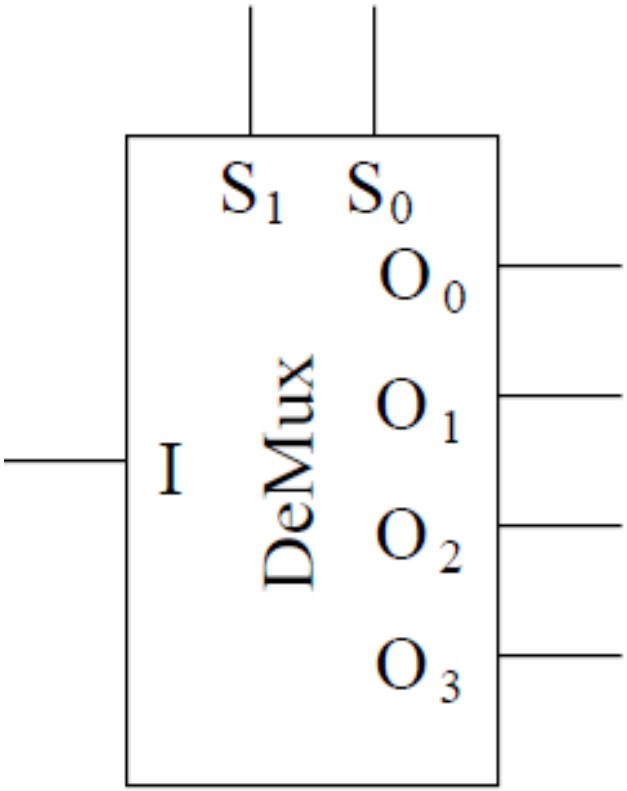
A	B	F_1
0	0	C
0	1	$\overline{C}$
1	0	$\overline{C}$
1	1	C



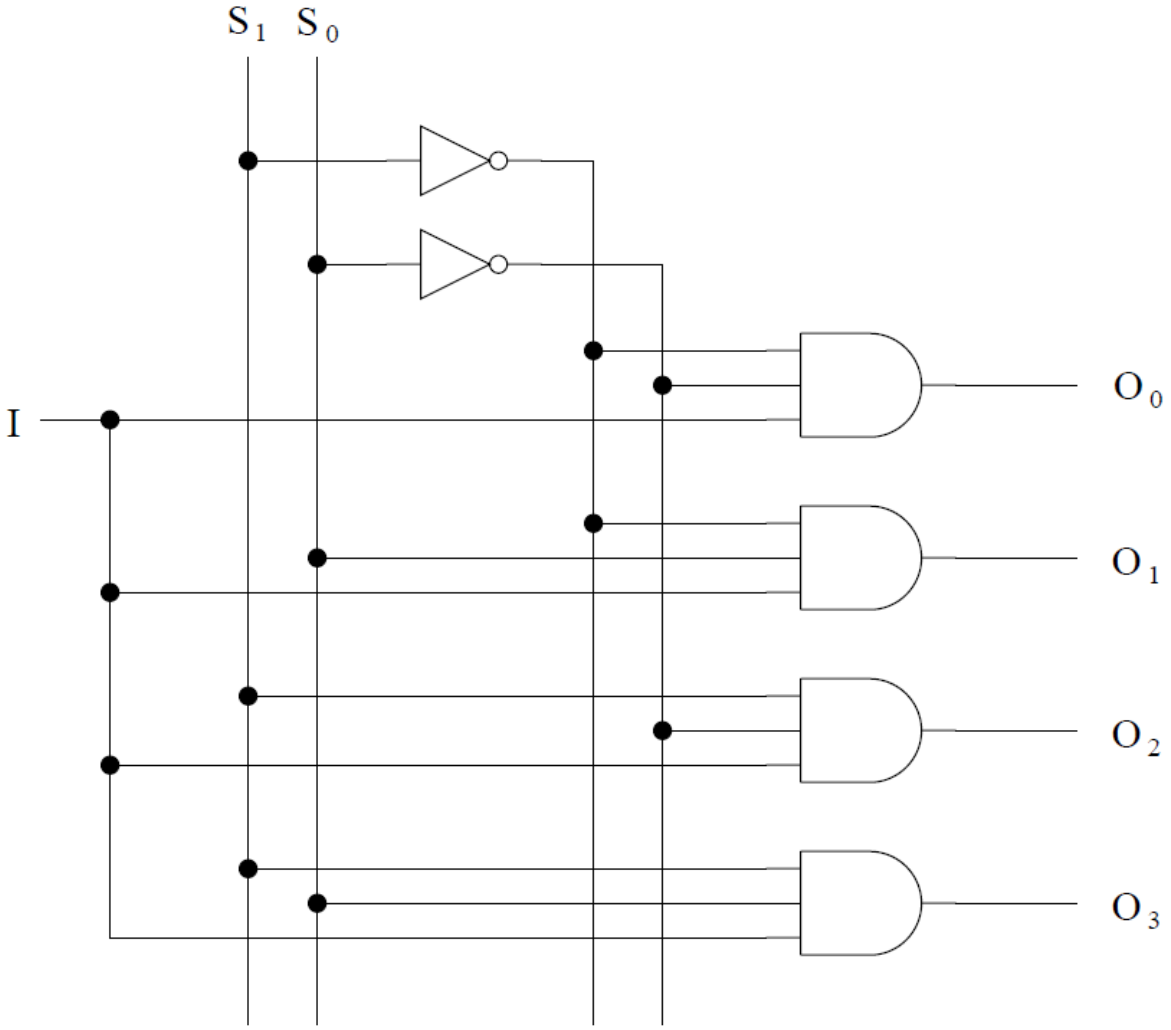
Демултиплексори

- *Демултиплексори* су комбинаторне мреже које имају:
  - n+1 улаза
    - 1 улазна вредност
    - n селекторских улаза
  - 2^n излаза
- Обављају инверзну функцију мултиплексора
  - Тачно на један излаз се пресликава вредност улаза
  - Сви остали излази имају вредност 0

Демултиплексор 1-4



Имплементација демултиплексора



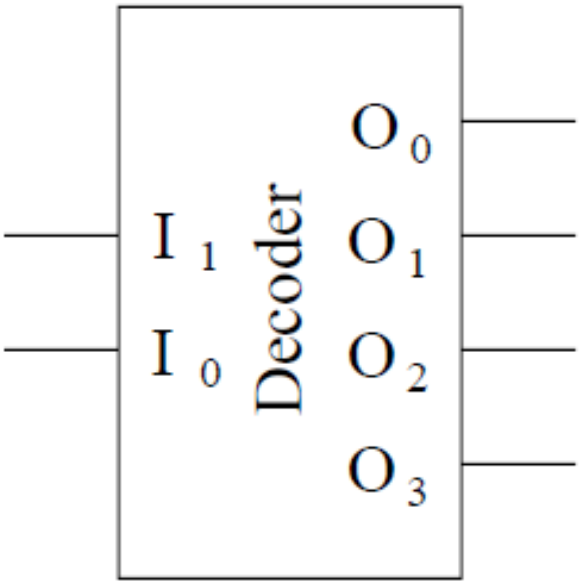
Декодери

- Декодери су комбинаторне мреже које имају:
  - n улаза
  - највише 2^n излаза
- У сваком тренутку је активан највише један излаз, а у зависности од улаза

Пример декодера

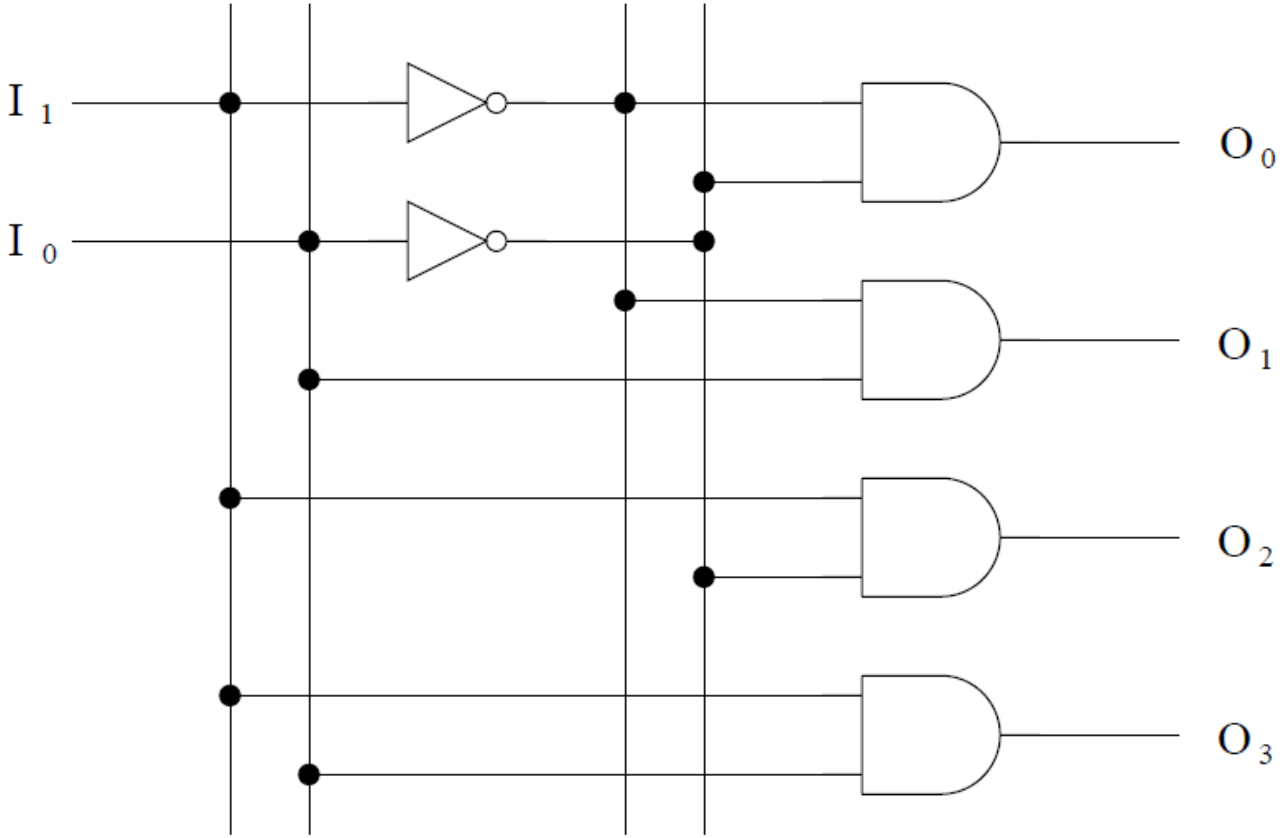
- На основу неозначеног 2-битног целог броја бира одговарајући од 4 улаза:

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



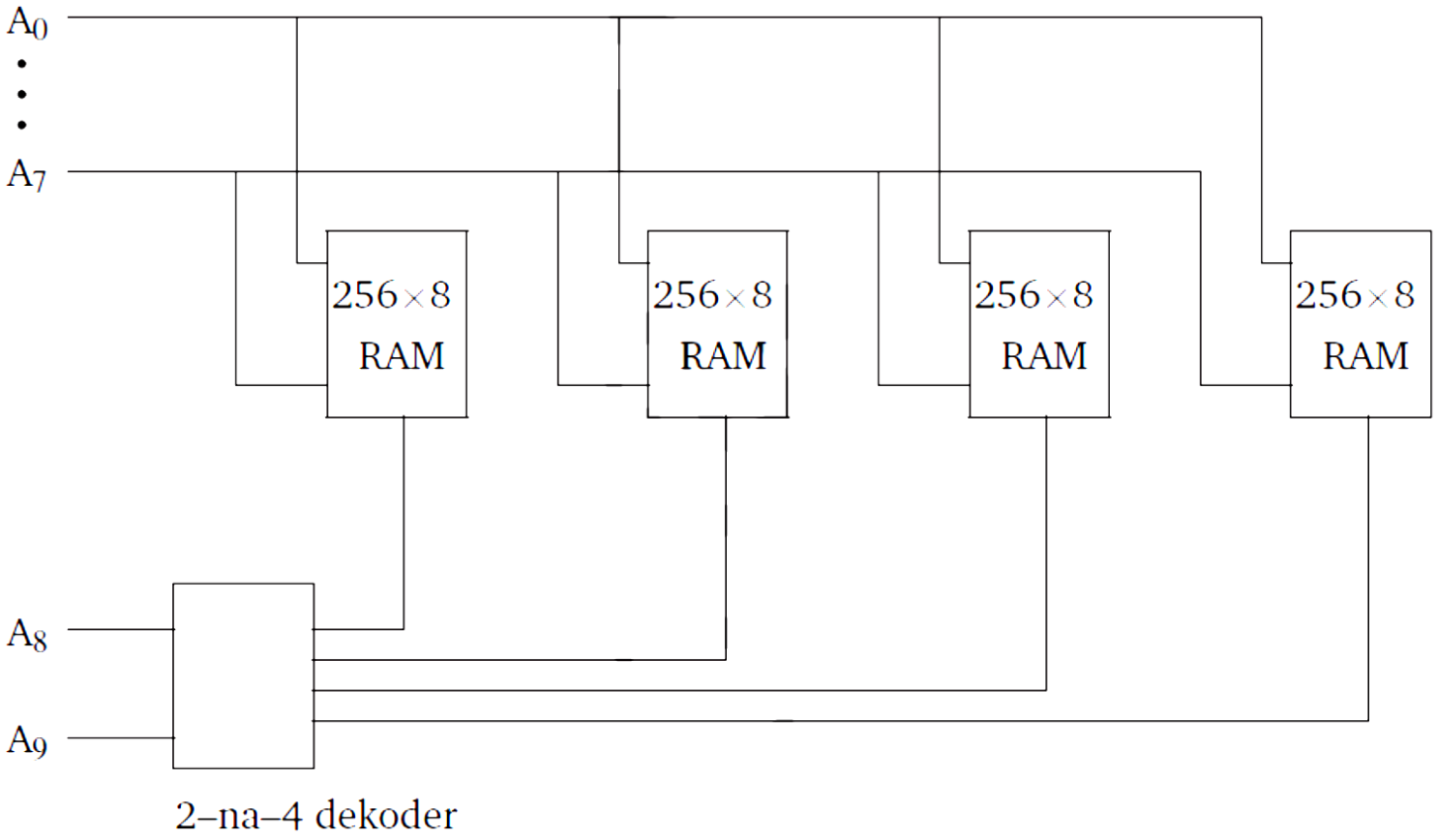
Пример декодера

- На основу неозначеног 2-битног целог броја бира одговарајући од 4 улаза:



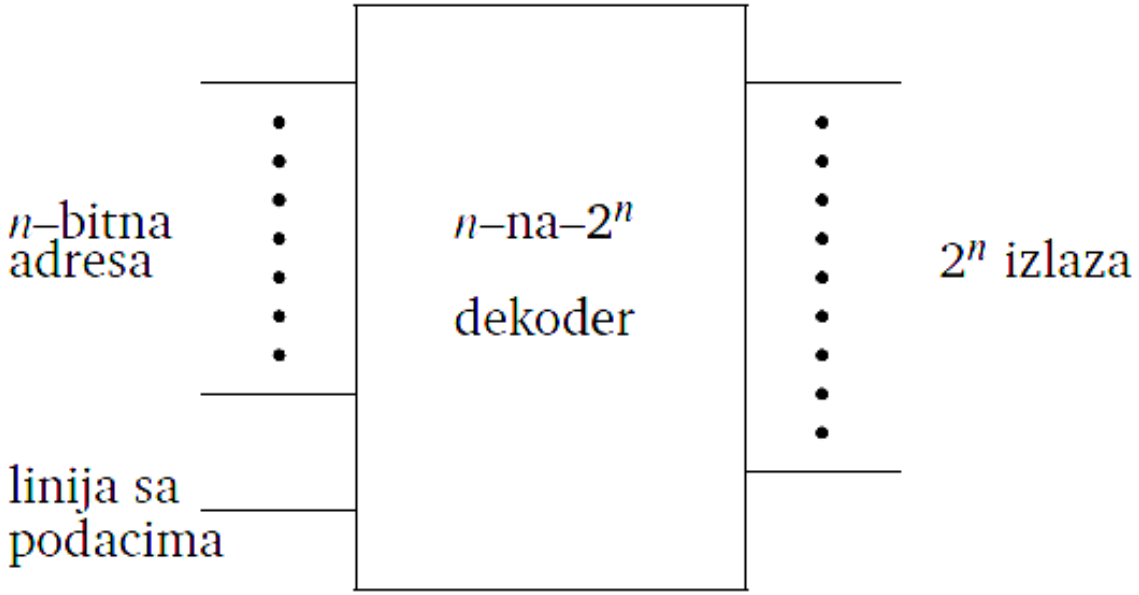
Пример – Декодирање адреса

- Декодирање адреса меморије величине 1KB од четири 256B (8-битна) RAM чипа:



Декодер као демултиплексор

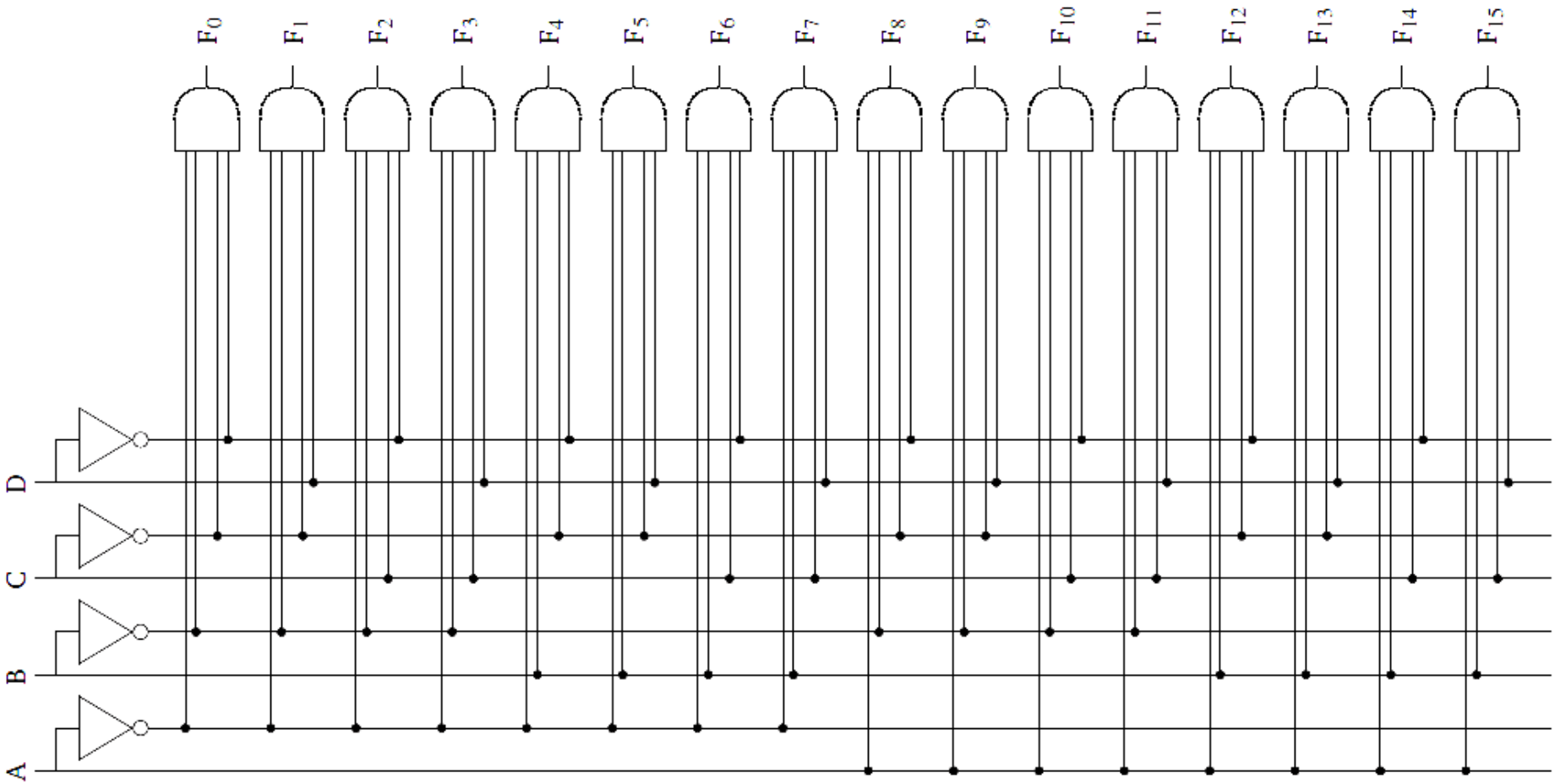
- Декодеру се уводи додатни улаз
  - улаз представља улаз демултиплексора
  - може да се тумачи и као контролни бит декодера:
    - декодер ради ако је улаз активан
    - ако је улаз неактиван, сви излази су неактивни



Пример – Декодирање *BCD* цифара

Ulaz				Izlaz													
				Dekadne cifre										Greška			
A	B	C	D	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Пример – Декодирање *BCD* цифара



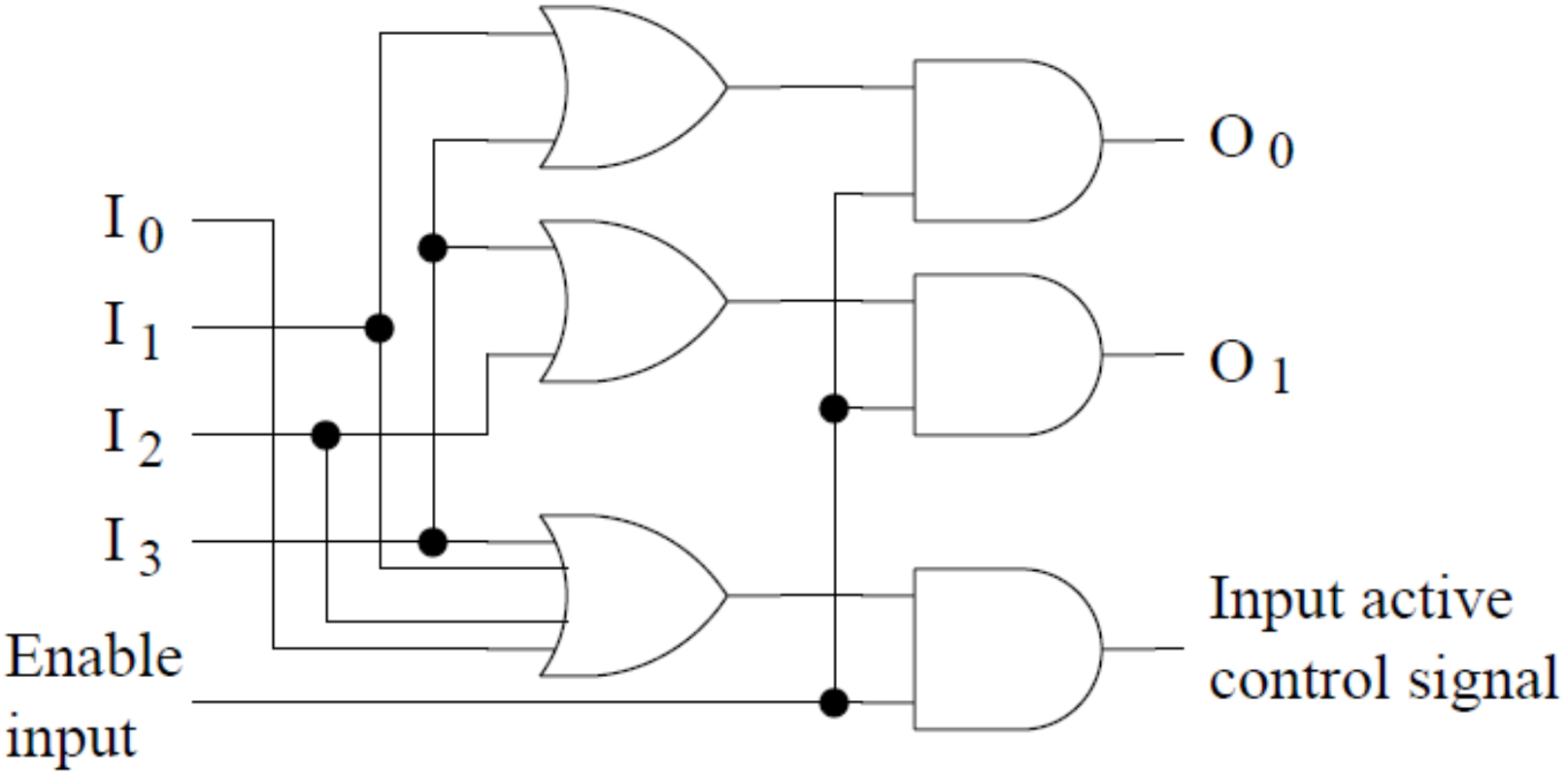
Енкодер

- Енкодери су комбинаторне мреже које имају:
  - 2^n улаза
  - n излаза
- Представљају инверзну операцију декодера
  - У сваком тренутку је активан највише један улаз
  - Излаз је одређен активним улазом
- Обично се додају
  - контролни улаз који укључује енкодер
  - контролни излаз који је активан ако су активни
    - контролни улаз и
    - бар један улазни бит

Пример енкодера

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

Пример енкодера

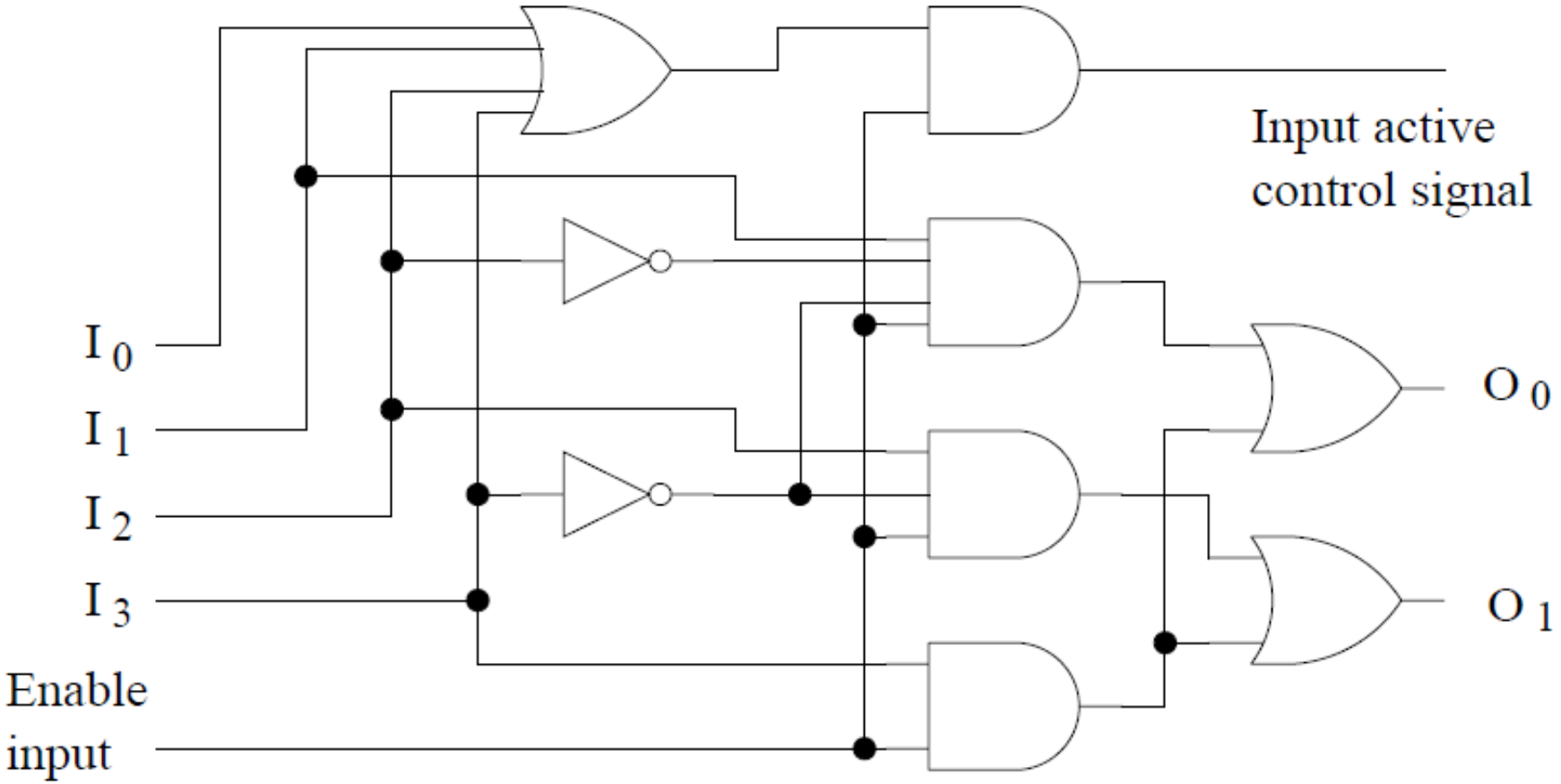


Енкодер са приоритетом

- У одређеним случајевима се, ако је неки улаз активан, неки други улази могу игнорисати:

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

Енкодер са приоритетом



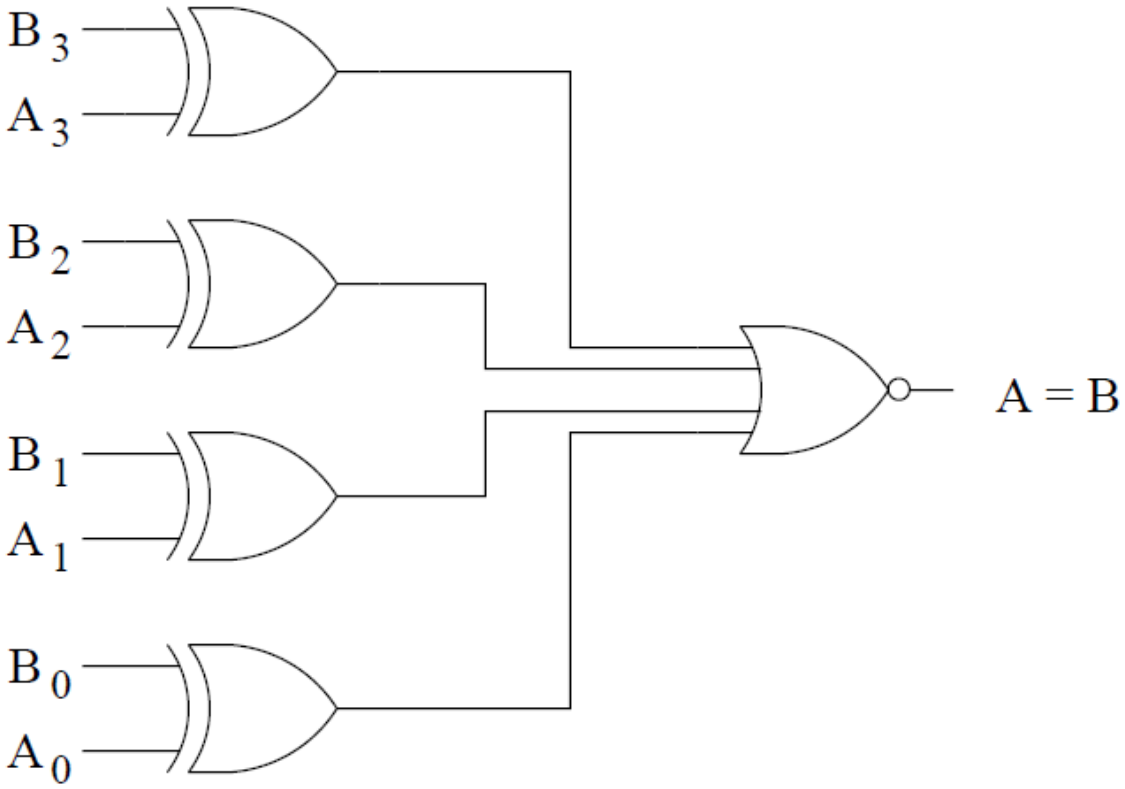
Компаратор

- Компаратори су мреже које пореде два низа битова на одговарајући начин
- Имају:
  - 2 x *n* улаза
  - одговарајући број излаза, зависно од врсте поређења

- Примери:
  - поређење једнакости – довољан је један излаз
  - поређење величине броја – довољна су два излаза, али се обично користе три
  - поређење са проширењем – ако се пореде бројеви који имају *m* x *n* битова, улазна информација обухвата и резултат поређења претходне групе од *n* битова

Компаратори – једнакост

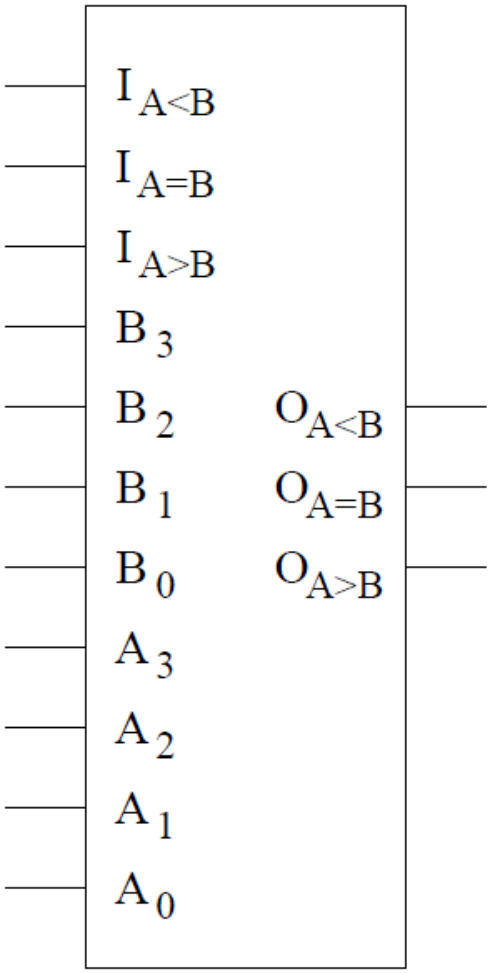
- Поређење једнакости се обично изводи помоћу *XOR* елемената
- Поредимо једнакост два низа од 4 бита:



Компаратори – поређење

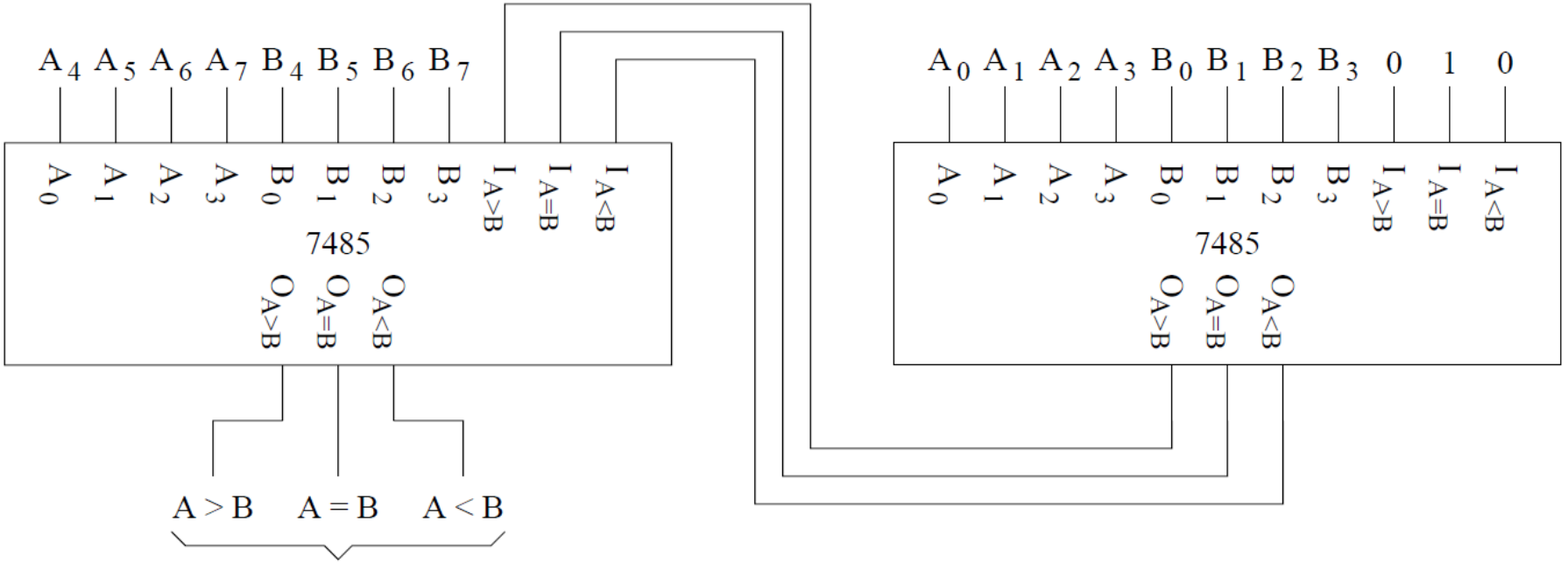
- Поређење два 4-битна броја, за вежбање

Компаратори – поређење са проширењем



Компаратори – поређење са проширењем

- Поређење два неозначена 8-битна цела броја помоћу два компаратора дужине 4:



Сабирачи

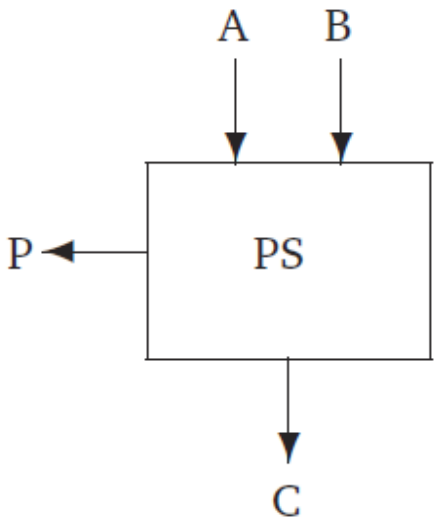
- Сабирачи су комбинаторне мреже које се користе при имплементацији сабирања

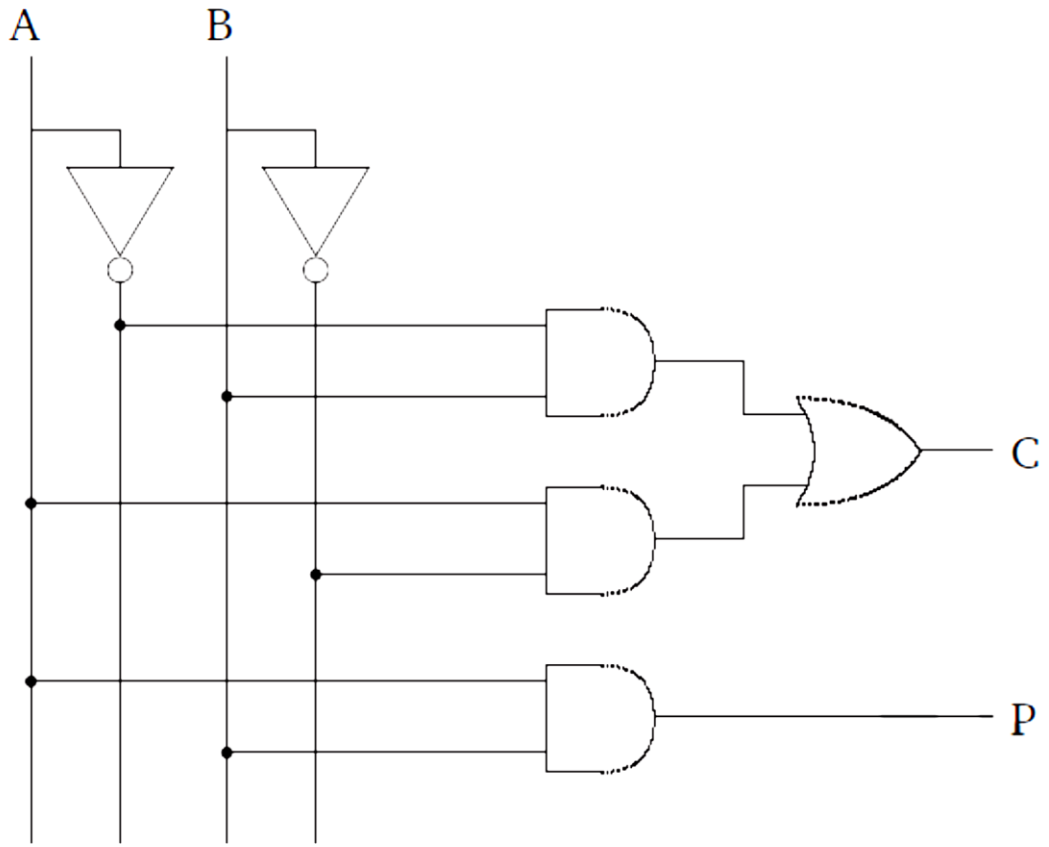
Бинарни полусабирач

- Бинарни полусабирач је комбинаторна мрежа која сабира два једноцифрена бинарна броја
  - Улази су два једноцифрена броја
  - Излази су један бит резултата и један бит преноса

Бинарни полусабирач

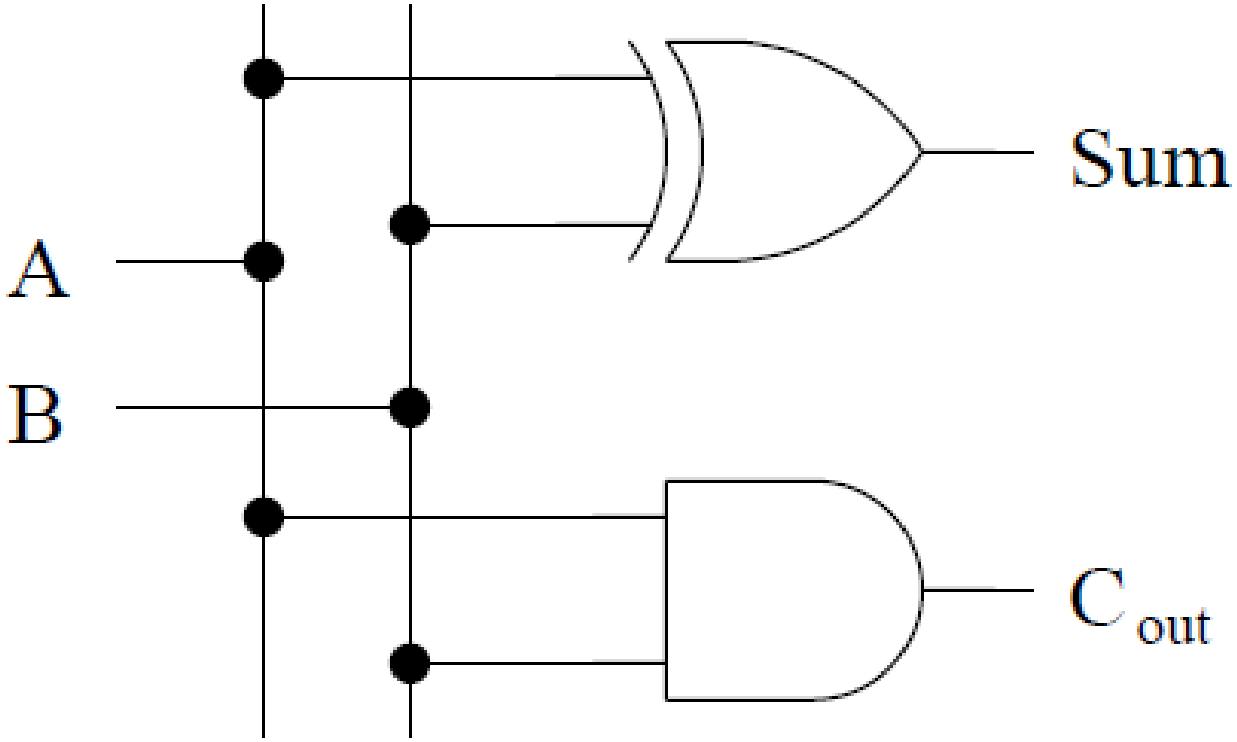
A	B	C	P
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1





Бинарни полусабирач

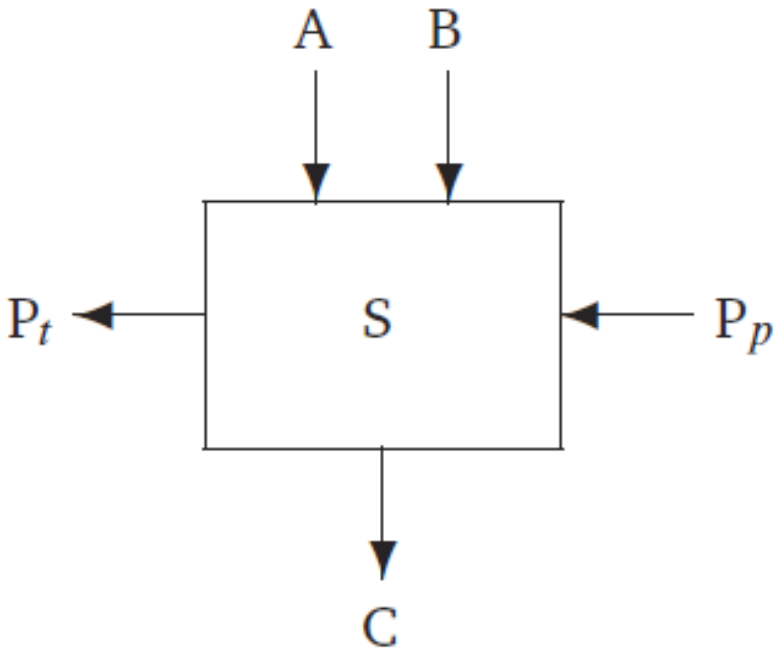
- Задатак: оптимизовати бинарни полусабирач



Бинарни сабирач

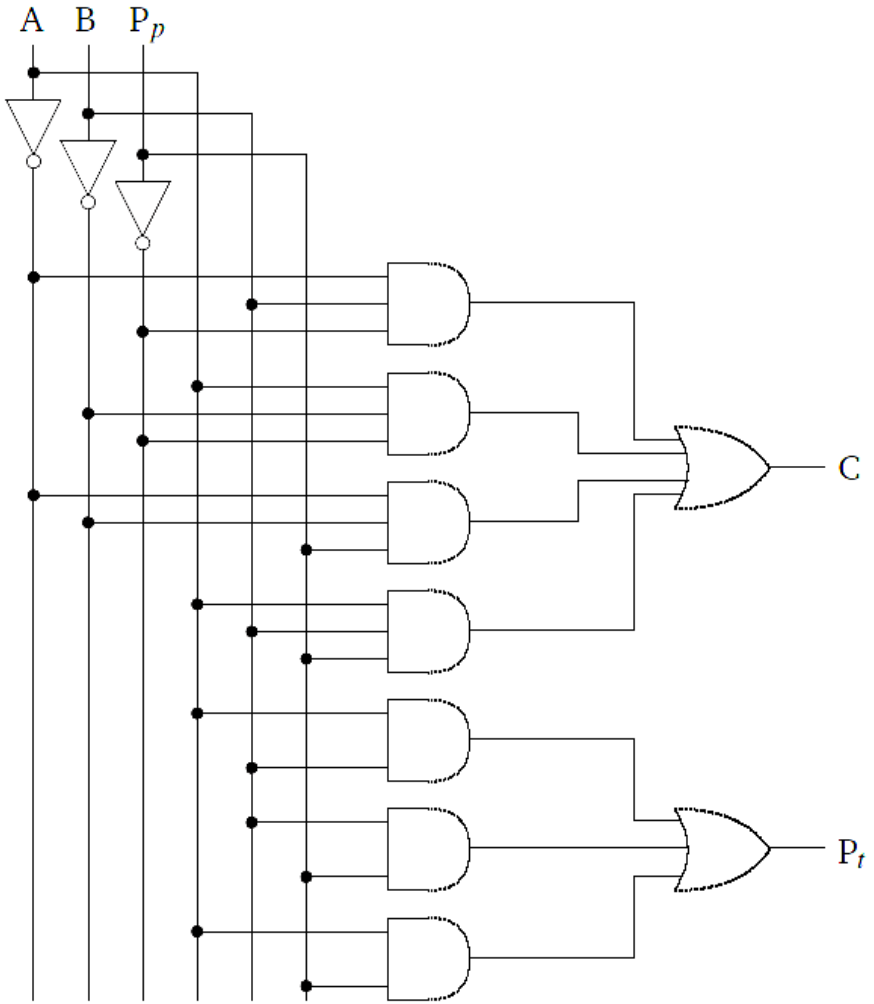
- Бинарни сабирач је комбинаторна мрежа која сабира два једноцифрена бинарна броја и дати пренос
  - Улази су два једноцифрена броја и један бит преноса
  - Излази су један бит резултата и један бит преноса
- Задатак: минимизовати бинарни сабирач

Бинарни сабирач



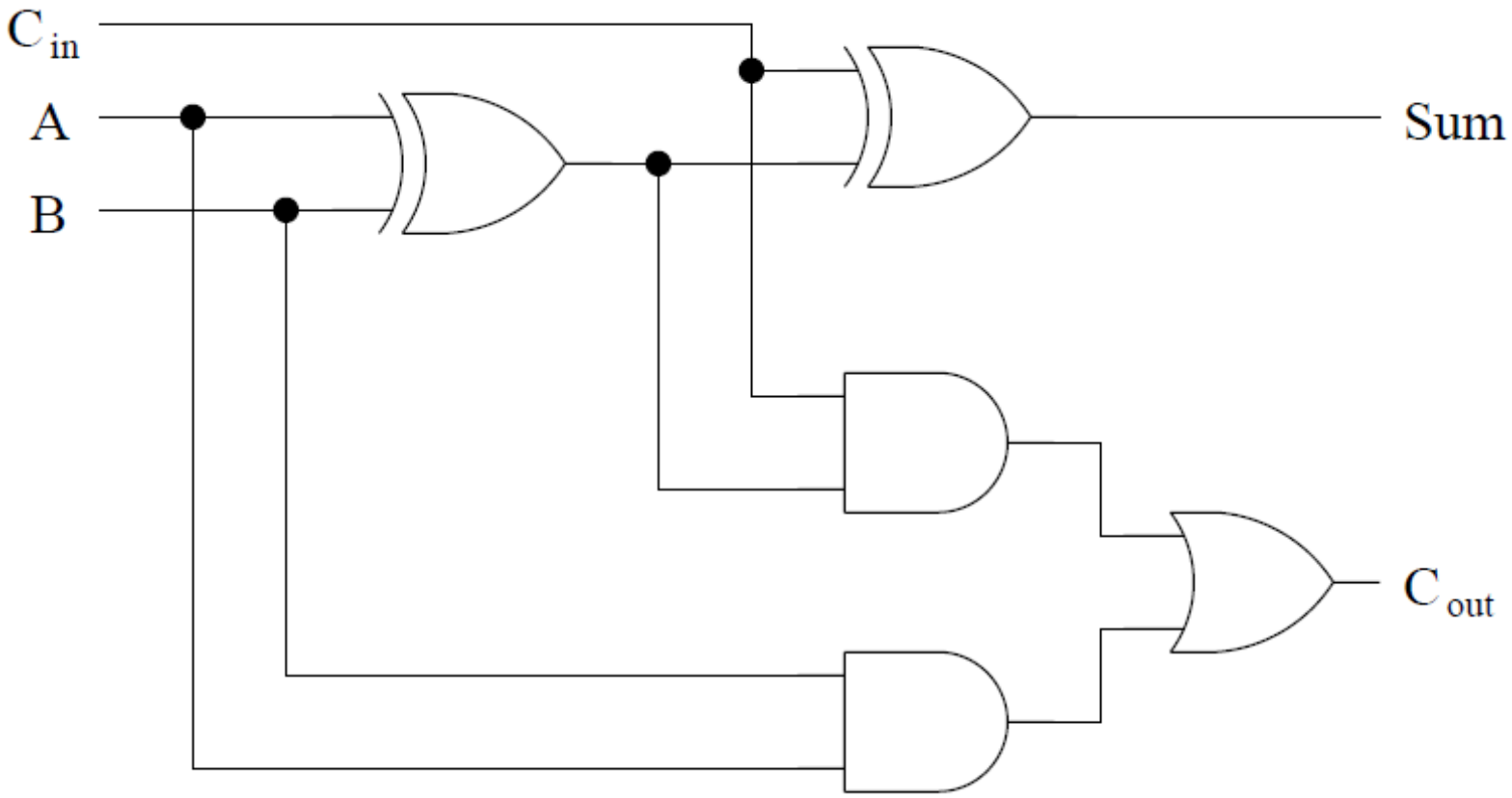
P_p	A	B	C	P_t
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Бинарни сабирач



Бинарни сабирач

- Задатак: оптимизовати бинарни сабирач

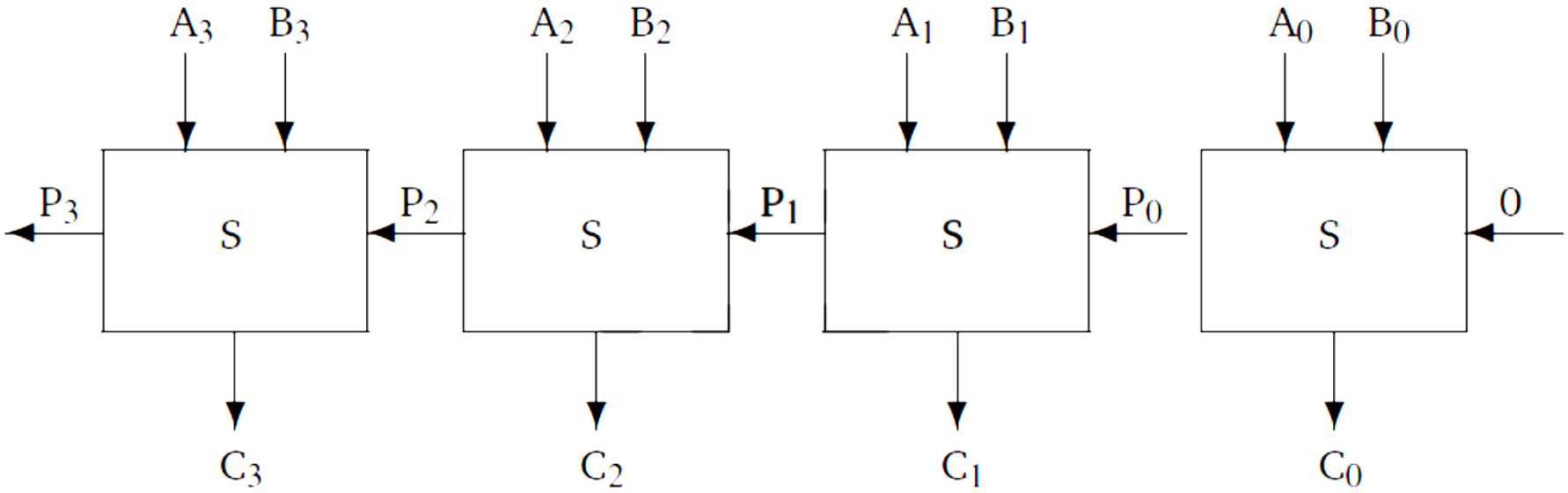


Сложени сабирач

- Сложени сабирач (или *вишебитни сабирач*) је комбинаторна мрежа која сабира два вишецифрена бинарна броја и дати пренос
- Имплементира се помоћу више бинарних сабирача

Сложени сабирач

- Сабирање два 4-битна броја, са прекорачењем



Проблем ефикасности

- Ако претпоставимо да сваки елемент производи кашњење од око 5ms, онда један оптимизован сабирач производи кашњење од 15ms
- Тада би 16-битни сабирач са постепеним рачунањем преноса имао кашњење од 16x15 = 240ms
- Да би се кашњење смањило, приступа се *рачунању преноса унапред*

Рачунање преноса унапред

- Први пренос се рачуна као:

  $P_0 = A_0B_0$

- Други пренос се рачуна као:

  $P_1 = A_1B_1 + (A_1+B_1)P_0$

- Ако заменимо први...

  $P_1 = A_1B_1 + (A_1+B_1)\,A_0B_0$

  $= A_1B_1 + A_1\,A_0B_0 + B_1A_0B_0$

- Слично, други пренос је:

  $P_2 = A_2B_2 + (A_2+B_2)P_1$

  $= A_2B_2 + (A_2+B_2)(A_1B_1 + (A_1+B_1)\,A_0B_0)$

  $= A_2B_2 + (A_2+B_2)(A_1B_1 + (A_1+B_1)\,A_0B_0)$

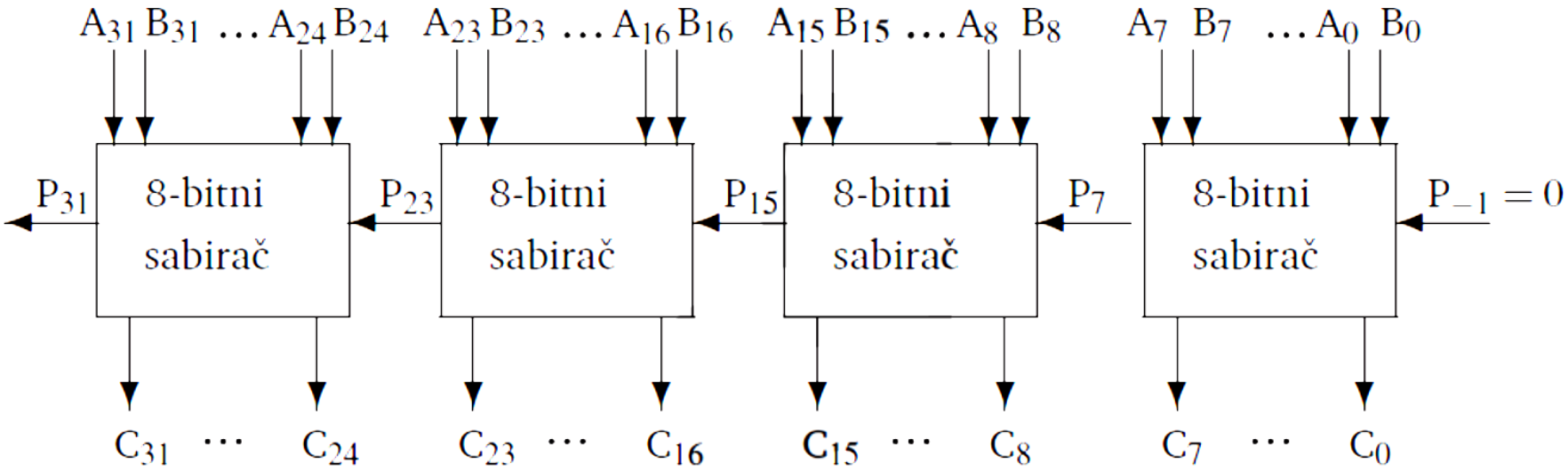
- и тако даље ...

Рачунање преноса унапред

- Очигледно је да се рачунање преноса унапред значајно компликује са сваким кораком
- Сложеност чини имплементацију непрактичном за више од 8 битова
- Уобичајено се пренос рачуна за 4 или 8 битова унапред

Рачунање преноса унапред

- Имплементација 32.битног сабирача помоћу 4 8-битна сабирача



Програмабилни низ логичких елемената

- Прва интегрисана кола су имала до 10 логичких елемената
- Свако коло је обликовано према намени

- Повећавање степена интеграције и броја уграђених логичких елемената је довело до повећавања броја различитих кола и отежавало производњу
- Одатле је настала потреба за развијање техника за производњу интегрисаних логичких кола опште намене, која се лако могу прилагодити специфичним потребама (тј. *програмирати*)

Програмабилни низ лог ел. (2)

- Програмабилни низ логичких елемената (енгл. *Programmable Logic Array - PLA*) почива на примени СДНФ

- Како се свака логичка функција може изразити у облику СДНФ, најопштија форма је да се:
  - сваки улаз “повеже” са НЕ елементом
  - сваки улаз и његова негација “повеже” са И елементима
  - сваки излаз из И елемента се “повеже” са ИЛИ елементима
  - излази ИЛИ елемената представљају излазе логичког кола

Програмабилни низ лог ел. (3)

- Назив потиче из чињенице да се имплементација састоји од два низа елемената
  - низа И елемената и
  - низа ИЛИ елемената

- Ако има n улаза и m излаза, потребно је:
  - до 2^n И елемената са по $2n$ улаза
  - m ИЛИ елемената са до 2^n улаза

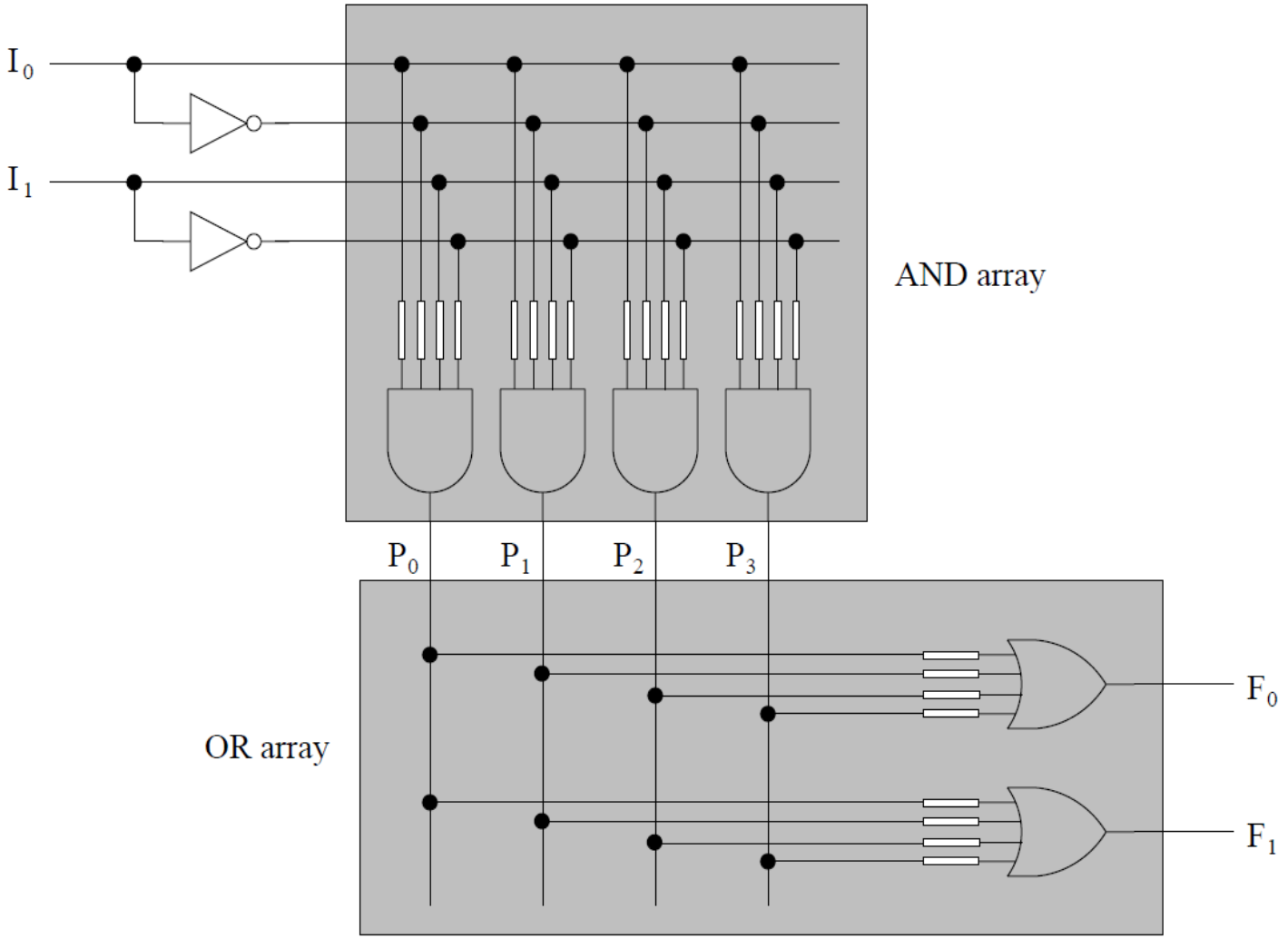
Програмабилни низ лог ел. (4)

- На сваком од улаза у И и ИЛИ елементе постоји по *прекидач*
- Програмирање се постиже искључивањем прекидача

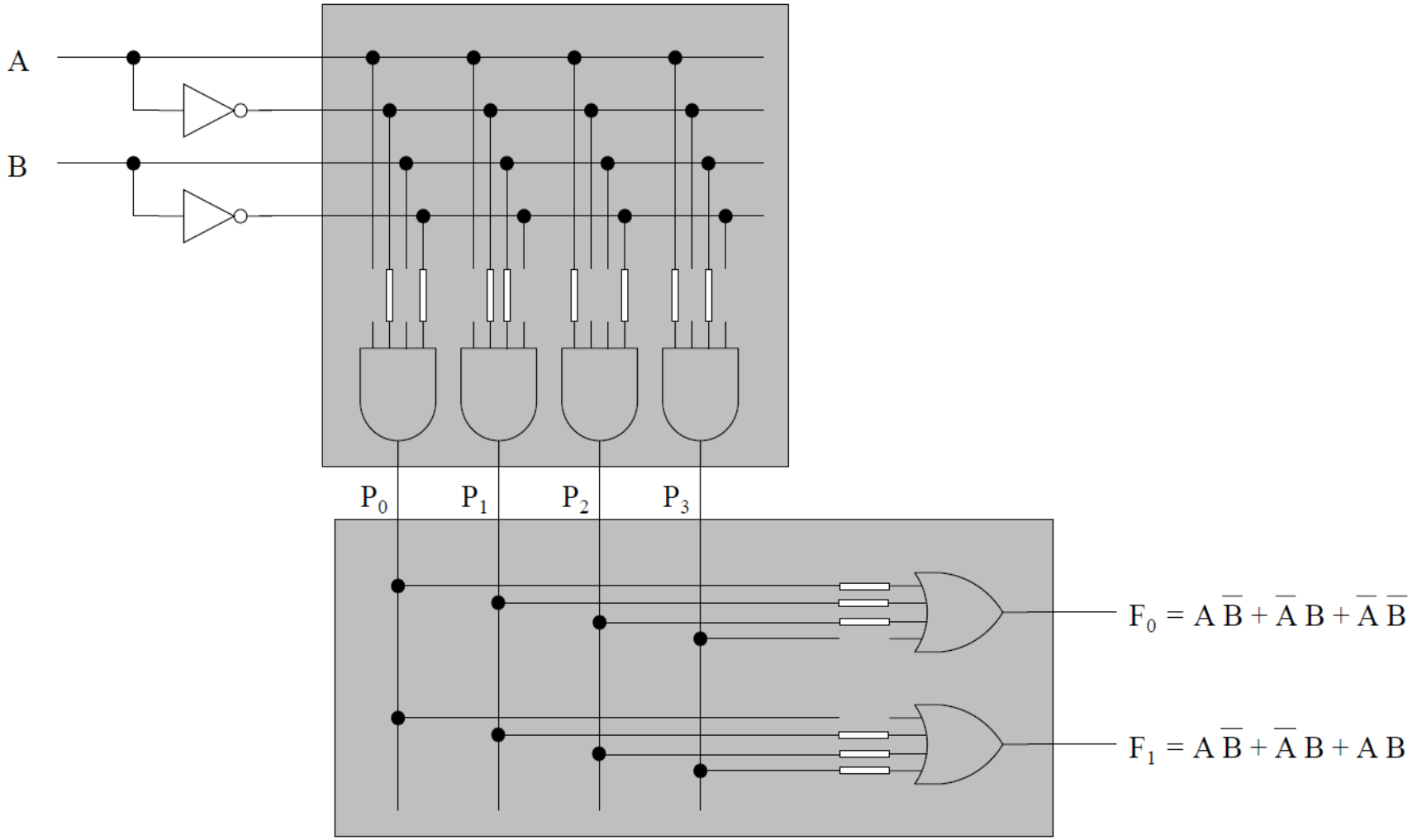
- Прекидачи се по правилу искључују једнократно
  - уобичајено је њихово спаљивање пропуштањем јаке ел. струје

- Имплементацијом се обезбеђује да се искључени прекидачи понашају као активни (1) улази за И кола и неактивни (0) улази за ИЛИ кола

Пример ПНЛЕ



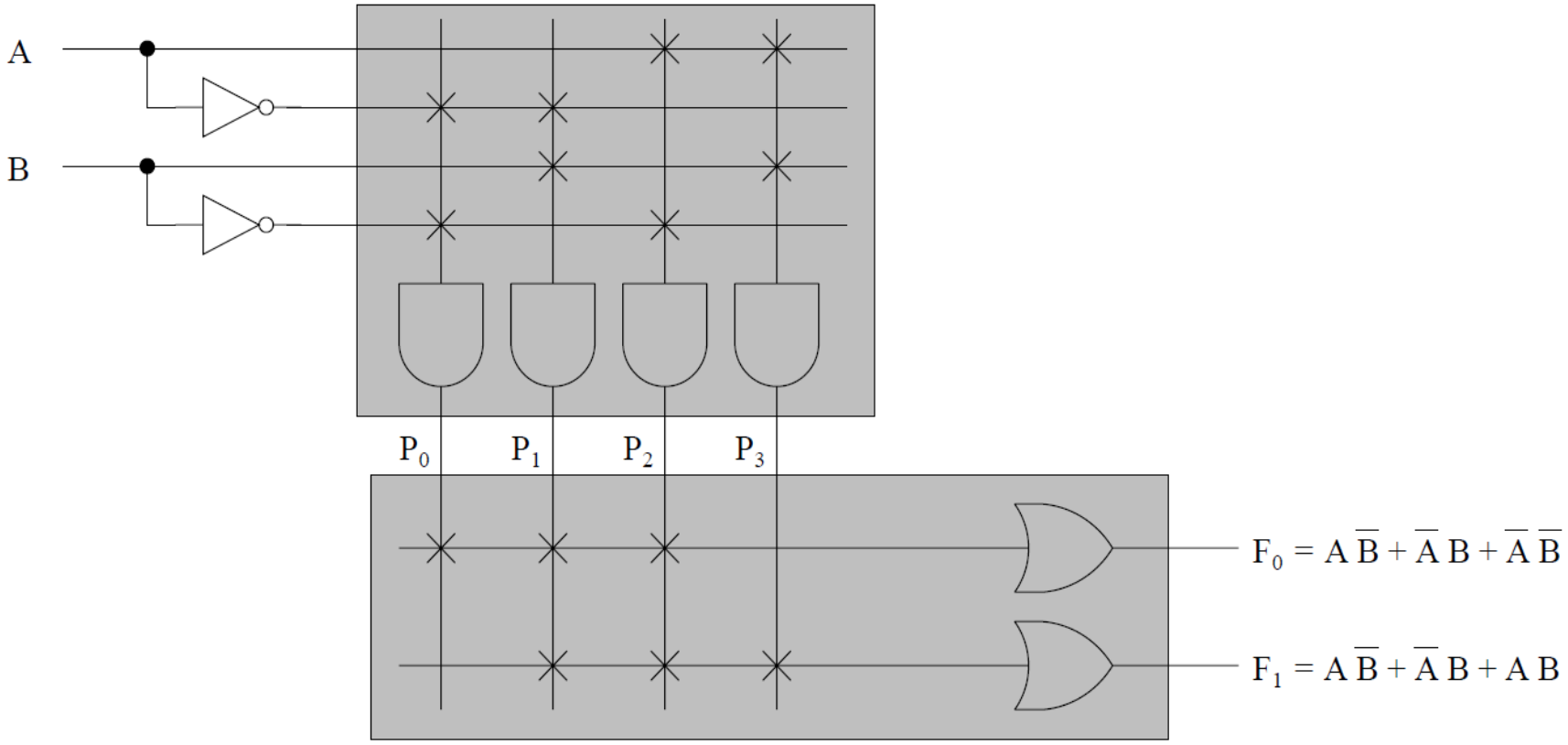
Пример ПНЛЕ (2)



Поједностављена нотација

- Да би се олакшало приказивање ПНЛЕ, уводи се нотација која претпоставља
  - изостављање прекидача из приказа
  - означавање проходних пресека водова са X

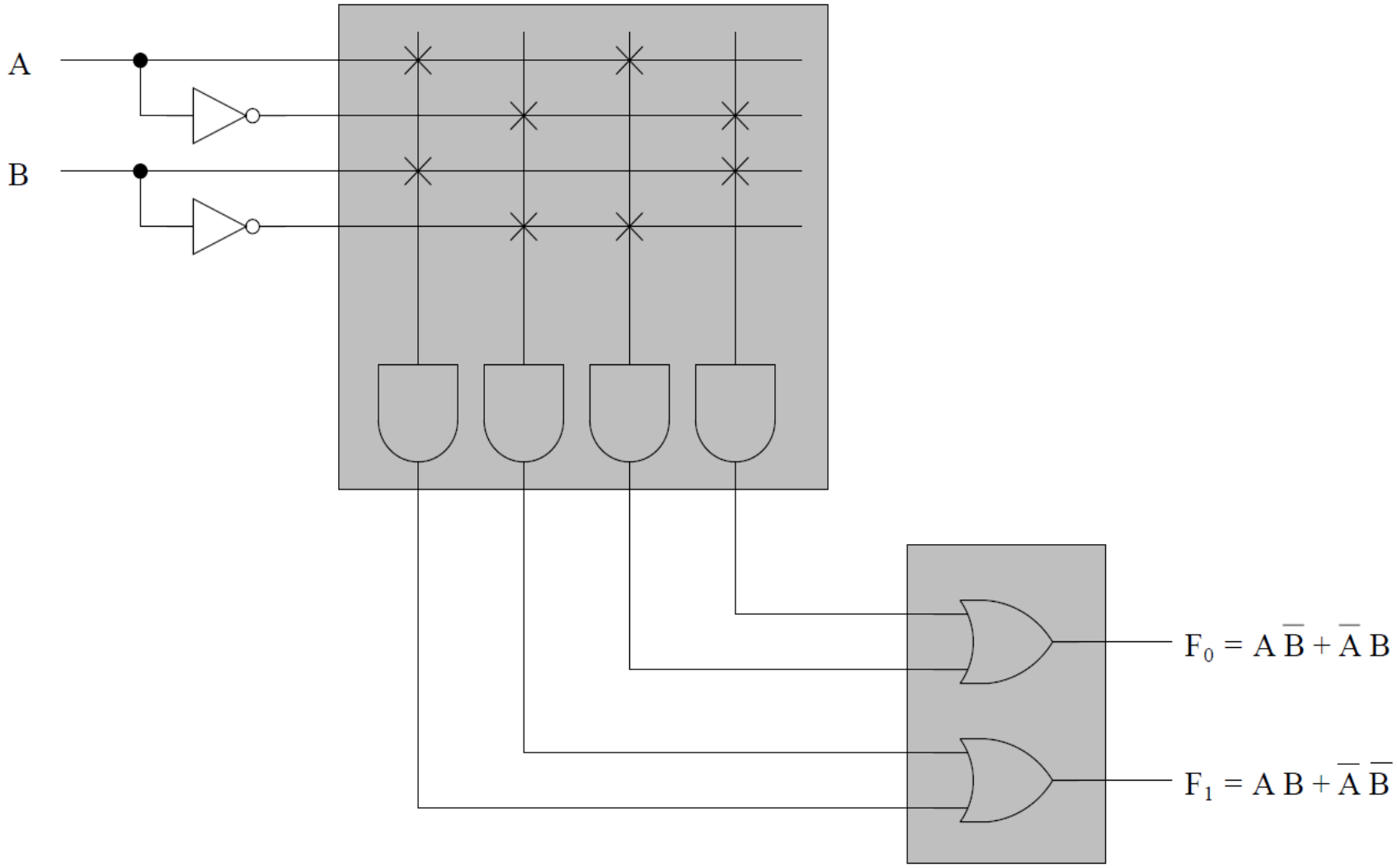
Пример ПНЛЕ (2)



Уређаји са програмибилним низом елемената

- Један од основних проблема у вези ПНЛЕ је висока цена уградње великог броја прекидача:
  - до $2(n + m) \times 2^n$
- Један начин решавања је елиминисање прекидача на улазима ИЛИ елемената
  - фиксирају се везе И и ИЛИ елемената
  - задржава се релативно висок ниво флексибилности
    - извесно нижи него у случају ПНЛЕ
- Такво решење се назива “Уређај са програмибилним низом логичких елемената” (енгл. *Programmable Array Logic Device - PAL*)

Пример

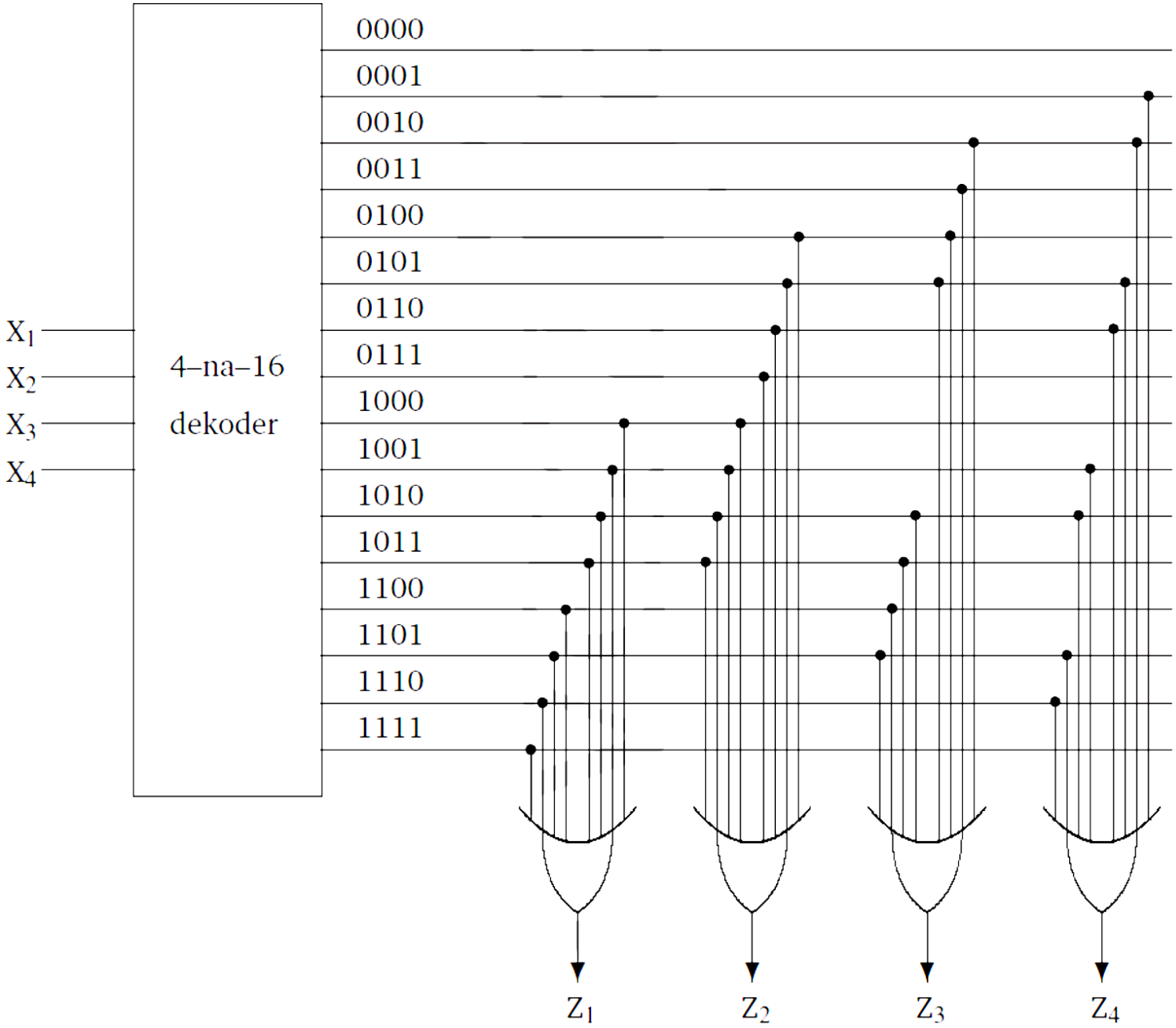


Меморија само за читање

- Комбинаторне мреже се називају и “мреже без меморије”, зато што резултати зависе искључиво од улаза
- Ипак, могу се употребљавати за имплементацију меморија које служе само за читање (*ROM*)
- Ако је дужина адресе *n* битова, а податка *m* битова, онда се имплементира помоћу:
  - декодера *n*-на-2^n, који препознаје адресу
  - низа од *m* ИЛИ елемената, који дају битове податка у зависности од адресе

Пример *ROM*-а

Ulaz				Izlaz			
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0



Литература

- *Ненад Митић,* ***Увод у организацију рачунара****, Математички факултет, 2009.*
- *Sivarama Dandamudi,* ***Fundamentals of Computer Organization and Design****, Springer, 2002.*