08 Multiplexers, Decoders, Demultiplexers

121-139

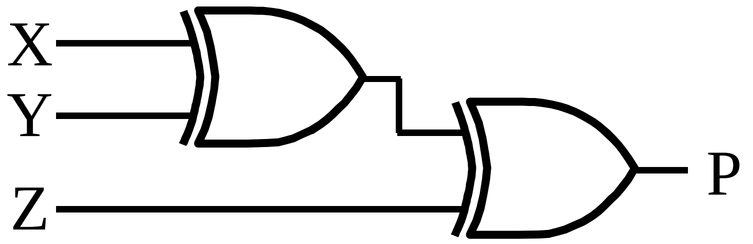
Sections 3.7, 3.8, 3.9

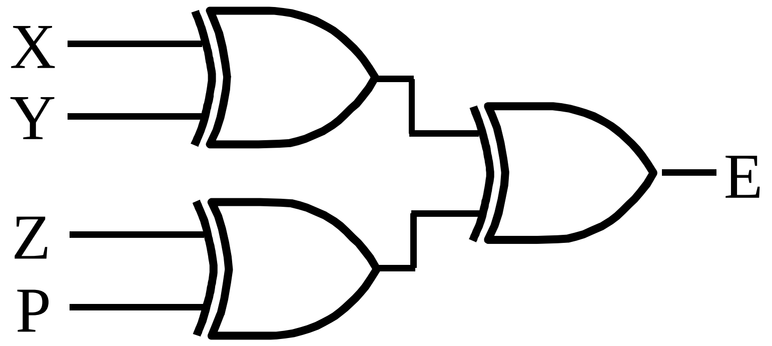
Parity Generators and Checkers

- A parity bit can be added to n-bit code to produce an n + 1 bit code:
  - Add odd parity bit to generate code words with even parity
  - Add even parity bit to generate code words with odd parity

Example: n = 3.

1. Generate even parity code words of length 4 with odd parity generator
2. Check even parity code words of length 4 with odd parity checker





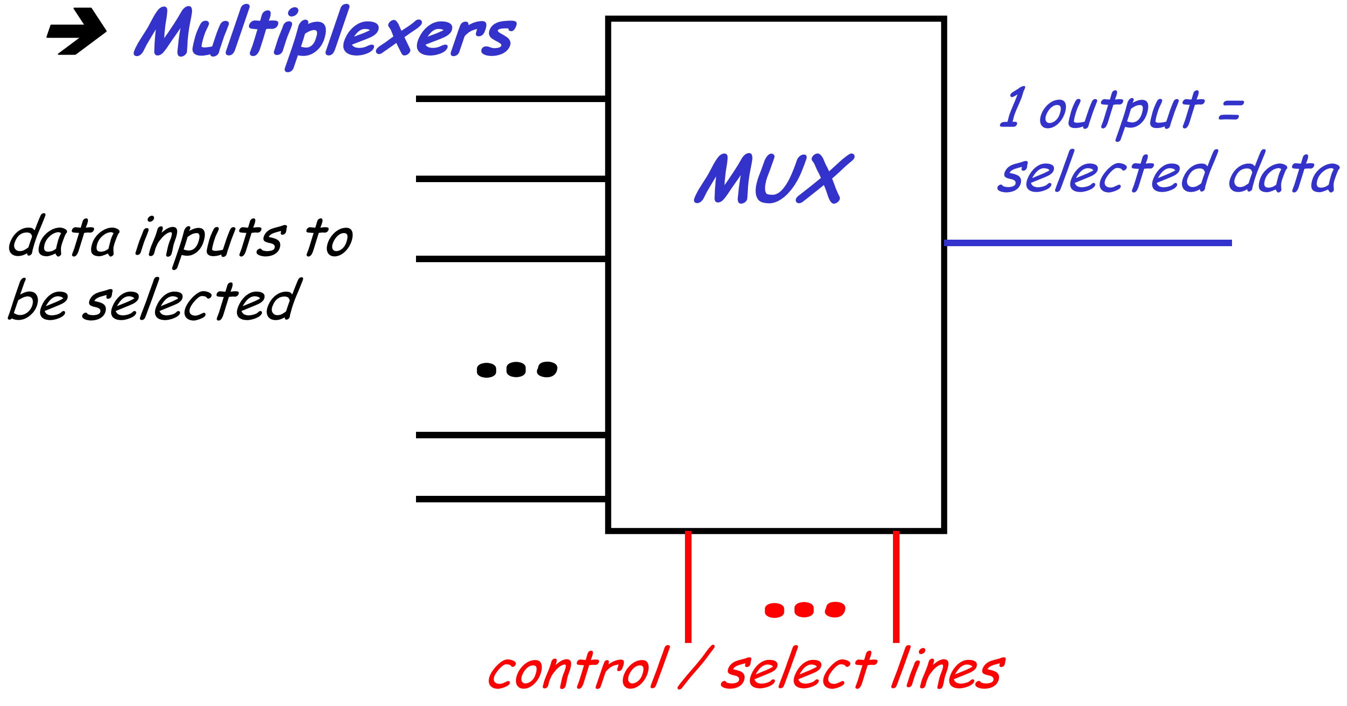
Operation: (X,Y,Z) = (0,0,1) gives (X,Y,Z,P) = (0,0,1,1) and E = 0.

If Y changes from 0 to 1 between generator and checker, then E = 1 indicates an error.

Selecting

- ❖ Selection of data is a critical function
- ❖ Circuits that perform selection have:
  1. A set of data inputs from which to select
  2. A single output
  3. A set of control lines for making the selection

➔ ***Multiplexers***



Multiplexers

131-135

- A multiplexer selects information from an input line and directs the information to an output line
- A typical multiplexer has *n* control inputs (S_{n-1}, … S_0) called *selection inputs*, 2^n information inputs (I_{2^n-1}, … I_0), and one output Y
- A multiplexer can be designed to have *m* information inputs with $m < 2^n$ as well as *n* selection inputs

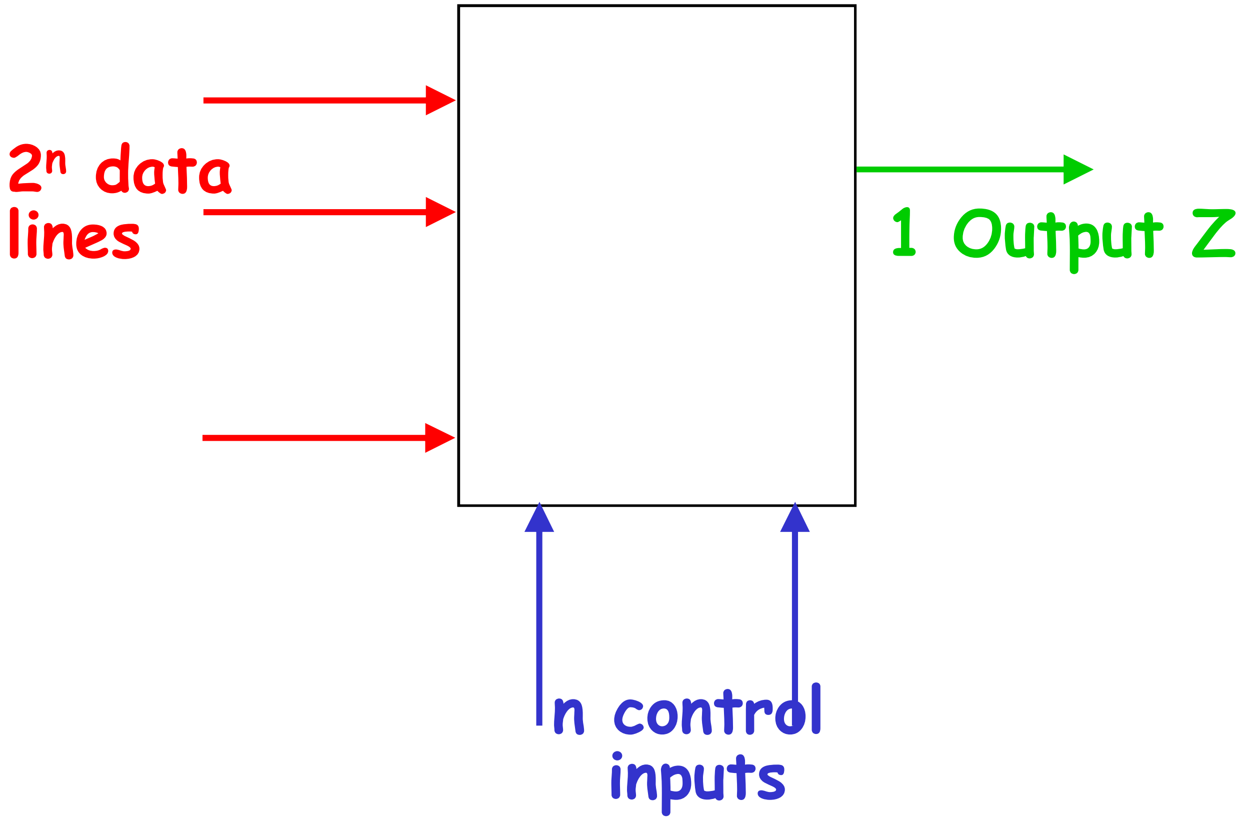
Selecting → Multiplexers

A multiplexer has

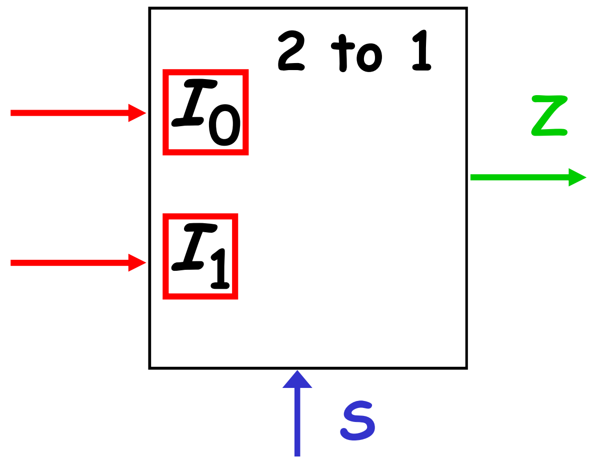
n control inputs (S_{n-1}, … S_0) or *selection inputs,*

2^n data inputs (I_{2^n-1}, … I_0),

and one output Z



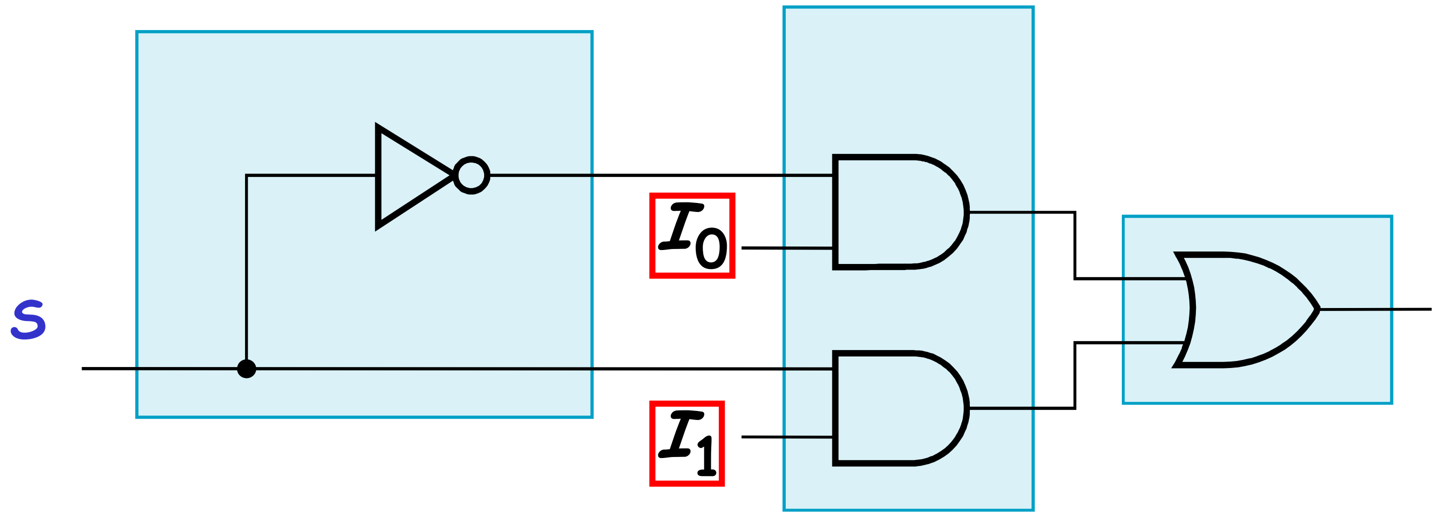
2-to-1-Line Multiplexer



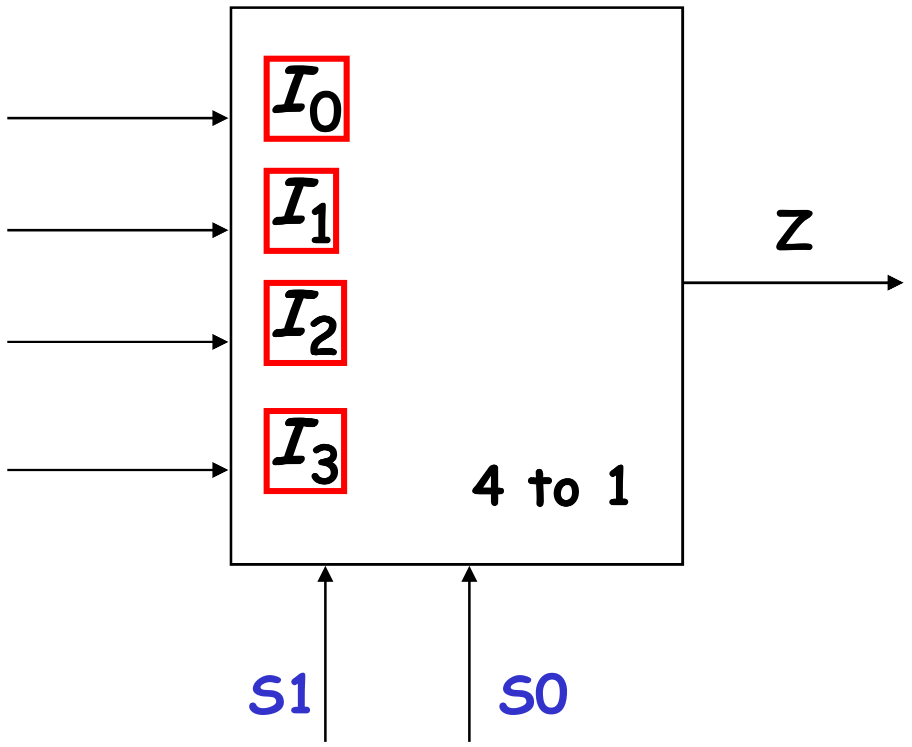
if S = 0, Z gets the $\boxed{I_0}$ signal

if S = 1, Z gets the $\boxed{I_1}$ signal

$$Z = S' I_0 + S I_1$$

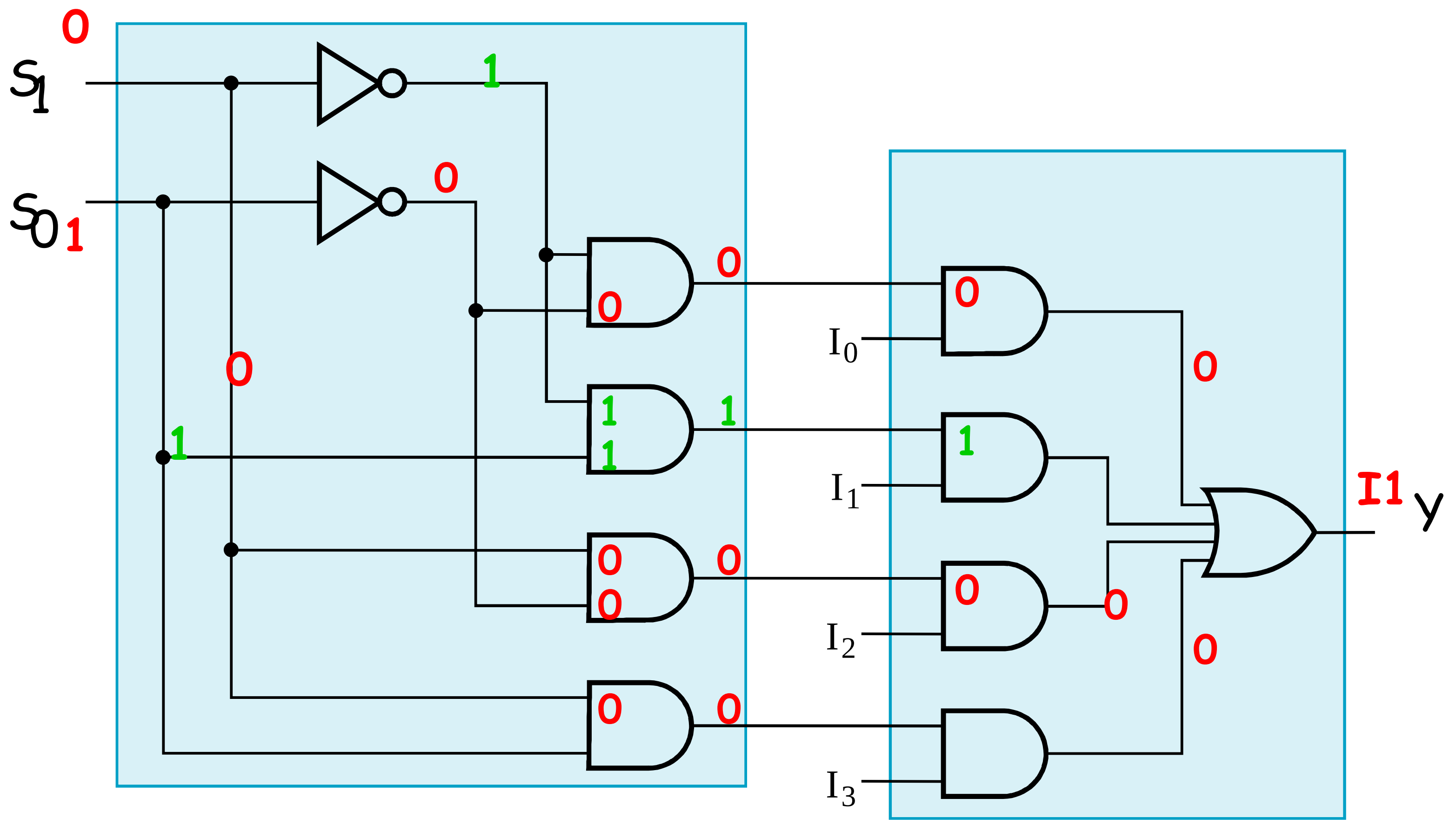


4-to-1-Line Multiplexer

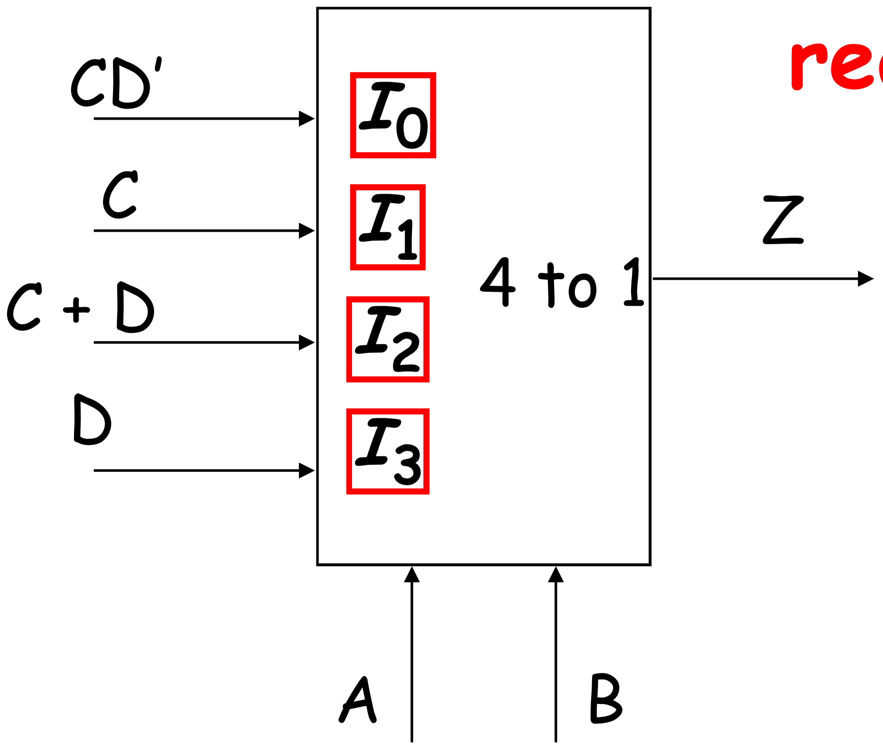


$$Z = S_1'S_0'I_0 + S_1'S_0 I_1 + S_1 S_0'I_2 + S_1 S_0 I_3$$

4-to-1-Line Multiplexer

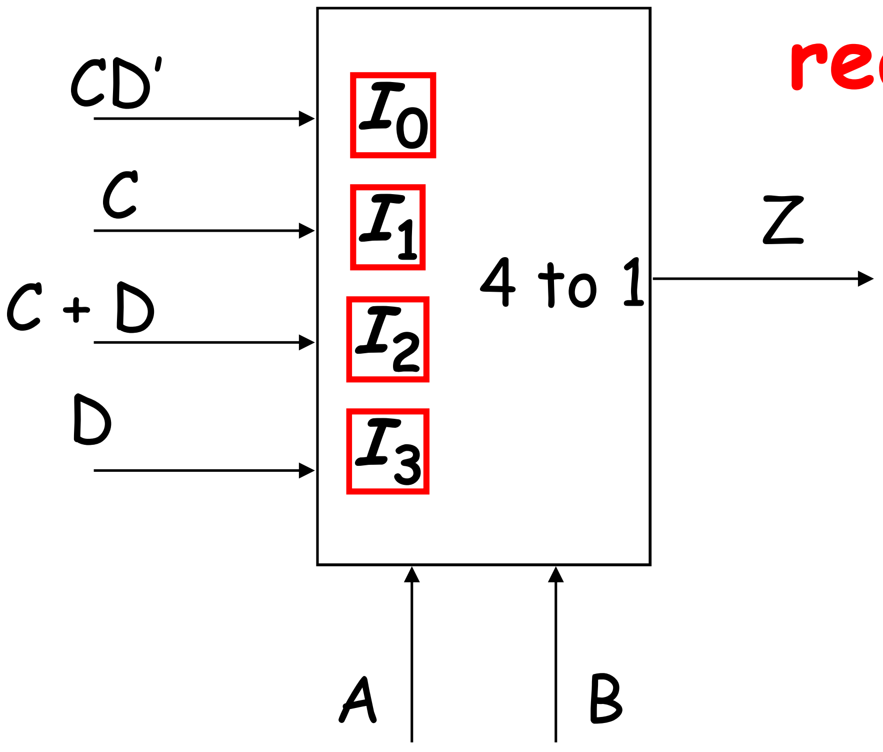


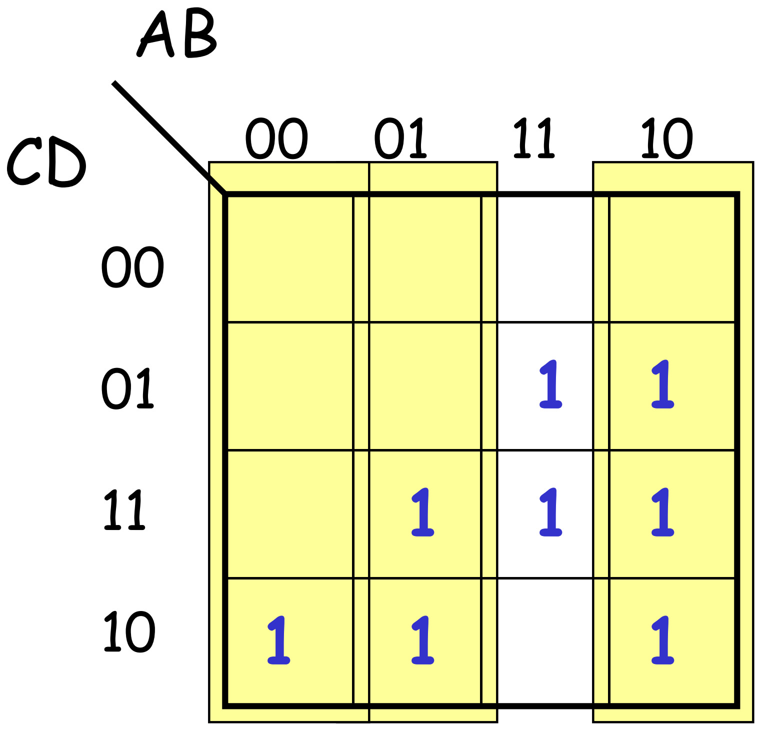
*What function is really implemented?



CD \ AB	00	01	11	10
00				
01				
11				
10				

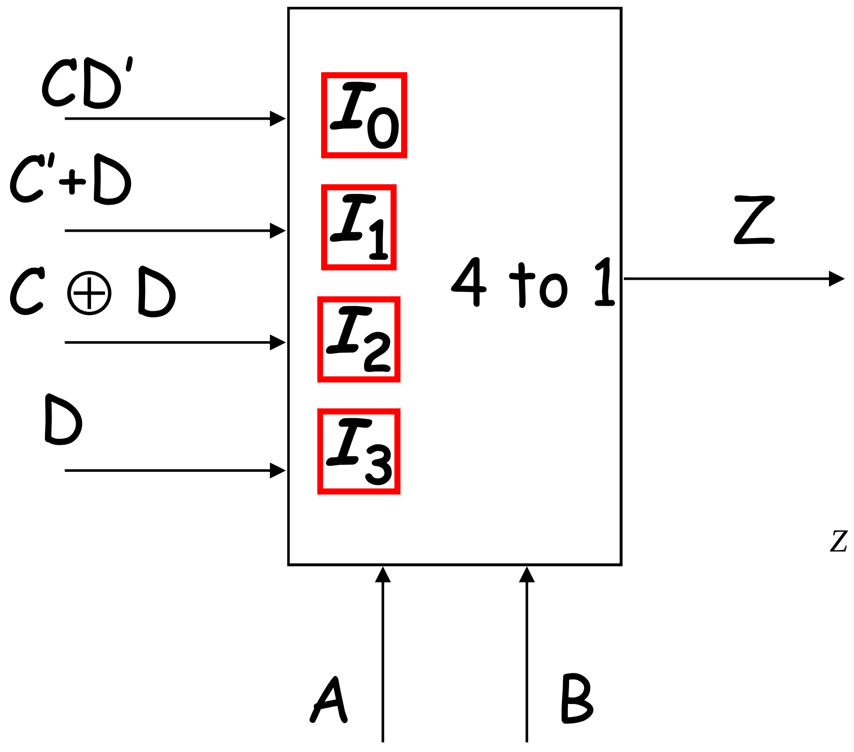
*What function is really implemented?

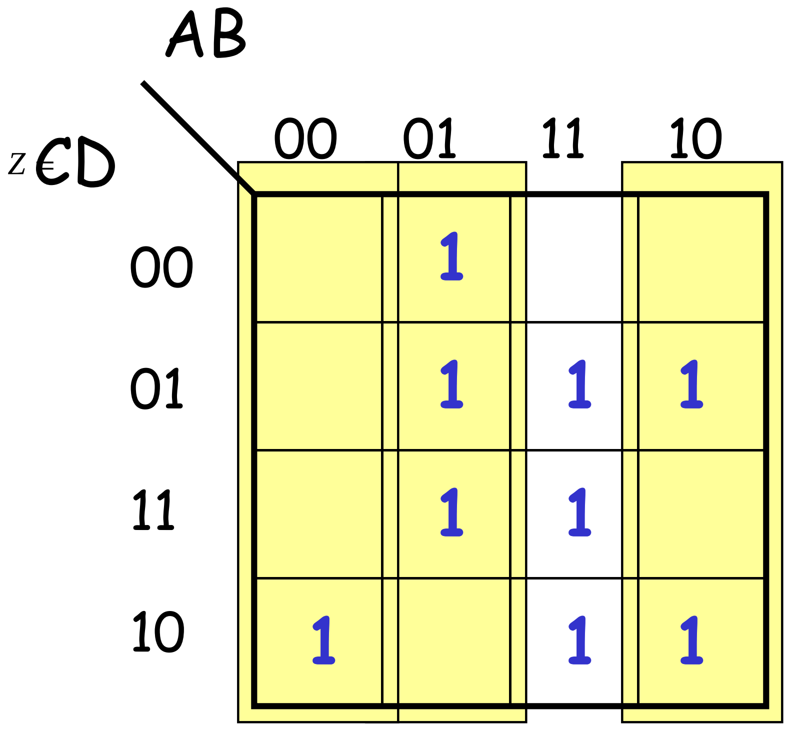




CD \ AB	00	01	11	10
00				
01			1	1
11		1	1	1
10	1	1		1

*Analysis: What function is really implemented?





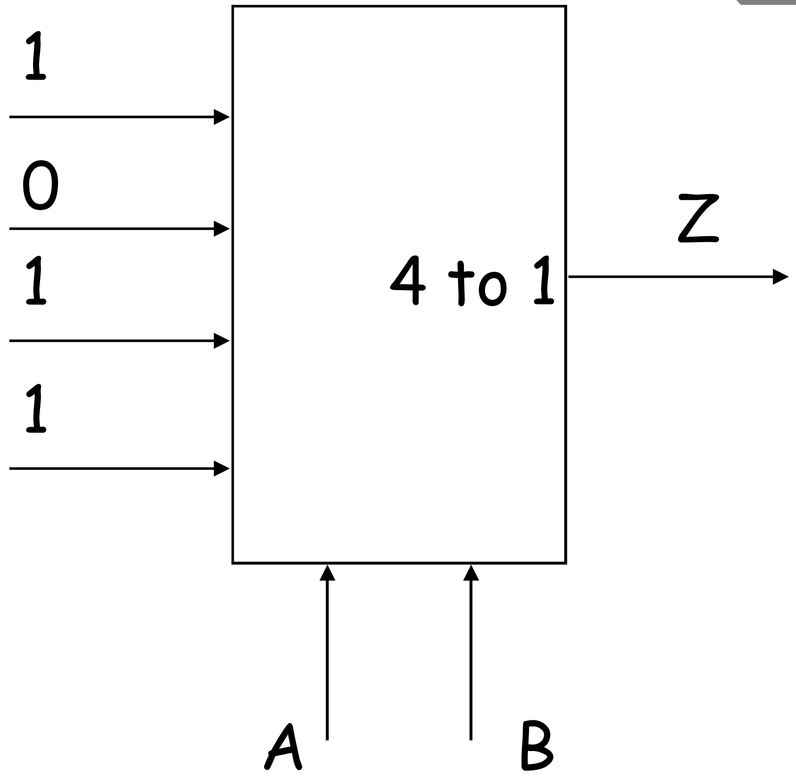
CD \ AB	00	01	11	10
00		1		
01		1	1	1
11		1	1	
10	1		1	1

$$Z = \overline{B}C\overline{D} + AC\overline{D} + \overline{A}B\overline{C} + A\overline{C}D + BD$$

Multiplexer design approach for arbitrary Boolean functions

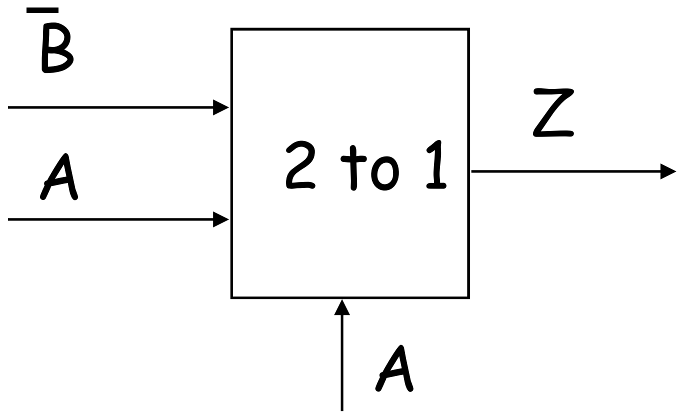
136-139

A	B	F
0	0	1
0	1	0
1	0	1
1	1	1



Implementation with 2-to-1

A	B	F
0	0	1
0	1	0
1	0	1
1	1	1

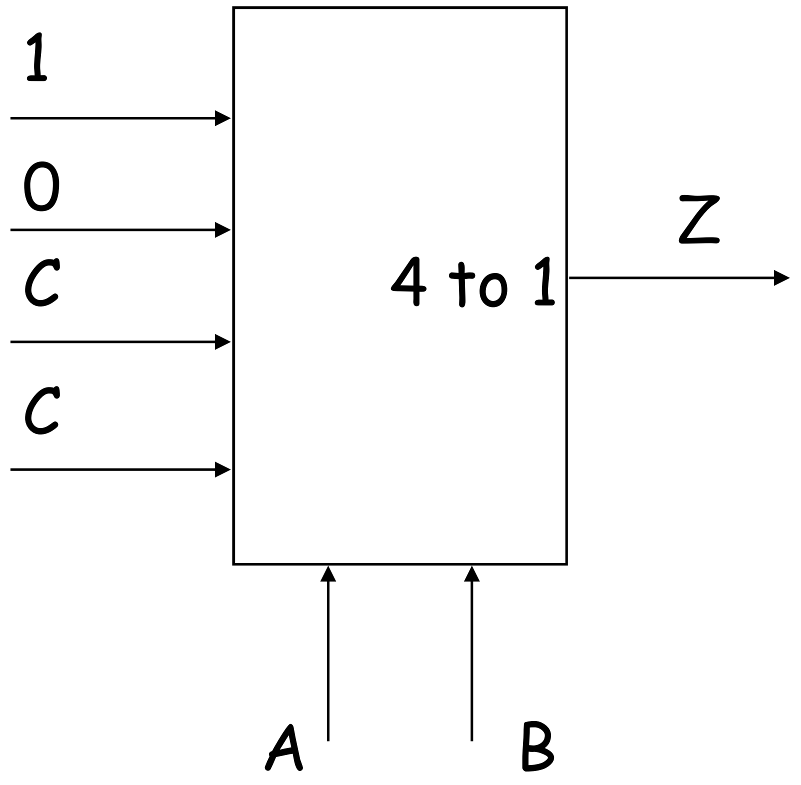


implementation with 4-to-1

AB \ C	0	1
00	1	1
01		
11		1
10		1

$F = A'\,B' + AC$

AB = 00	➔ 1
AB = 01	➔ 0
AB = 11 or 10	➔ C

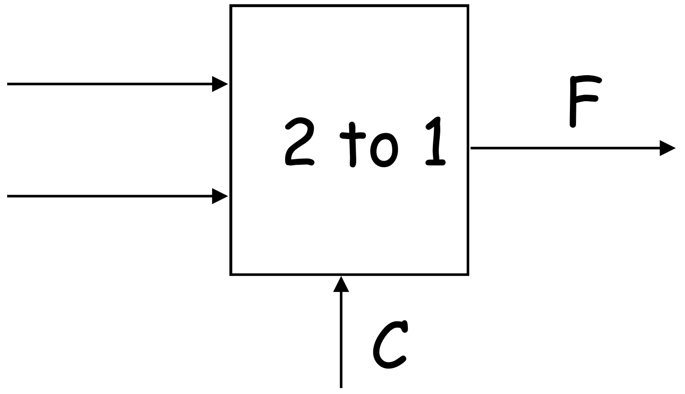


important is choice of the "best" control variables

implementation with 2-to-1

AB \ C	0	1
00	1	1
01		
10		1
11		1

$F = A'B' + AC$



B \ A	0	1
0		
1		

when C=0

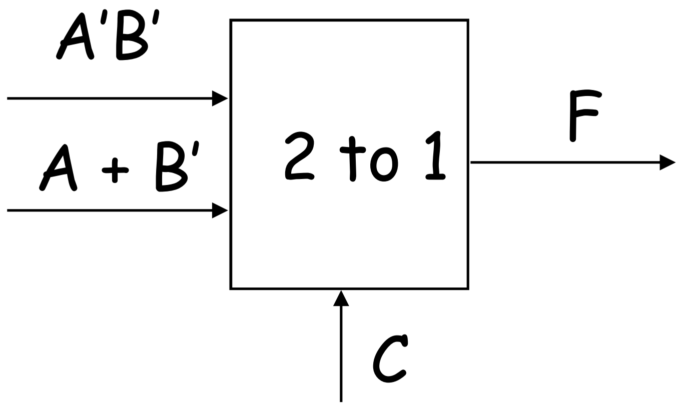
B \ A	0	1
0		
1		

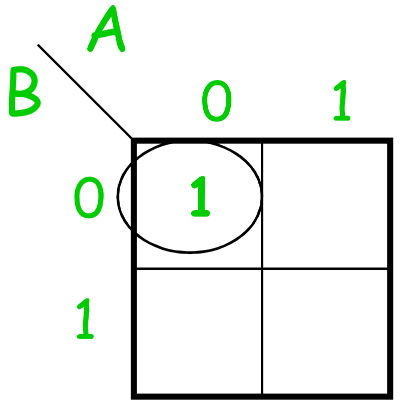
when C=1

implementation with 2-to-1

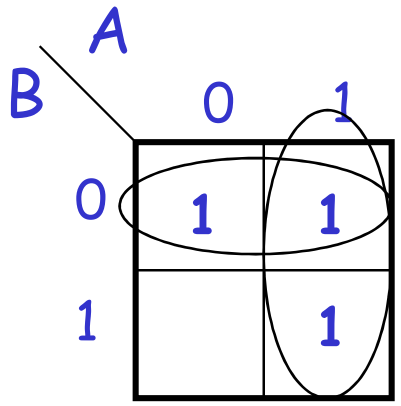
AB \ C	0	1
00	1	1
01		
10		1
11		1

$F = A'B' + AC$

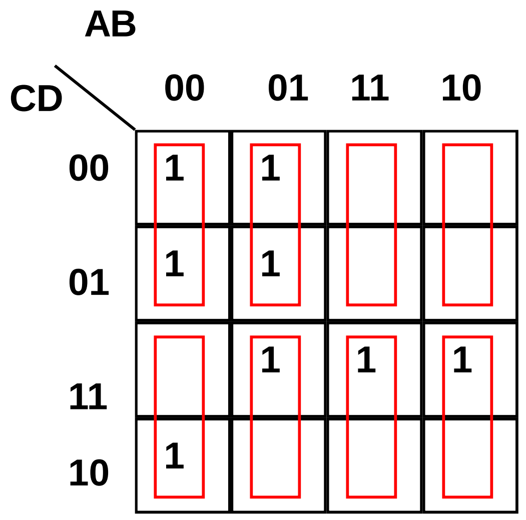




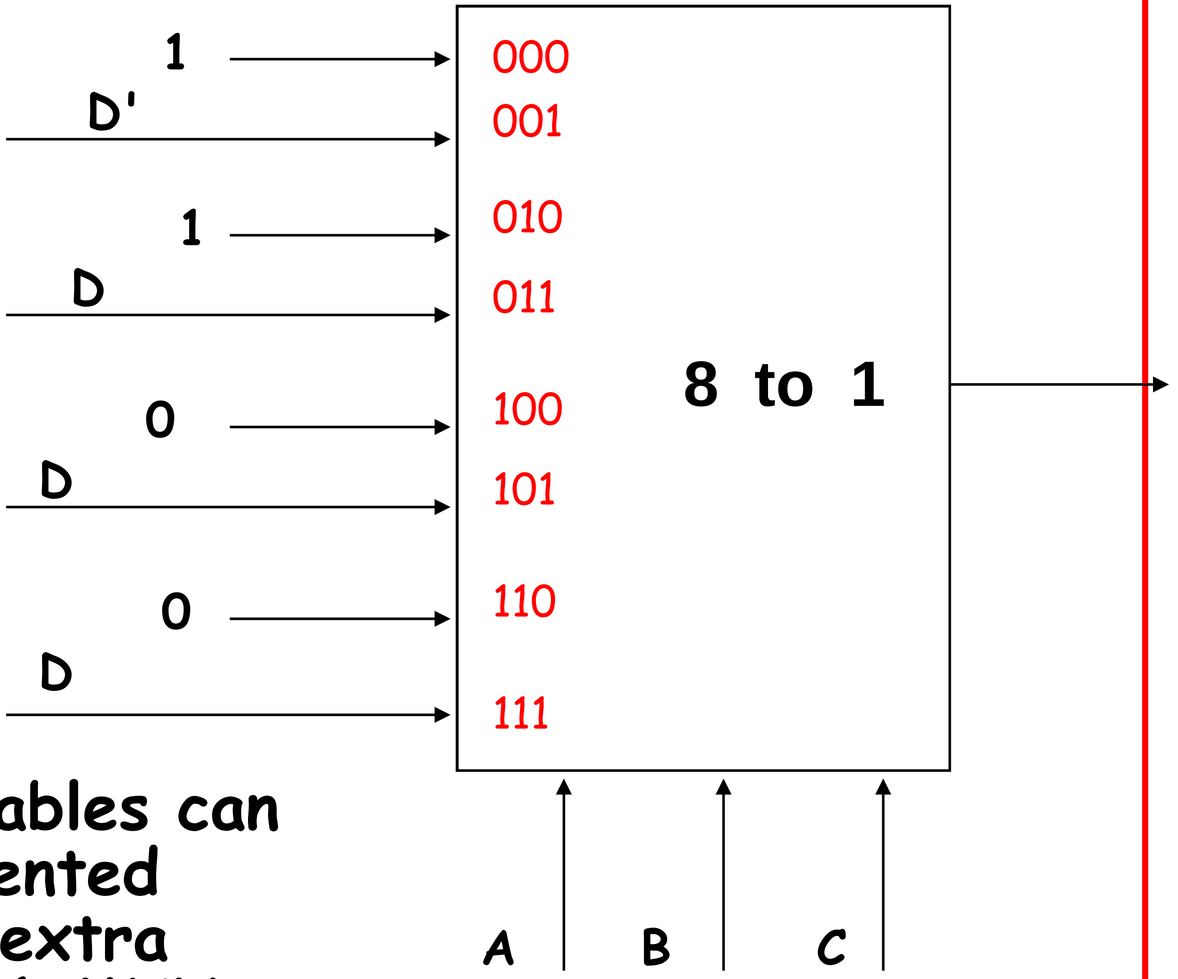
when C=0



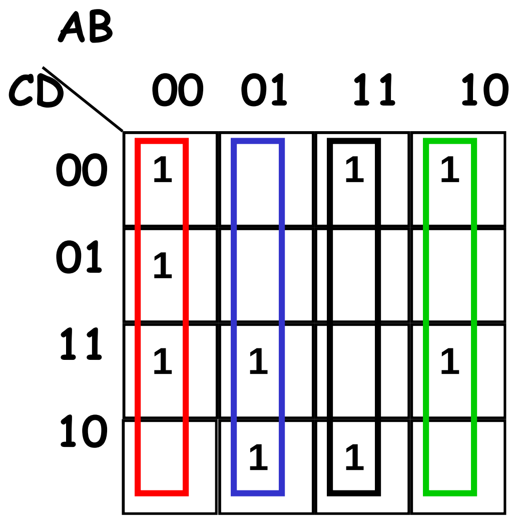
when C=1



choose ABC as controls



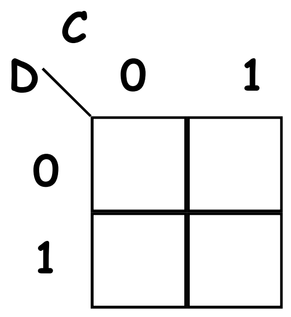
a function of n variables can always be implemented directly, with no extra gates, with an n-1 MUX



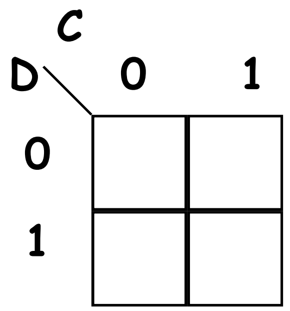
CD \ AB	00	01	11	10
00	1		1	1
01	1			
11	1	1		1
10		1	1	

- choose A, B as controls

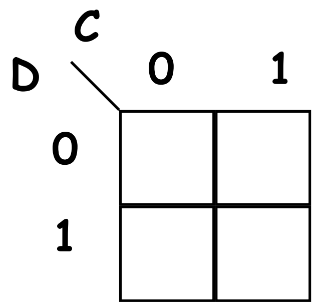
- look at each column as a 2 variable map for CD



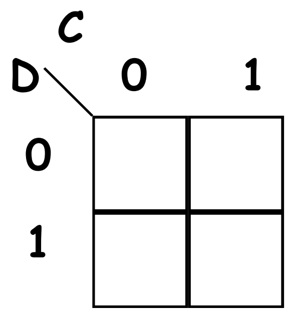
AB=10



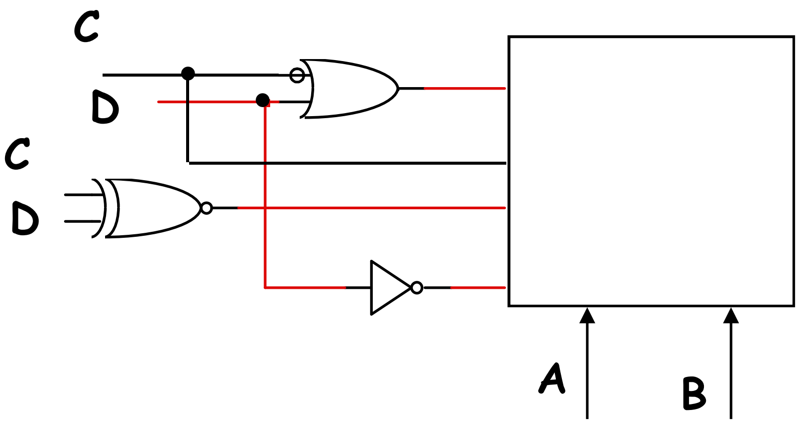
AB=11



AB=00



AB=01



CD \ AB	00	01	11	10
00	1		1	1
01	1			
11	1	1		1
10		1	1	

- choose A, B as controls
- look at each column as a 2 variable map for CD

D \ C	0	1
0	1	
1		1

AB=10

D \ C	0	1
0	1	1
1		

AB=11

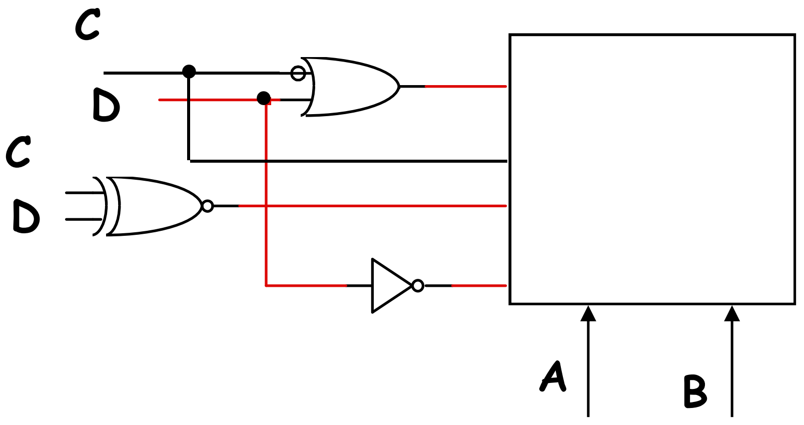
D \ C	0	1
0	1	
1	1	1

AB=00

D \ C	0	1
0		1
1		1

AB=01

$$I_0 = C' + D$$
$$I_1 = C$$
$$I2 = (C \oplus D)'$$
$$I_3 = D'$$



Decoding

- Decoding - the conversion of an *n*-bit input code to an *m*-bit output code with
  $n \le m \le 2^n$ such that each valid code word produces a unique output code
- Circuits that perform decoding are called *decoders*
- Here, functional blocks for decoding are
  - called *n*-to-*m* line decoders, where $m \le 2^n$, and
  - generate 2^n (or fewer) minterms for the *n* input variables

❖ **Used a lot for memory addressing**

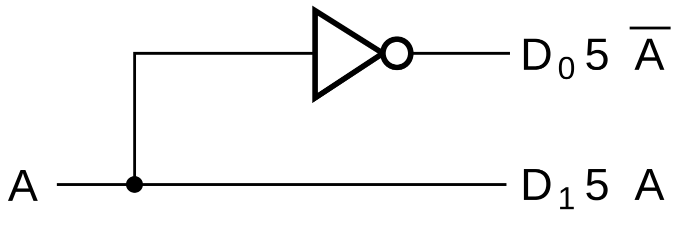
i.e. n-bit address on address bus gets decoded to location x in memory, where 0 <= x < 2^n

Decoder Examples

- 1-to-2-Line Decoder

A	D_0	D_1
0	1	0
1	0	1

(a)

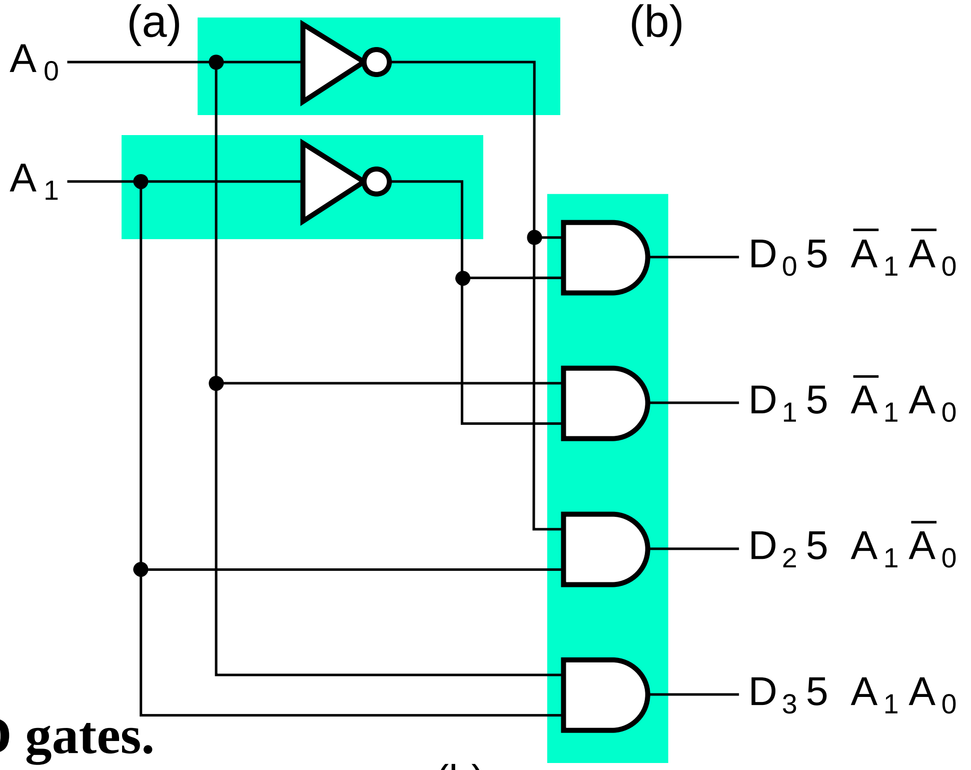


(b)

- 2-to-4-Line Decoder

A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

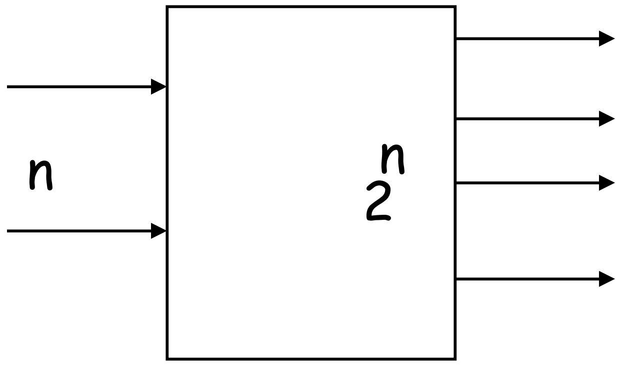
(a)



A_0
A_1
$D_0 5 \overline{A}_1 \overline{A}_0$
$D_1 5 \overline{A}_1 A_0$
$D_2 5 A_1 \overline{A}_0$
$D_3 5 A_1 A_0$

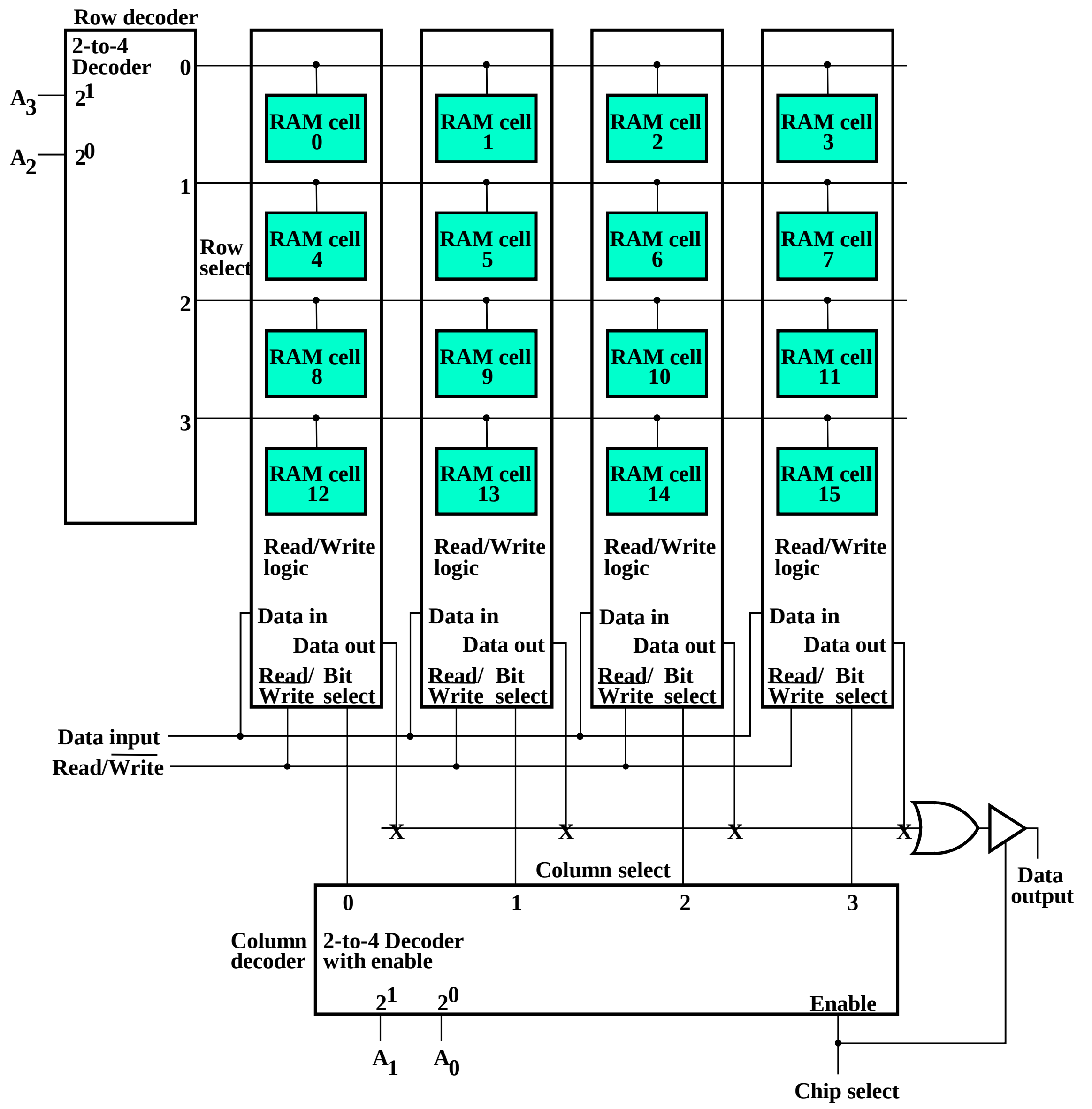
(b)

- **Note that the 2-4-line made up of 2 1-to-2-line decoders and 4 AND gates.**

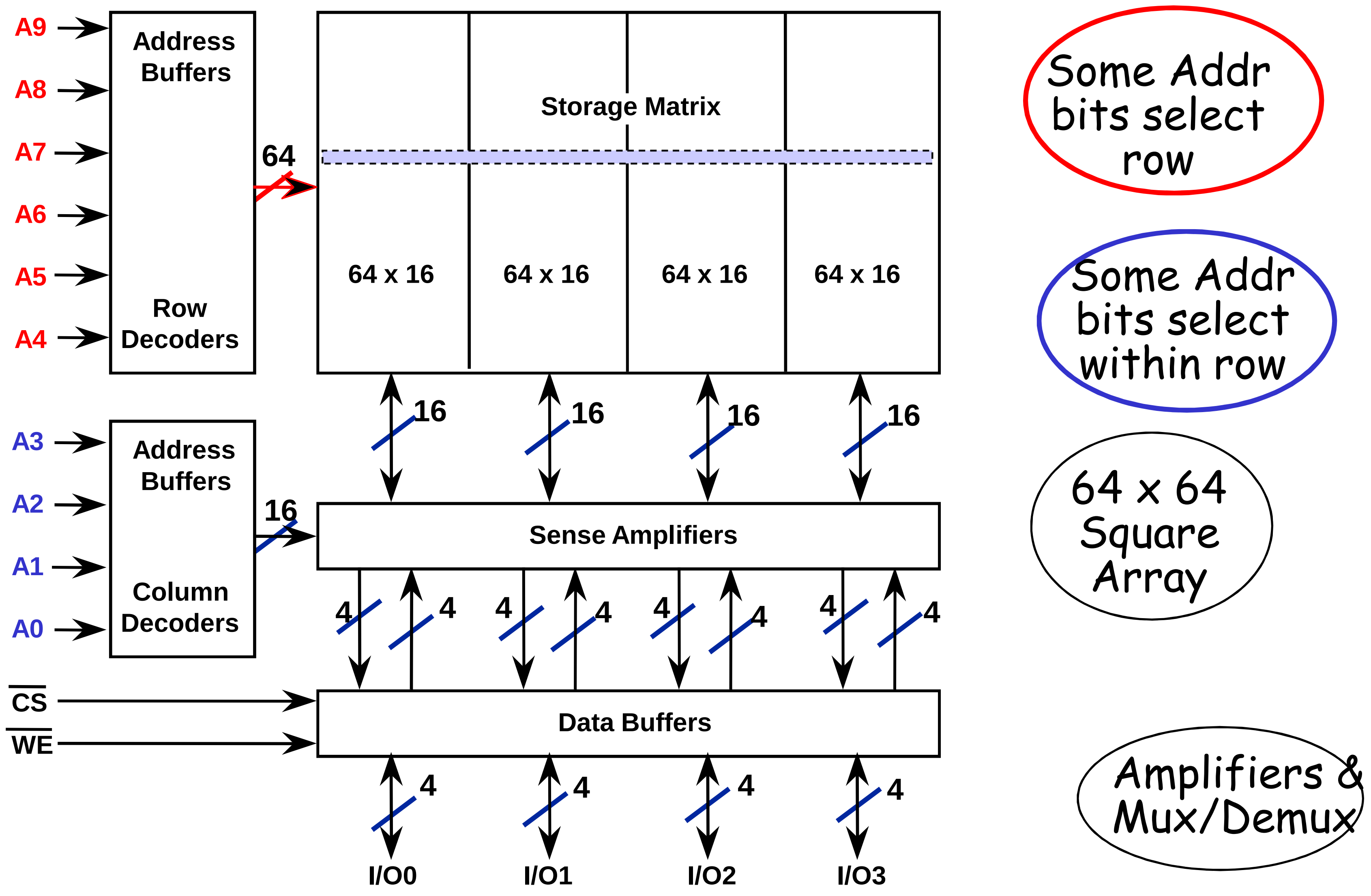


given n-bits on input lines, only 1 of 2^n output lines is activated (high or low)

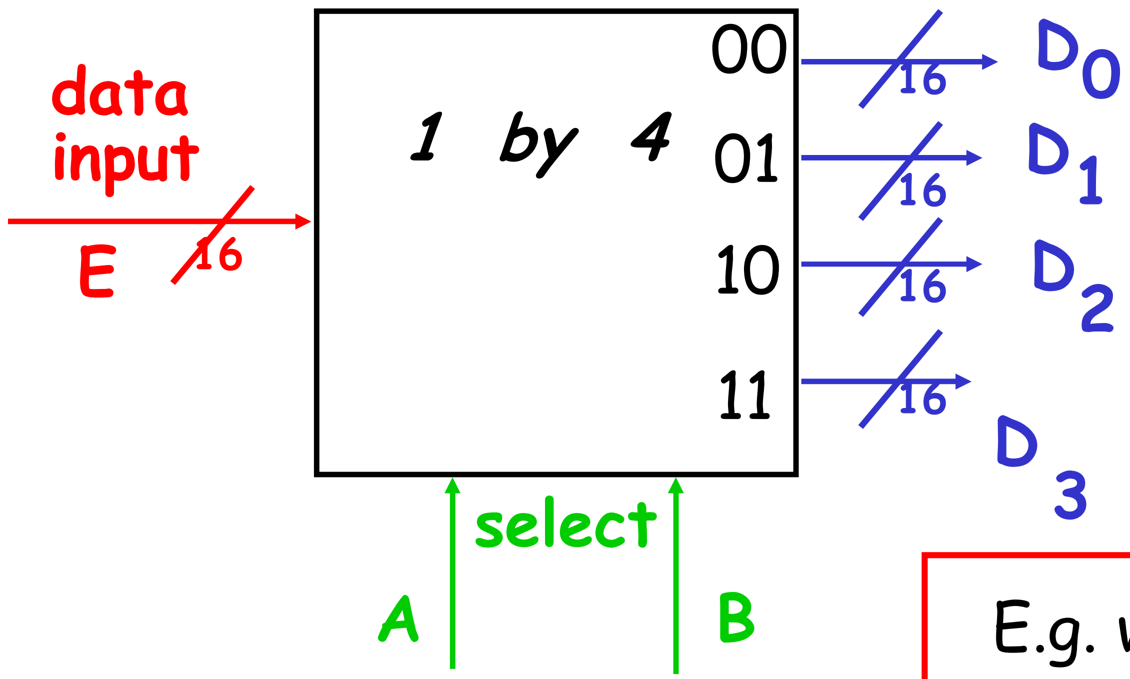
Using Decoders: 16x1 RAM using 4x4 RAM Cell Array



Memory Access Block Diagram



Demultiplexers



D_0 = A′ B′ E

D_1 = A′ B E

D_2 = A B′ E

D_3 = A B E

E.g. when A = B = 1,

D_3 = E

while other outputs = 0

1 data input

2^n outputs

selected by *n* control lines

outputs NOT selected are = 0