Functions of Combinational logic

EIC 0844091

Digital Circuit and Logic Design

Associate Prof. Luo Jie

Huazhong University of Science & Technology

Functions of Combinational logic

- A practical combinational circuit may be vary complex. Thus, most real combinational logic design problems are too large and difficult.
- How do we design such a complex logic circuit?
- In the first place, the key is structured thinking. A complex circuit or system should be divided into some smaller subsystems or modules.
- And then, we can perform these subsystems or modules by using same specific function modules.
- In this lecture, we will take several specific function modules those are used frequently.

Presentation Outline

- Encoders
- Decoders
- Multiplexers (Data selectors)
- Demultiplexers
- Comparators
- Adders

Encoders

- Decimal-to-BCD encoder
- Priority encoder
- 8-line-to-3-line priority encoder (74HC148)

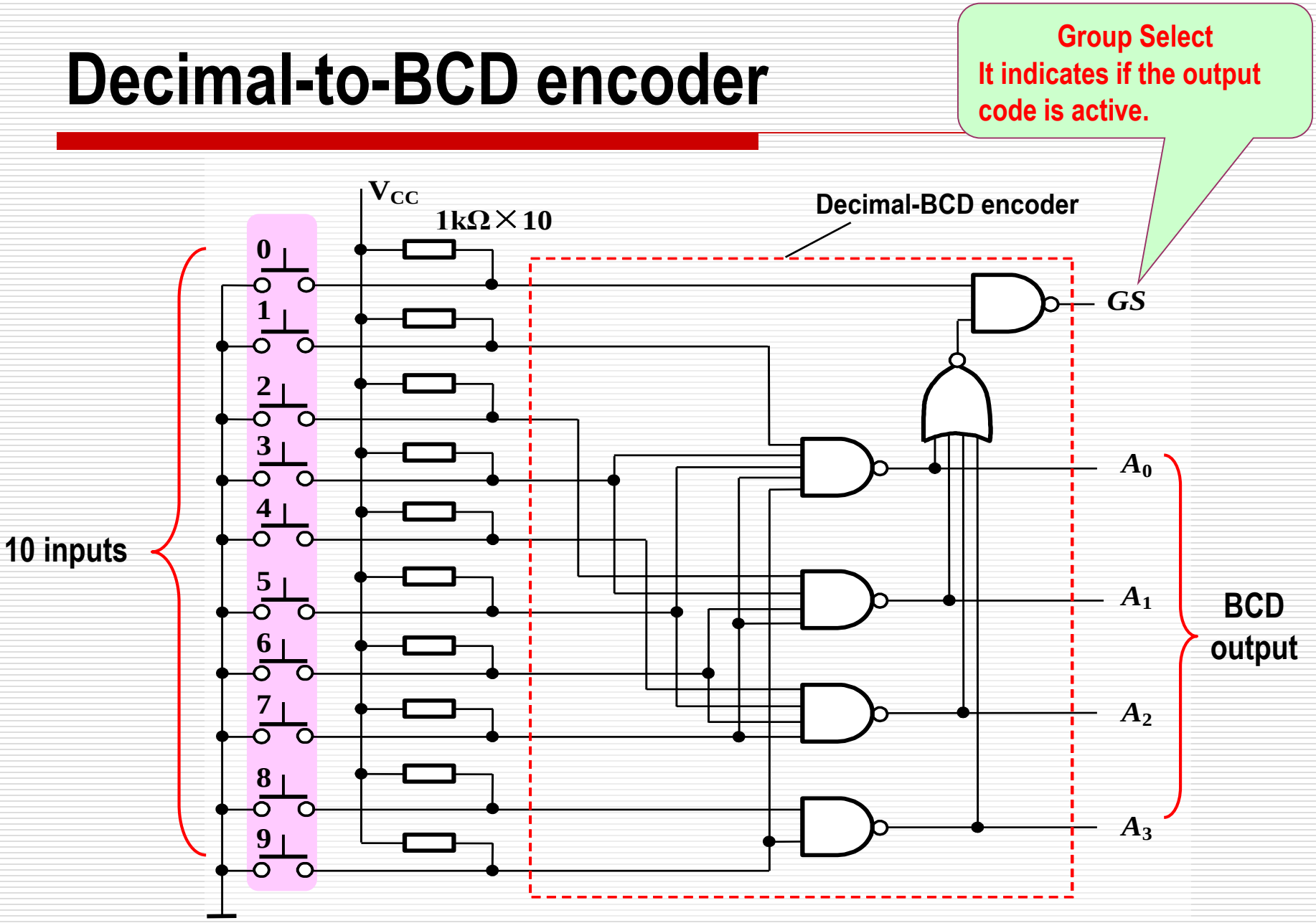
Decimal-to-BCD encoder

Encoder

It is the device that can perform the process of converting from familiar symbols or numbers to a coded format.

This process is called encoding.

Decimal-to-BCD encoder



Decimal-to-BCD encoder

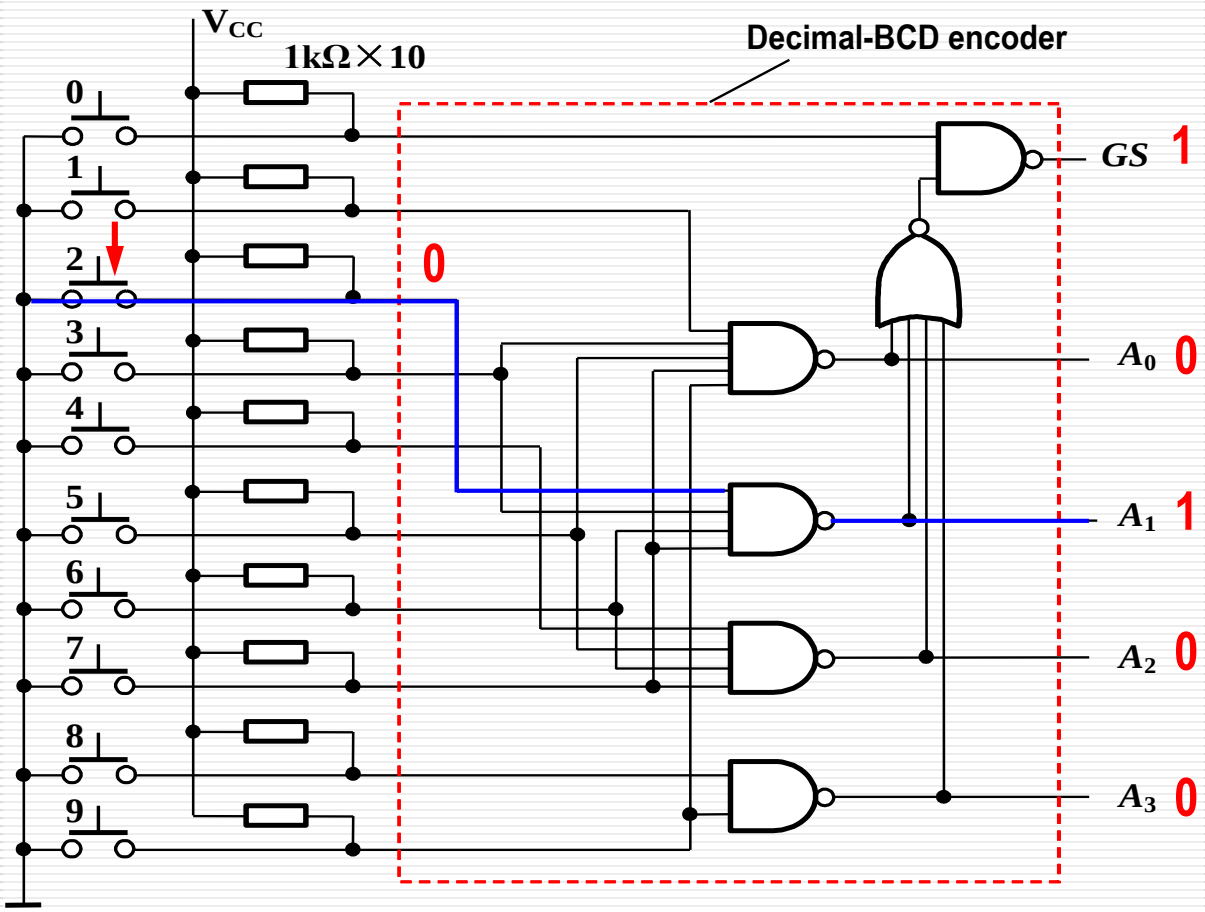
Analyzing the function

When the key is depressed, corresponding input line is LOW, Or else it is HIGH

- When the key 2 is depressed

Output code: $A_3A_2A_1A_0 = 0010$

$GS = 1$



Decimal-to-BCD encoder

Analyzing the function

When the key is depressed, corresponding input line is LOW, Or else it is HIGH

- When the key 2 is depressed

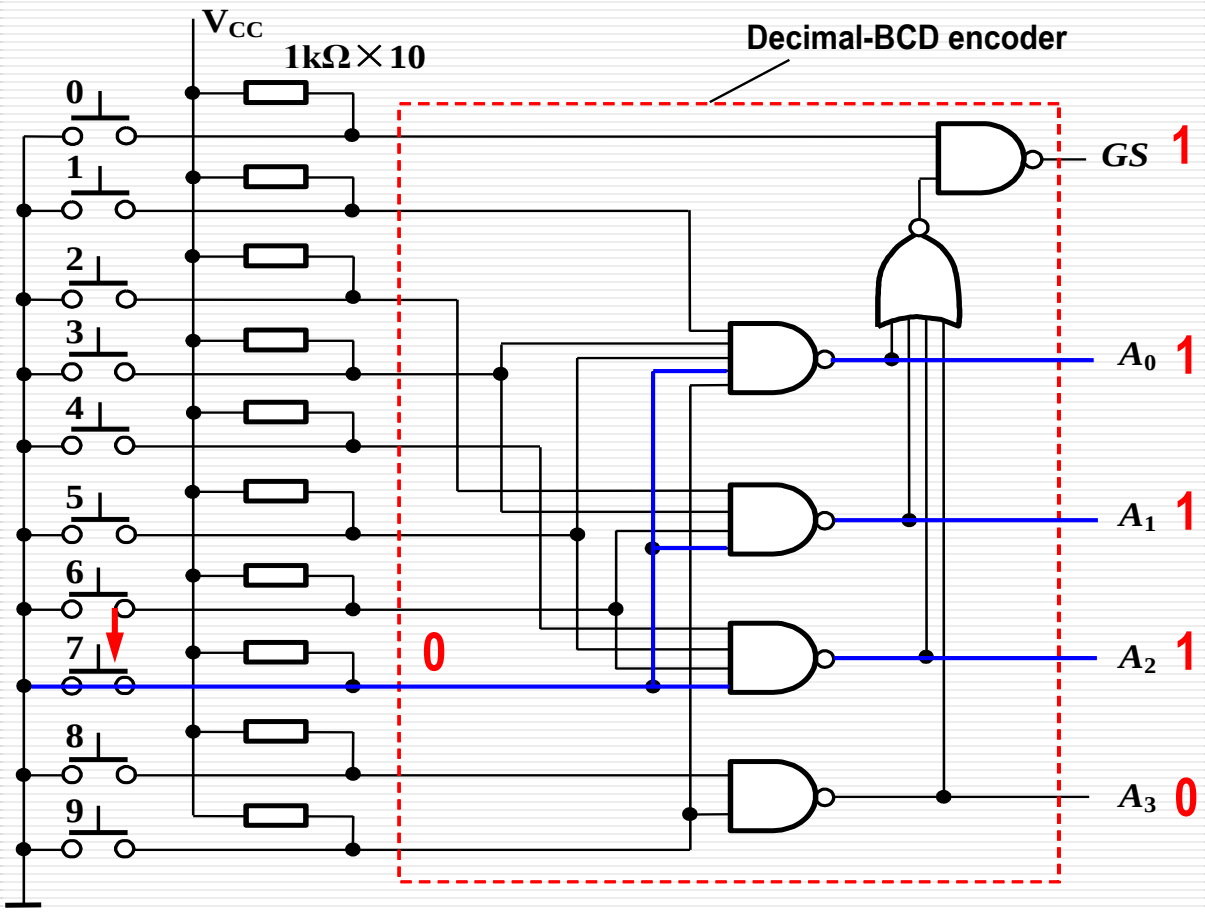
  Output code: $A_3A_2A_1A_0 = 0010$

  $GS = 1$

- When the key 7 is depressed

  Output code: $A_3A_2A_1A_0 = 0111$

  $GS = 1$



Decimal-to-BCD encoder

Analyzing the function

When the key is depressed, corresponding input line is LOW, Or else it is HIGH

- When the key 2 is depressed

  Output code: $A_3A_2A_1A_0 = 0010$

  $GS = 1$

- When the key 7 is depressed

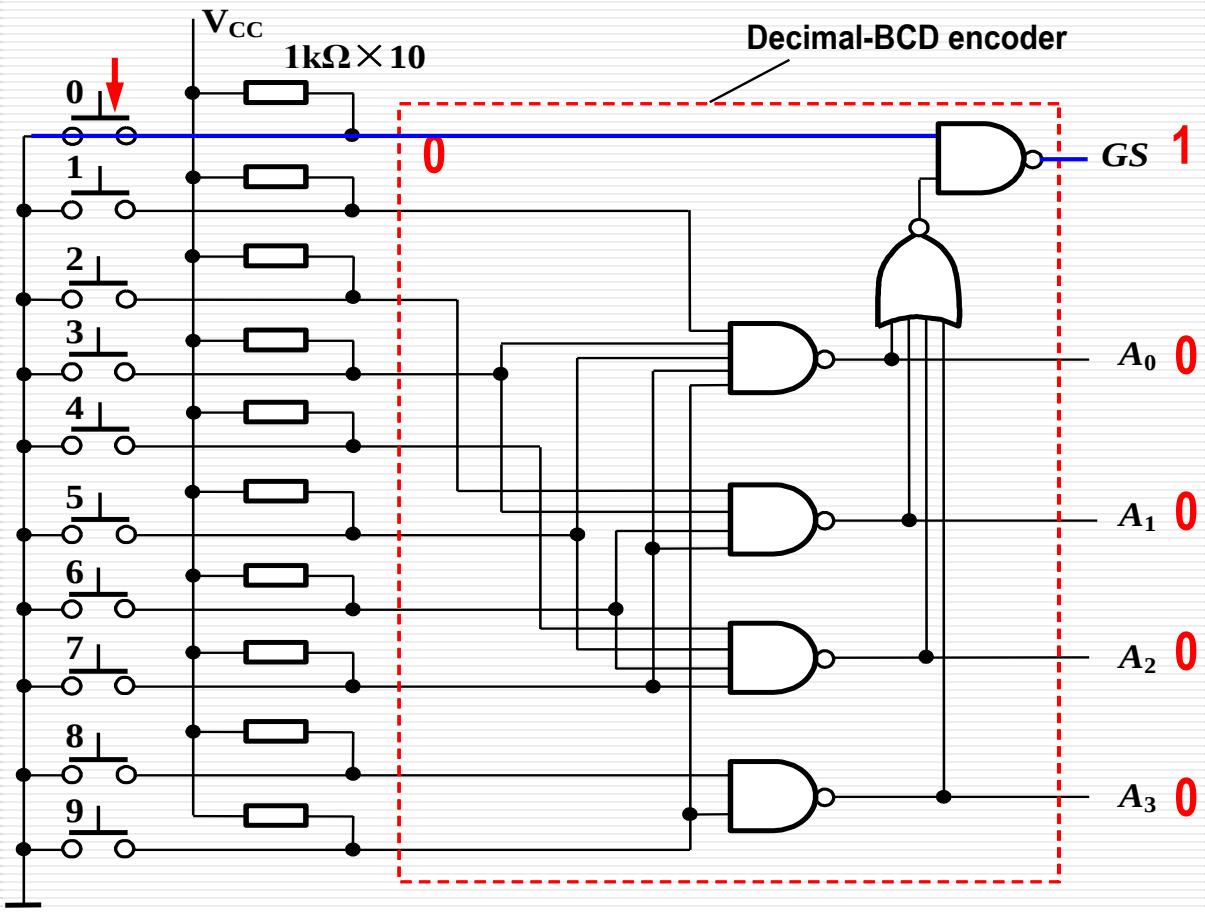
  Output code: $A_3A_2A_1A_0 = 0111$

  $GS = 1$

- When the key 0 is depressed

  Output code: $A_3A_2A_1A_0 = 0000$

  $GS = 1$



Decimal-to-BCD encoder

Analyzing the function

When the key is depressed, corresponding input line is LOW, Or else it is HIGH

- When the key 2 is depressed

  Output code: $A_3A_2A_1A_0 = 0010$

  $GS = 1$

- When the key 7 is depressed

  Output code: $A_3A_2A_1A_0 = 0111$

  $GS = 1$

- When the key 0 is depressed

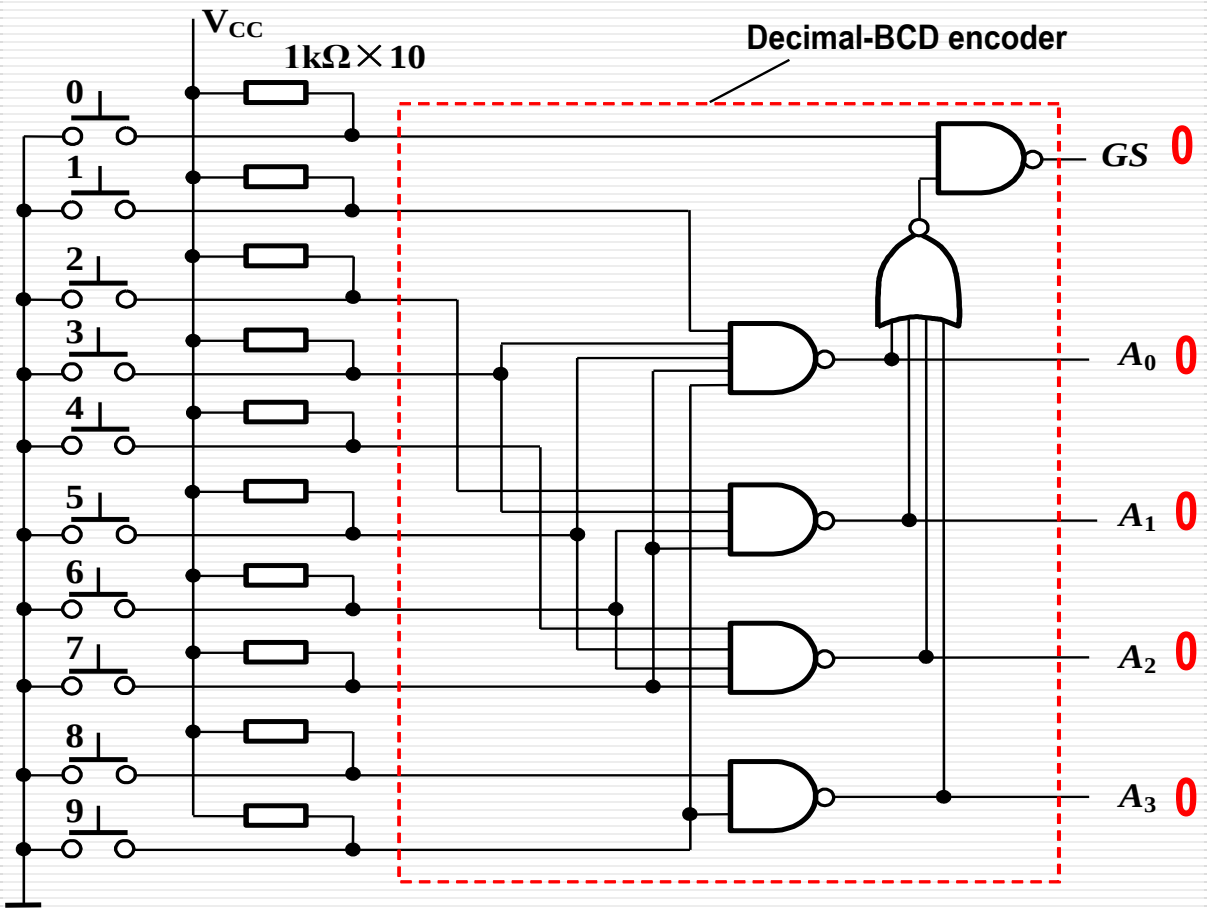
  Output code: $A_3A_2A_1A_0 = 0000$

  $GS = 1$

- When none of keys is depressed

  Output code: $A_3A_2A_1A_0 = 0000$

  $GS = 0$

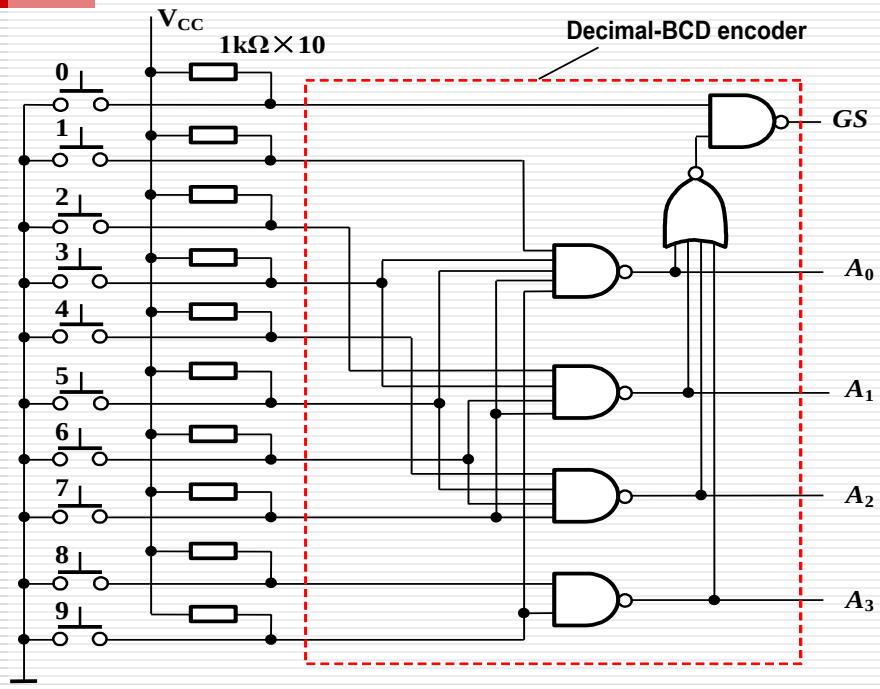


Decimal-to-BCD encoder

Analyzing the function

Function table

inputs										outputs				
0	1	2	3	4	5	6	7	8	9	A_3	A_2	A_1	A_0	GS
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0	1	0	0	1	1
1	1	1	1	1	1	1	1	0	1	1	0	0	0	1
1	1	1	1	1	1	1	0	1	1	0	1	1	1	1
1	1	1	1	1	1	0	1	1	1	0	1	1	0	1
1	1	1	1	1	0	1	1	1	1	0	1	0	1	1
1	1	1	1	0	1	1	1	1	1	0	1	0	0	1
1	1	1	0	1	1	1	1	1	1	0	0	1	1	1
1	1	0	1	1	1	1	1	1	1	0	0	1	0	1
1	0	1	1	1	1	1	1	1	1	0	0	0	1	1
0	1	1	1	1	1	1	1	1	1	0	0	0	0	1



The active input level is LOW.

Why is the GS required ?

What is happen when more then one keys are depressed ?

Decimal-to-BCD encoder

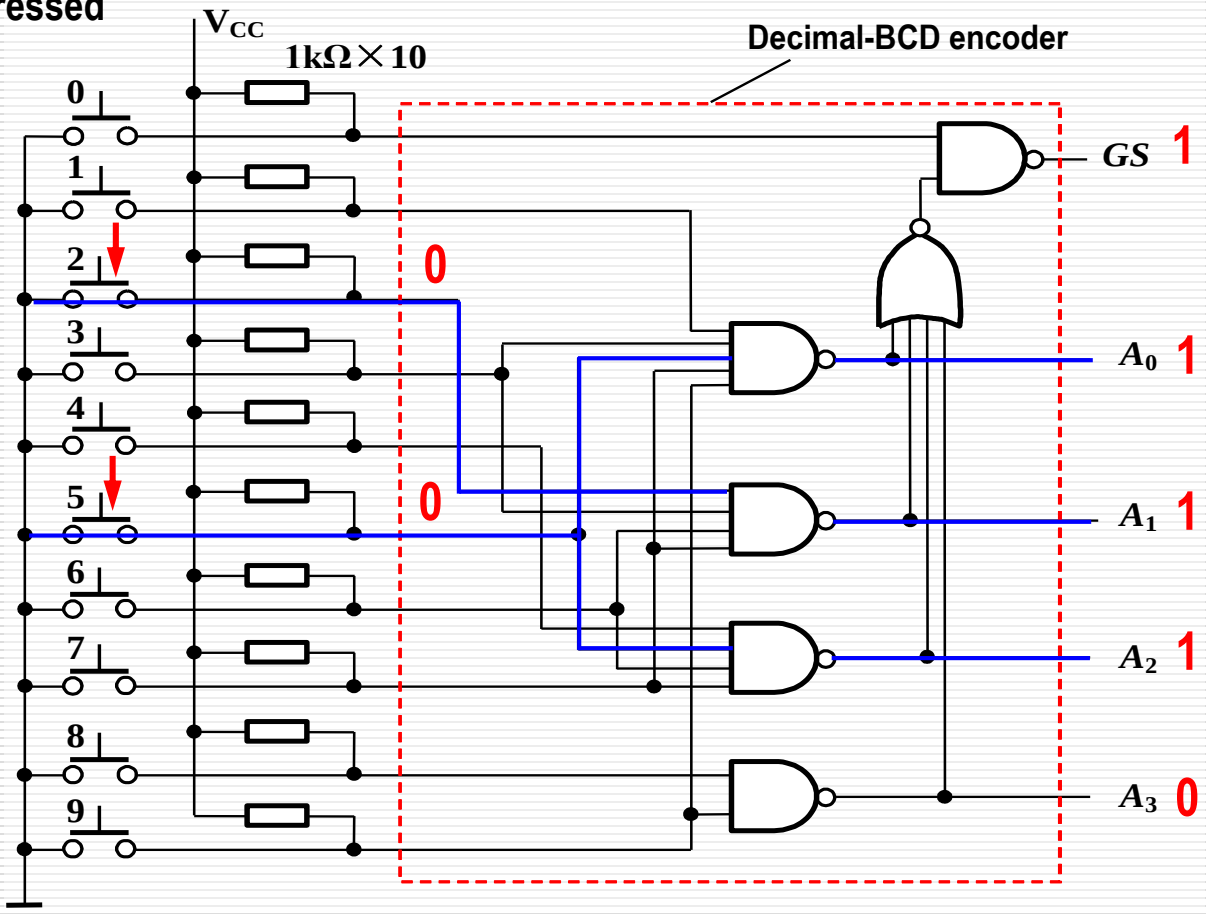
What is happen when more then one keys are depressed ?

- When the keys 2 and 5 are depressed

Output code: $A_3A_2A_1A_0 = 0111$

$GS = 1$

Wrong code



Next item we are going to discuss

- Decimal-to-BCD encoder
- Priority encoder
- 8-line-to-3-line priority encoder (74HC148)

Priority encoder

- The priority encoder is such encoder which inputs are assigned priority, so that when multiple inputs are active, it will only produce the code for the highest-order input and ignore any other lower-order active inputs.

Priority encoder

Decimal-to-BDC priority encoder

The code 0101 indicates input 5

The code 1001 indicates input 9

Which is the highest-priority input ?

We can derive the output expressions from this table and draw logic diagram.

$$A_3 = \overline{9} + 9\overline{8}$$

$$A_2 = 98\overline{7} + 987\overline{6} + 9876\overline{5} + 98765\overline{4}$$

...

Note:

In the expressions, 9,8,7… only represent the input variables.

truth table for priority encoder

inputs										outputs				
0	1	2	3	4	5	6	7	8	9	A_3	A_2	A_1	A_0	*GS*
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
x	x	x	x	x	0	x	x	x	0	1	0	0	1	1
x	x	x	x	x	x	x	x	0	1	1	0	0	0	1
x	x	x	x	x	x	x	0	1	1	0	1	1	1	1
x	x	x	x	x	x	0	1	1	1	0	1	1	0	1
x	x	0	x	x	0	1	1	1	1	0	1	0	1	1
x	x	x	x	0	1	1	1	1	1	0	1	0	0	1
x	x	x	0	1	1	1	1	1	1	0	0	1	1	1
x	x	0	1	1	1	1	1	1	1	0	0	1	0	1
x	0	1	1	1	1	1	1	1	1	0	0	0	1	1
0	1	1	1	1	1	1	1	1	1	0	0	0	0	1

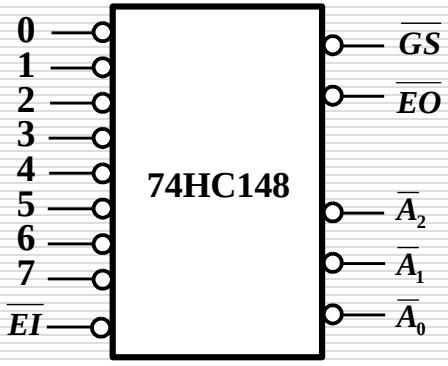
Next item we are going to discuss

- Decimal-to-BCD encoder
- Priority encoder
- 8-line-to-3-line priority encoder (74HC148)

8-line-to-3-line priority

Note: Each of $\overline{A}_2, \overline{A}_1, \overline{A}_0, \overline{EI}, \overline{EO}$ and $\overline{GS}$ is regarded as a whole variable. These are active-LOW.

- Its inputs and outputs are active-LOW (Code 000 is corresponding to input 7)
- The highest-priority input is 7
- Enable input $\overline{EI}$ is active-LOW (If $\overline{EI} = 0$, the encoder can be allowed to encode)
- Enable output $\overline{EO}$ is active-LOW
- $\overline{EO}$ can be connected to the $\overline{EI}$ of another encoder for expending
- Group select $\overline{GS}$ is active-LOW



Logic diagram

Truth table for 8-line-to-3-line priority encoder

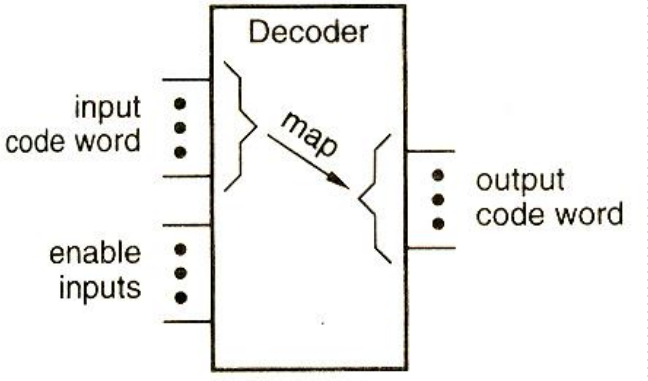
inputs									outputs				
$\overline{EI}$	0	1	2	3	4	5	6	7	$\overline{A}_2$	$\overline{A}_1$	$\overline{A}_0$	$\overline{GS}$	$\overline{EO}$
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	x	x	x	x	x	x	x	0	0	0	0	0	1
0	x	x	x	x	x	x	0	1	0	0	1	0	1
0	x	x	x	x	x	0	1	1	0	1	0	0	1
0	x	x	x	x	0	1	1	1	0	1	1	0	1
0	x	x	x	0	1	1	1	1	1	0	0	0	1
0	x	x	0	1	1	1	1	1	1	0	1	0	1
0	x	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0

The next topic

- Encoders
- Decoders
- Multiplexers (Data selectors)
- Demultiplexers
- Comparators
- Adders

Decoders

- A decoder is a combinational logic circuit that essentially performs a “reverse” encoder function.
- It is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs.
- The input code generally has fewer bits than the output code, and there is a one-to-one mapping from input code words into output code words.



Decoders

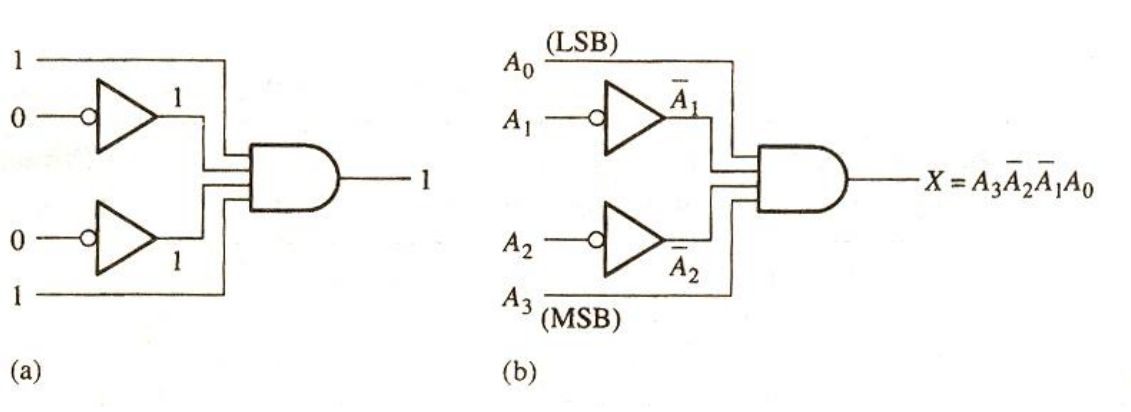
Binary decoder

A binary decoder is an n-to-2^n decoder. Such a decoder has n inputs and 2^n outputs. It is also called 1-of-2^n decoder because for any given code on the inputs, one of the 2^n outputs is activated.

Basic decoding element

To detect the state when a binary 1001 occurs on the inputs of a digital circuit.

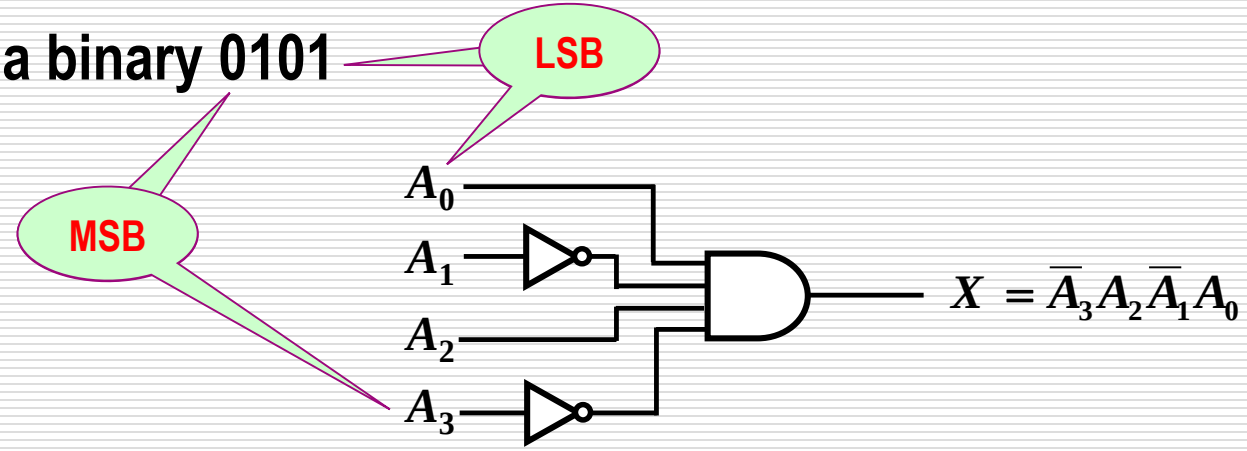
Using AND gate, the output is active-HIGH



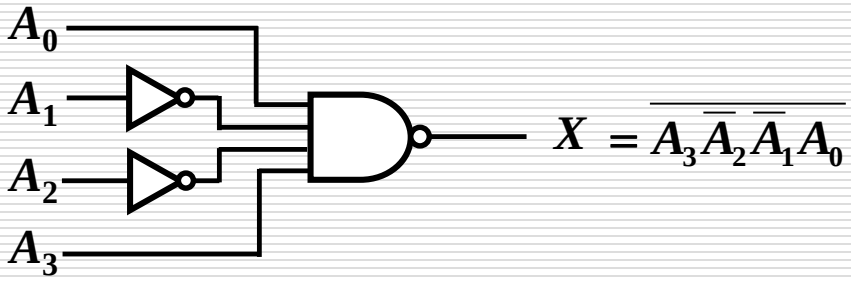
Decoders

Basic decoding element

To detect a binary 0101



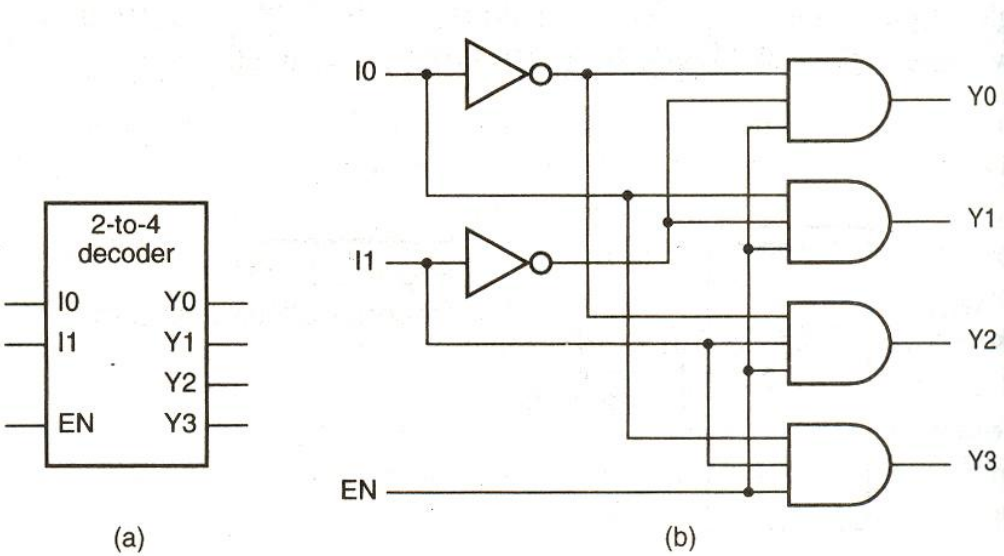
Using NAND gate, the output is active-LOW



Decoders

2-to-4 decoder

- The output is active-HIGH
- Enable input is active-HIGH



Inputs			Outputs			
EN	I1	I0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

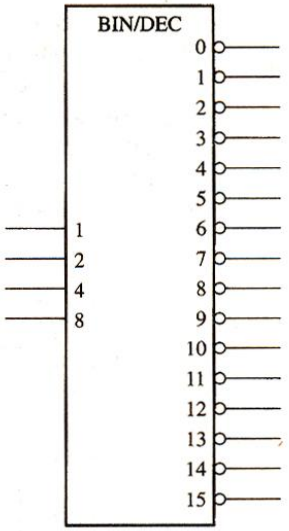
Decoders

4-to-16 decoder

- The output is active-LOW

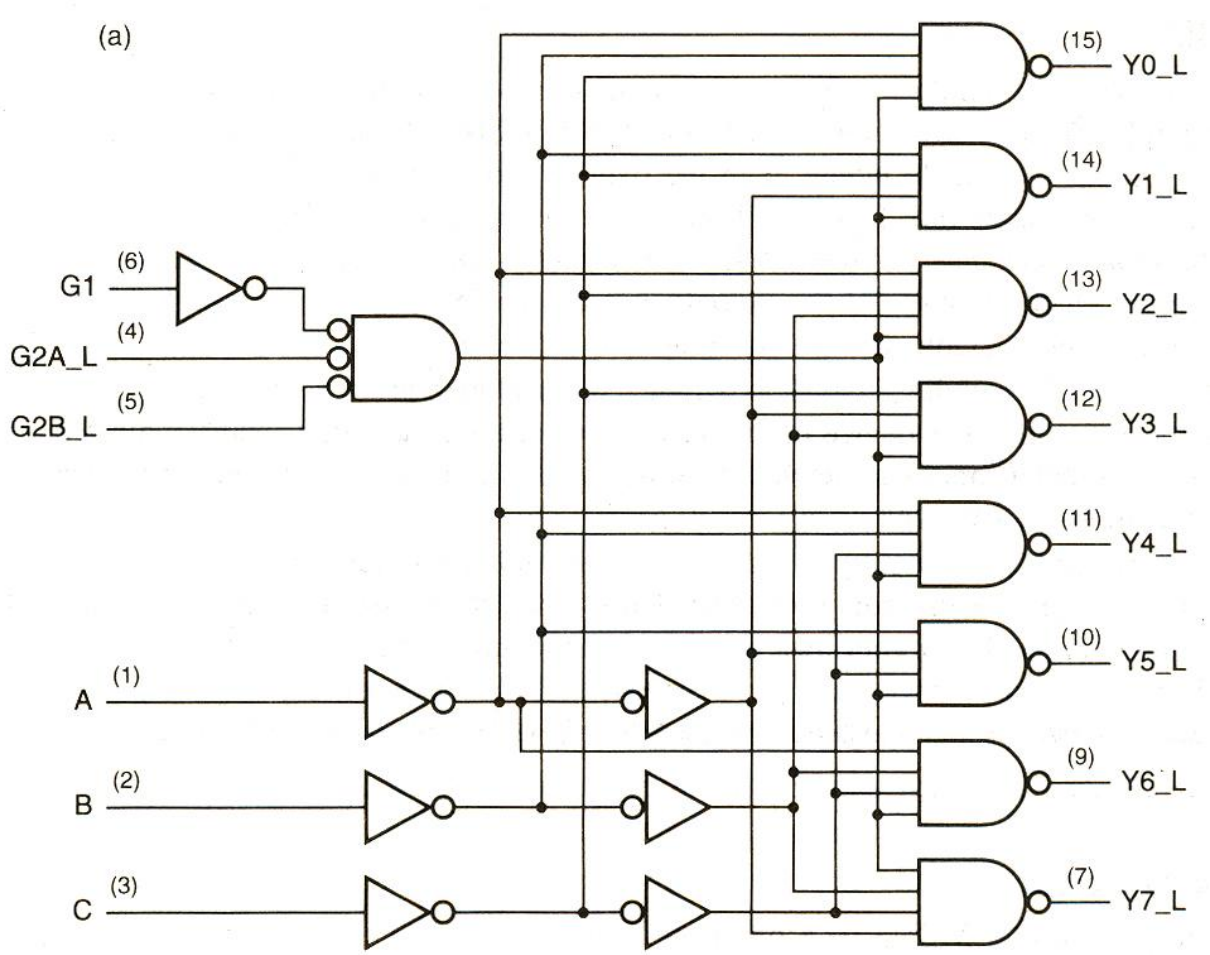
Decoding functions and truth table for a 4-line-to-16-line decoder with active-LOW outputs.

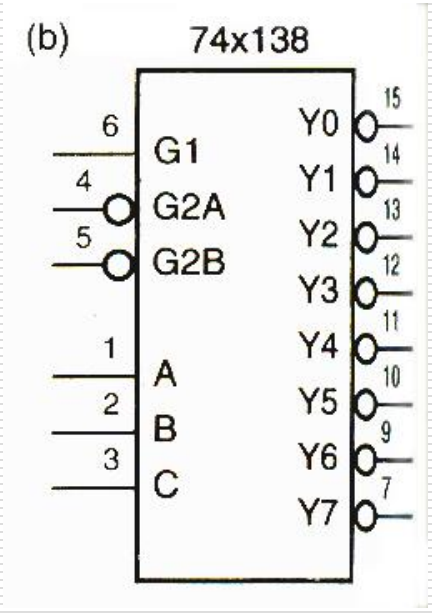
Decimal Digit	Binary Inputs A_3	A_2	A_1	A_0	Decoding Function	Outputs 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	0	$\bar{A}_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0	1	0	1	$\bar{A}_3A_2\bar{A}_1A_0$	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0	1	1	0	$\bar{A}_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1	0	0	0	$A_3\bar{A}_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1	0	0	1	$A_3\bar{A}_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1	0	1	0	$A_3\bar{A}_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1	0	1	1	$A_3\bar{A}_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1	1	0	0	$A_3A_2\bar{A}_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1	1	0	1	$A_3A_2\bar{A}_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1	1	1	0	$A_3A_2A_1\bar{A}_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1	1	1	1	$A_3A_2A_1A_0$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0



Decoders

- 3-to-8 MSI decoder





Decoders

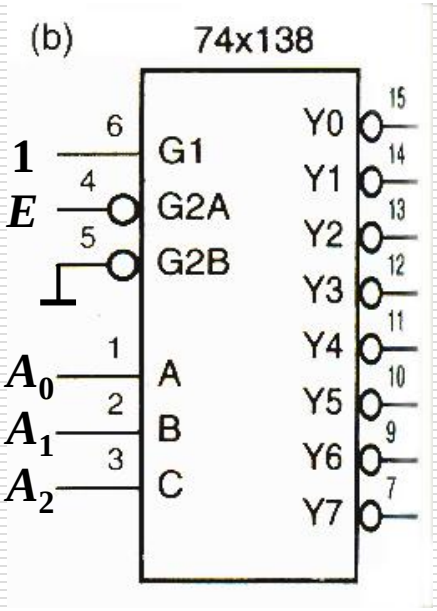
3-to-8 MSI decoder

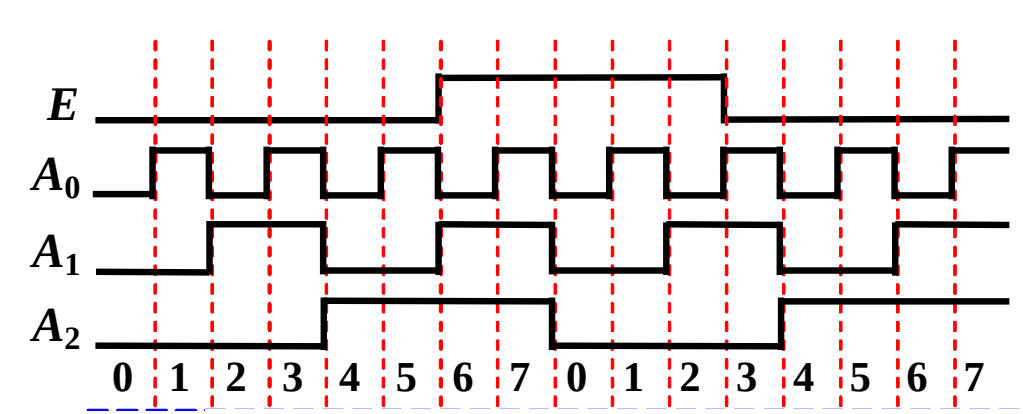
Table 5-7 Truth table for a 74x138 3-to-8 decoder.

Inputs						*Outputs*							
G1	G2A_L	G2B_L	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

Decoders

Pulse Operation





Y0_L

Y1_L

Y2_L

Y3_L

Y4_L

Y5_L

Y6_L

Y7_L

Decoders

The extension of the decoder

How to implement a 4-to-16 decoder with two 74HC138 decoders ?

Expand by enable inputs

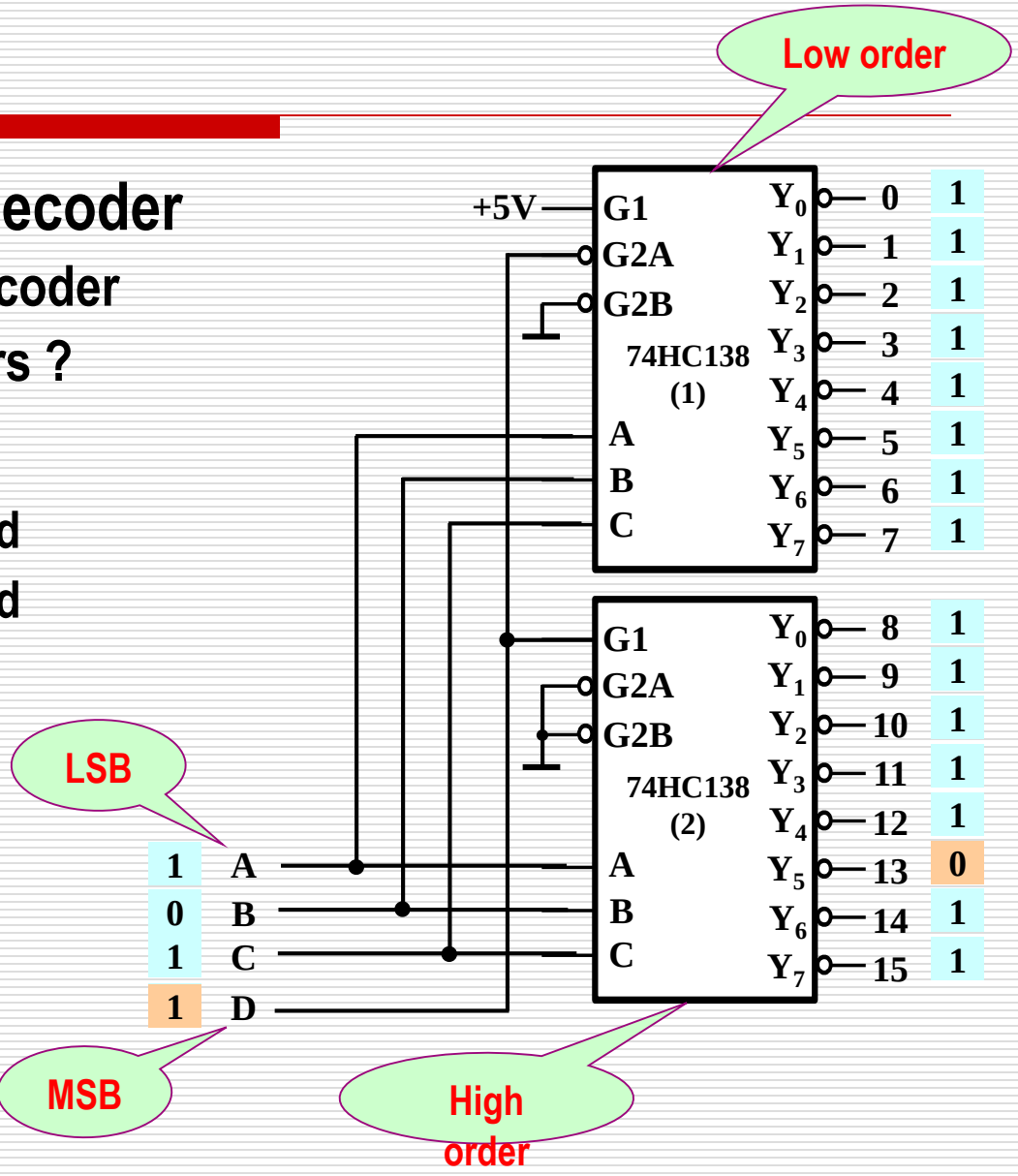
When D=0, chip (1) is enabled

When D=1, chip (2) is enabled

For example

When DCBA=0101,
the output 5 is 0,
others are 1s.

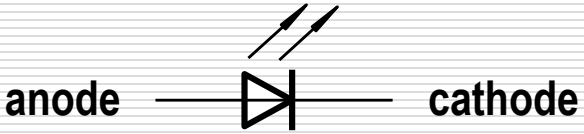
When DCBA=1101,
the output 13 is 0,
others are 1s.

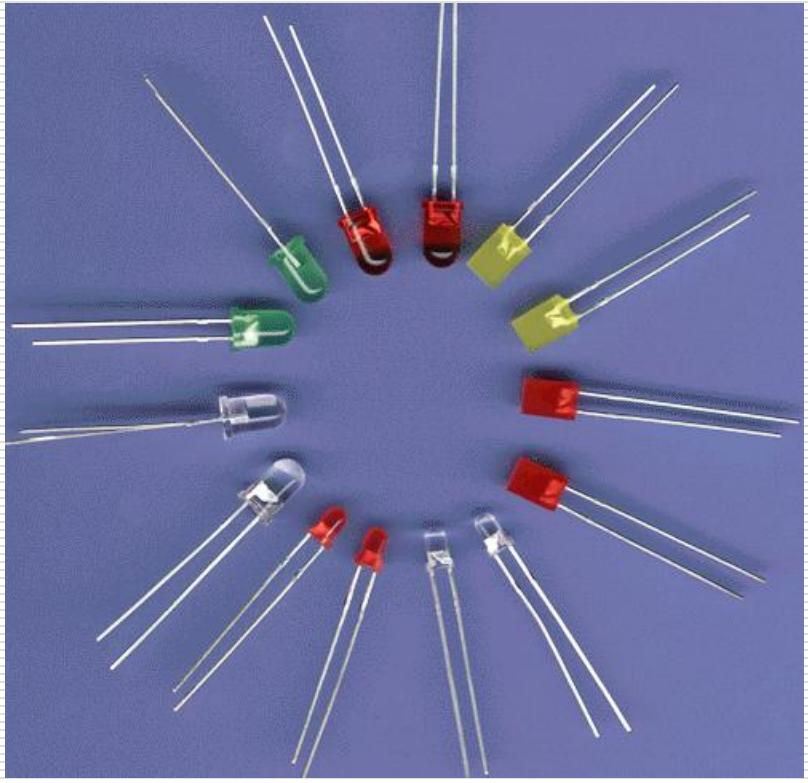


Decoders

The BCD-to-7-segment decoder

The light-emitting diode (LED)

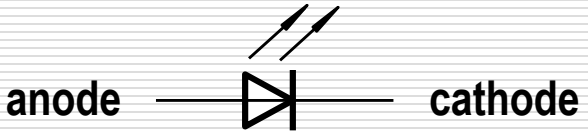




Decoders

The BCD-to-7-segment decoder

The light-emitting diode (LED)



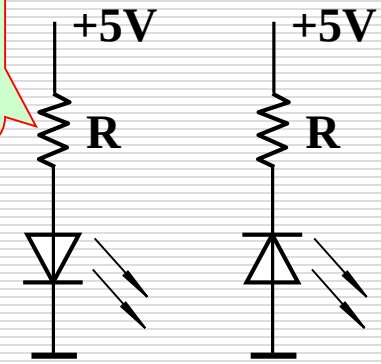
- If the voltage level of anode is higher than cathode,

  the diode turns on, there is current through it, and it emits light.

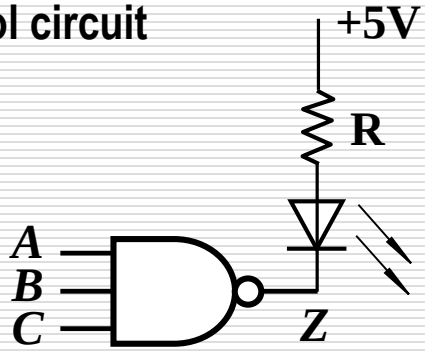
- If the voltage level of anode is lower than cathode,

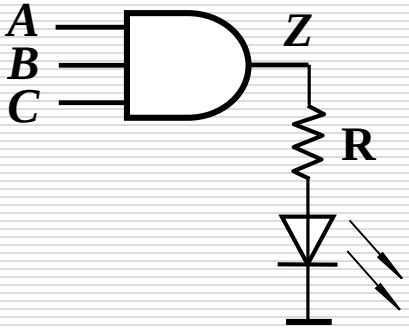
  the diode turns off, there is not current through it, and it is blanking.

Limiting current resistor to prevent that LED is burned out



Control circuit



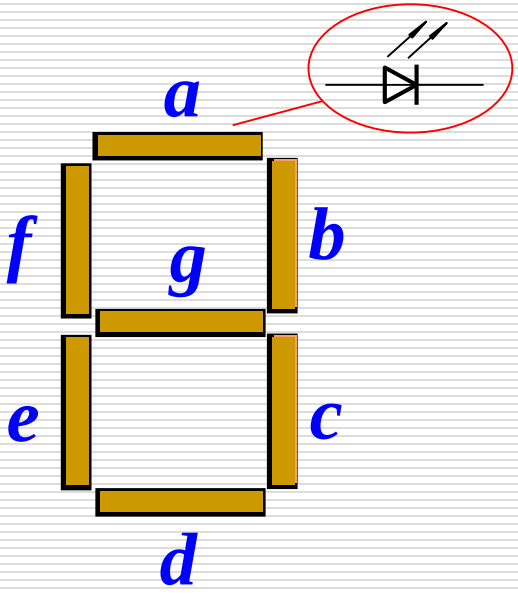


Decoders

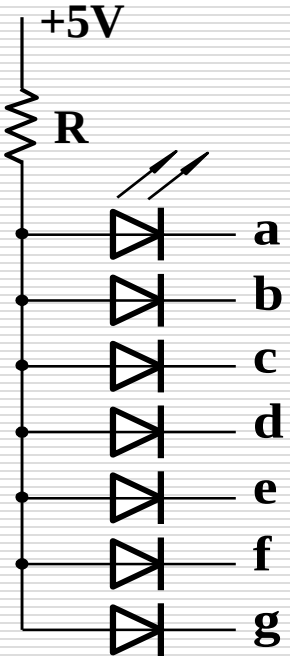
- The BCD-to-7-segment decoder

The 7-segment display

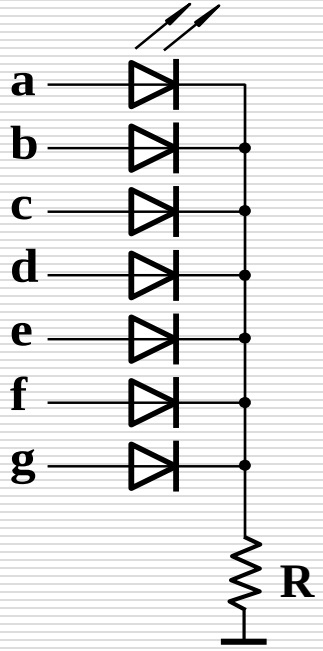
(refer to 4-12 of Textbook)



Segment identification



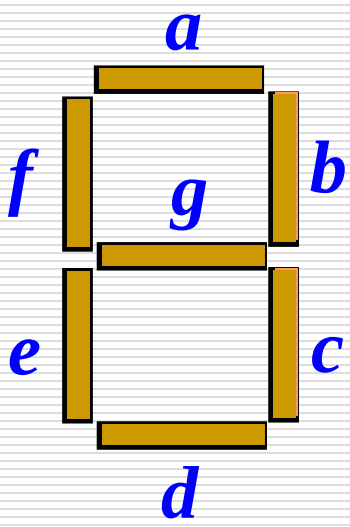
Common anode



Common cathode

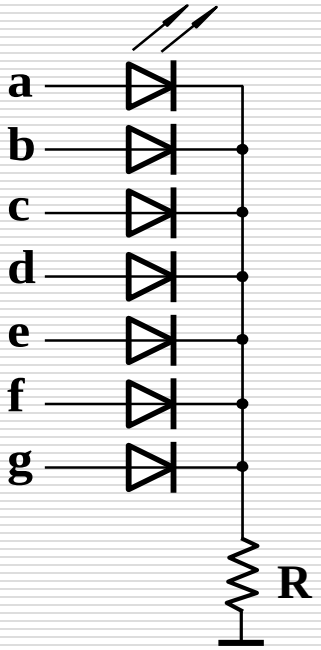
Decoders

- The 7-segment display

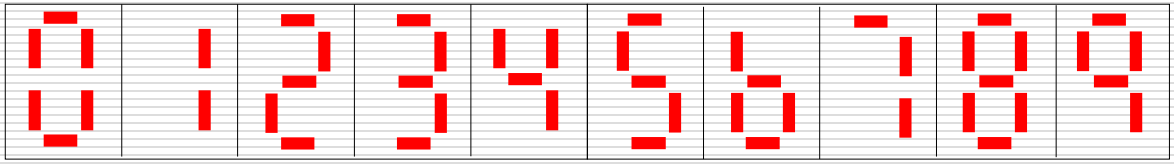


Segment identification

a	b	c	d	e	f	g
1	1	1	1	1	1	0
0	1	1	0	0	0	0
1	1	0	1	1	0	1

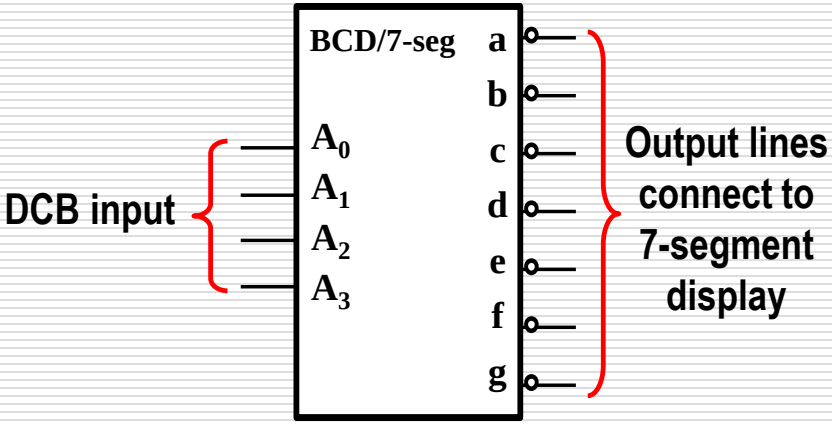


Common cathode

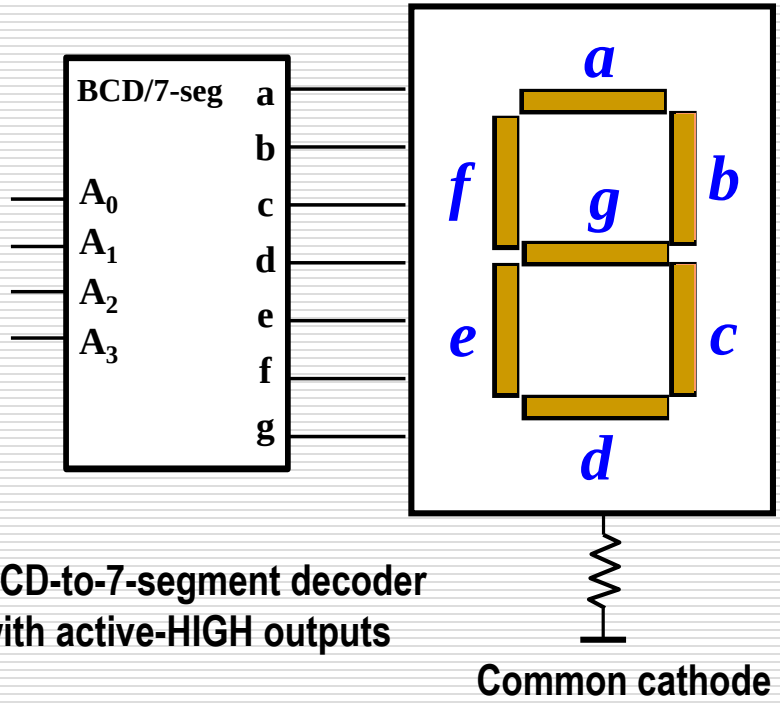


Decoders

The BCD-to-7-segment decoder



A BCD-to-7-segment decoder with active-LOW outputs

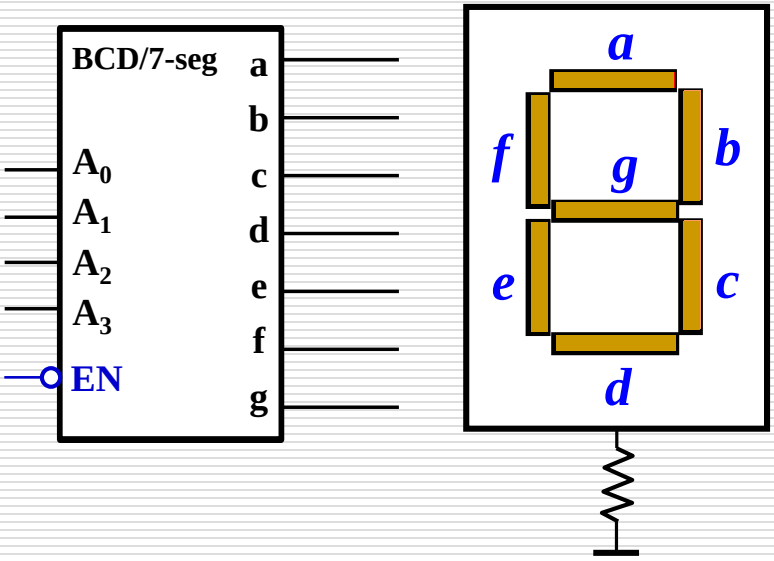


A BCD-to-7-segment decoder with active-HIGH outputs

Common cathode

The BCD-to-7-segment decoder

- The truth table for the decoder with active-HIGH outputs



Inputs	Outputs	Decimal digit
$A_3 A_2 A_1 A_0$	a b c d e f g	
0 0 0 0	1 1 1 1 1 1 0	0
0 0 0 1	0 1 1 0 0 0 0	1
0 0 1 0	1 1 0 1 1 0 1	2
0 0 1 1	1 1 1 1 0 0 1	3
0 1 0 0	0 1 1 0 0 1 1	4
0 1 0 1	1 0 1 1 0 1 1	5
0 1 1 0	1 0 1 1 1 1 1	6
0 1 1 1	1 1 1 0 0 0 0	7
1 0 0 0	1 1 1 1 1 1 1	8
1 0 0 1	1 1 1 1 0 1 1	9
Other states	0 0 0 0 0 0 0	Dead

According to this truth table, we can design the logic circuit of BCD-to-7-segment decoder.

MSI 74HC4511 is a BCD-to-7-segment decoder

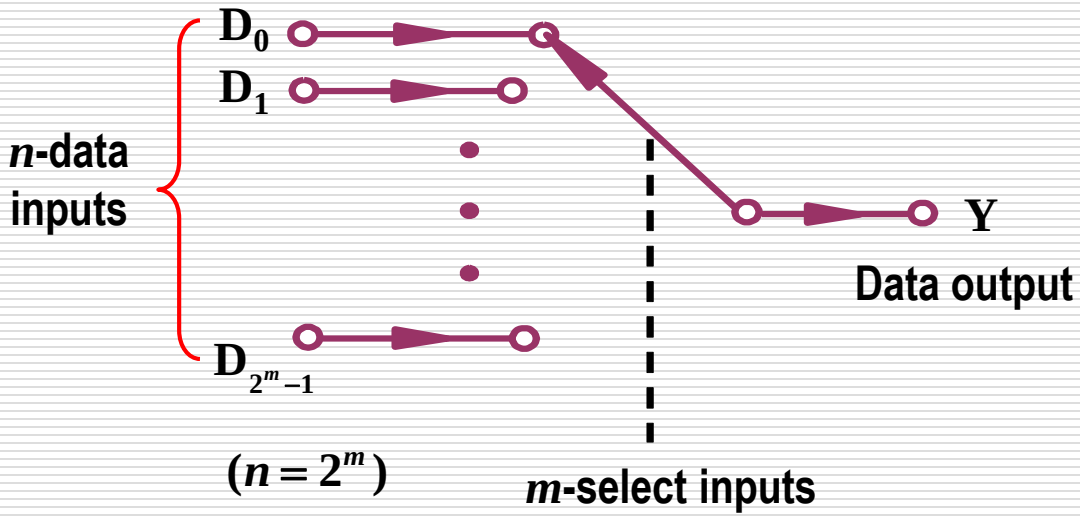
The next topic

- Encoders
- Decoders
- Multiplexers (Data selectors)
- Demultiplexers
- Comparators
- Adders

Multiplexers (Data selectors)

Multiplexers

A multiplexer (MUX) is a digital switch which connects data from one of n sources to its output.



The data-select inputs control which route is chosen.

Multiplexers (Data selectors)

- The simplest multiplexer (1-of-2 data selector)

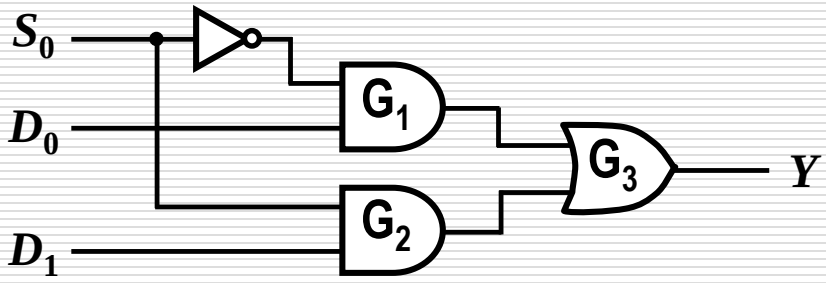
$$Y = \overline{S}_0 D_0 + S_0 D_1$$

If S_0=0,

then Y=D_0

If S_0=1,

then Y=D_1

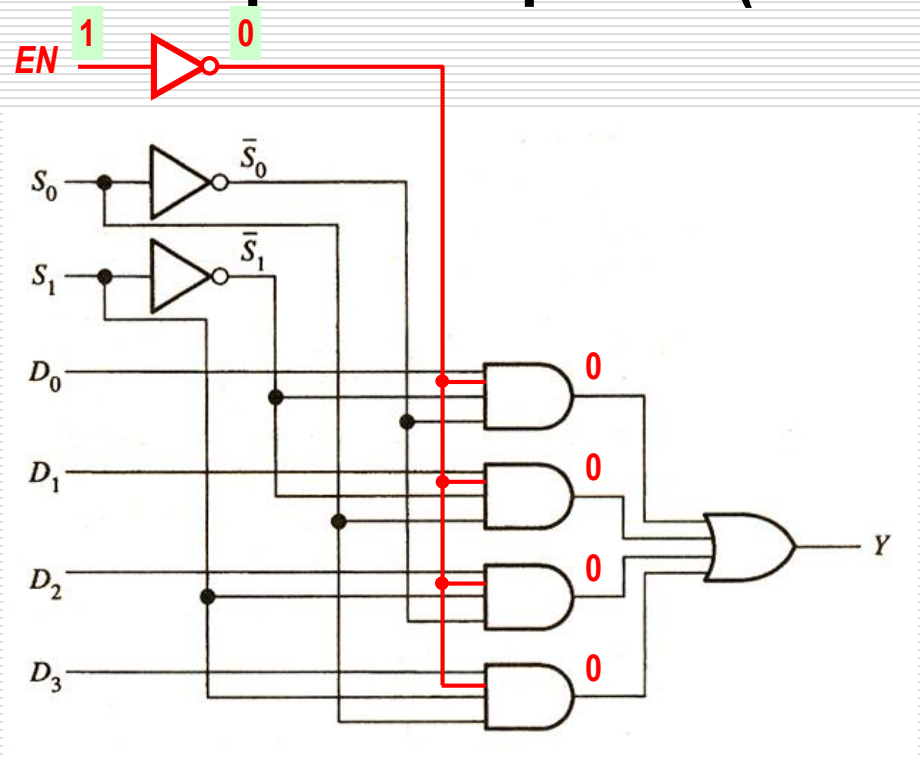


select input: S_0

data input: D_0, D_1

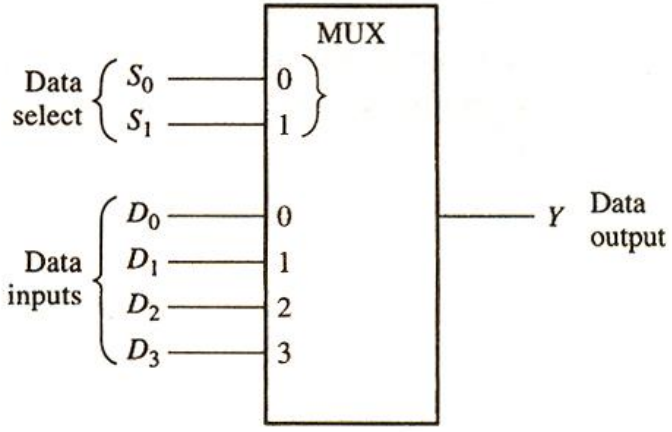
Multiplexers (Data selectors)

A 4-input multiplexer (1-of-4 data selector)



function table

EN	Data-select Inputs S_1	S_0	Input Selected
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	x	x	0



Logic symbol

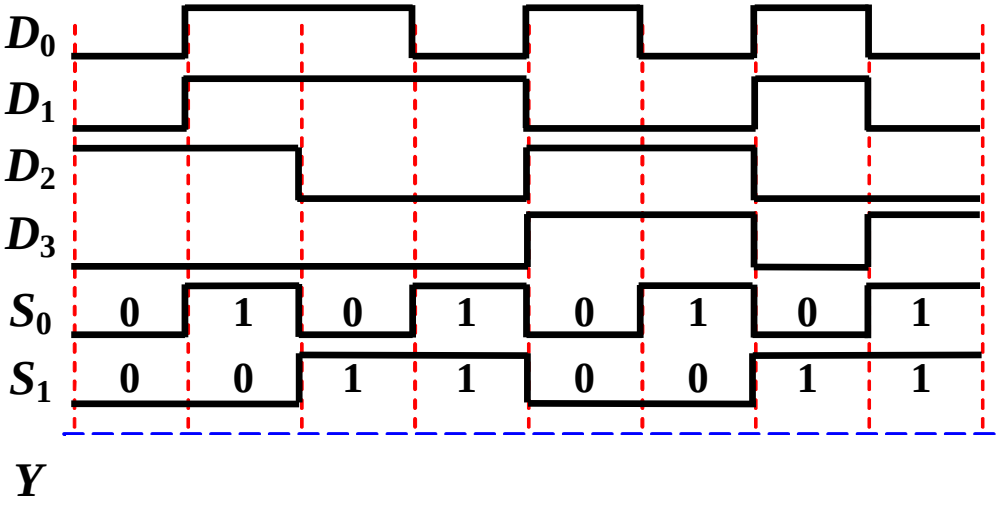
$$Y = D_0\bar{S}_1\bar{S}_0 + D_1\bar{S}_1S_0 + D_2S_1\bar{S}_0 + D_3S_1S_0$$

Multiplexers (Data selectors)

A 4-input multiplexer (1-of-4 data selector)

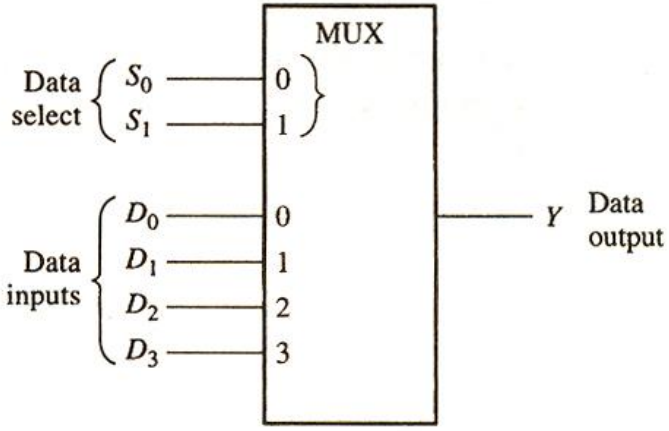
Example : the data-input and data-select waveforms are applied to multiplexer.

Determine the output in relation to the inputs.



function table

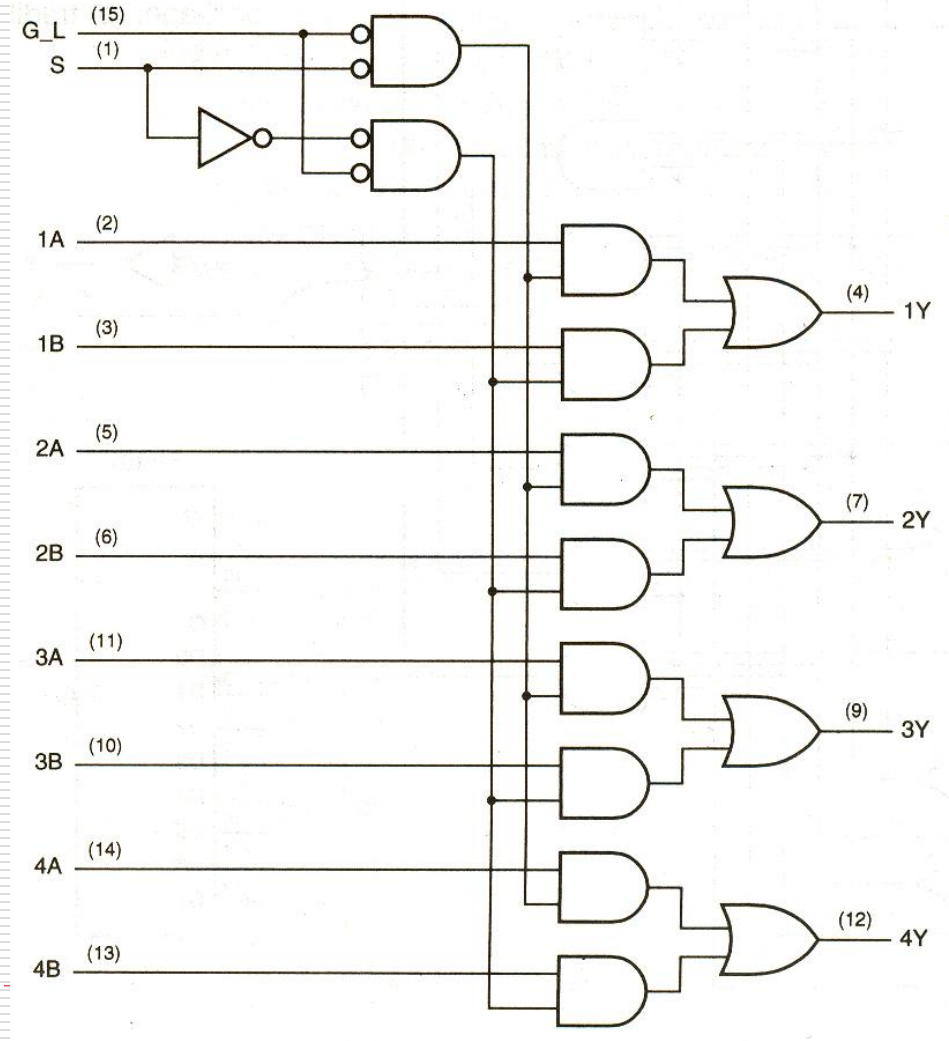
Data-select Inputs S_1	S_0	Input Selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



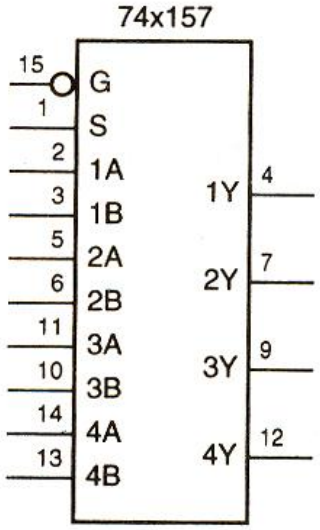
Logic symbol

Multiplexers (Data selectors)

- The 2-input, 4-bit multiplexer (MSI 74x157)



Inputs		Outputs			
G_L	S	1Y	2Y	3Y	4Y
1	x	0	0	0	0
0	0	1A	2A	3A	4A
0	1	1B	2B	3B	4B



2 sources of data

4 bits wide for each

Multiplexers

Application example

Two BCD digits: $A_3A_2A_1A_0$, $B_3B_2B_1B_0$

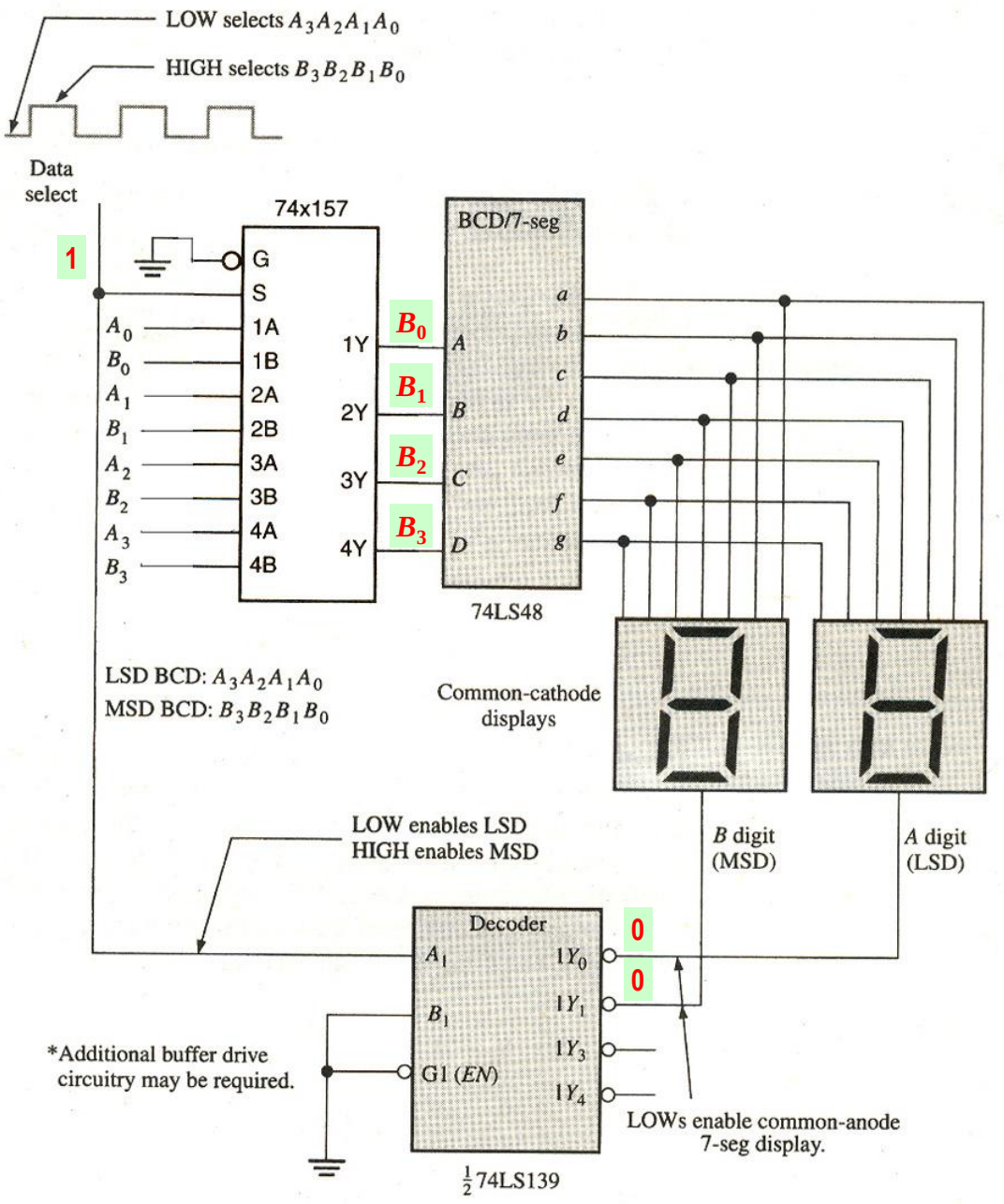
A square wave is applied to data-select line

When data-select is LOW,

- The output of MUX select the A bits
- The BCD/7-seg decoder decodes data A
- The output Y_0 of 2-to-4 decoder is LOW and enable A-digit display

When data-select is HIGH,

- The output of MUX select input B
- The BCD/7-seg decoder decodes data B
- The output Y_1 of 2-to-4 decoder is LOW and enable B-digit display



Multiplexers

Application example

Two BCD digits: $A_3A_2A_1A_0$, $B_3B_2B_1B_0$

A square wave is applied to data-select line

When the cycle repeats at frequency of the data-select square wave,

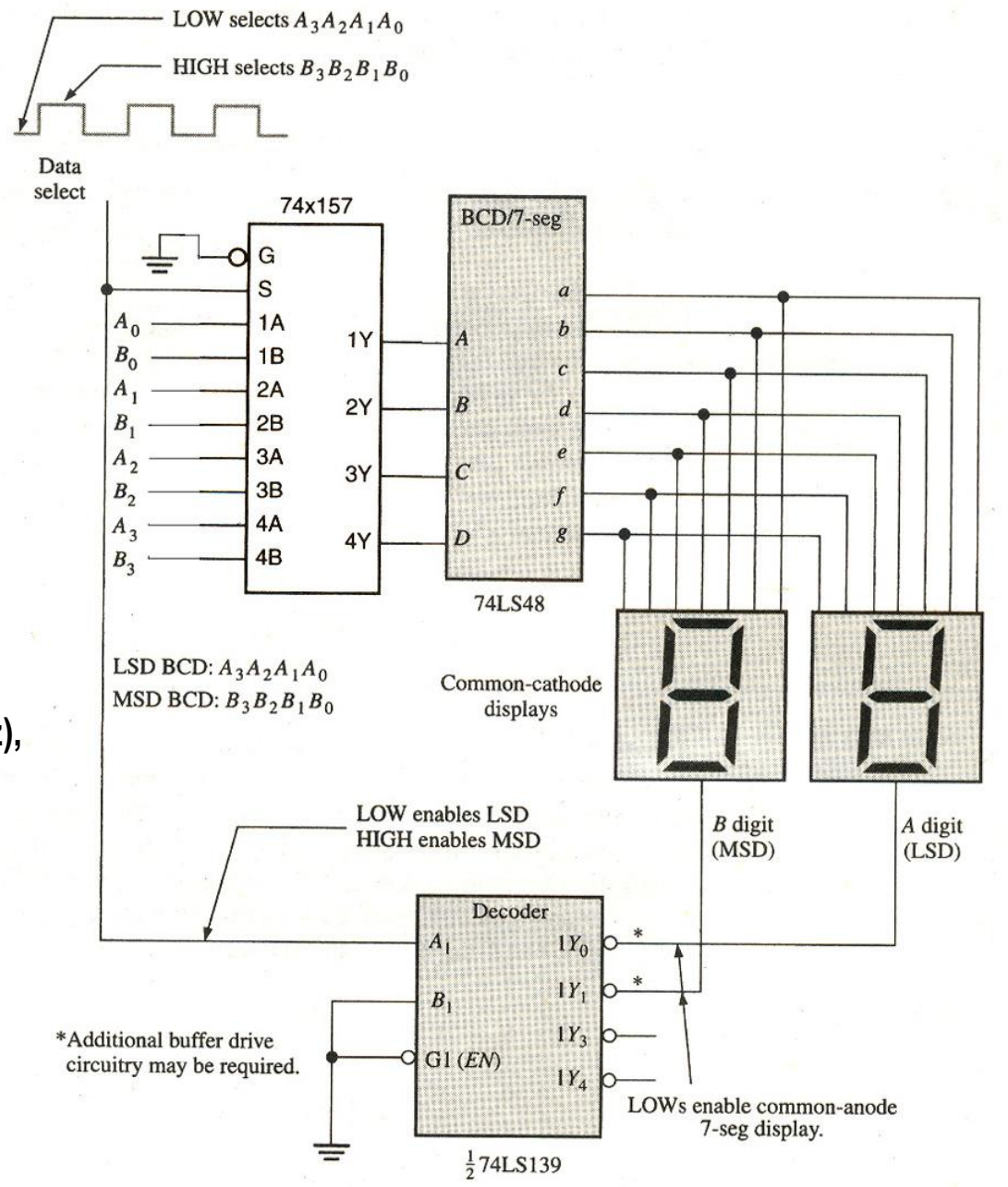
each digit display alternates between on and off in different interval.

If this frequency is high enough (about 30 Hz),

all digit displays look like always being on and they are not visual flicker

How to extend it to display 4-digit ?

(This question is left for you)



The next topic

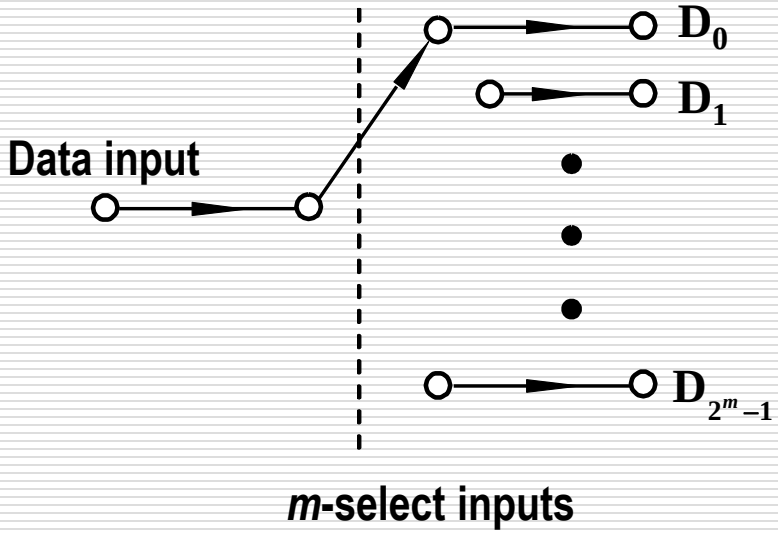
- Encoders
- Decoders
- Multiplexers (Data selectors)
- Demultiplexers
- Comparators
- Adders

Demultiplexers

Demultiplexers

The function of a demultiplexer (DMUX) is just the inverse of a multiplexer's. It takes data from one line and distributes them to a given number of output lines. It is also known as a data distributor.

Decoders can be used as demultiplexers

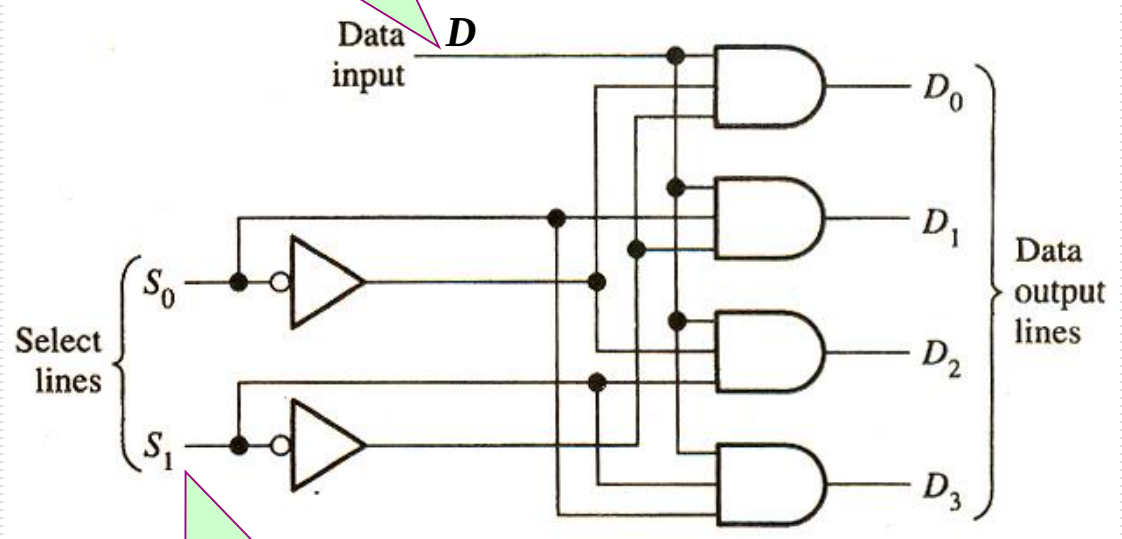


Demultiplexers

Note: in this case, the decoder must has the enable input

A 2-to-4 decoder is used as a 1-to-4 demultiplexer

Enable input in decoder

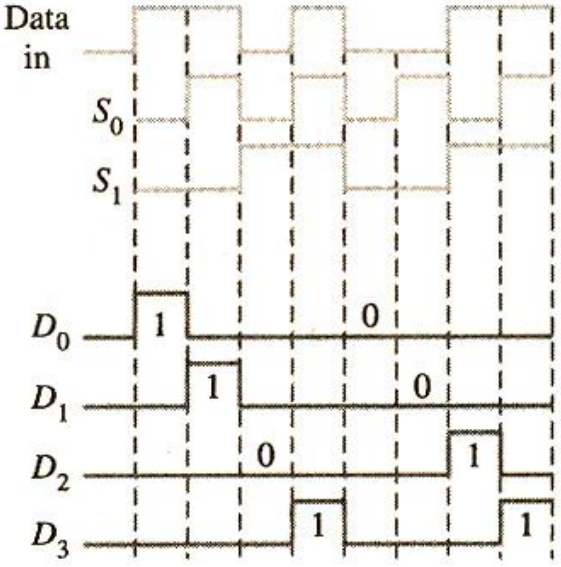


Code inputs in decoder

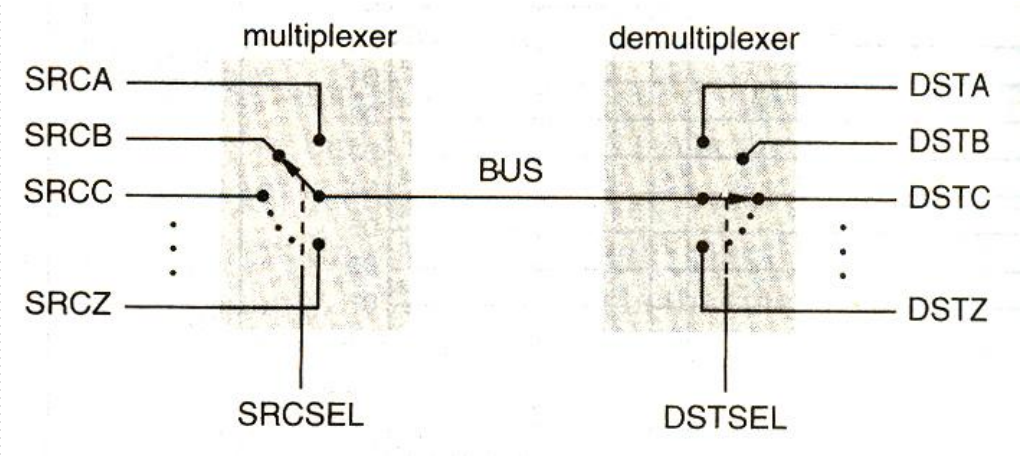
$$D_0 = D\bar{S}_1\bar{S}_0 \qquad D_2 = DS_1\bar{S}_0$$

$$D_1 = D\bar{S}_1S_0 \qquad D_3 = DS_1S_0$$

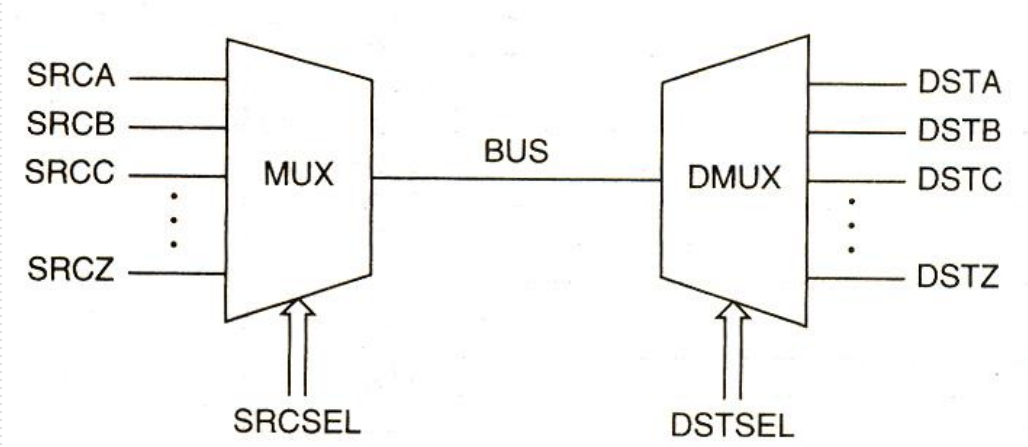
Given different state of select lines, the input data will be distributed to corresponding output.



Multiplexers, Demultiplexers and Buses



A selected one of multiple data sources gets directed onto a bus and routed to a selected one of multiple destinations.



Block diagram

The next topic

- Encoders
- Decoders
- Multiplexers (Data selectors)
- Demultiplexers
- Comparators
- Adders

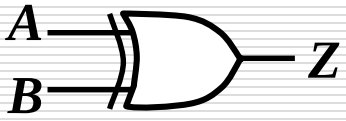
Comparators

Comparators

The basic function of a comparator is to compare the magnitudes of two binary quantities to determine the relationship of those quantities.

For simplest case, only indicating whether two numbers are equal.

We can use a XOR gate to implement 1-bit equality comparator.

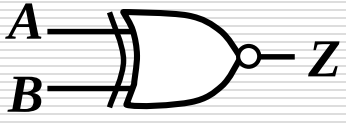


XOR gate

If $A=B$, then $Z=0$

If $A \neq B$, then $Z=1$

The output is active-LOW



XNOR gate

The output is active-HIGH

Comparators

The magnitude comparator

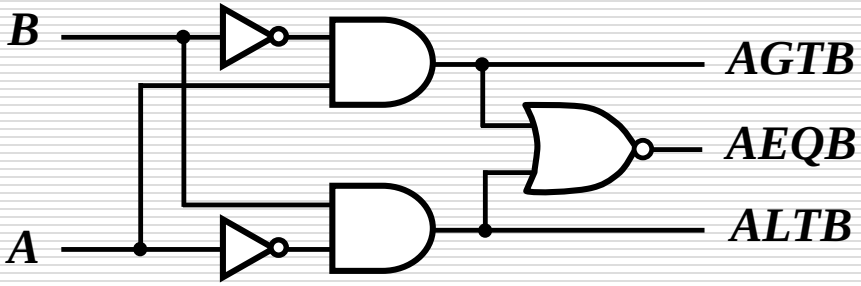
For 1-bit case

$$AGTB = A\overline{B}$$

$$ALTB = \overline{A}B$$

$$AEQB = AB + \overline{A}\,\overline{B} = \overline{A\overline{B} + \overline{A}B}$$

Inputs		Outputs		
A	*B*	*AEQB*	*AGTB*	*ALTB*
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0



Comparators

The magnitude comparator

For the more then one bit case

First examine the highest-order bit in each number, when they are equal, then examine the next lower-order bit, or else ignore other lower-order bits.

A 2-bit comparator as example

Compare A_1A_0 and B_1B_0

If $A_1>B_1$, the result is that number A is greater than number B

If $A_1<B_1$, the result is that number A is less than number B

If $A_1=B_1$, then examine the next lower-order bit position, A_0 and B_0

Comparators

A 2-bit magnitude comparator

Compare A_1A_0 and B_1B_0

If $A_1>B_1$, the result is that number A is greater then number B

If $A_1<B_1$, the result is that number A is less then number B

If $A_1=B_1$, then examine the next lower-order bit position, A_0 and B_0

thus, If $A_0=B_0$, the result is that number A is equal to number B

If $A_0>B_0$, the result is that number A is greater then number B

If $A_0<B_0$, the result is that number A is less then number B

Comparators

A 2-bit magnitude comparator

Implement a 2-bit magnitude comparator by using two 1-bit comparators and an additive logic circuit

Here

$AEQB_1: A_1=B_1$

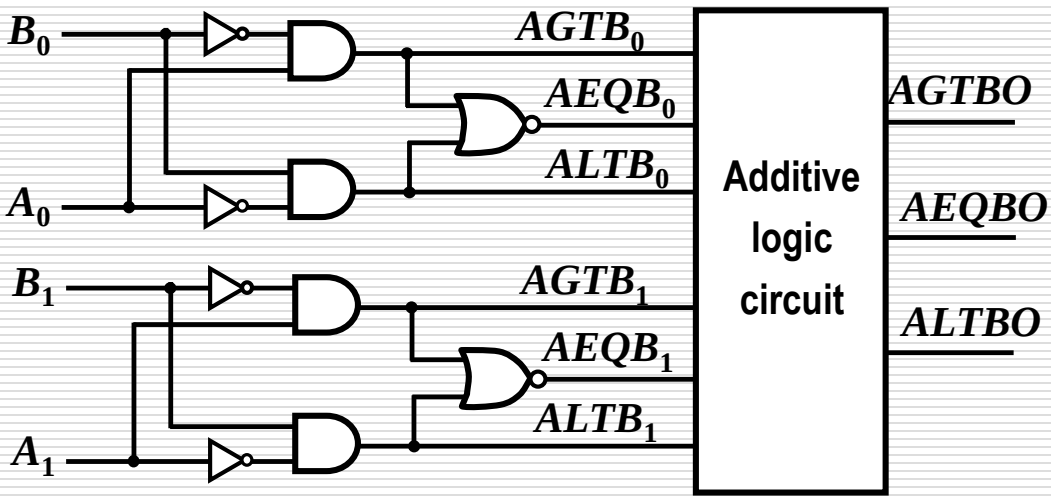
$AGTB_1: A_1>B_1$

$ALTB_1: A_1<B_1$

$AEQB_0: A_0=B_0$

$AGTB_0: A_0>B_0$

$ALTB_0: A_0<B_0$



Comparators

A 2-bit magnitude comparator

If $A_1 > B_1$, the result is that number A is greater then number B

If $A_1 < B_1$, the result is that number A is less then number B

When $A_1 = B_1$, examine the next lower-order bit position, A_0 and B_0

thus, If $A_0 = B_0$, the result is that number A is equal to number B

If $A_0 > B_0$, the result is that number A is greater then number B

If $A_0 < B_0$, the result is that number A is less then number B

For additive logic circuit

If $AEQB_1$ and $AEQB_0$ are true, $AEQBO$ is true

If $AGTB_1$ is true or $AEQB_1$ and $AGTB_0$ are true, $AGTBO$ is true

If $ALTB_1$ is true or $AEQB_1$ and $ALTB_0$ are true, $ALTBO$ is true

Comparators

A 2-bit magnitude comparator

Design additive logic circuit

If $AEQB_1$ and $AEQB_0$ are true, $AEQBO$ is true

If $AGTB_1$ is true or $AEQB_1$ and $AGTB_0$ are true, $AGTBO$ is true

If $ALTB_1$ is true or $AEQB_1$ and $ALTB_0$ are true, $ALTBO$ is true

The expressions

$$AEQBO = AEQB_1 \cdot AEQB_0$$

$$AGTBO = AGTB_1 + AEQB_1 \cdot AGTB_0$$

$$ALTBO = ALTB_1 + AEQB_1 \cdot ALTB_0$$

Comparators

A 2-bit magnitude comparator

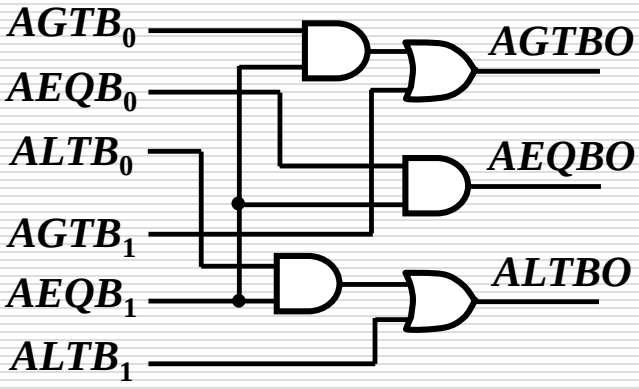
The expressions of additive logic circuit

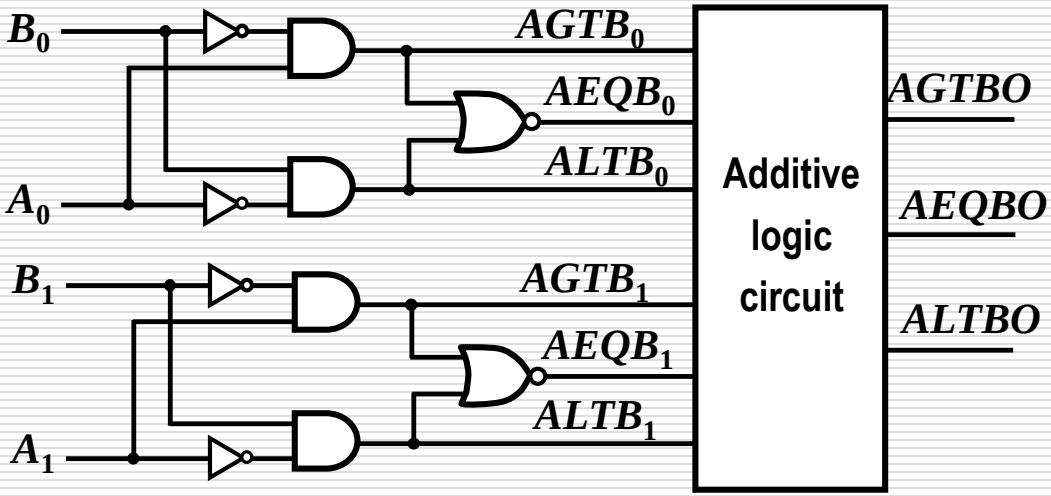
$$AEQBO = AEQB_1 \cdot AEQB_0$$

$$AGTBO = AGTB_1 + AEQB_1 \cdot AGTB_0$$

$$ALTBO = ALTB_1 + AEQB_1 \cdot ALTB_0$$

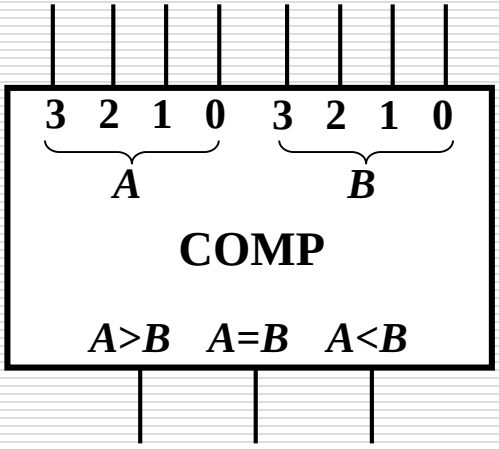
Three AND gates and two OR gates are required.

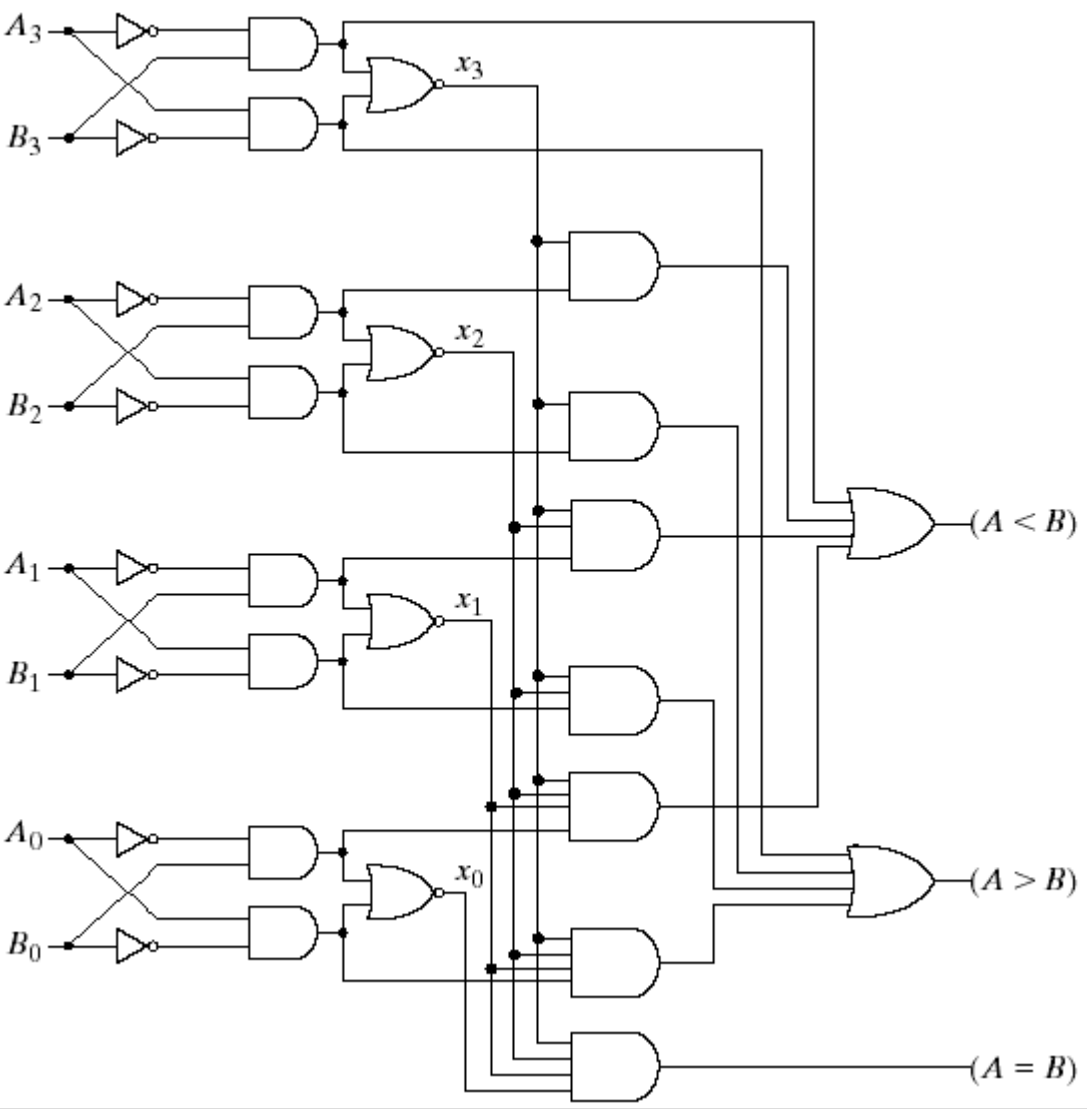




Comparators

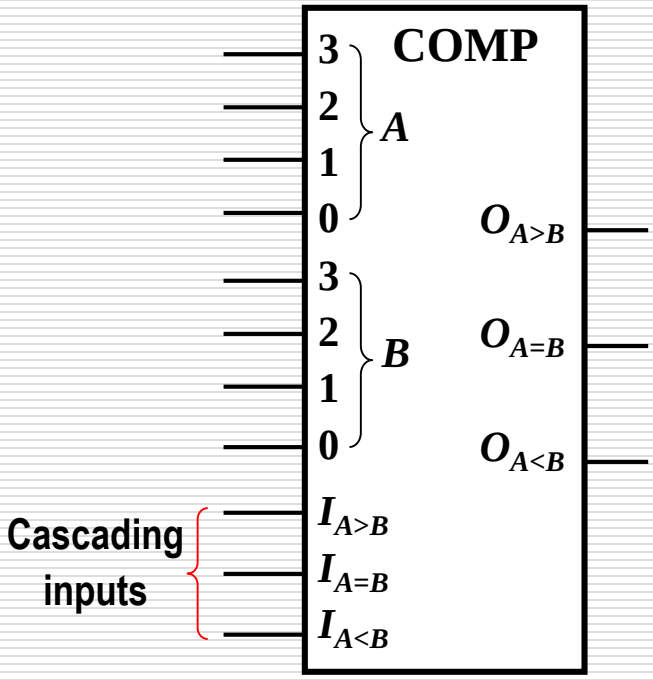
- A 4-bit magnitude comparator





Comparators

An MSI 4-bit magnitude comparator (74HC85)

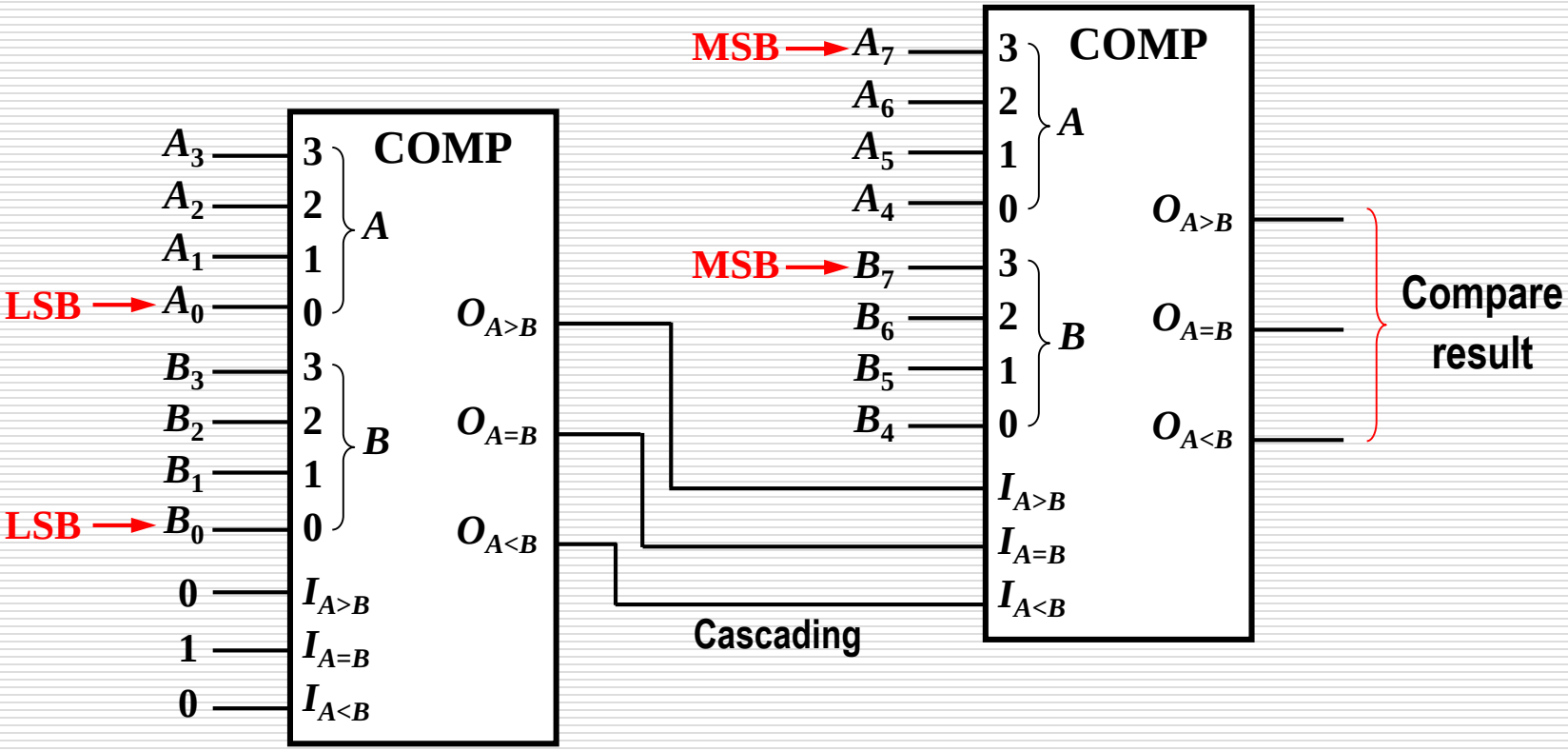


Inputs										Outputs		
A_3 B_3	A_2 B_2	A_1 B_1	A_0 B_0	$I_{A>B}$	$I_{A=B}$	$I_{A<B}$	$O_{A>B}$	$O_{A=B}$	$O_{A<B}$
1 0	x x	x x	x x	x	x	x	1	0	0
0 1	x x	x x	x x	x	x	x	0	0	1
$A_3=B_3$ 1 0	x x	x x	x x	x	x	x	1	0	0
$A_3=B_3$ 0 1	x x	x x	x x	x	x	x	0	0	1
$A_3=B_3$	$A_2=B_2$ 1 0	x x	x x	x	x	x	1	0	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	1	0	0	1	0	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	0	0	1	0	0	1
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	x	1	x	0	1	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	1	0	1	0	0	0
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	0	0	0	1	0	1

Comparators

A MSI 4-bit magnitude comparator (74HC85)

Use 74HC85 comparators to compare two 8-bit numbers



The next topic

- Encoders
- Decoders
- Multiplexers (Data selectors)
- Demultiplexers
- Comparators
- Adders

Adders

The half-adder

A combinational circuit that performs the addition of two bits is called a half adder.

The basic rules for binary addition

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$

Carry bit — Sum bit

(arithmetic operation)

Truth table for half adder

Inputs		Outputs	
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Carry bit — Sum bit

Adders

Inputs		Outputs	
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

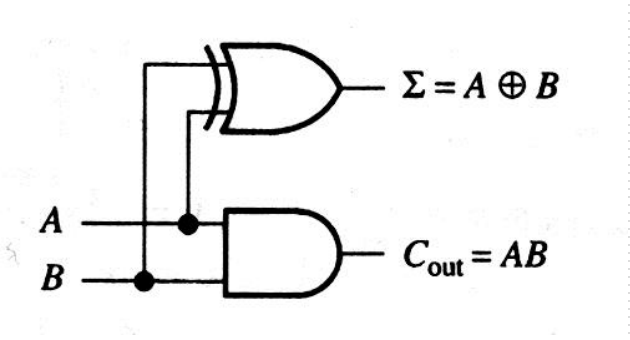
The half-adder

The expression

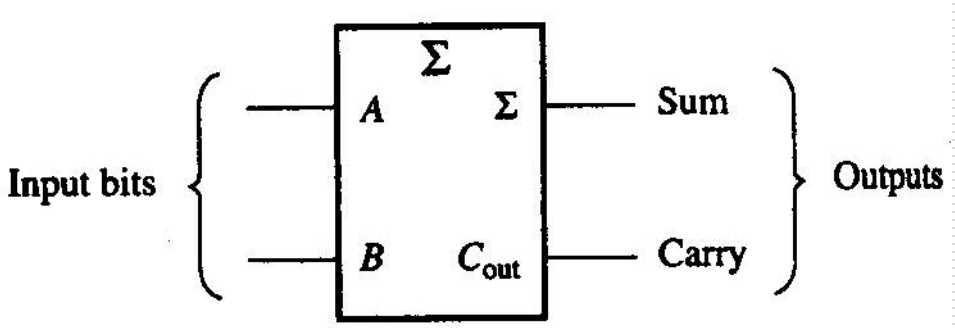
$$C_{out} = AB$$

$$\Sigma = A\overline{B} + \overline{A}B = A \oplus B$$

Logic diagram

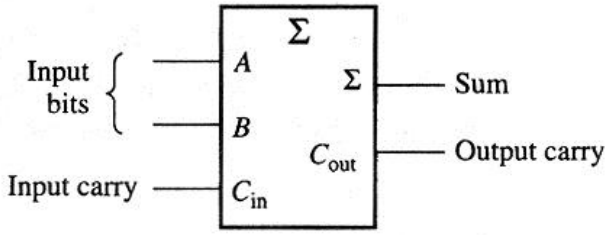


Logic symbol



Here, we use logic circuit to perform arithmetic operation.

Adders



Logic symbol

The full-adder

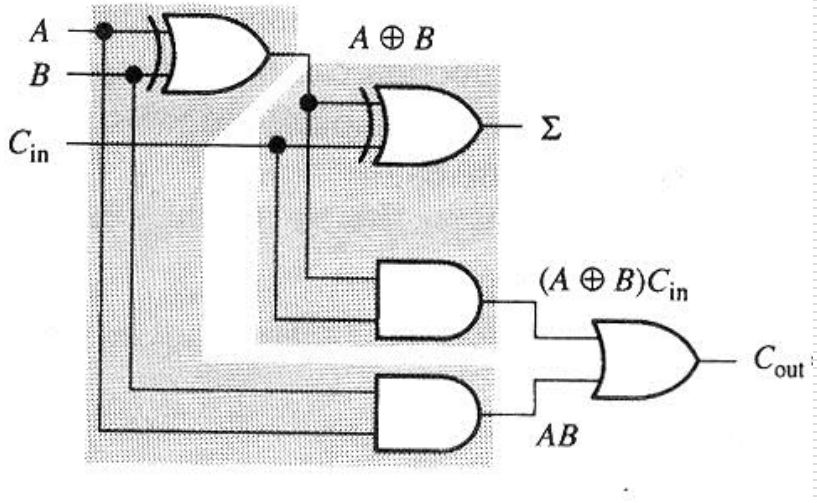
The basic difference between a full-adder and a half-adder is that the full-adder accept an input carry coming from lower-order bit.

The expression (using K-Map to simplify)

$$\Sigma = (A \oplus B) \oplus C_{in} \qquad C_{out} = AB + (A \oplus B)C_{in}$$

A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Logic diagram



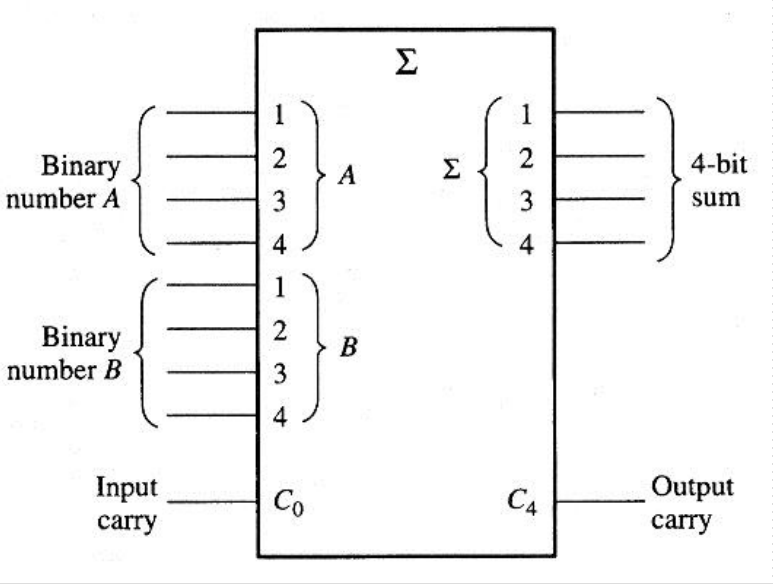
Adders

Parallel binary adders

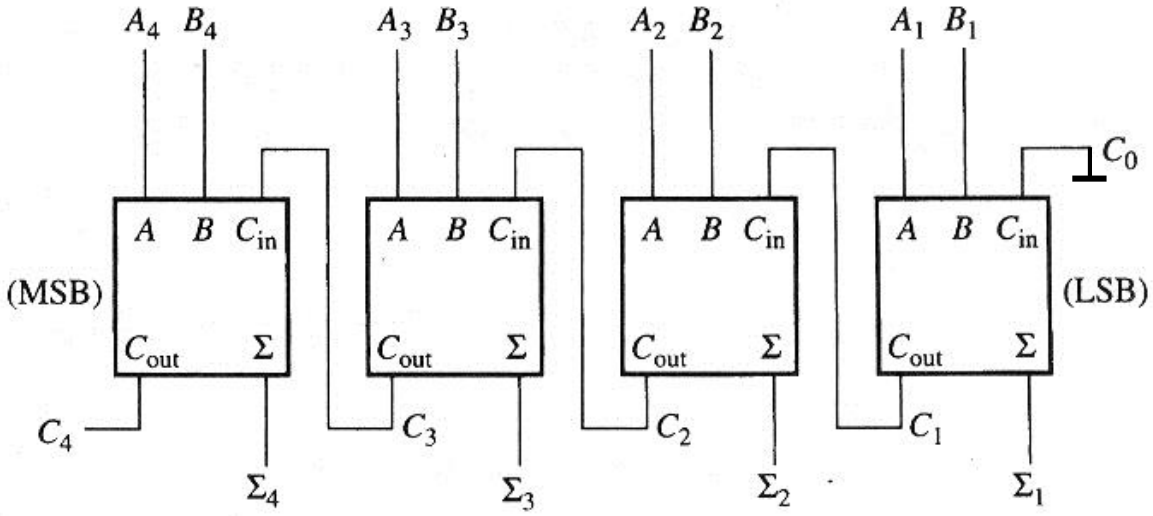
Use four full-adders to implement a 4-bit parallel binary adder

The carry output of each adder is connected to the carry input of the next higher-order adder.

To the LSB position there is no carry input, so C_0 should be connected to ground

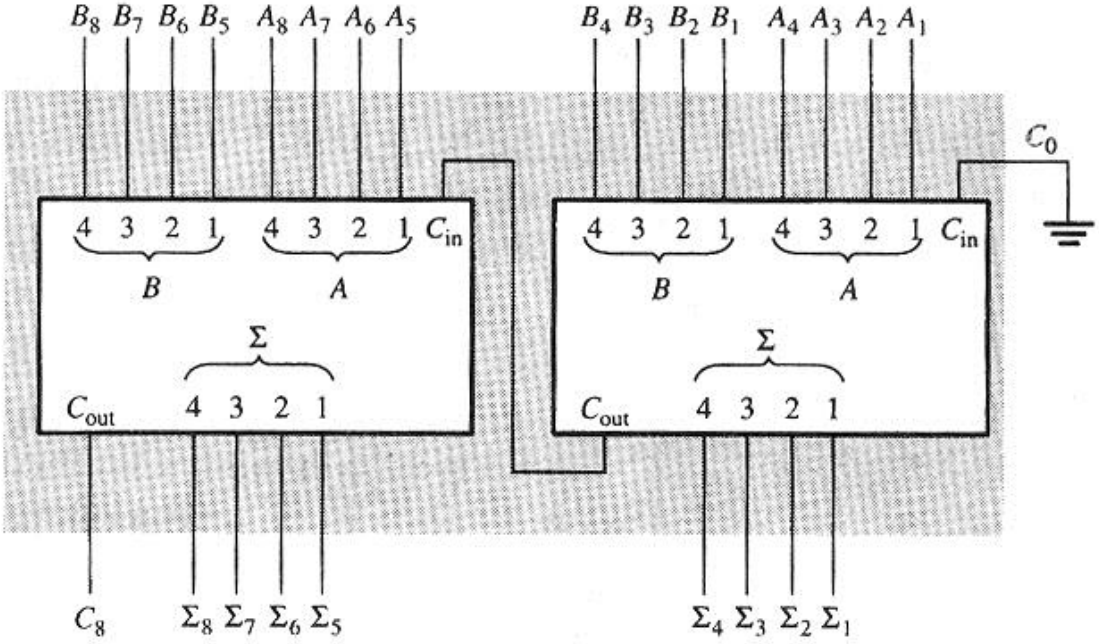


(MSI 74x283)



Adders

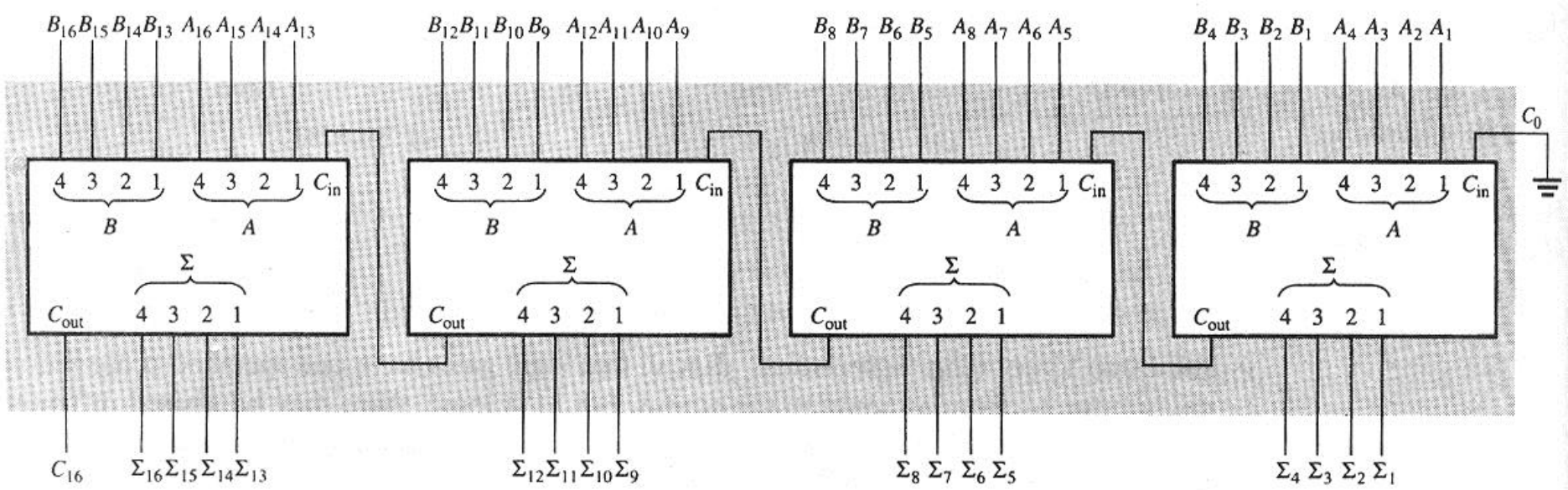
Adder expansion



(a) Cascading of 4-bit adders to form an 8-bit adder

Adders

Adder expansion



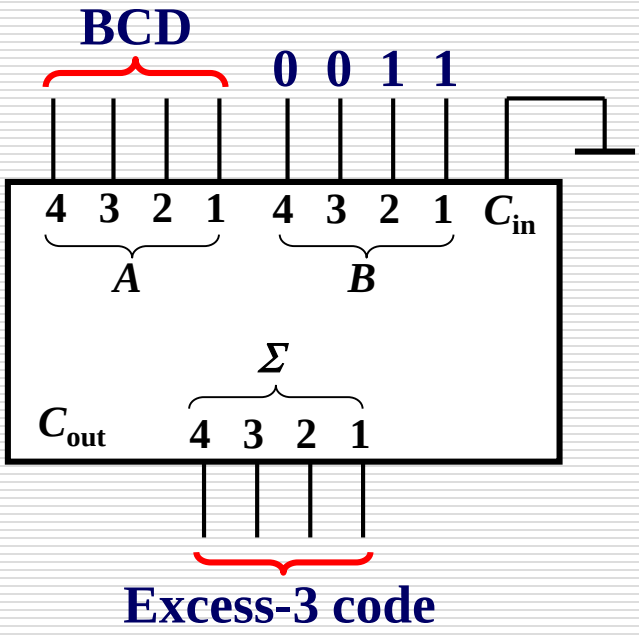
(b) Cascading of 4-bit adders to form a 16-bit adder

Adders

- Application example

Use a 4-bit parallel adder to implement BCD-to-Excess-3 conversion.

BCD + 0011 = Excess-3



Dec	BCD $A_3A_2A_1A_0$	Excess-3 $Y_3Y_2Y_1Y_0$
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0
10	1 0 1 0	x x x x
11	1 0 1 1	x x x x
12	1 1 0 0	x x x x
13	1 1 0 1	x x x x
14	1 1 1 0	x x x x
15	1 1 1 1	x x x x

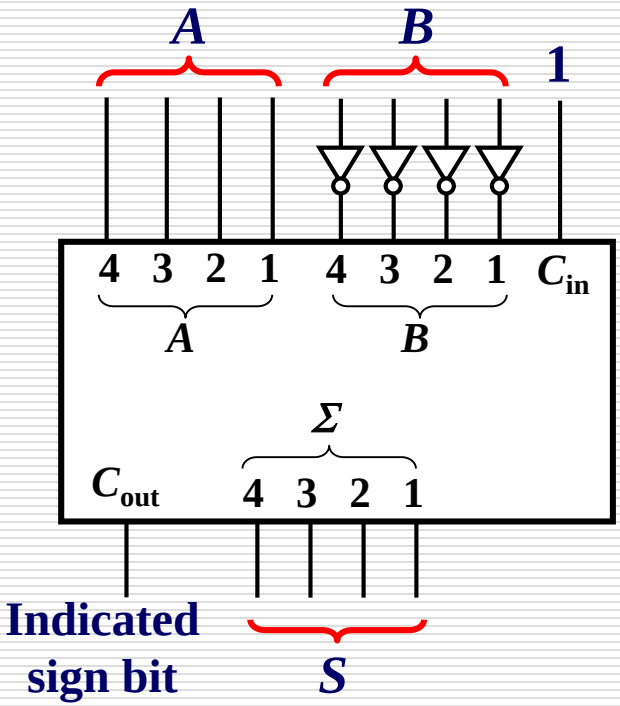
Adders

Application example

A 4-bit magnitude subtraction logic circuit

A and B are 4-bit magnitude numbers

$$S = A - B = A + (-B) = A + (B)_{2'com}$$
$$= A + (B)_{1'com} + 1$$



if **Indicated sign bit = 1,**

difference S is positive number in true

if **Indicated sign bit = 0,**

difference S is negative number in 2' complement

How to perform that the difference is always in true ?

Adders

Application example

A 4-bit magnitude subtraction logic circuit

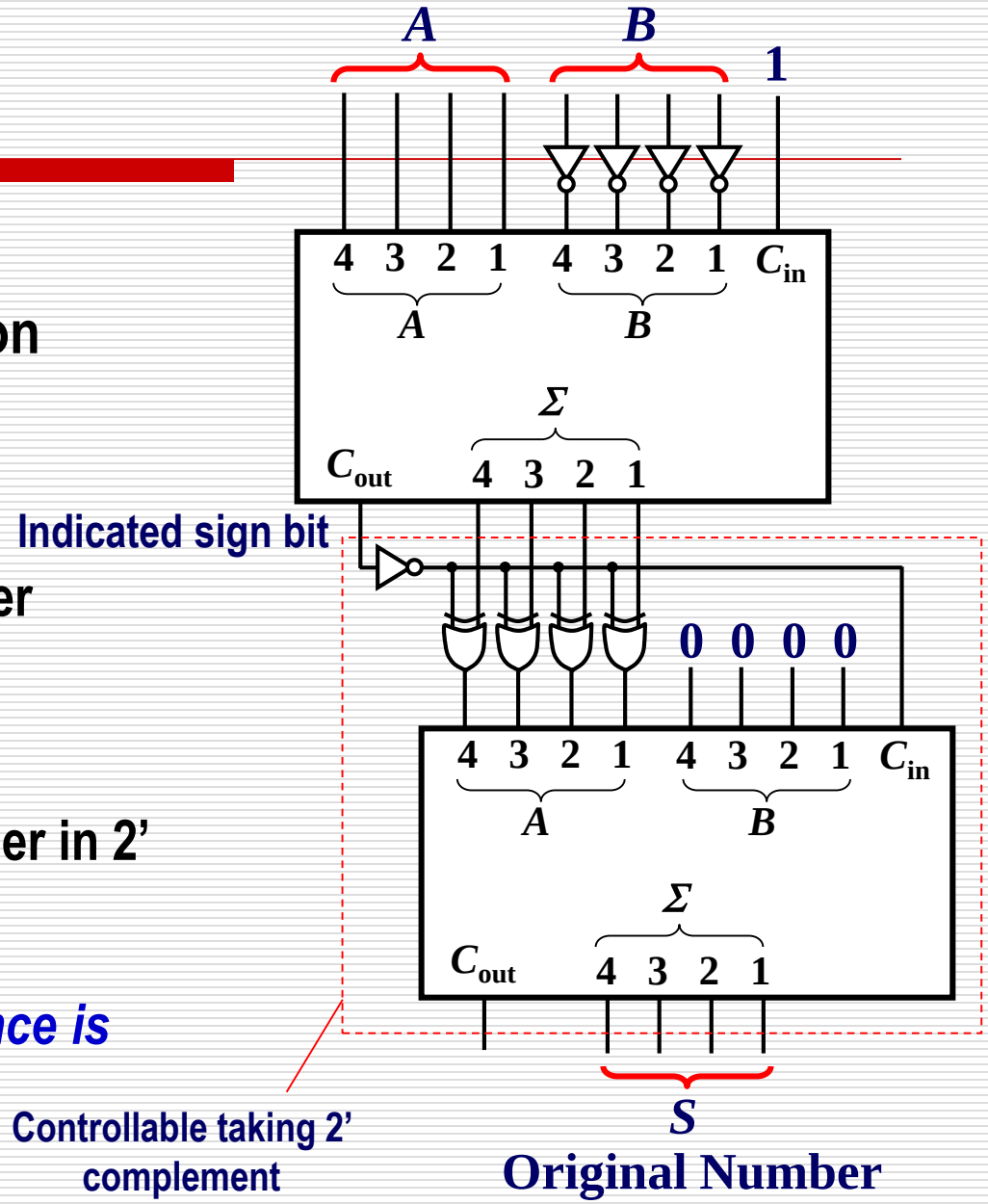
if **Indicated sign bit = 1,**

difference S is positive number in true

if **Indicated sign bit = 0,**

difference S is negative number in 2' complement

How to perform that the difference is always in true ?



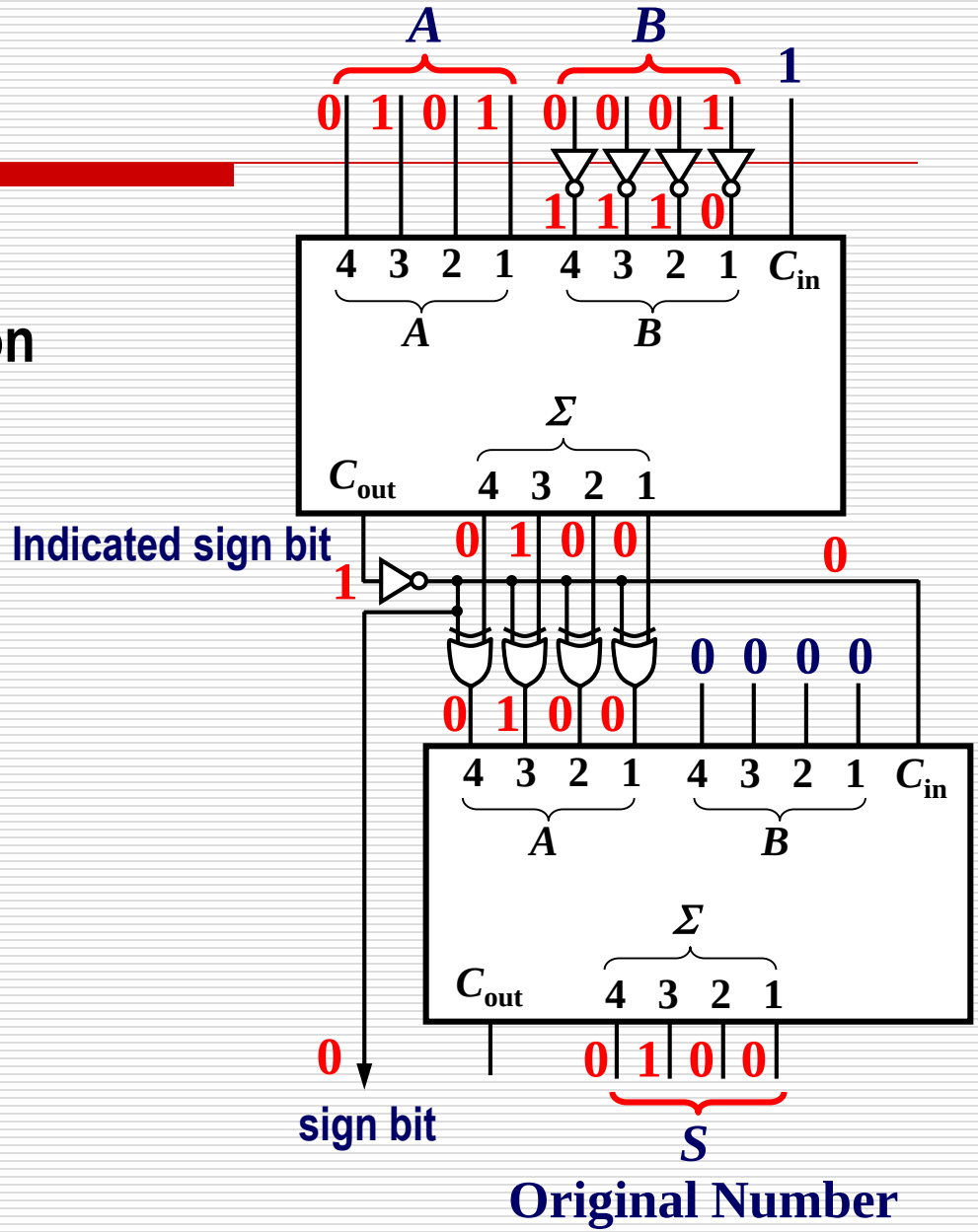
Adders

Application example

A 4-bit magnitude subtraction logic circuit

A=0101 ， B=0001

$$
\begin{array}{r}
0\ 1\ 0\ 1 \\
1\ 1\ 1\ 0 \\
+\qquad 1 \\
\hline
1\ 0\ 1\ 0\ 0 \\
\qquad\qquad\ \ \downarrow \\
0\ 1\ 0\ 0 \\
0\ 0\ 0\ 0 \\
+\qquad 0 \\
\hline
0\ 1\ 0\ 0
\end{array}
$$



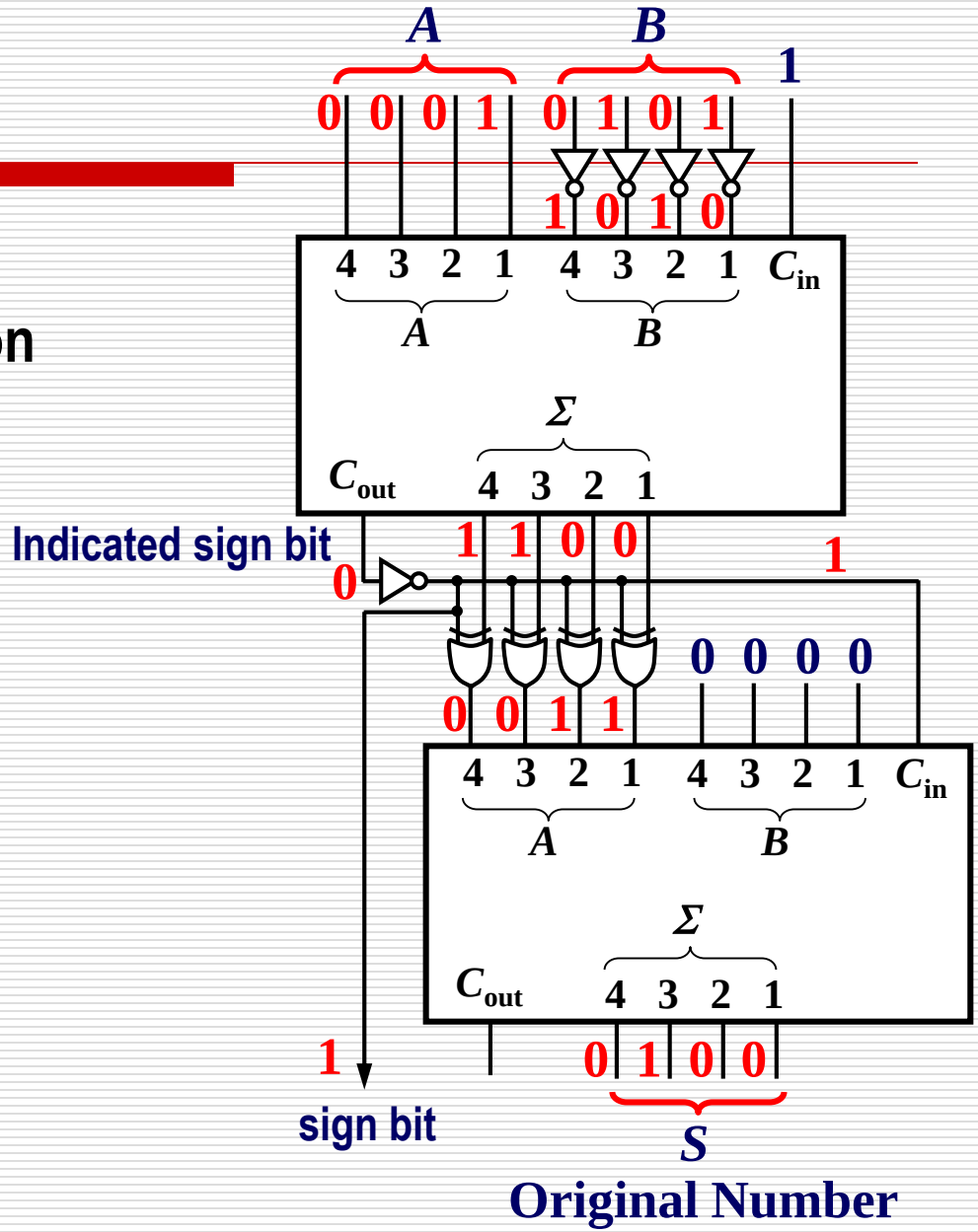
Adders

Application example

A 4-bit magnitude subtraction logic circuit

A=0001 ， B=0101

$$
\begin{array}{rr}
 & 0\ 0\ 0\ 1 \\
 & 1\ 0\ 1\ 0 \\
+ & \qquad 1 \\
\hline
 & 0\ 1\ 1\ 0\ 0 \\
 & 0\ 0\ 1\ 1 \\
 & 0\ 0\ 0\ 0 \\
+ & \qquad 1 \\
\hline
 & 0\ 1\ 0\ 0
\end{array}
$$



Adders

Application example

A 5-bit signed binary numbers addition/subtraction circuit. Here all numbers are in 2' complement.

Input numbers:

$A_4 A_3 A_2 A_1 A_0$

$B_4 B_3 B_2 B_1 B_0$

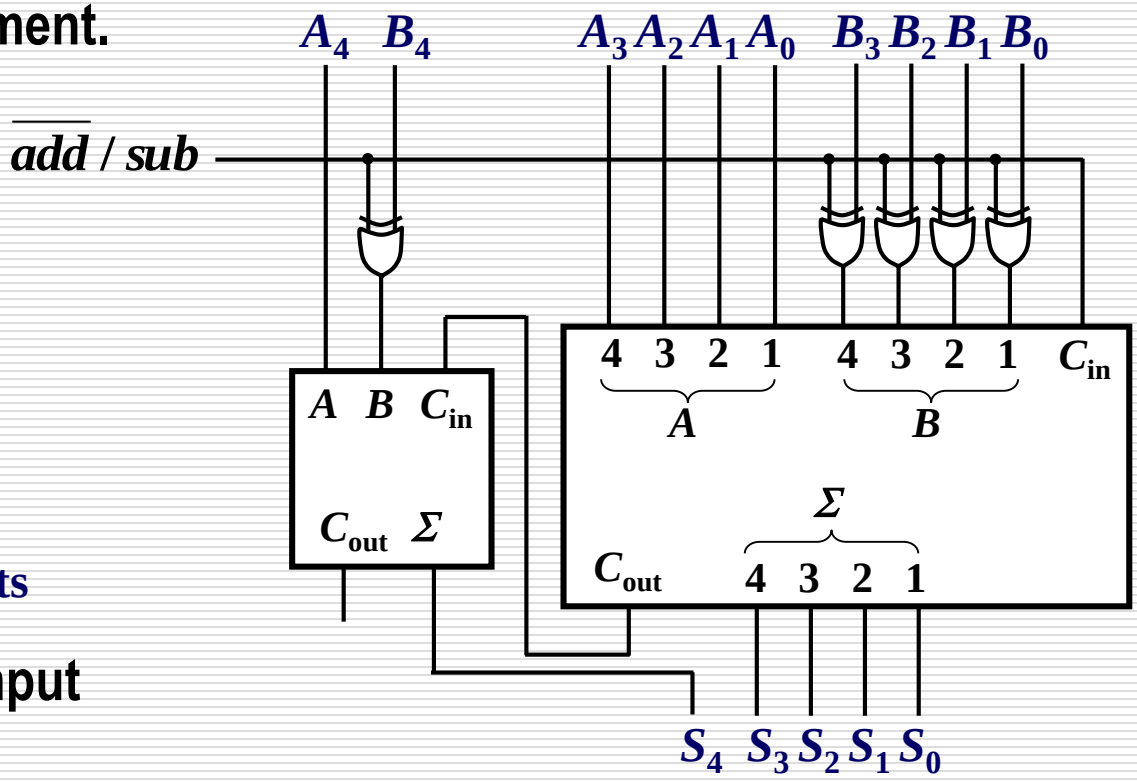
Output number:

$S_4 S_3 S_2 S_1 S_0$

Hereinto:

A_4 , B_4 and S_4 are sign bits

The $\overline{add}/sub$ is control input



Adders

Application example

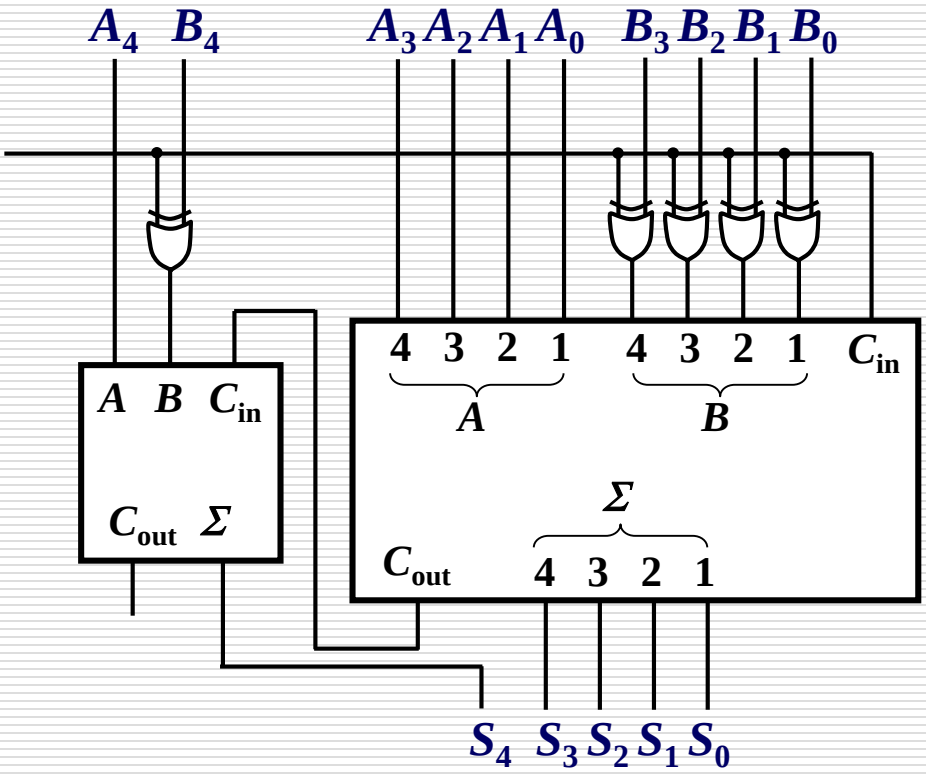
A 5-bit signed numbers addition/subtraction circuit. Here all numbers are in 2' complement.

When $\overline{add} / sub = 0$

The circuit performs addition operation

$$S_4S_3S_2S_1S_0 = A_4A_3A_2A_1A_0 + B_4B_3B_2B_1B_0$$

$$\begin{array}{r} A_4\,A_3\,A_2\,A_1\,A_0 \\ B_4\,B_3\,B_2\,B_1\,B_0 \\ +\qquad\qquad 0 \\ \hline S_4\,S_3\,S_2\,S_1\,S_0 \end{array}$$



(A_4 , B_4 and S_4 are sign bits)

Adders

Application example

A 5-bit signed numbers addition/subtraction circuit. Here all numbers are in 2' complement.

When $\overline{add}/sub = 1$ $\overline{add}/sub$

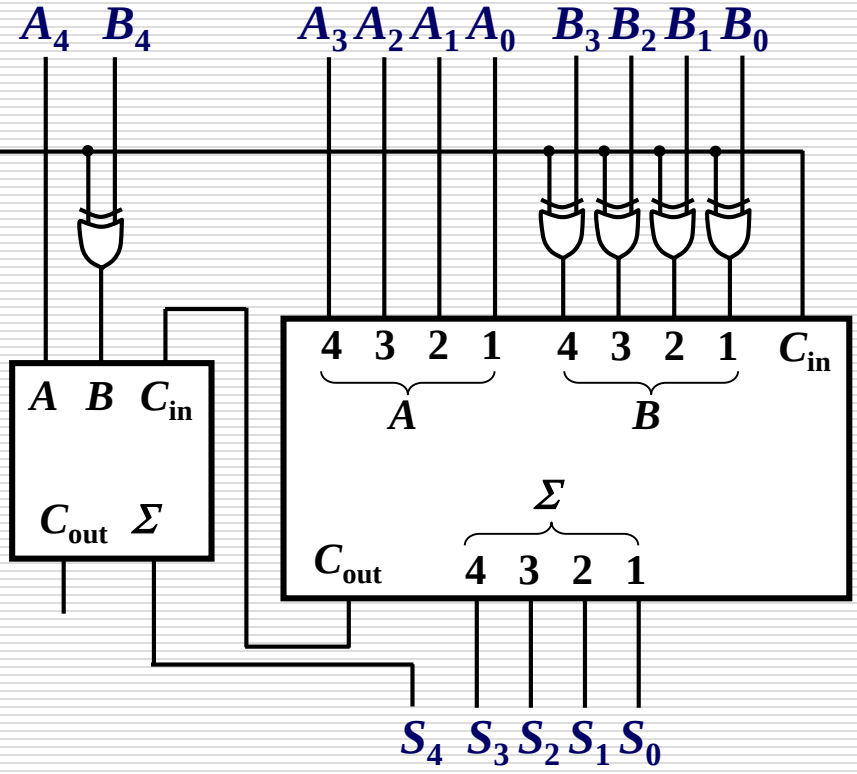
The circuit performs subtraction operation

$$S_4S_3S_2S_1S_0 = A_4A_3A_2A_1A_0 - B_4B_3B_2B_1B_0$$

$$= A_4A_3A_2A_1A_0 + (B_4B_3B_2B_1B_0)_{2'com}$$

$$\begin{array}{r} A_4\ A_3\ A_2\ A_1\ A_0 \\ \overline{B}_4\ \overline{B}_3\ \overline{B}_2\ \overline{B}_1\ \overline{B}_0 \\ +\qquad\qquad\quad 1 \\ \hline S_4\ S_3\ S_2\ S_1\ S_0 \end{array}$$

This circuit is maybe a danger of overflow



(A_4, B_4 and S_4 are sign bits)

Adders

Application example

A 5-bit signed numbers addition/subtraction circuit. Here all numbers are in 2' complement.

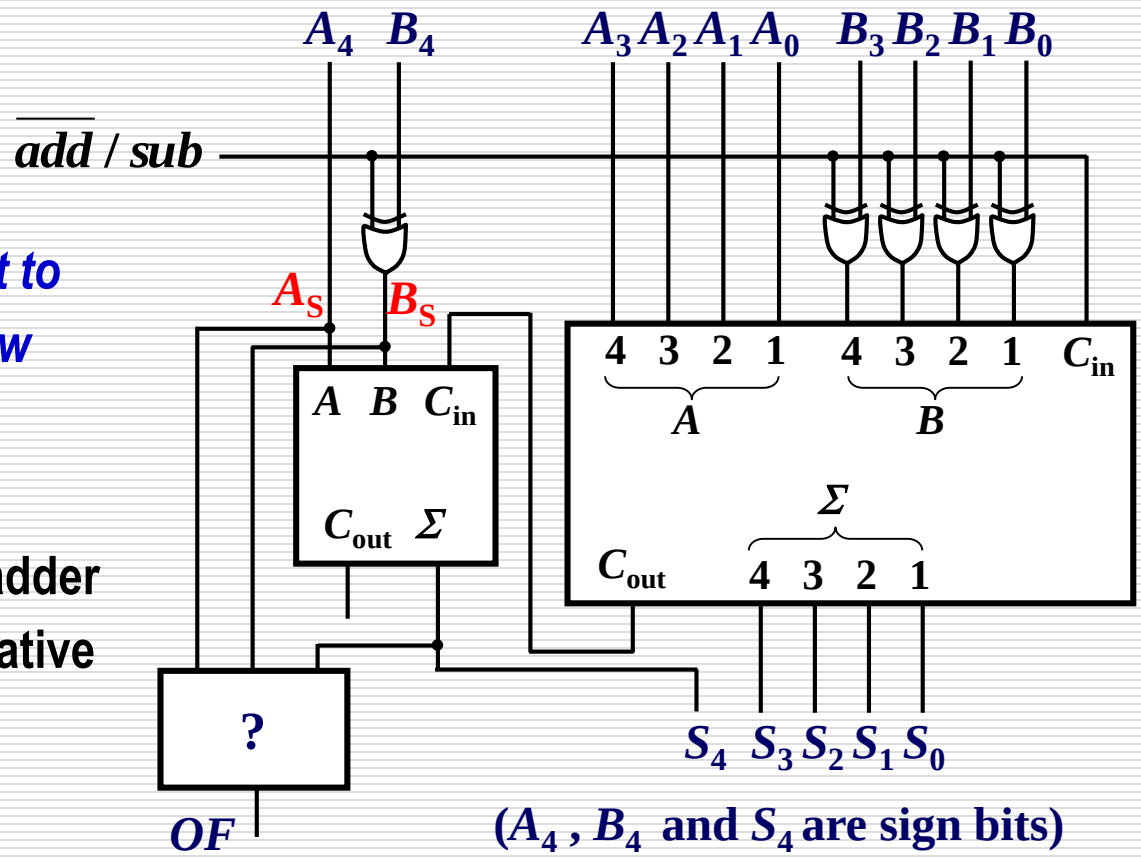
The circuit is maybe a danger of overflow

How to design a logic circuit to indicate whether the overflow occurs?

The condition of overflow :

Both input numbers of the adder are positive or both are negative but the result is opposite.

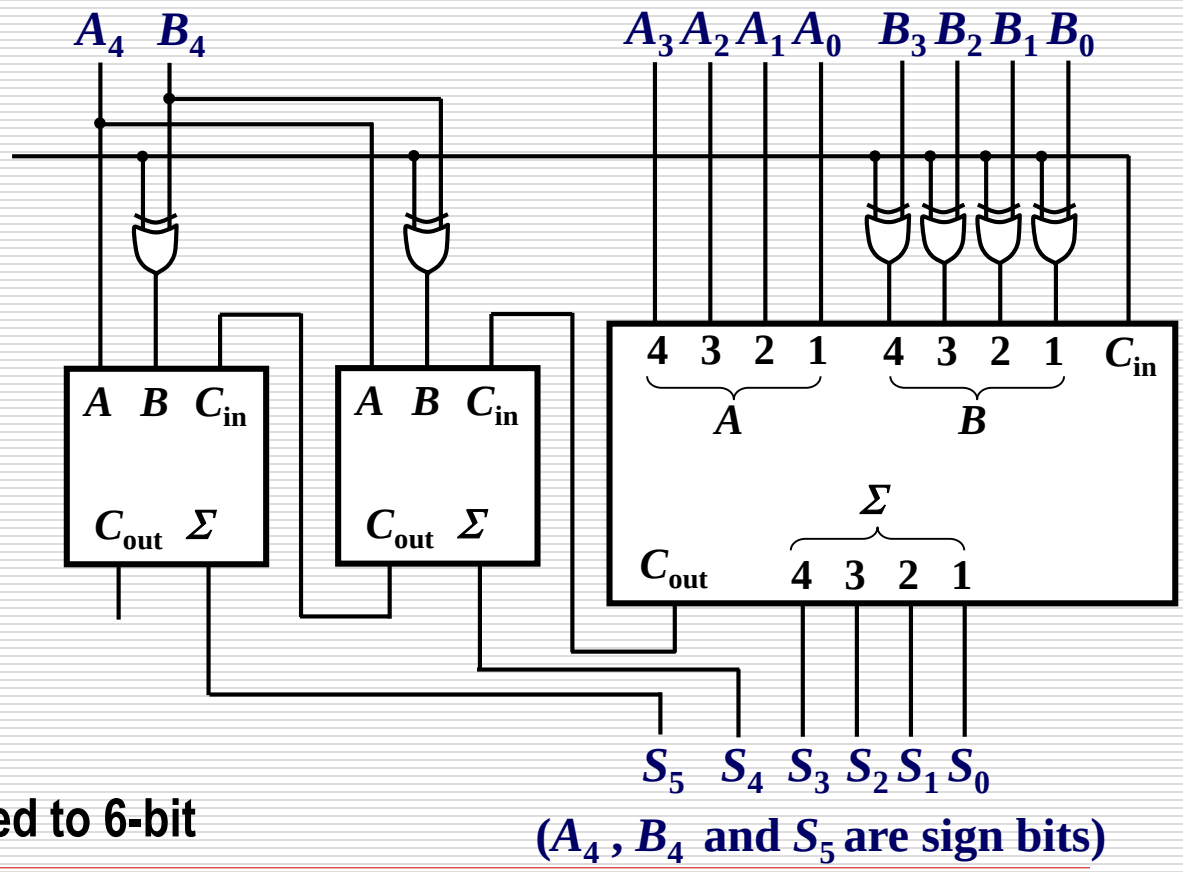
$$OF = \overline{A}_S \overline{B}_S S_4 + A_S B_S \overline{S}_4$$



Adders

Application example

A 5-bit signed numbers addition/subtraction circuit. Here all numbers are in 2' complement.



How to solve overflow?

by extending magnitude bits

If the number is positive, the extending bits need be supplied with **0**s

If the number is negative, the extending bits need be supplied with **1**s

The output has been expanded to 6-bit

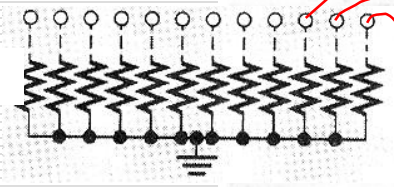
Adders

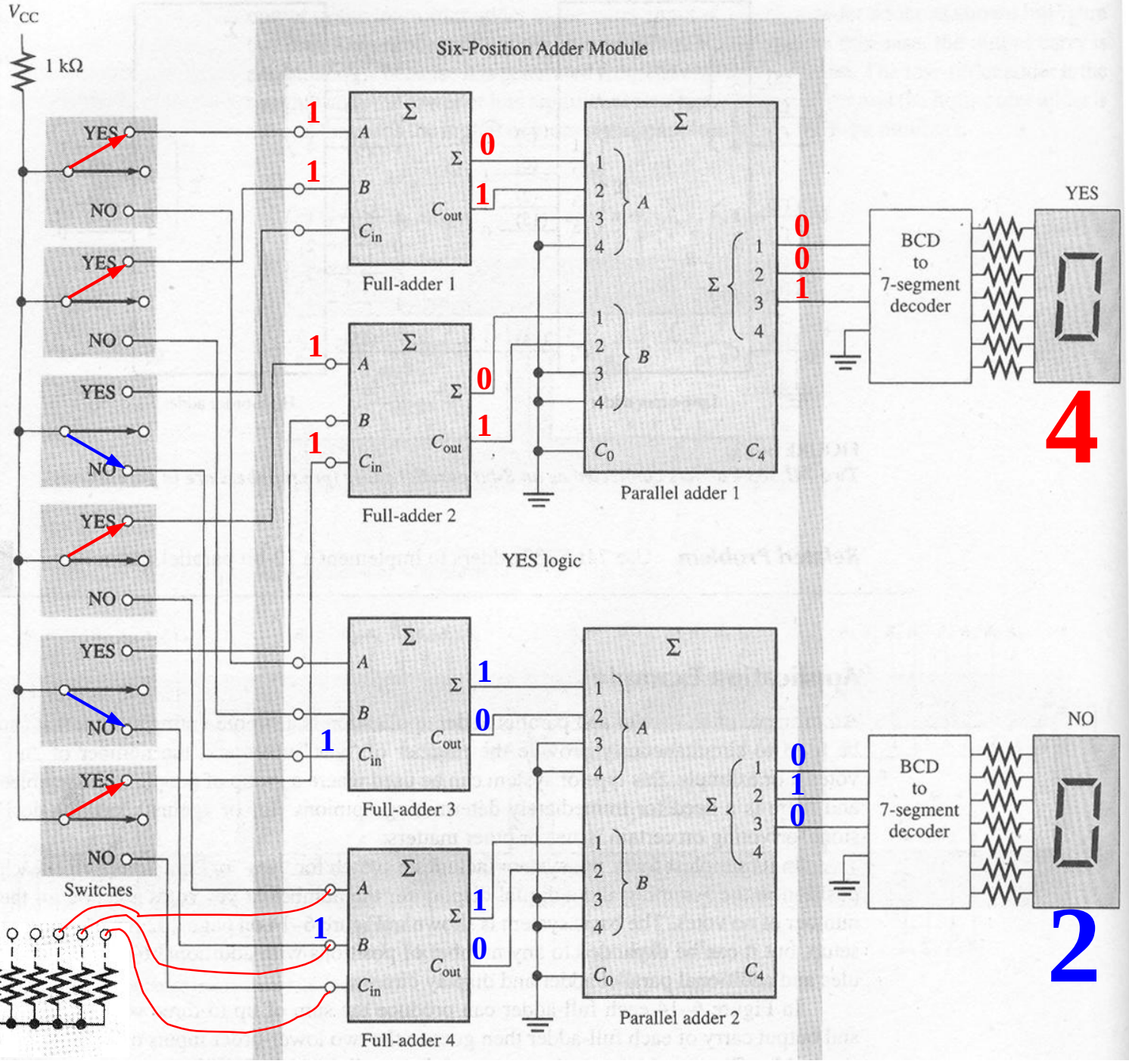
Application example

6-input voting system

Display the number of “yes” votes and the number of “no” votes

Each resistor connects to a full-adder input





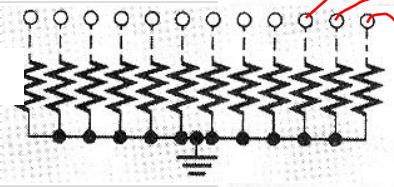
Adders

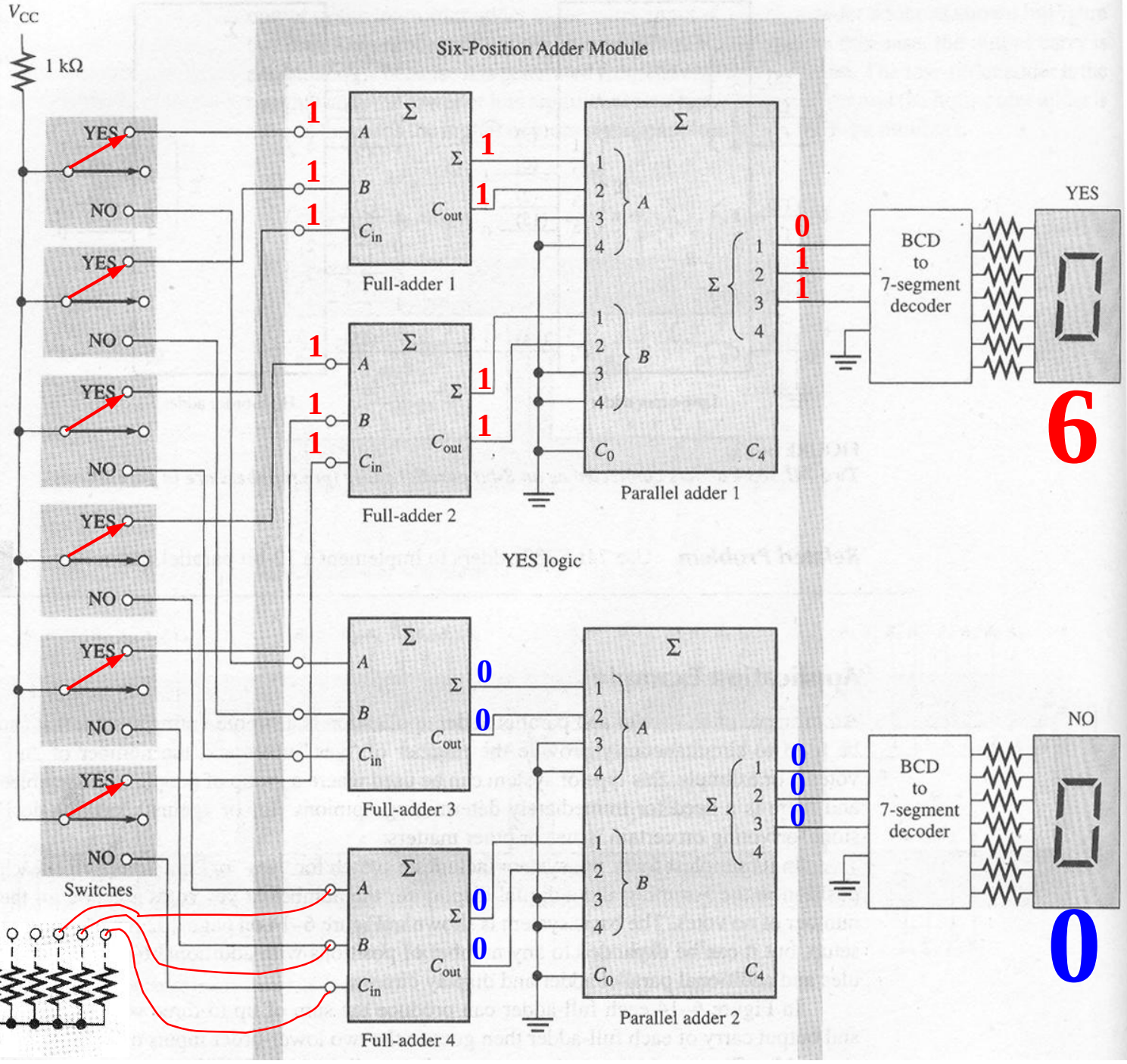
Application example

6-input voting system

Display the number of “yes” votes and the number of “no” votes

Each resistor connects to a full-adder input





The brief statement for this lecture content

- A practical combinational circuit may have dozens of inputs and outputs and could require hundreds, thousands, even millions of terms to describe as a sum of products, and billions and billions of rows to describe in a truth table. Thus, most real combinational logic design problems are too large to solve by "brute-force" application of theoretical techniques.
- How could any human being conceive of such a complex logic circuit in the first place? The key is structured thinking. A complex circuit or system is conceived as a collection of smaller subsystems, each of which has a much simpler description.

The brief statement for this lecture content

- In combinational logic design, there are several straightforward structures-encoders, decoders, multiplexers, comparators, adders and the like-that turn up quite regularly as *building blocks* in larger systems.
- As we have seen these structures, that are regarded as building blocks usually, have their *specific functions* and simpler description.

The brief statement for this lecture content

- Although the applications of MSI are declining since PLD and FPGA are used more and more, the standard MSI functions like encoders, decoders, multiplexers …, are widely used in PLD, FPGA and ASIC as building blocks (or "*standard cells*" or "*macros*").
- We should more pay attention to the functions of these building blocks and make light of them as MSI.