Digital Electronics Basic Concepts

Digital Electronics

Internshala Trainings

NUMBER SYSTEM

Numbers

Every number system is associated with a base or radix

A positional notation is commonly used to express numbers

$$(a_5a_4a_3a_2a_1a_0)_r = a_5r^5 + a_4r^4 + a_3r^3 + a_2r^2 + a_1r^1 + a_0r^0$$

The decimal system has a base of 10 and uses symbols (0,1,2,3,4,5,6,7,8,9) to represent numbers

$$(2009)_{10} = 2\times10^3 + 0\times10^2 + 0\times10^1 + 9\times10^0$$

$$(123.24)_{10} = 1\times10^2 + 2\times10^1 + 3\times10^0 + 2\times10^{-1} + 4\times10^{-2}$$

An octal number system has a base 8 and uses symbols (0,1,2,3,4,5,6,7)

$$(2007)_8 = 2\times8^3 + 0\times8^2 + 0\times8^1 + 7\times8^0$$

What decimal number does it represent?

$$(2007)_8 = 2\times512 + 0\times64 + 0\times8^1 + 7\times8^0 = 1033$$

A hexadecimal system has a base of 16

Number	Symbol
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

$$(2BC9)_{10} = 2\times16^{3} + B\times16^{2} + C\times16^{1} + 9\times16^{0}$$

How do we convert it into decimal number?

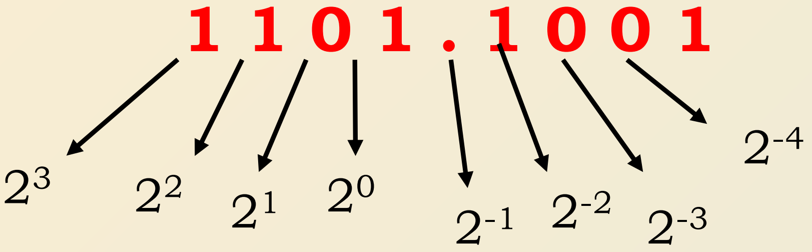
$$(2BC9)_{10} = 2\times4096 + 11\times256 + 12\times16^{1} + 9\times16^{0} = 11209$$

A Binary system has a base 2 and uses only two symbols 0, 1 to represent all the numbers

$$(1101)_2 = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0$$

Which decimal number does this correspond to ?

$$(1101)_2 = 1\times 8 + 1\times 4 + 0\times 2^1 + 1\times 2^0 = 13$$



1 1 0 1 . 1 0 0 1

2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024(K)
2^{20}	1048576(M)

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}
0.5	0.25	0.125	0.0625	0.03125	0.015625

Developing Fluency with Binary Numbers

1 1 0 0 1 = ?	25
1100001 = ?	64+32+1=97
0.101 = ?	0.5+0.125=0.625
11.001 = ?	3+0.125=3.125

Converting decimal to binary number

Convert 45 to binary number

$$(45)_{10} = b_n b_{n-1} ....b_0$$

$$45 = b_n 2^n + b_{n-1} 2^{n-1} ....b_1 2^1 + b_0$$

Divide both sides by 2

$$\frac{45}{2} = 22.5 = b_n 2^{n-1} + b_{n-1} 2^{n-2} ....b_1 2^0 + b_0 \times 0.5$$

$$22 + 0.5 = b_n 2^{n-1} + b_{n-1} 2^{n-2} .... + b_1 2^0 + b_0 \times 0.5$$

$$\Rightarrow \; b_0 = 1$$

$$22+0.5=b_n 2^{n-1}+b_{n-1}2^{n-2}\ldots\ldots\ldots+b_1 2^0+b_0\times 0.5$$ $$\Rightarrow\ b_0=1$$

$$22=b_n 2^{n-1}+b_{n-1}2^{n-2}\ldots\ldots\ldots b_2 2^1+b_1 2^0$$

Divide both sides by 2

$$\frac{22}{2}=11=b_n 2^{n-2}+b_{n-1}2^{n-3}\ldots\ldots\ldots b_2 2^0+b_1\times 0.5$$ $$\Rightarrow\ b_1=0$$

$$11=b_n 2^{n-2}+b_{n-1}2^{n-3}\ldots\ldots+b_3 2^1+b_2 2^0$$

$$5.5=b_n 2^{n-3}+b_{n-1}2^{n-4}\ldots\ldots+b_3 2^0+0.5b_2$$ $$\Rightarrow\ b_2=1$$

$$5=b_n 2^{n-3}+b_{n-1}2^{n-4}\ldots\ldots b_4 2^1+b_3 2^0$$

$$5 = b_n 2^{n-3} + b_{n-1} 2^{n-4} \ldots\ldots b_4 2^1 + b_3 2^0$$

$$2.5 = b_n 2^{n-4} + b_{n-1} 2^{n-5} \ldots\ldots b_4 2^0 + 0.5 b_3$$

$$\Rightarrow \; b_3 = 1$$

$$2 = b_n 2^{n-4} + b_{n-1} 2^{n-5} \ldots\ldots b_5 2^1 + b_4 2^0$$

$$1 = b_n 2^{n-5} + b_{n-1} 2^{n-6} \ldots\ldots b_5 2^0 + 0.5 b_4$$

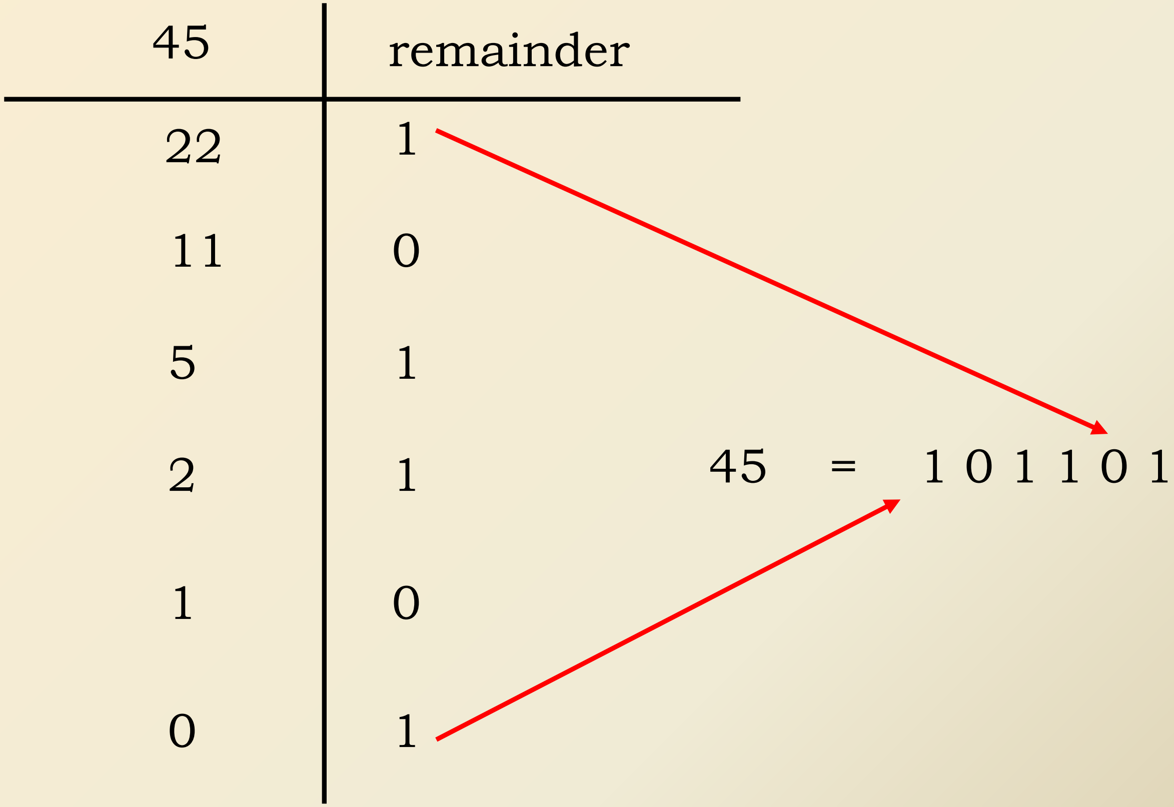
$$\Rightarrow \; b_4 = 0$$

$$\Rightarrow \; b_5 = 1$$

$$(45)_{10} = b_5 b_4 b_3 b_2 b_1 b_0 = 101101$$

Converting decimal to binary number

Method of successive division by 2



45	remainder
22	1
11	0
5	1
2	1
1	0
0	1

45 = 1 0 1 1 0 1

Convert $(153)_{10}$ to octal number system

$$(153)_{10} = (b_n b_{n-1}.......b_0)_8$$

$$(153)_{10} = b_n 8^n + b_{n-1} 8^{n-1}.......b_1 8^1 + b_0$$

Divide both sides by 8

$$\frac{153}{8} = 19.125 = b_n 8^{n-1} + b_{n-1} 8^{n-2}.......b_1 8^0 + \frac{b_0}{8}$$

$$\Rightarrow \frac{b_0}{8} = 0.125$$

$$\Rightarrow b_0 = 1$$

153	remainder
19	1
2	3
0	2

$153 = (231)_8$

Converting decimal to binary number

Convert $(0.35)_{10}$ to binary number

$$(0.35)_{10} = 0.b_{-1}b_{-2}b_{-3}.......b_{-n}$$

$$0.35 = 0 + b_{-1}2^{-1} + b_{-2}2^{-2} + ...b_{-n}2^{-n}$$

How do we find the b_{-1} b_{-2} ...coefficients?

Multiply both sides by 2

$$0.7 = b_{-1} + b_{-2}2^{-1} + ...b_{-n}2^{-n+1}$$

$$\Rightarrow b_{-1} = 0$$

$$0.7 = b_{-2}2^{-1} + b_{-3}2^{-2} + ...b_{-n}2^{-n+1}$$

$$0.7 = b_{-2}2^{-1} + b_{-3}2^{-2} + ...b_{-n}2^{-n+1}$$

Multiply both sides by 2

$$1.4 = b_{-2} + b_{-3}2^{-1} + ...b_{-n}2^{-n+2}$$

Note that ½+1/4+1/8+……≤1

$$\Rightarrow b_{-2} = 1$$

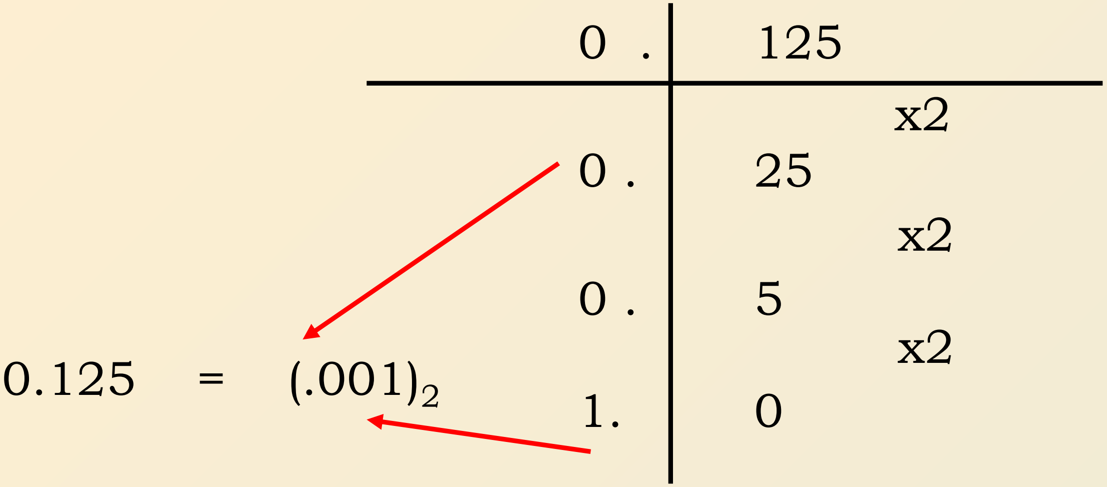
$$0.4 = b_{-3}2^{-1} + b_{-4}2^{-2}.......b_{-n}2^{-n+2}$$

$$0.8 = b_{-3} + b_{-4}2^{-1}.......b_{-n}2^{-n+3}$$

$$\Rightarrow b_{-3} = 0$$

Converting decimal to binary number

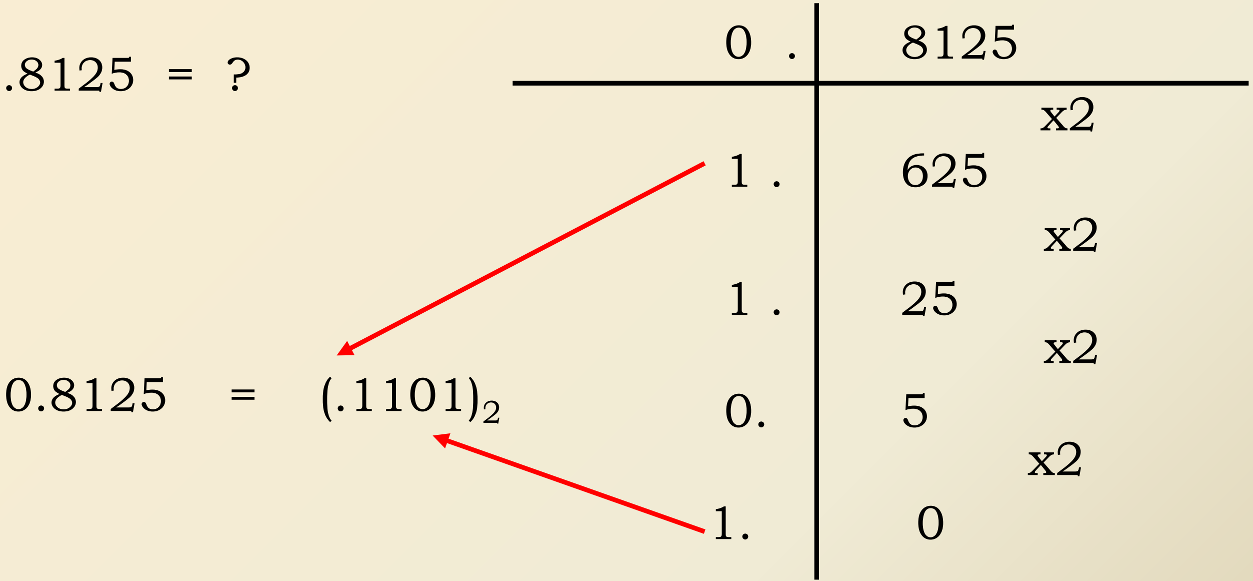
0.125 = ?



0 .	125
	x2
0 .	25
	x2
0 .	5
	x2
1.	0

$0.125 = (.001)_2$

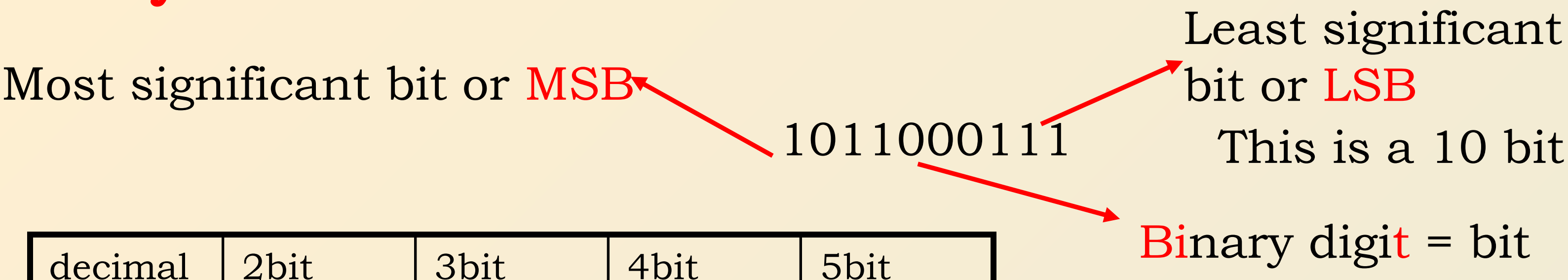
0.8125 = ?



0 .	8125
	x2
1 .	625
	x2
1 .	25
	x2
0.	5
	x2
1.	0

$0.8125 = (.1101)_2$

Binary numbers



Most significant bit or MSB

1011000111

Least significant bit or LSB

This is a 10 bit number

Binary digit = bit

decimal	2bit	3bit	4bit	5bit
0	00	000	0000	00000
1	01	001	0001	00001
2	10	010	0010	00010
3	11	011	0011	00011
4		100	0100	00100
5		101	0101	00101
6		110	0110	00110
7		111	0111	00111
8			1000	01000
9			1001	01001
10			1010	01010
11			1011	01011
12			1100	01100
13			1101	01101
14			1110	01110
15			1111	01111

N-bit binary number can represent numbers from 0 to 2^N -1

Converting Binary to Hex and Hex to Binary

$$(b_7b_6b_5b_4b_3b_2b_1b_0)_b = (h_1, h_0)_{Hex}$$

$$b_7 2^7 + b_6 2^6 + b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 b_1 2^1 + b_0 = h_1 16^1 + h_0$$

$$\underbrace{(b_7 2^3 + b_6 2^2 + b_5 2^1 + b_4)}_{h_1} 2^4 + \underbrace{(b_3 2^3 + b_2 2^2 b_1 2^1 + b_0)}_{h_0} = h_1 16^1 + h_0$$

$$(10110011)_b = (1011)(0011) = (B3)_{Hex}$$

$$(110011)_b = (11)(0011) = (33)_{Hex}$$

$$(EC)_{Hex} = (1110)(1100) = (11101100)_b$$

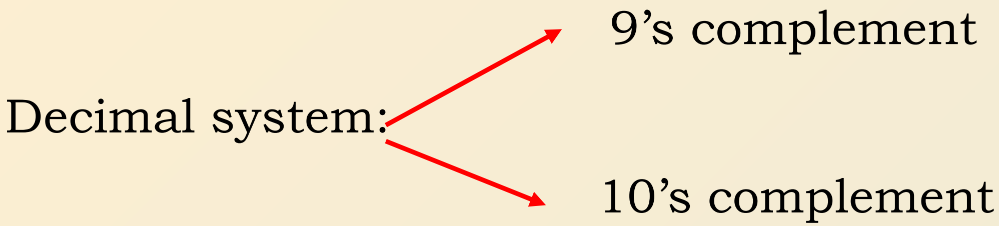
Number	Symbol
0(0000)	0
1(0001)	1
2(0010)	2
3(0011)	3
4(0100)	4
5(0101)	5
6(0110)	6
7(0111)	7
8(1000)	8
9(1001)	9
10(1010)	A
11(1011)	B
12(1100)	C
13(1101)	D
14(1110)	E
15(1111)	F

Binary Addition/Subtraction

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

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

Complement of a number



Decimal system:

9’s complement

10’s complement

9’s complement of n-digit number x is 10^n -1 -x

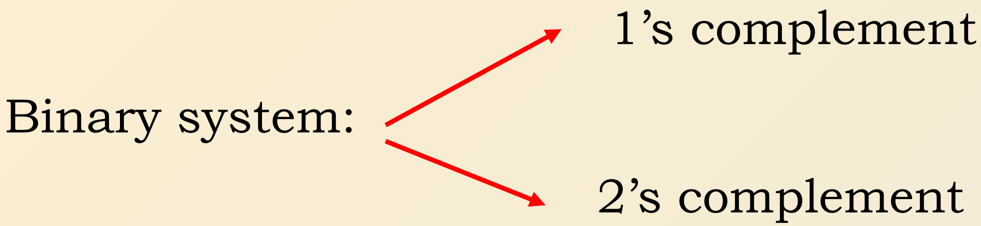
10’s complement of n-digit number x is 10^n -x

9’s complement of 85 ? $10^2 - 1 - 85$ $99 - 85 = 14$

$$9\text{'s complement of } 123 = 999 - 123 = 876$$

$$10\text{'s complement of } 123 = 9\text{'s complement of } 123 + 1 = 877$$

Complement of a binary number



Binary system:

1’s complement

2’s complement

1’s complement of n-bit number x is $2^n - 1 - x$

2’s complement of n-bit number x is $2^n - x$

1’s complement of 1011 ? $2^4 - 1 - 1011$ $1111 - 1011 = 0100$

1’s complement is simply obtained by flipping a bit (changing 1 to 0 and 0 to 1)

1's complement of 1001101 = ?

0110010

$$2\text{'s complement of } 1010 = 1\text{'s complement of } 1010 + 1 = 0110$$

$$2\text{'s complement of } 110010 =$$

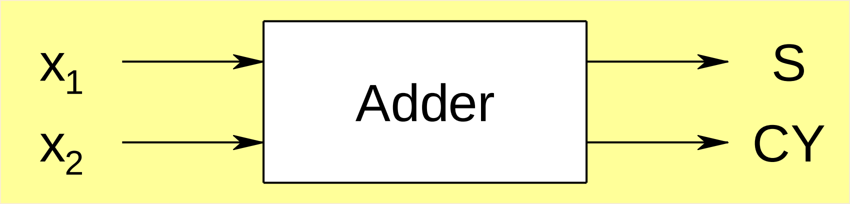
Leave all least significant 0’s as they are, leave first 1 unchanged and then flip all subsequent bits

$$001110$$

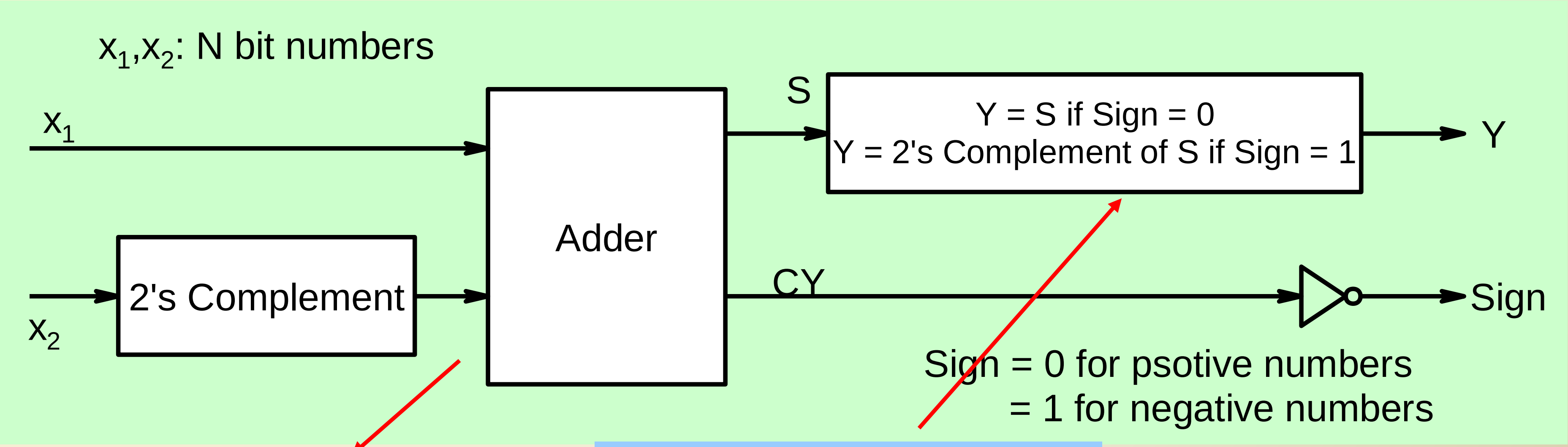
$$1011 \rightarrow 0101$$

$$101101100 \rightarrow 010010100$$

Advantages of using 2's complement



Can we carry out Y = $X_1 - X_2$ using such an adder?



x_1, x_2: N bit numbers

Y = S if Sign = 0
Y = 2's Complement of S if Sign = 1

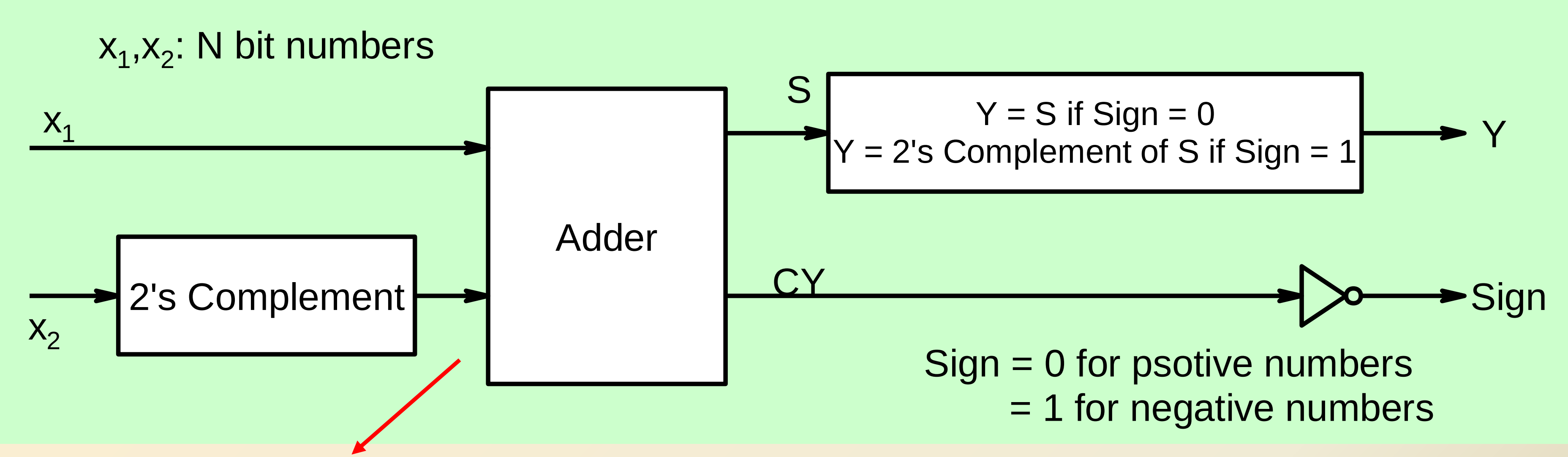
Sign = 0 for psotive numbers
= 1 for negative numbers

$$2^N - x_2$$

$$(CY, S) = x_1 + 2^N - x_2$$

Note that carry will be there only if $x_1 - x_2$ is positive as 2^N is N+1 bits (1 followed by N zeros)

Advantages of using 2's complement



x_1,x_2: N bit numbers
x_1
x_2
2's Complement
Adder
S
Y = S if Sign = 0
Y = 2's Complement of S if Sign = 1
Y
CY
Sign
Sign = 0 for psotive numbers
= 1 for negative numbers

$$2^N - x_2$$

$$(CY, S) = x_1 + 2^N - x_2$$

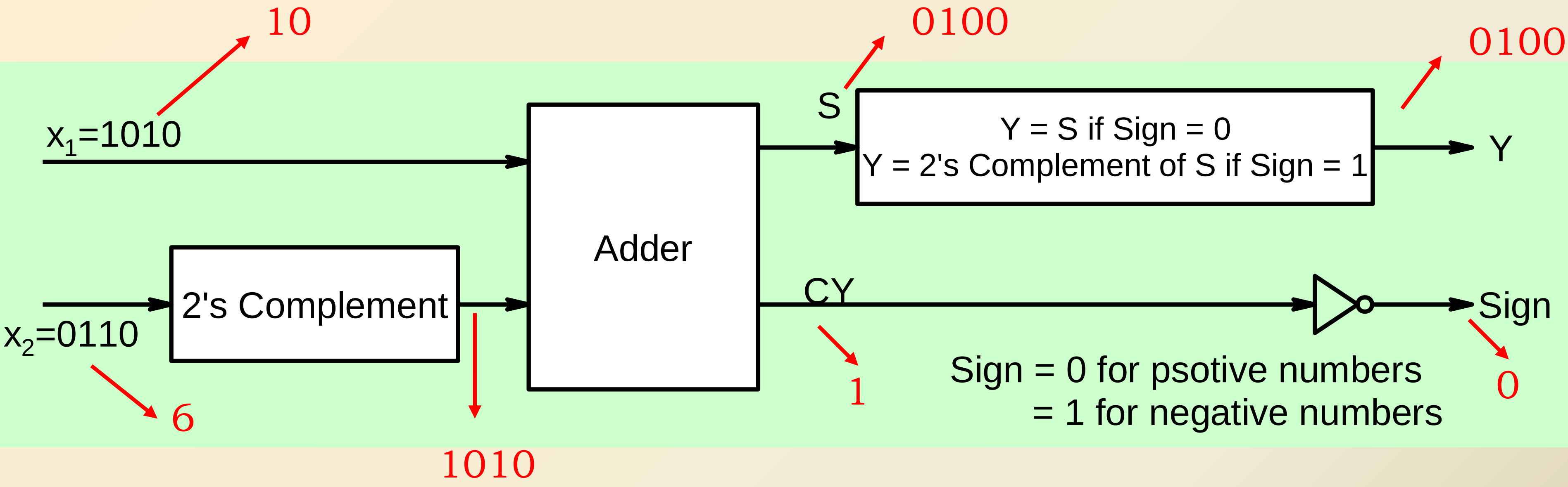
Note that carry will be there only if $x_1 - x_2$ is positive as 2^N is N+1 bits (1 followed by N zeros)

A zero carry implies a negative number whose magnitude ($x_2 - x_1$) can be found as follows:

$$S = x_1 + 2^N - x_2$$

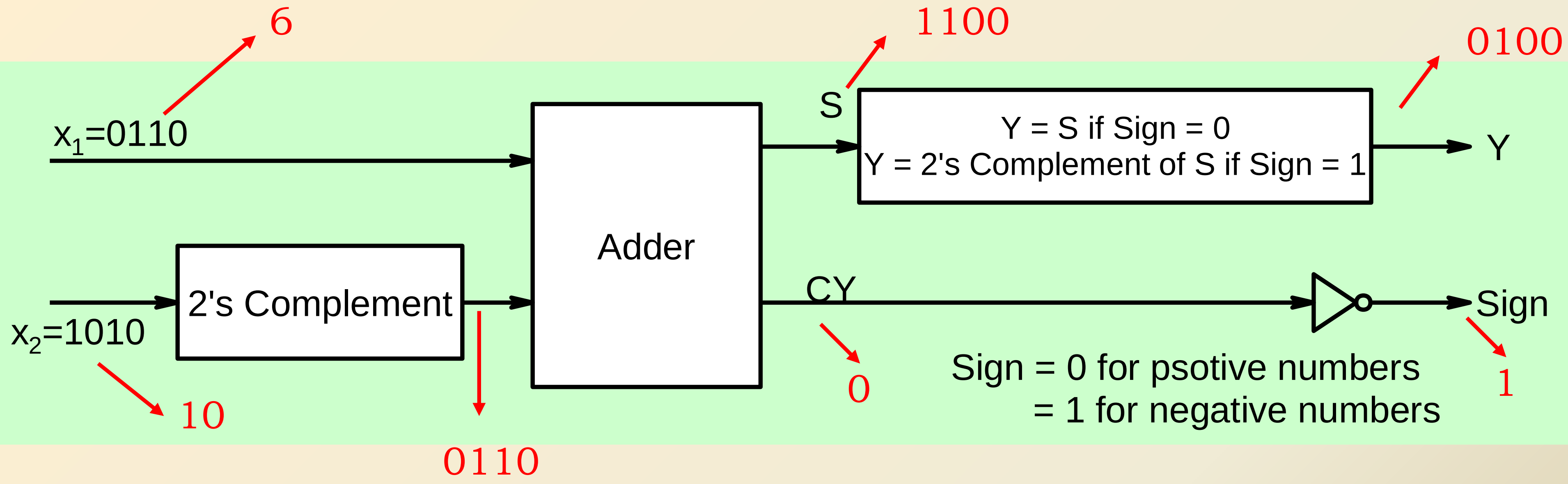
$$\text{2'scomplement of } S = 2^N - (x_1 + 2^N - x_2) = x_2 - x_1$$

Example



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

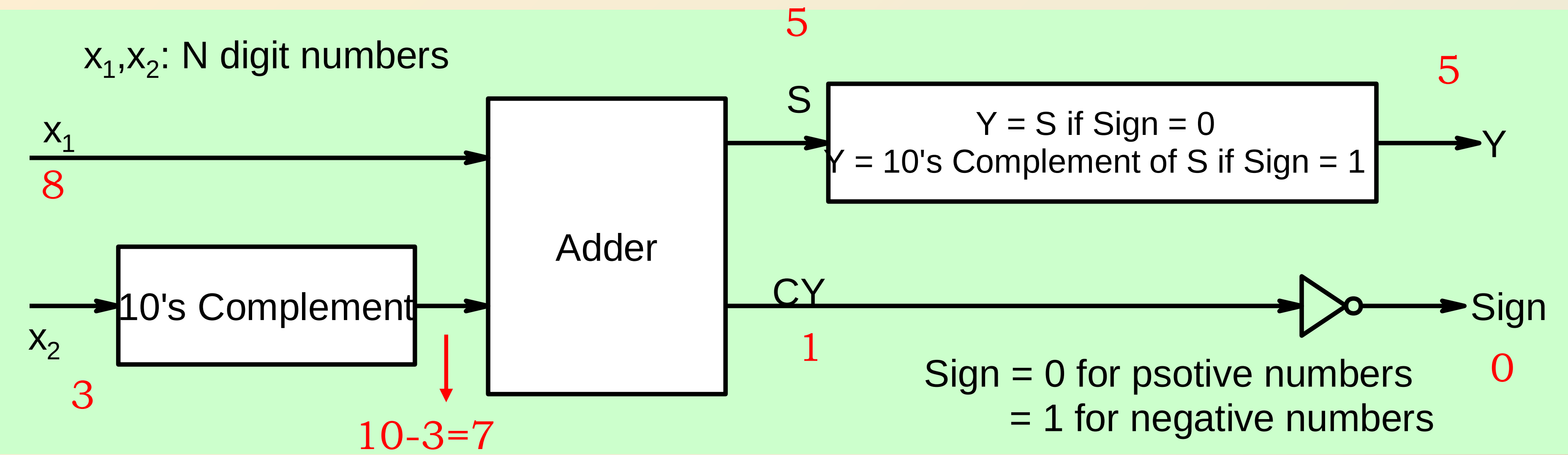
Example



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

It makes sense to use adder as a subtractor as well provided additional circuit required for carrying out 2's complement is simple

Subtraction using 10's complement



x_1,x_2: N digit numbers
x_1
8
x_2
3
10's Complement
10-3=7
Adder
5
S
Y = S if Sign = 0
Y = 10's Complement of S if Sign = 1
5
Y
CY
1
Sign
0
Sign = 0 for psotive numbers
= 1 for negative numbers

This way of subtraction would make sense only if subtracting a number x_2 from 10^N is much simpler than directly subtracting it directly from x_1

Representing positive and negative binary numbers

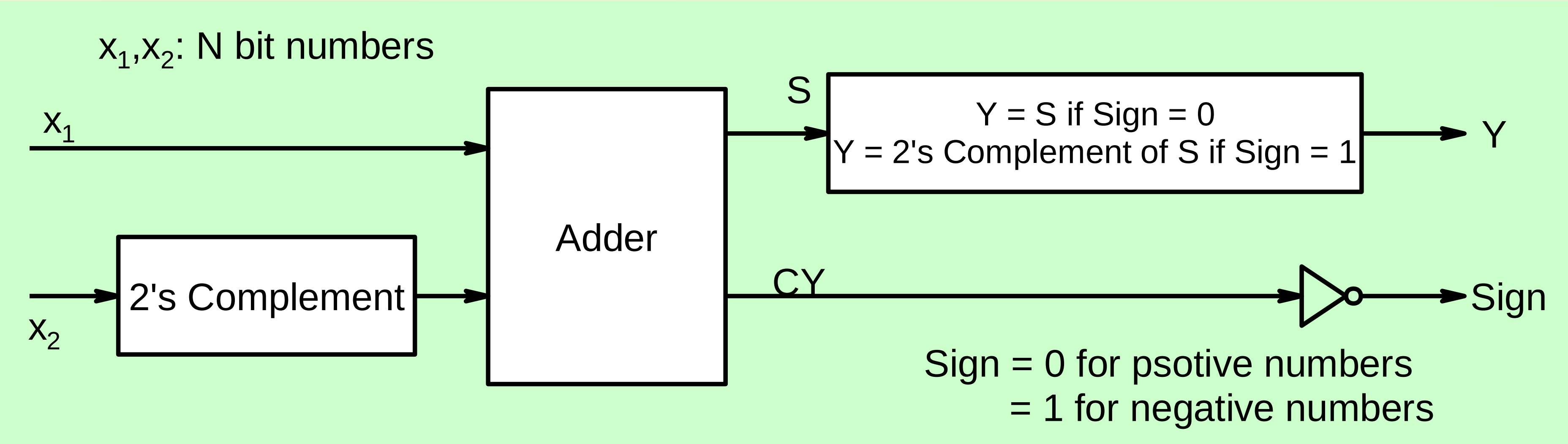
One extra bit is required to carry sign information. Sign bit = 0 Represents positive number and Sign bit = 1 represents negative number

decimal	Signed Magnitude
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

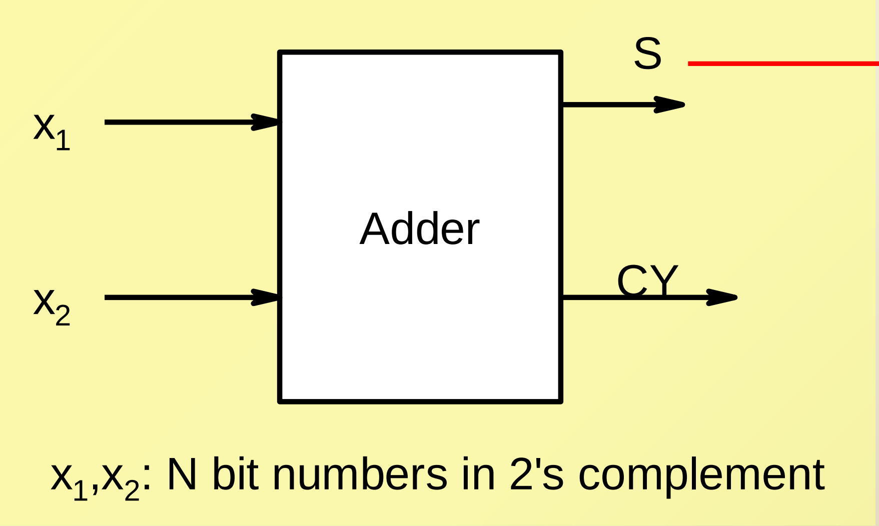
decimal	Signed 1's complement
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

decimal	Signed 2's complement
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001

If we represent numbers in 2’s complement form carrying out subtraction is same as addition

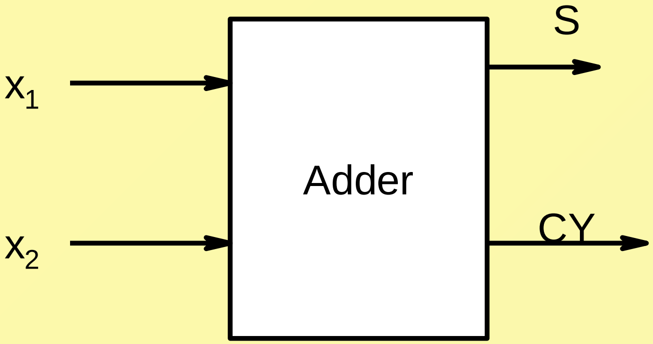


x_1,x_2: N bit numbers
x_1
x_2
2's Complement
Adder
S
Y = S if Sign = 0
Y = 2's Complement of S if Sign = 1
Y
CY
Sign
Sign = 0 for psotive numbers
= 1 for negative numbers



Answer is in 2’s complement form

Example



x_1,x_2: N bit numbers in 2's complement

$$\begin{array}{r} +5 \\ +2 \\ \hline +7 \\ \hline \end{array} \qquad \begin{array}{r} 0\,1\,0\,1 \\ +\,0\,0\,1\,0 \\ \hline 0\,1\,1\,1 \\ \hline \end{array}$$

$$\begin{array}{r} +5 \\ -2 \\ \hline +3 \\ \hline \end{array} \qquad \begin{array}{r} 0\,1\,0\,1 \\ +\,1\,1\,1\,0 \\ \hline 0\,0\,1\,1 \\ \hline \end{array}$$

$$\begin{array}{r} -5 \\ +2 \\ \hline -3 \\ \hline \end{array} \qquad \begin{array}{r} 1\,0\,1\,1 \\ +\,0\,0\,1\,0 \\ \hline 1\,1\,0\,1 \\ \hline \end{array}$$

$$\begin{array}{r} -5 \\ -2 \\ \hline -7 \\ \hline \end{array} \qquad \begin{array}{r} 1\,0\,1\,1 \\ +\,1\,1\,1\,0 \\ \hline 1\,0\,0\,1 \\ \hline \end{array}$$

2’s complement is 0011 = 3

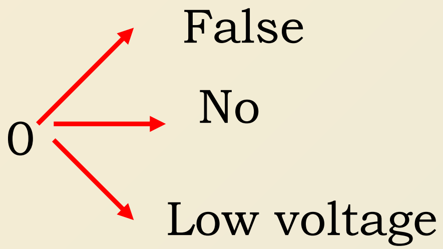
2’s complement is 0111 = 7

BOOLEAN ALGEBRA

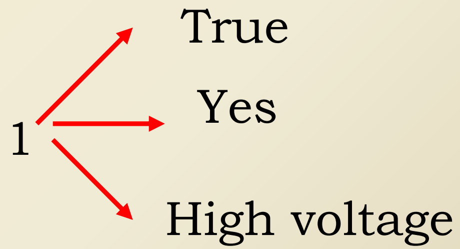
Boolean Algebra

Algebra on Binary numbers

A variable x can take two values {0,1}



Basic operations:



AND: $y = x_1 . x_2$

Y is 1 if and only if both x_1 and x_2 are 1, otherwise zero

Truth Table

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Basic operations:

OR: $y = x_1 + x_2$

Y is 1 if either x_1 and x_2 is 1. Or y= 0 if and only if both variables are zero

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

NOT: $y = \overline{x}$

x	y
0	1
1	0

Boolean Algebra

Basic Postulates

P1: $x + 0 = x$

P2: $x + y = y + x$

P3: $x.(y+z) = x.y+x.z$

P4: $x + \overline{x} = 1$

P1: $x . 1 = x$

P2: $x . y = y . x$

P3: $x+y.z = (x+y).(x+z)$

P4: $x . \overline{x} = 0$

Basic Theorems

T1: $x + x = x$

T2: $x + 1 = 1$

T3: $(\overline{\overline{x}}) = x$

T4: $x + (y+z) = (x+y)+z$

T5 $\overline{(x+y)} = \overline{x} . \overline{y}$ (DeMorgan's theorem)

T6: $x+ x.y = x$

T1: $x . x = x$

T2: $x . 0 = 0$

T4: $x . (y.z) = (x.y).z$

T5 $\overline{(x.y)} = \overline{x} + \overline{y}$ (DeMorgan's theorem)

T6: $x.(x+y) = x$

Proving theorems

P1: $x + 0 = x$

P2: $x + y = y + x$

P3: $x.(y+z) = x.y+x.z$

P4: $x + \overline{x} = 1$

P1: $x . 1 = x$

P2: $x . y = y . x$

P3: $x+y.z = (x+y).(x+z)$

P4: $x . \overline{x} = 0$

Prove T1: $x + x = x$

$x + x = (x+x). 1$ (P1)

$= (x+x). (x+\overline{x})$ (P4)

$= x + x.\overline{x}$ (P3)

$= x + 0$ (P4)

$= x$ (P1)

Prove T1: $x . x = x$

$x . x = x.x+ 0$ (P1)

$= x.x + x.\overline{x}$ (P4)

$= x . (x+\overline{x})$ (P3)

$= x . 1$ (P4)

$= x$ (P1)

Proving theorems

P1: $x + 0 = x$

P2: $x + y = y + x$

P3: $x.(y+z) = x.y+x.z$

P4: $x + \overline{x} = 1$

P1: $x . 1 = x$

P2: $x . y = y . x$

P3: $x+y.z = (x+y).(x+z)$

P4: $x . \overline{x} = 0$

Prove : $x + 1 = 1$

$x + 1 = x+(x+ \overline{x})$

$= (x+x)+ \overline{x}$

$= x + \overline{x}$

$= 1$

$x + x .y = x$

$= x . 1 + x. y$

$= x. (1+ y)$

$= x . 1$

$=x$

$x + \overline{x} .y = x+y$

$= (x + \overline{x}). (x+ y)$

$= 1. (x+ y)$

$= x + y$

DeMorgan's theorem

$$\overline{(x_1 + x_2 + x_3 + \text{....})} = \overline{x_1} . \overline{x_2} . \overline{x_3} .$$

$$\overline{(x_1 . x_2 . x_3 \text{.....})} = (\overline{x_1} + \overline{x_2} + \overline{x_3} + \text{.....})$$

Simplification of Boolean expressions

$$\overline{(x_1 + x_2 + x_3 + \dots)} = \overline{x_1} . \overline{x_2} . \overline{x_3} .$$

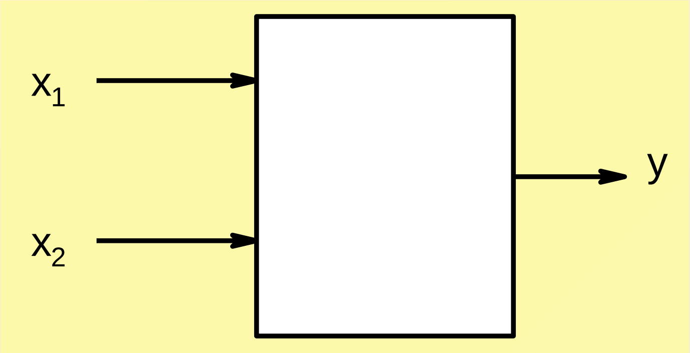
$$\overline{(\overline{x_1}.x_2 + \overline{x_2}.x_3)} = ?$$

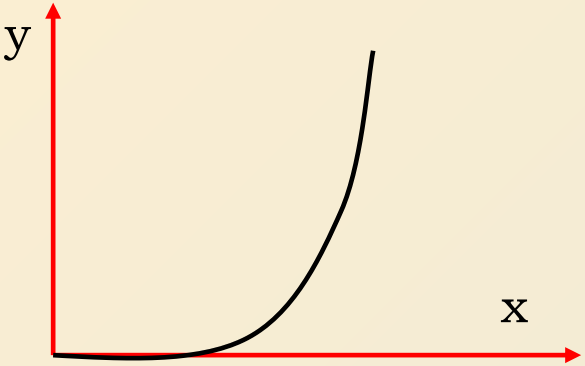
$$\overline{(x_1 . x_2 . x_3 \dots)} = (\overline{x_1} + \overline{x_2} + \overline{x_3} + \dots)$$

$$= (x_1 + \overline{x_2}) . (x_2 + \overline{x_3})$$

$$= x_1 . x_2 + x_1 . \overline{x_3} + \overline{x_2} . \overline{x_3}$$

Function of Boolean variables





$$y = x^2$$

x_1	x_2	y
0	0	0
0	1	1
1	0	0
1	1	0

Y = 1 when x_1 is 0 and x_2 is 1

$$y = \overline{x_1} \, . \, x_2$$

Boolean expression

Obtaining Boolean expressions from truth Table

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	0

$$y = \overline{x_1} \,.\, \overline{x_2}$$

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0

$$y = x_1 \,.\, \overline{x_2}$$

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

$$\overline{x_1} \,.\, \overline{x_2}$$

$$x_1 \,.\, x_2$$

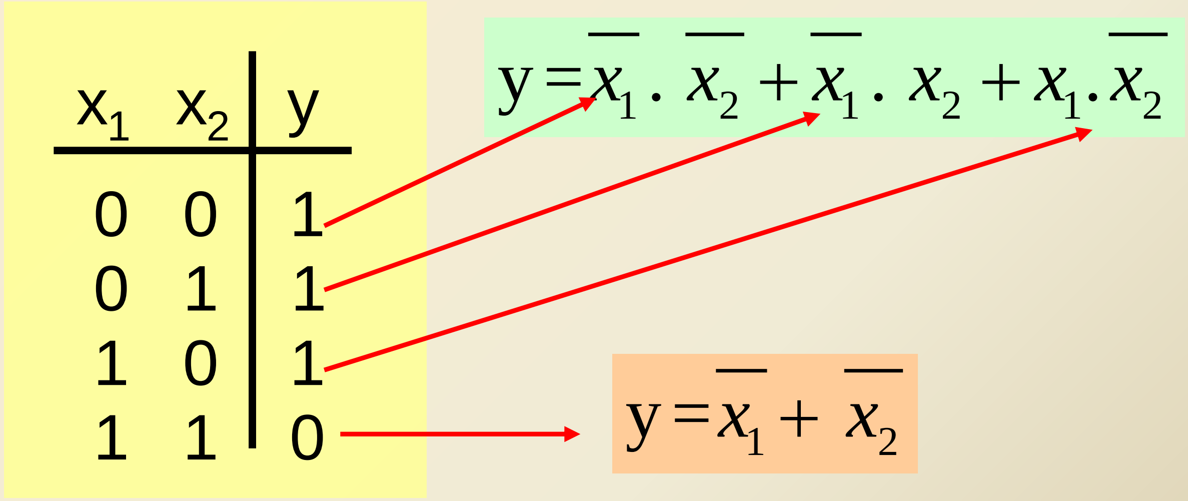
$$y = \overline{x_1} \,.\, \overline{x_2} + x_1 . x_2$$

Obtaining Boolean expressions from truth Table

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$y = \overline{x_1} . x_2 + x_1 . \overline{x_2}$$

Instead of writing expressions as sum of terms that make y equal to 1, we can also write expressions using terms that make y equal to 0



x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

$$y = \overline{x_1} . \overline{x_2} + \overline{x_1} . x_2 + x_1 . \overline{x_2}$$

$$y = \overline{x_1} + \overline{x_2}$$

x_1	x_2	y
0	0	1
0	1	0
1	0	1
1	1	1

$$y = x_1 + \overline{x_2}$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$$y = x_1 + x_2$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$x_1 + x_2$$

$$y = (x_1 + x_2).(\overline{x_1} + \overline{x_2})$$

$$\overline{x_1} + \overline{x_2}$$

Obtaining Boolean expressions from truth Table

x_1	x_2	x_3	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$y = \overline{x_1}.\overline{x_2}.x_3 + \overline{x_1}.x_2.x_3 + x_1.\overline{x_2}.x_3 + x_1.x_2.x_3$$

Sum of Products (SOP) form

$$y = (x_1 + x_2 + x_3).(x_1 + \overline{x_2} + x_3).(\overline{x_1} + x_2 + x_3).(\overline{x_1} + \overline{x_2} + x_3)$$

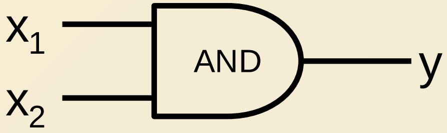
Product of Sum (POS) form

IMPLEMENTATION OF BOOLEAN EXPRESSIONS

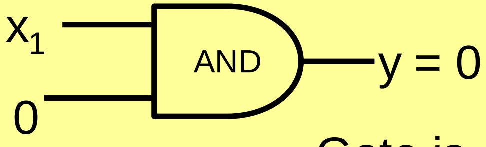
Implementing Boolean expressions

Elementary Gates

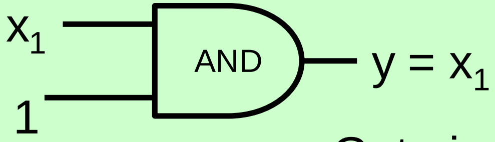
AND: $y = x_1 . x_2$



Why call it a gate?

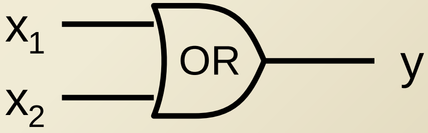


Gate is closed

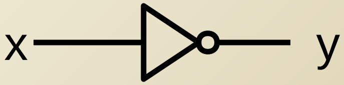


Gate is open

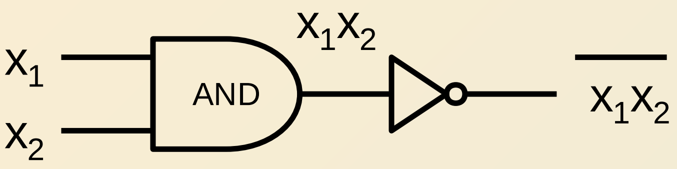
OR: $y = x_1 + x_2$

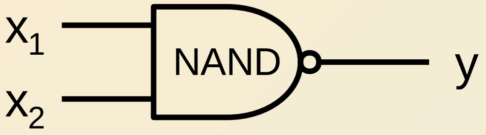


NOT: $y = \overline{x}$

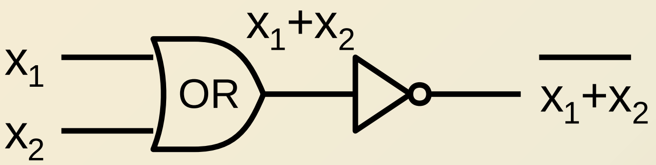


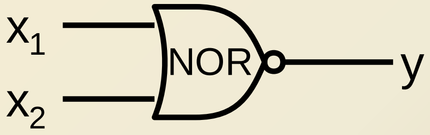
NAND: $y = \overline{x_1 . x_2}$





NOR: $y = \overline{x_1 + x_2}$

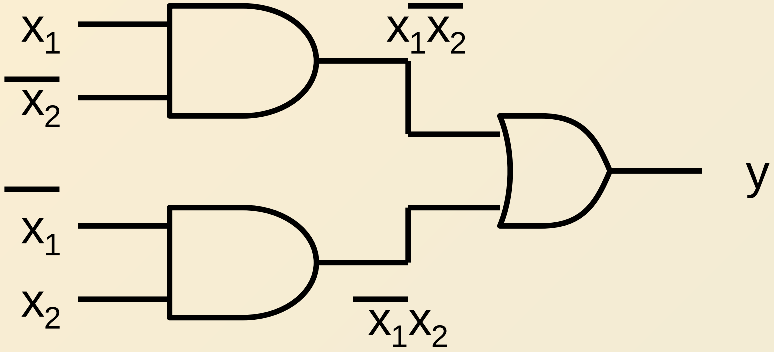


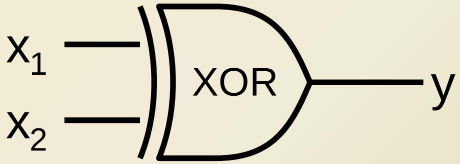


XOR: $y = x_1 \oplus x_2 = x_1 . \overline{x_2} + \overline{x_1} . x_2$

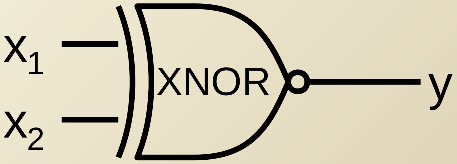
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Y is 1 if only one variable is 1 and the other is zero





XNOR: $y = x_1 \square\ x_2 = x_1 . x_2 + \overline{x_1} . \overline{x_2}$

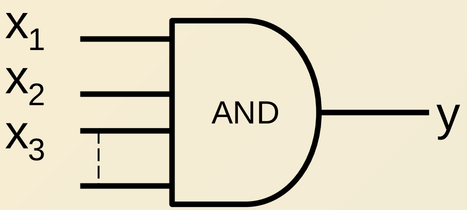


Y is 1 if only both variables are either 0 or 1

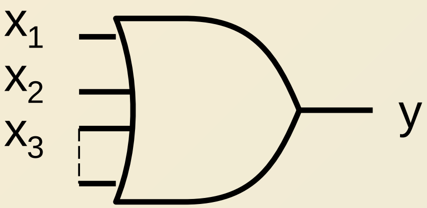
$$y = x_1 \square\ x_2 = \overline{x_1 \oplus x_2}$$

Gates with more than 2 inputs

AND: $y = x_1 . x_2 . x_3 \ldots$



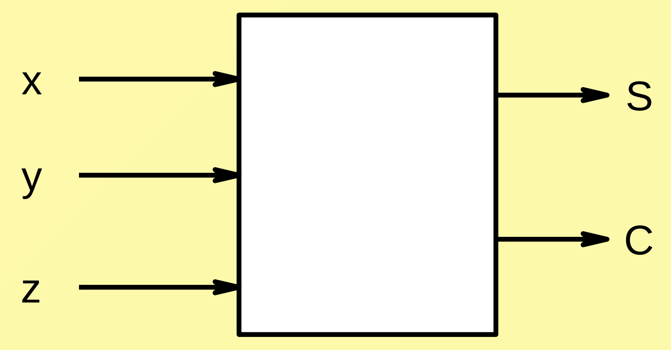
OR: $y = x_1 + x_2 + x_3 + \ldots.$



XOR: $y = x_1 \oplus x_2 \oplus x_3 = x_1 . \overline{x_2}\,\overline{x_3} + \overline{x_1} . x_2 . \overline{x_3} + \overline{x_1} . \overline{x_2} . x_3 + x_1 . x_2 . x_3$

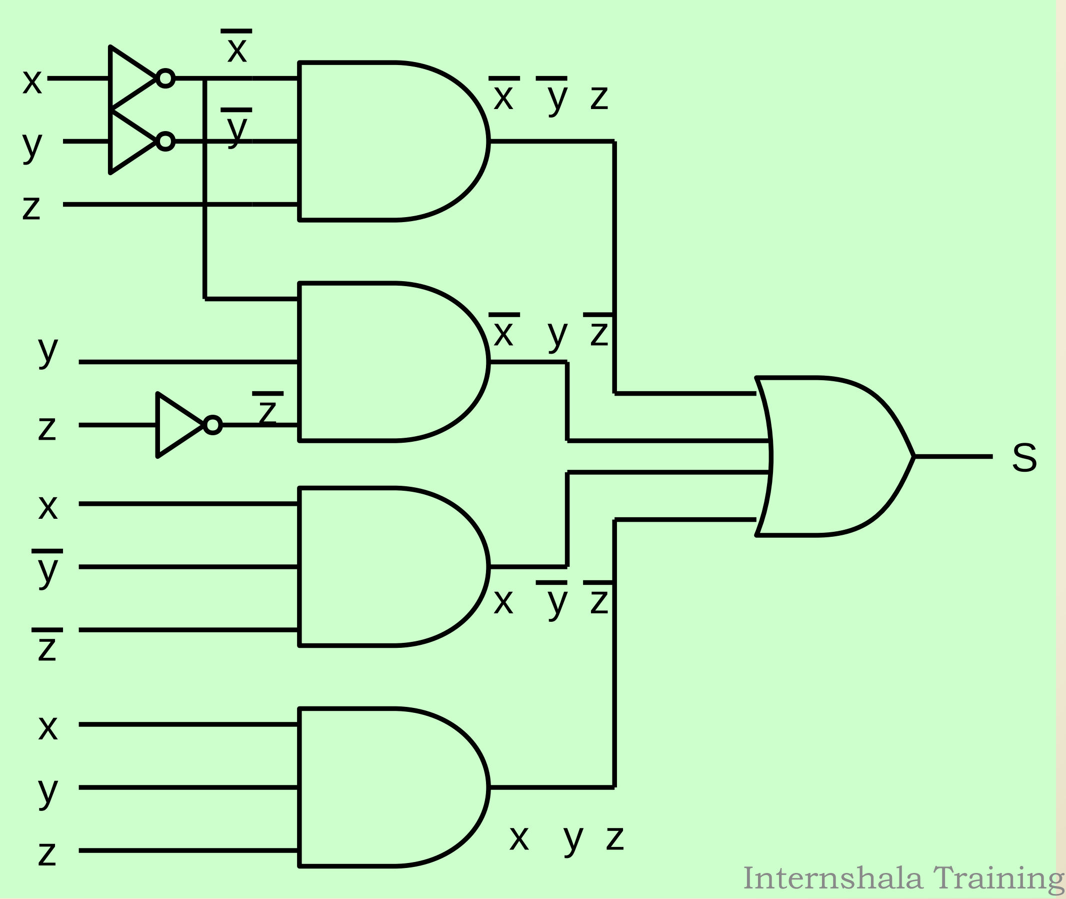
Y = 1 only if odd number of inputs is 1

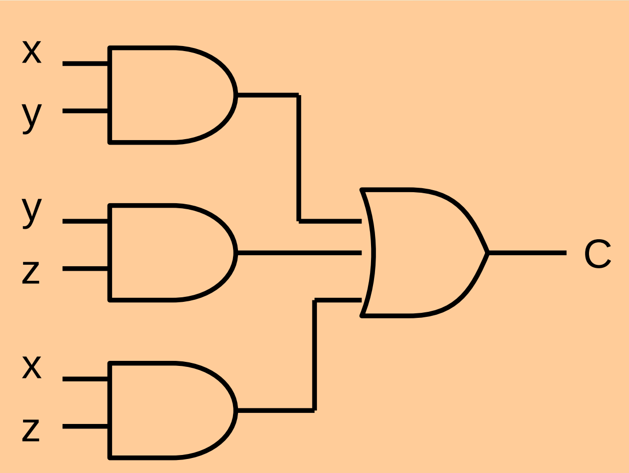
Implementing Boolean expressions using gates



$$S = \bar{x}.\bar{y}.z + \bar{x}.y.\bar{z} + x.\bar{y}.\bar{z} + x.y.z$$

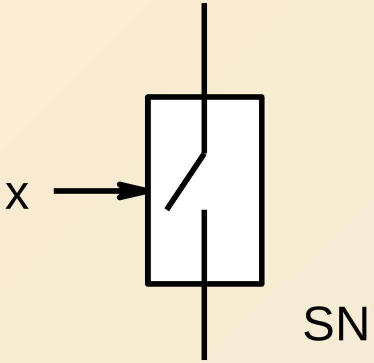
$$C = x.y + x.z + y.z$$





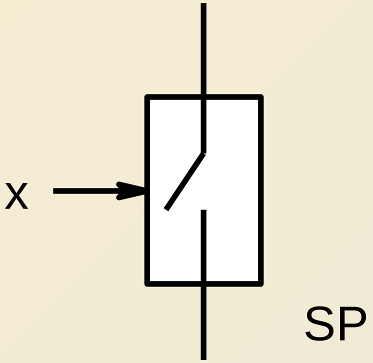
Implementing gates using Switches

Voltage controlled Switch
SN:



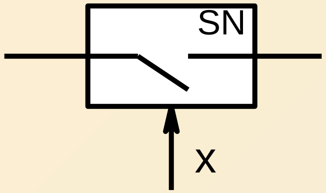
Switch is closed if voltage x is HIGH
Switch is open if voltage x is LOW

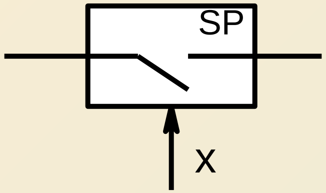
Voltage controlled Switch
SP:



Switch is closed if voltage x is LOW
Switch is open if voltage x is HIGH

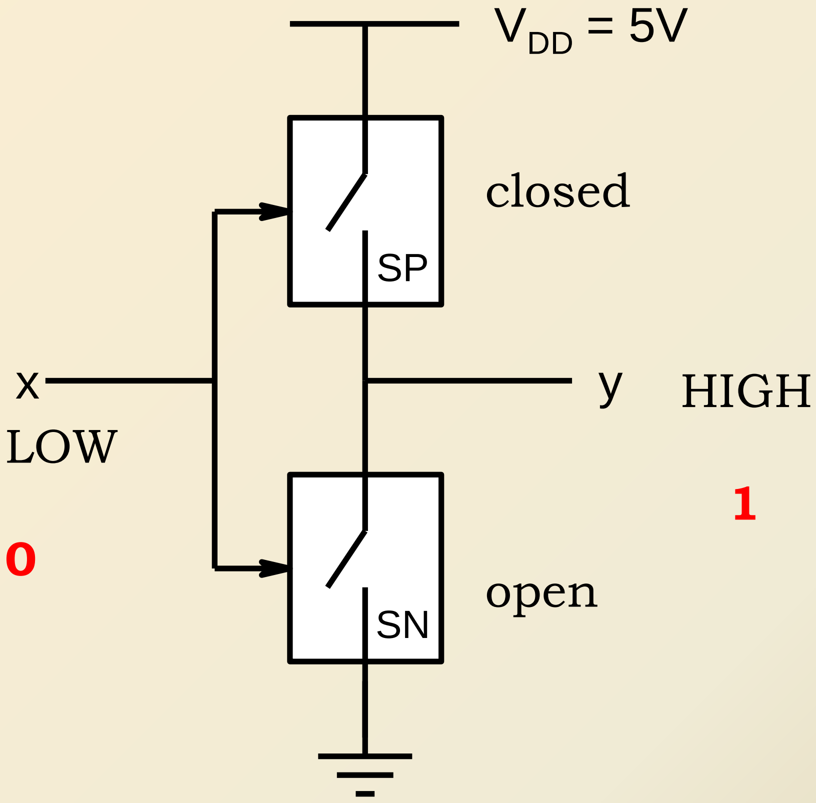
We have seen earlier (in class 12) that transistors act as switches !

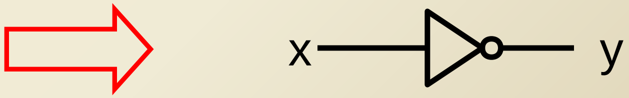




Switch is closed if voltage x is HIGH
Switch is open if voltage x is LOW

Switch is closed if voltage x is LOW
Switch is open if voltage x is HIGH

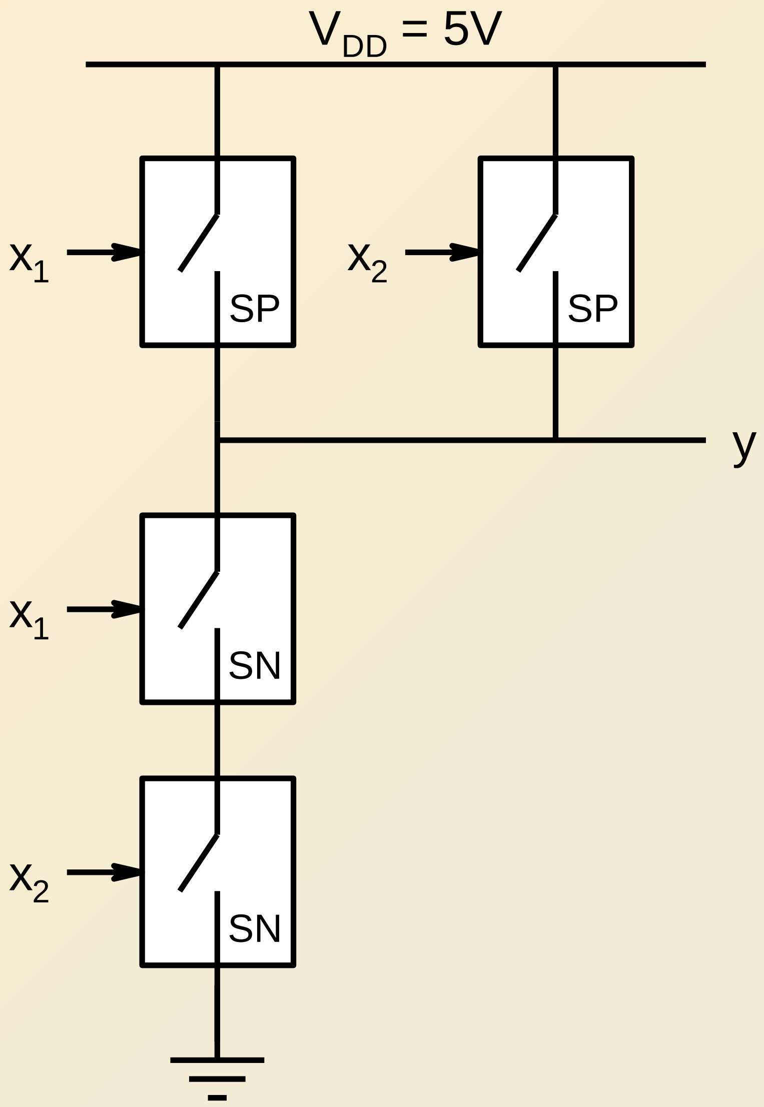




NOT gate

NAND Gate

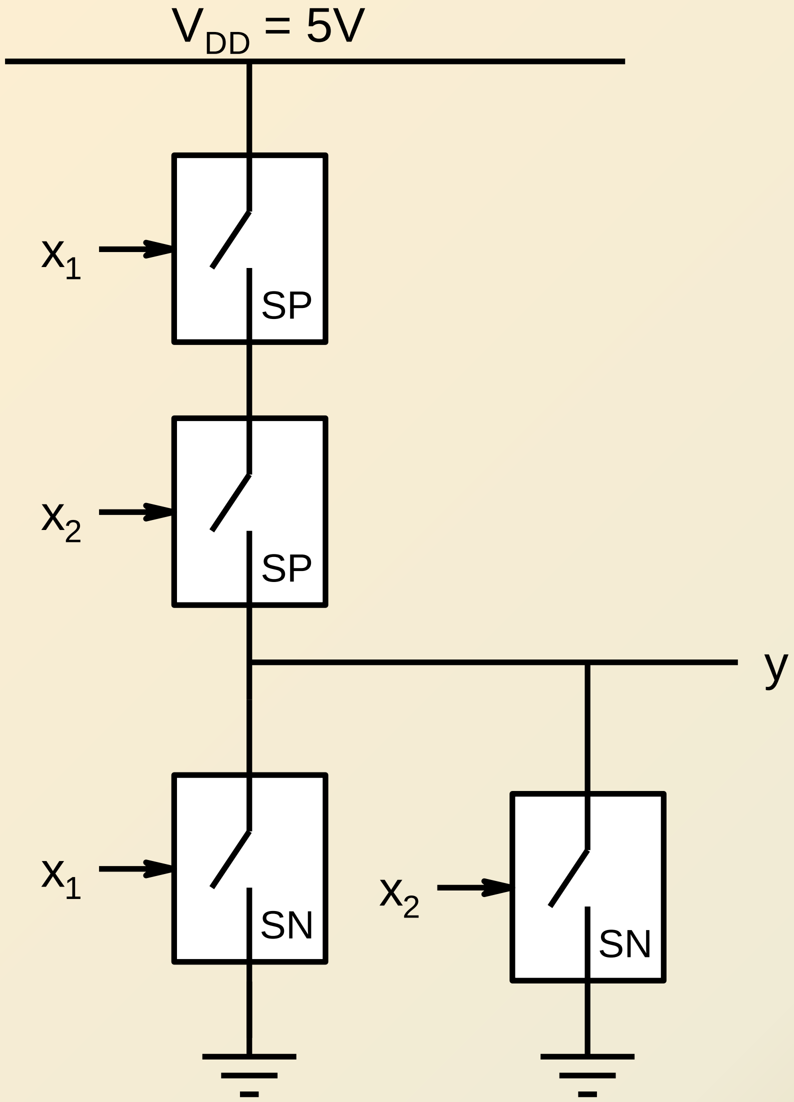
NAND: $y = \overline{x_1 . x_2}$



x_1	x_2	y
LOW	LOW	HIGH
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	LOW

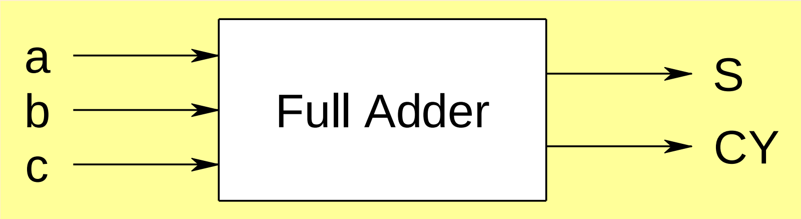
NOR Gate

$$\text{NOR:} \quad y = \overline{x_1 + x_2}$$



x_1	x_2	y
LOW	LOW	HIGH
LOW	HIGH	LOW
HIGH	LOW	LOW
HIGH	HIGH	LOW

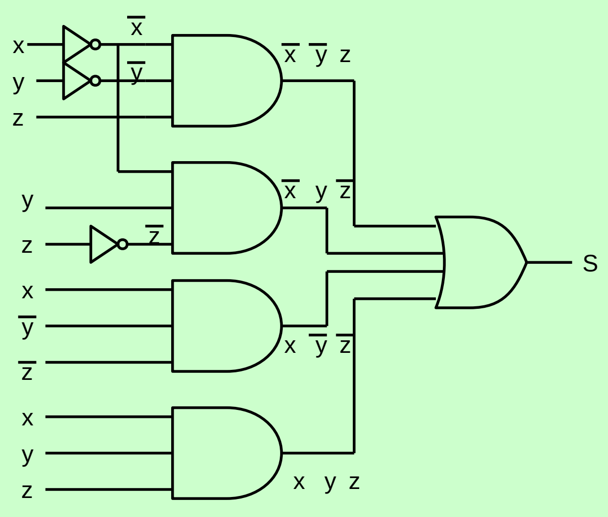
Design Overview

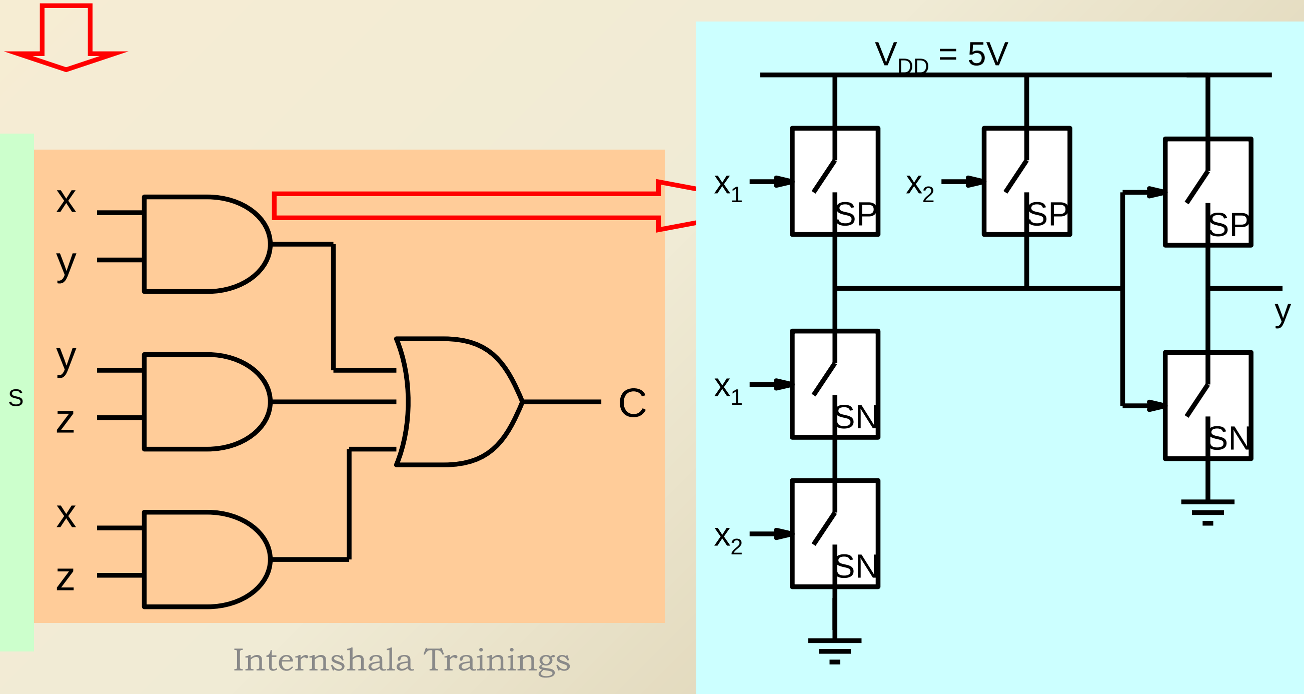


a	b	c	S	CY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \bar{x}.\bar{y}.z + \bar{x}.y.\bar{z} + x.\bar{y}.\bar{z} + x.y.z$$

$$C = x.y + x.z + y.z$$





SOP AND POS REPRESENTATIONS

Representation of Boolean Expressions

x	y	f_1
0	0	0
0	1	1
1	0	1
1	1	0

x	y	min term	
0	0	$\bar{x}\,.\,\bar{y}$	m0
0	1	$\bar{x}\,.\,y$	m1
1	0	$x\,.\,\bar{y}$	m2
1	1	$x\,.\,y$	m3

$$f_1 = \bar{x}.\ y + x.\bar{y}$$

$$f_1 = m_1 + m_2$$

$$f_1 = \sum(1,2)$$

$$f_2 = \sum(0,2,3) = ?$$

$$f_2 = \bar{x}.\ \bar{y} + x.\bar{y} + x.y$$

A minterm is a product that contains all the variables used in a function

Three variable functions

x	y	z	min terms	
0	0	0	$\overline{x} . \overline{y} . \overline{z}$	m0
0	0	1	$\overline{x} . \overline{y} . z$	m1
0	1	0	$\overline{x} . y . \overline{z}$	m2
0	1	1	$\overline{x} . y . z$	m3
1	0	0	$x . \overline{y} . \overline{z}$	m4
1	0	1	$x . \overline{y} . z$	m5
1	1	0	$x . y . \overline{z}$	m6
1	1	1	$x . y . z$	m7

$$f_2 = \sum(1,4,7) = ?$$

$$f_2 = \overline{x}.\ \overline{y}.z + x.\overline{y}.\overline{z} + x.y.z$$

Product of Sum Terms Representation

x	y	f_1
0	0	0
0	1	1
1	0	1
1	1	0

x	y	Max term	
0	0	$x + y$	M0
0	1	$x + \overline{y}$	M1
1	0	$\overline{x} + y$	M2
1	1	$\overline{x} + \overline{y}$	M3

F1 = (x+y)(x’ + y’) $= M_0.M_3$ $= \prod M_0M_3$

x	y	z	Max. terms	
0	0	0	$x + y + z$	M0
0	0	1	$x + y + \overline{z}$	M1
0	1	0	$x + \overline{y} + z$	M2
0	1	1	$x + \overline{y} + \overline{z}$	M3
1	0	0	$\overline{x} + y + z$	M4
1	0	1	$\overline{x} + y + \overline{z}$	M5
1	1	0	$\overline{x} + \overline{y} + z$	M6
1	1	1	$\overline{x} + \overline{y} + \overline{z}$	M7

$$f_1 = \Pi(1,5,7) = ?$$

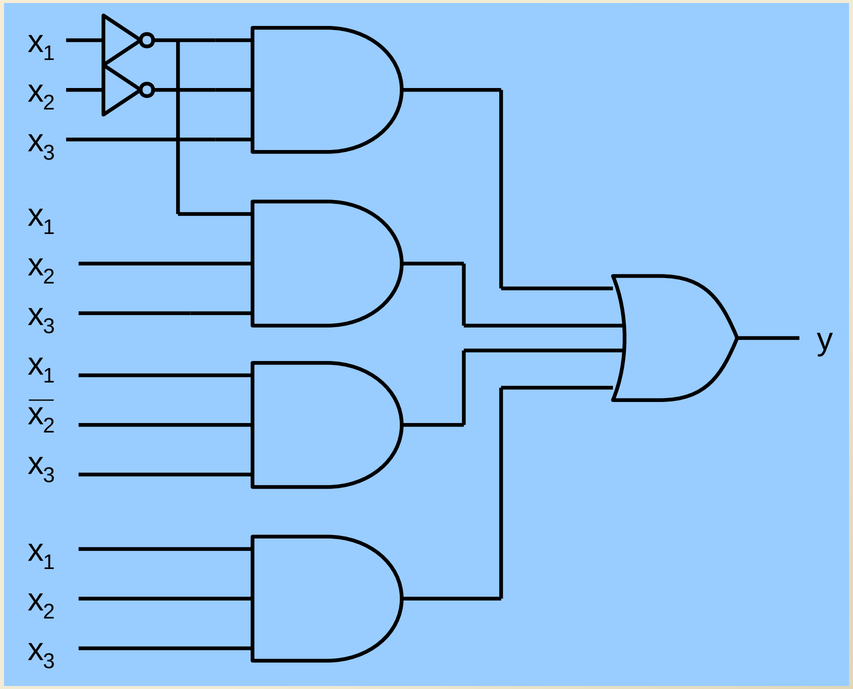
$$f_2 = (x + y + \overline{z}).(\overline{x} + y + \overline{z}).(\overline{x} + \overline{y} + \overline{z})$$

Simplification of Boolean Expressions

x_1	x_2	x_3	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$y = \sum(1,3,5,7)$$

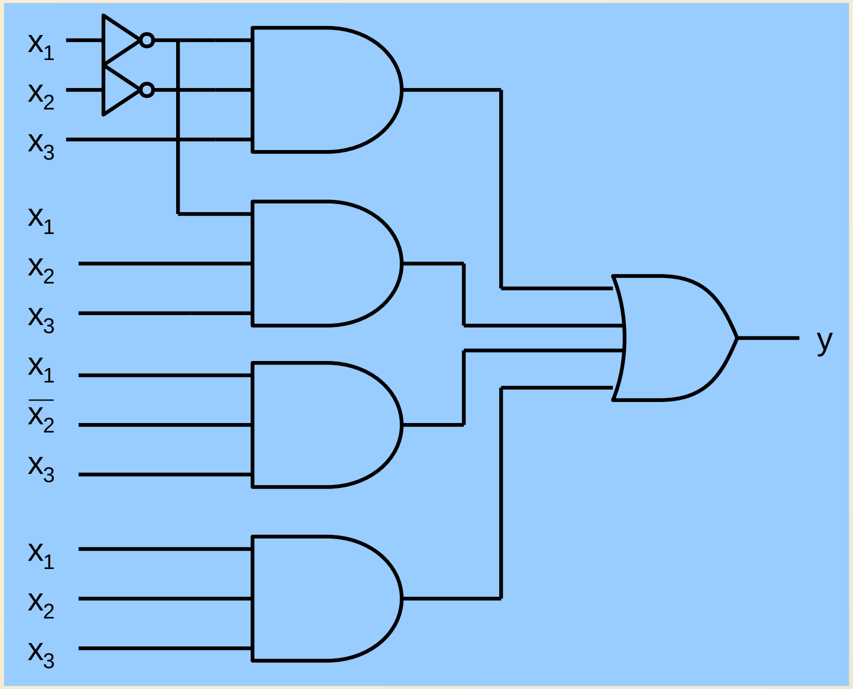
$$y = \overline{x_1}.\,\overline{x_2}.x_3 + \overline{x_1}.x_2.x_3 + x_1.\overline{x_2}.x_3 + x_1.x_2.x_3$$



Simplification of Boolean expression yields : $y = x_3$!! which does not require any gates at all !

Goal of Simplification

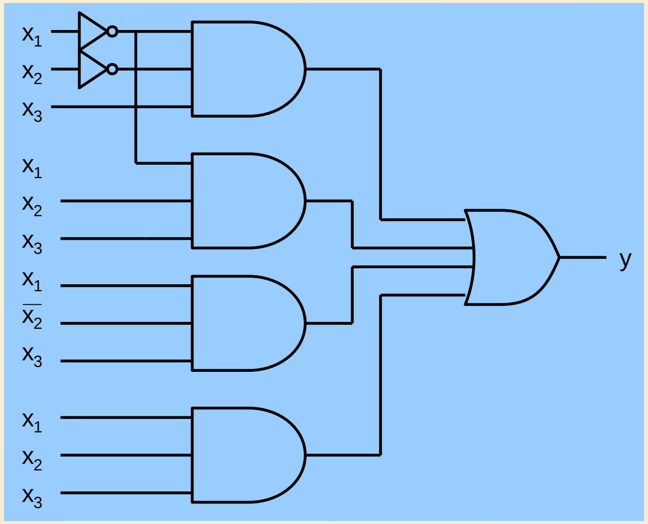
$$y = \overline{x}_1 . \overline{x}_2 . x_3 + \overline{x}_1 . x_2 . x_3 + x_1 . \overline{x}_2 . x_3 + x_1 . x_2 . x_3$$

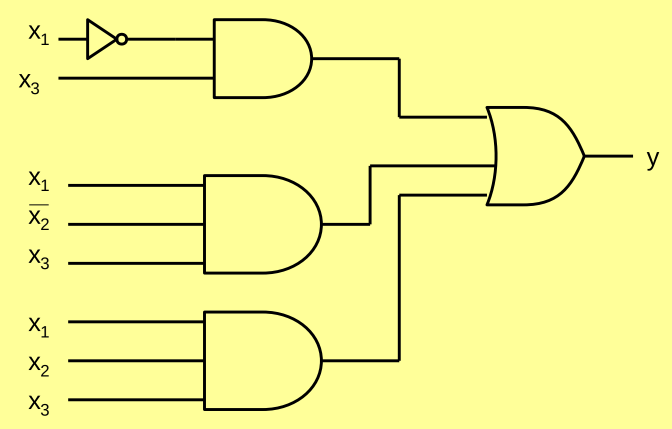


Goal of simplification is to reduce the complexity of gate circuit. This requires that we minimize the number of gates. Since number of gates depends on number of minterms, one of the goals of simplification is to **minimize the number of minterms in SOP expression**

$$y = \overline{x}_1 . \overline{x}_2 . x_3 + \overline{x}_1 . x_2 . x_3 + x_1 . \overline{x}_2 . x_3 + x_1 . x_2 . x_3$$

$$\Rightarrow \quad y = \overline{x}_1 . x_3 + x_1 . \overline{x}_2 . x_3 + x_1 . x_2 . x_3$$





This circuit is simpler not just because it uses 4 gates instead of 5 but also because circuit-2 uses one 2-input and three 3-input gates as compared to five 3-input gates used in circuit-1

Goal of Simplification

In the SOP expression:

1. Minimize number of product terms

2. Minimize number of literals in each term

Simplification $\Rightarrow$ Minimization

Minimization

$$y=\overline{x_1}.\overline{x_2}.x_3+\overline{x_1}.x_2.x_3+x_1.\overline{x_2}.x_3+x_1.x_2.x_3$$

$$y=\overline{x_1}.x_3.(\overline{x_2}+x_2)+x_1.x_3.(\overline{x_2}+x_2)$$

$$y=\overline{x_1}.x_3+x_1.x_3$$

$$y=(\overline{x_1}+x_1).x_3$$

$$y=x_3$$

Principle used: $x+\overline{x}=1$

$$f = \overline{x}.\overline{y} + \overline{x}.y + x.\overline{y}$$

Apply the Principle: $x + \overline{x} = 1$ to simplify

$$f = \overline{x}.(\overline{y} + y) + x.\overline{y}$$

$$f = \overline{x} + x.\overline{y}$$

How do we simplify further?

$$f = \overline{x}.\overline{y} + \overline{x}.y + x.\overline{y} = \overline{x}.\overline{y} + \overline{x}.\overline{y} + \overline{x}.y + x.\overline{y}$$

Principle used : $x + x = x$

$$f = \overline{x}.\overline{y} + \overline{x}.y + \overline{x}.\overline{y} + x.\overline{y}$$
$$= \overline{x}.(\overline{y} + y) + (\overline{x} + x).\overline{y} = \overline{x} + \overline{y}$$

Simplify

$$f = \overline{x_1}.x_2.\overline{x_3}.x_4 + \overline{x_1}.x_2.x_3.x_4 + x_1.x_2.\overline{x_3}.x_4 + x_1.x_2.x_3.x_4 +$$
$$x_1.\overline{x_2}.\overline{x_3}.x_4 + x_1.\overline{x_2}.x_3.x_4$$

Principle: $x + \overline{x} = 1$ and $x + x = x$

Need a systematic and simpler method for applying these two principles

Karnaugh Map (K map) is a popular technique for carrying out simplification

It represents the information in problem in such a way that the two principles become easy to apply

KARNAUGH MAPS

K-map representation of truth table

x	y	min term	
0	0	$\overline{x} . \overline{y}$	m0
0	1	$\overline{x} . y$	m1
1	0	$x . \overline{y}$	m2
1	1	$x . y$	m3

$x \backslash y$	0	1
0	m_0	m_1
1	m_2	m_3

x	y	f_1
0	0	0
0	1	1
1	0	1
1	1	0

⇒

$x \backslash y$	0	1
0	0	1
1	1	0

$f_2 = \sum(0,2,3)$

x \ y	0	1
0	1	0
1	1	1

x \ y	0	1
0	1	0
1	0	1

$f = \overline{x}.\overline{y} + x.y$

3-variable K-map representation

x	y	z	min terms	
0	0	0	$\overline{x} . \overline{y} . \overline{z}$	m0
0	0	1	$\overline{x} . \overline{y} . z$	m1
0	1	0	$\overline{x} . y . \overline{z}$	m2
0	1	1	$\overline{x} . y . z$	m3
1	0	0	$x . \overline{y} . \overline{z}$	m4
1	0	1	$x . \overline{y} . z$	m5
1	1	0	$x . y . \overline{z}$	m6
1	1	1	$x . y . z$	m7

x \ yz	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

x \ yz	00	01	11	10
0	0	1	1	0
1	0	1	1	0

x \ yz	00	01	11	10
0	1	0	1	0
1	0	1	1	0

$$f = \overline{x}.\overline{y}.\overline{z} + \overline{x}.y.z + x.\overline{y}.z + x.y.z$$

4-variable K-map representation

w	x	y	z	min terms
0	0	0	0	m_0
0	0	0	1	m_1
0	0	1	0	m_2
0	0	1	1	m_3
⋮	⋮	⋮	⋮	⋮
1	1	1	0	m_{14}
1	1	1	1	m_{15}

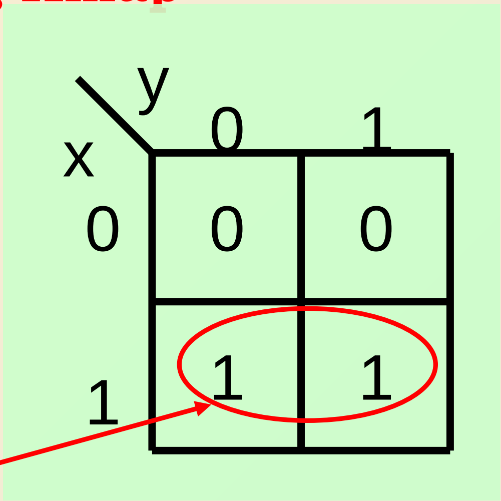
wx \ yz	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

wx \ yz	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	0

$$f = \overline{w}.\overline{x}.\overline{y}.\overline{z} + \overline{w}.\overline{x}.y.z + \overline{w}.x.\overline{y}.z + \overline{w}.x.y.z$$
$$+ w.x.\overline{y}.\overline{z} + w.x.y.\overline{z} + w.\overline{x}.\overline{y}.\overline{z}$$

Minimization using Kmap

$$f_2 = \sum(2,3)$$



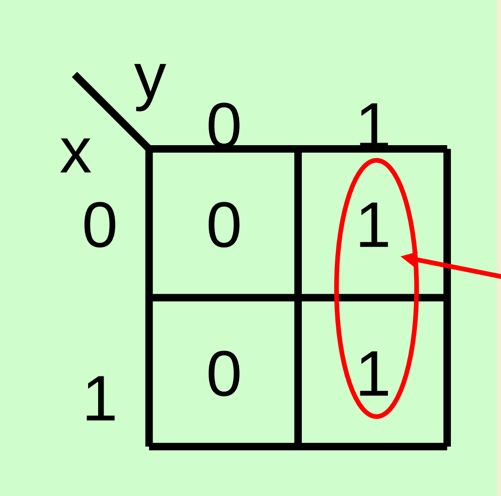
x \ y	0	1
0	0	0
1	1	1

$$f = x.\overline{y} + x.y$$

$$f = x.(\overline{y} + y)$$

$$f = x$$

Combine terms which differ in only one bit position. As a result, whatever is common remains.

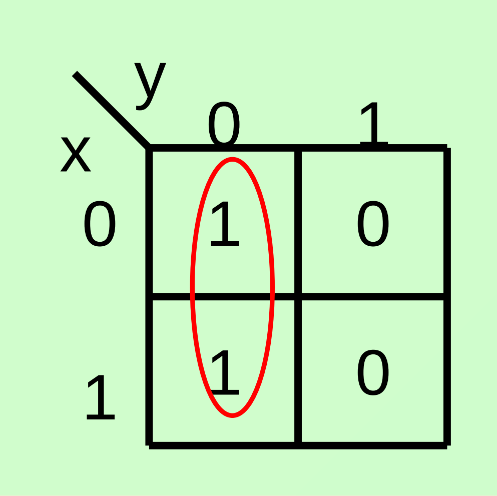


x \ y	0	1
0	0	1
1	0	1

$$f = \overline{x}.y + x.y$$

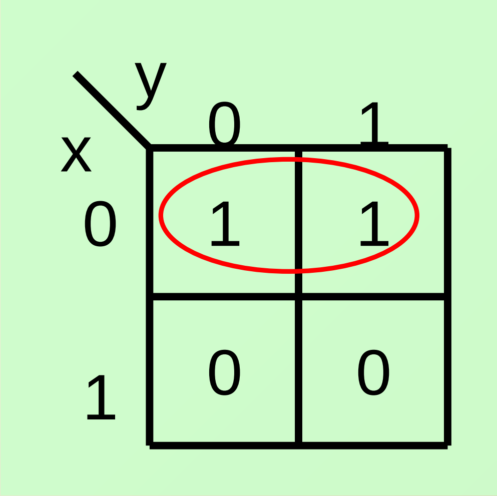
$$f = (\overline{x}. + x).y$$

$$\Rightarrow f = y$$



x \ y	0	1
0	1	0
1	1	0

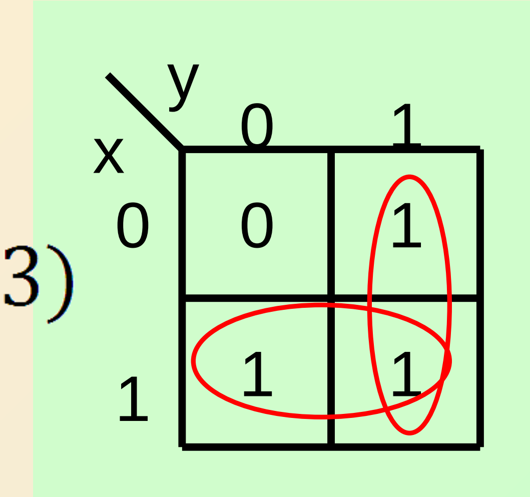
$$\Rightarrow f = \overline{y}$$



x \ y	0	1
0	1	1
1	0	0

$$\Rightarrow f = \overline{x}$$

$$F2 = \sum(1, 2, 3)$$



x \ y	0	1
0	0	1
1	1	1

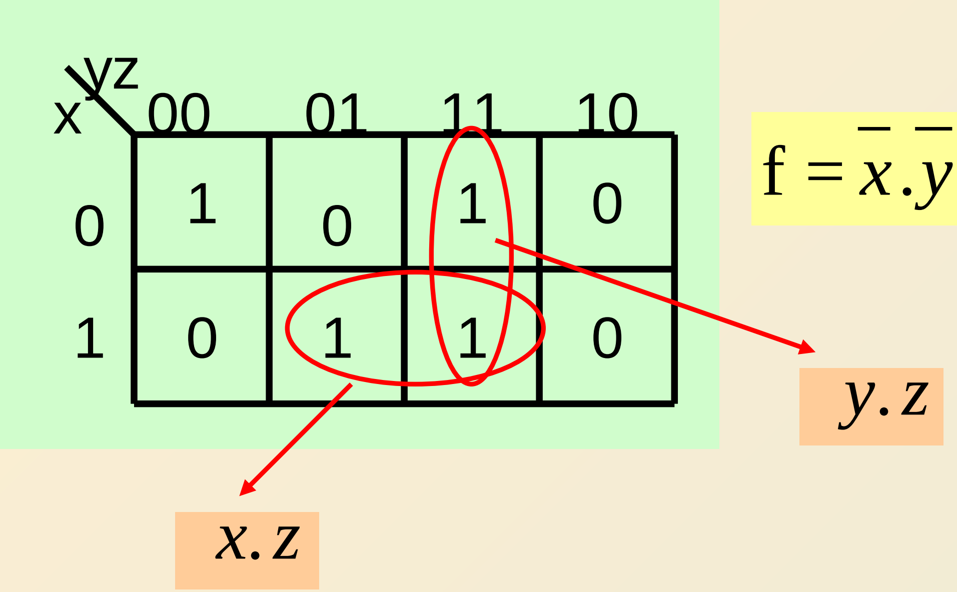
$$f = x.\bar{y} + x.y + \bar{x}.y$$

$$\begin{aligned} f &= x.(\bar{y} + y) + \bar{x}.y \\ &= x + \bar{x}.y \end{aligned}$$

$$\begin{aligned} f &= x + \bar{x}.y + x.y \\ &= x + (\bar{x} + x).y \\ &= x + y \end{aligned}$$

The idea is to cover all the 1's with as few and as simple terms as possible

3-variable minimization



x \ yz	00	01	11	10
0	1	0	1	0
1	0	1	1	0

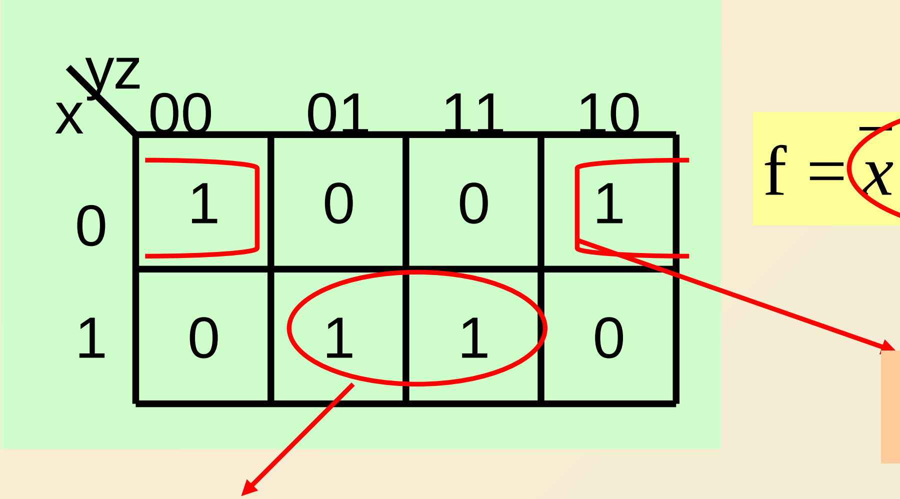
$y.z$

$x.z$

$$f = \overline{x}.\overline{y}.\overline{z} + \overline{x}.y.z + x.y.z + x.\overline{y}.z$$

$$f = \overline{x}.\overline{y}.\overline{z} + y.z + x.z$$

3-variable minimization



x \ yz	00	01	11	10
0	1	0	0	1
1	0	1	1	0

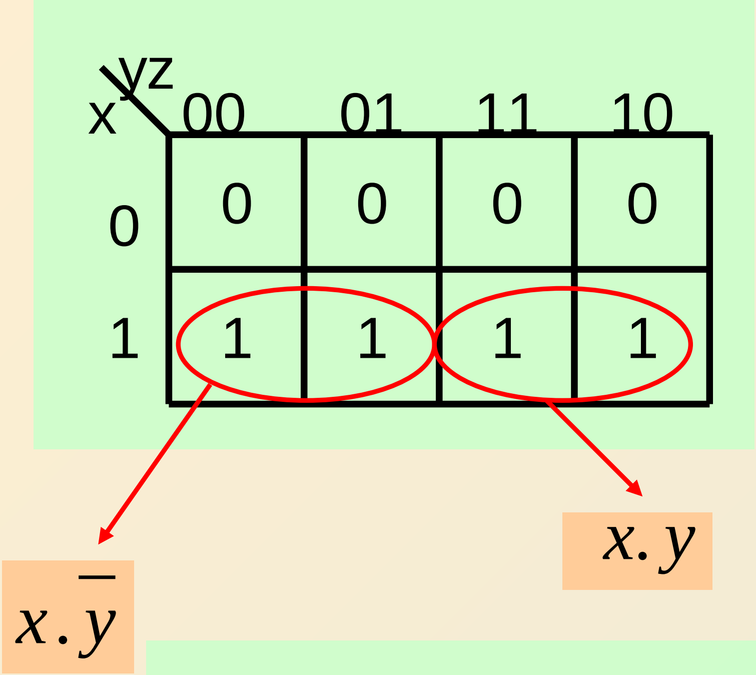
$$f = \overline{x}.\overline{y}.\overline{z} + \overline{x}.y.\overline{z} + x.y.z + x.\overline{y}.z$$

$\overline{x}.\overline{z}$

$x.z$

$$f = \overline{x}\,.\overline{z} + x.z$$

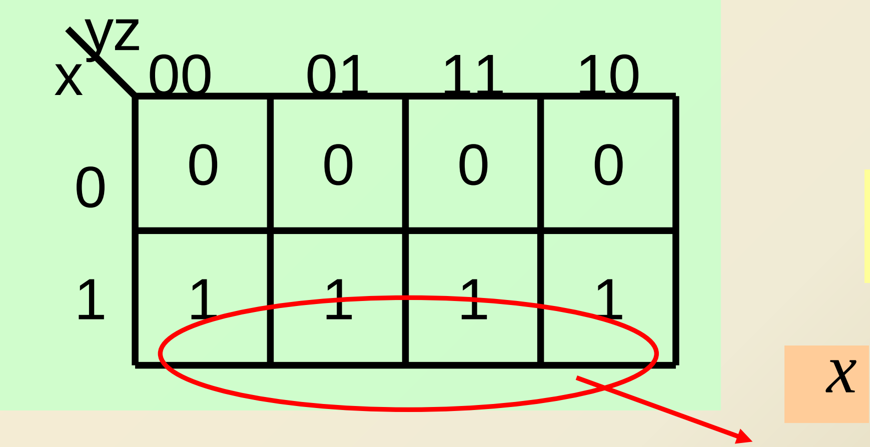
3-variable minimization



x \ yz	00	01	11	10
0	0	0	0	0
1	1	1	1	1

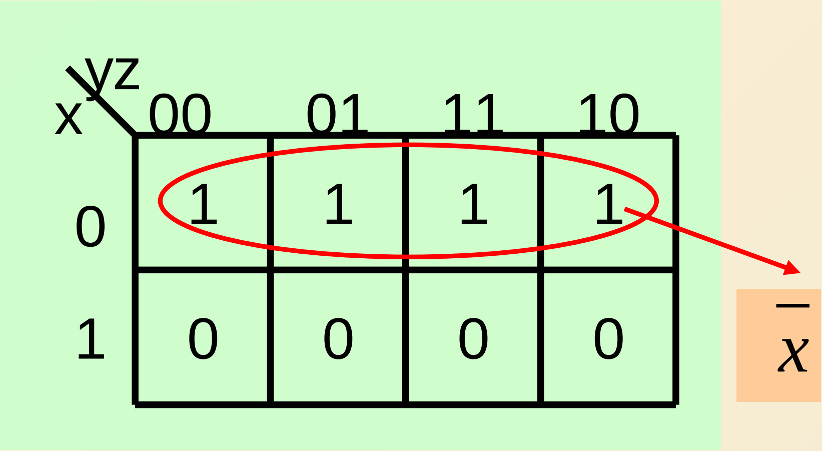
$$f = x.\overline{y}.\overline{z} + x.\overline{y}.z + x.y.z + x.y.\overline{z}$$

$$f = x.\overline{y} + x.y$$

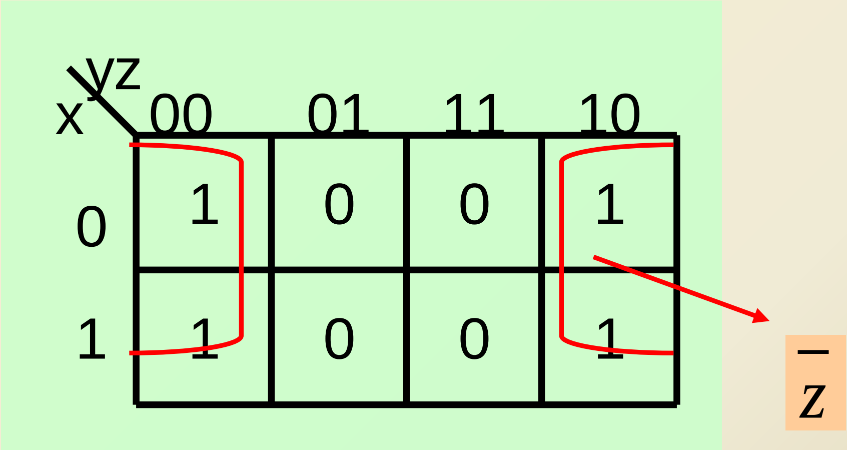


x \ yz	00	01	11	10
0	0	0	0	0
1	1	1	1	1

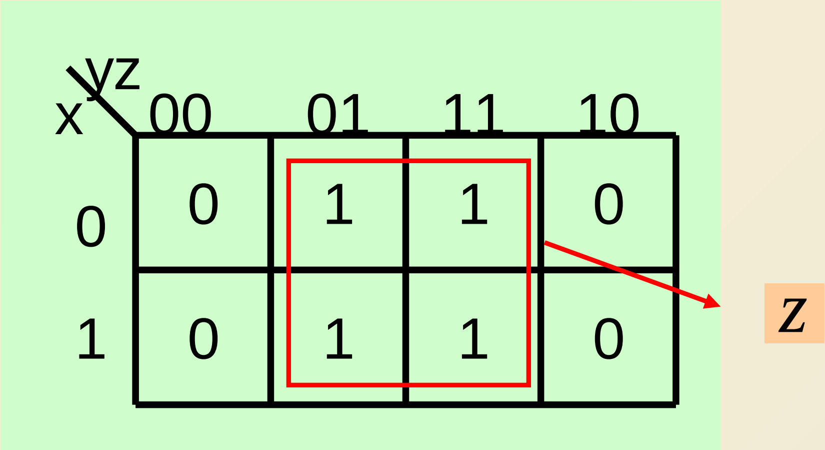
$$f = x.(\overline{y} + y) = x$$



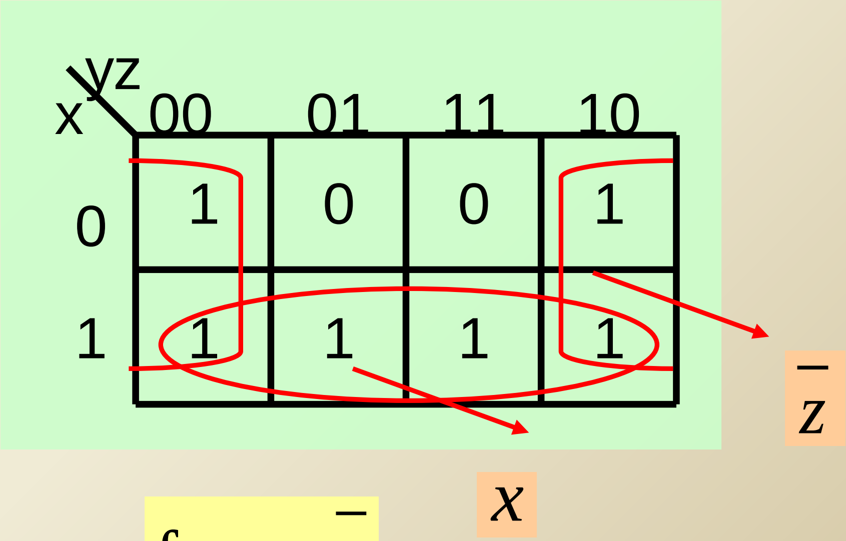
x \ yz	00	01	11	10
0	1	1	1	1
1	0	0	0	0



x \ yz	00	01	11	10
0	1	0	0	1
1	1	0	0	1



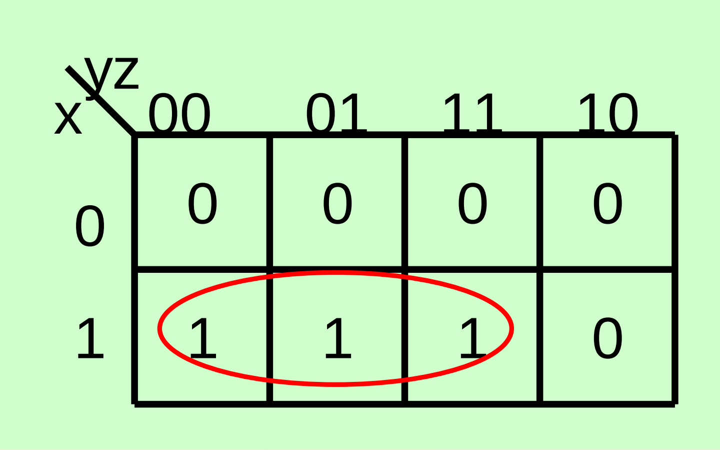
x \ yz	00	01	11	10
0	0	1	1	0
1	0	1	1	0



x \ yz	00	01	11	10
0	1	0	0	1
1	1	1	1	1

$$f = x + \bar{z}$$

Can we do this ?



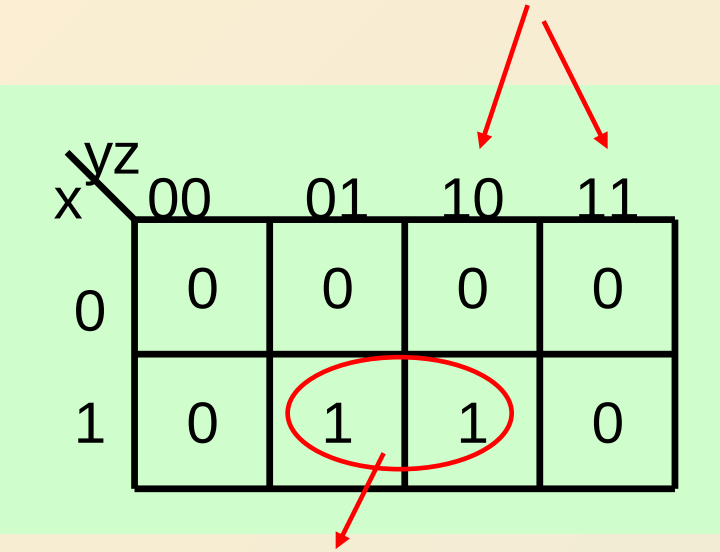
x \ yz	00	01	11	10
0	0	0	0	0
1	1	1	1	0

Note that each encirclement should represent a single product term. In this case it does not.

$$f = x.\overline{y}.\overline{z} + x.\overline{y}.z + x.y.z$$
$$= x.\overline{y} + x.z$$

We do not get a single product term.
In general we cannot make groups of 3 terms.

Can we use kmap with the following ordering of variables?



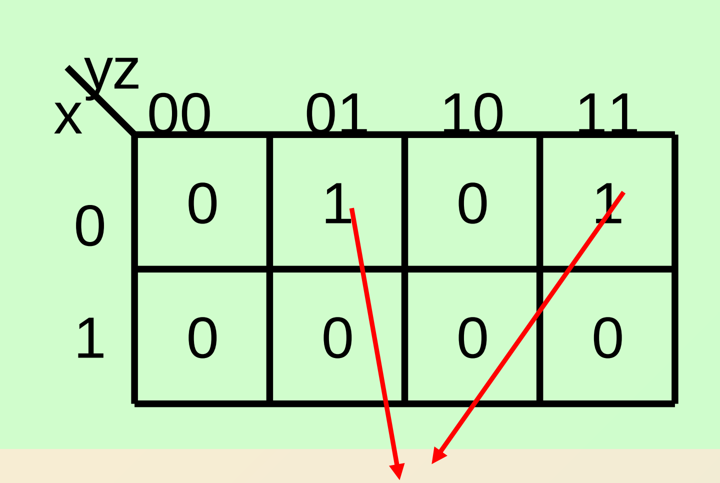
x \ yz	00	01	10	11
0	0	0	0	0
1	0	1	1	0

Can we combine these two terms into a single term ?

$$f = x.\overline{y}.z + x.y.\overline{z}$$

$$= x.(\overline{y}.z + y.\overline{z})$$

Note that no simplification is possible.
Kmap requires information to be represented



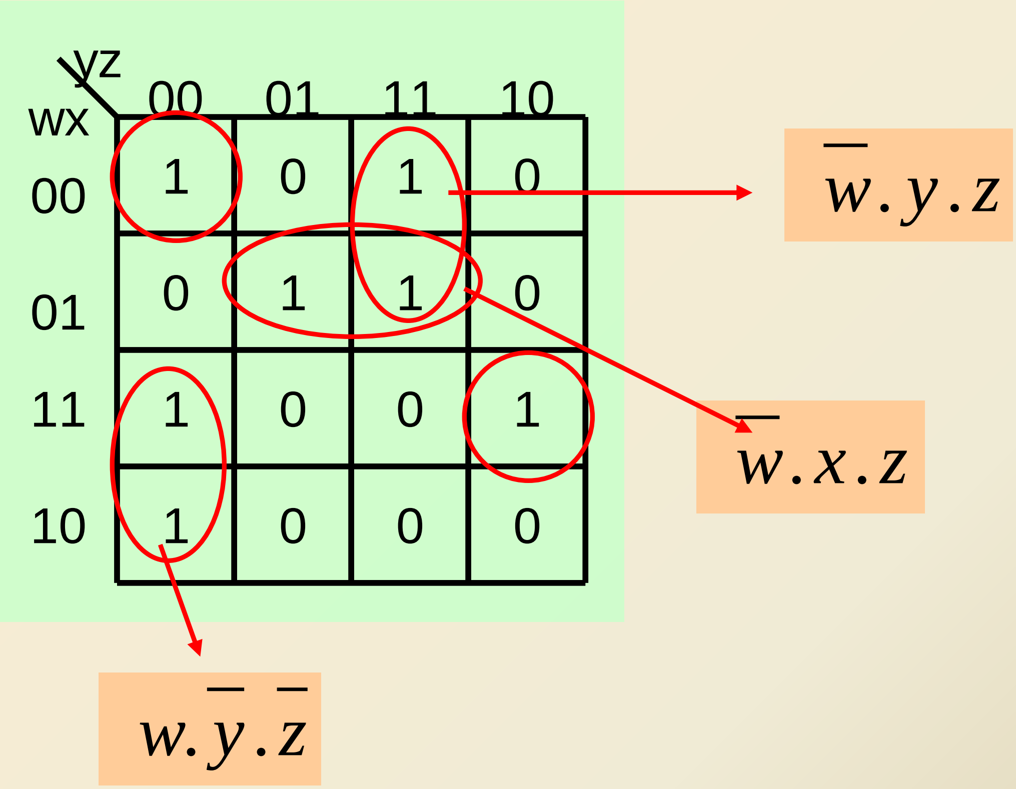
x \ yz	00	01	10	11
0	0	1	0	1
1	0	0	0	0

These two terms can be combined into a single term but it is not easy to show that on the diagram.

$$\begin{aligned} f &= \bar{x}.\bar{y}.z + \bar{x}.y.z \\ &= \bar{x}.(\bar{y} + y).z = \bar{x}.z \end{aligned}$$

Kmap requires information to be represented in such a way that it is easy to apply the principle $x + x = 1$

4-variable minimization



wx \ yz	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	0

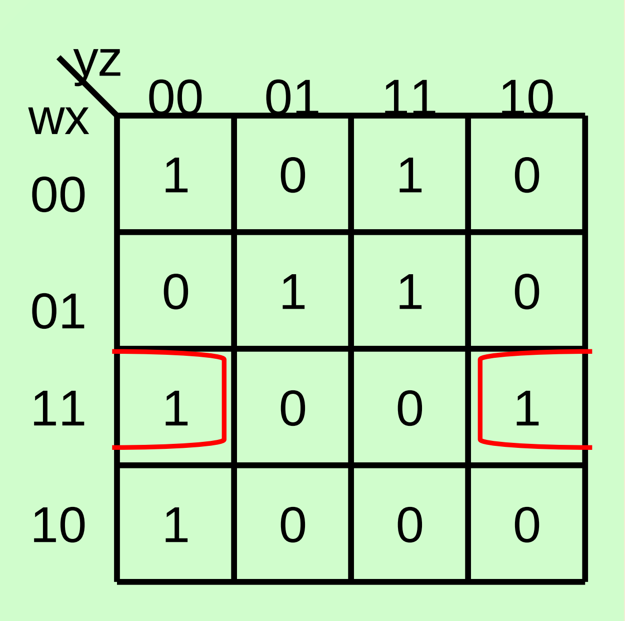
$\overline{w}.y.z$

$\overline{w}.x.z$

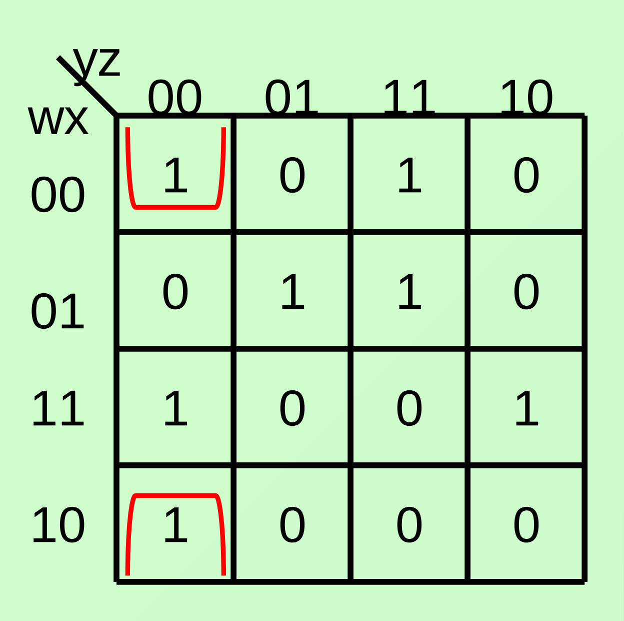
$w.\overline{y}.\overline{z}$

$$f = \overline{w}.y.z + \overline{w}.x.z + w.\overline{y}.\overline{z} + \overline{w}.\overline{x}.\overline{y}.\overline{z} + w.x.y.\overline{z}$$

But is this the simplest expression ?



wx \ yz	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	0

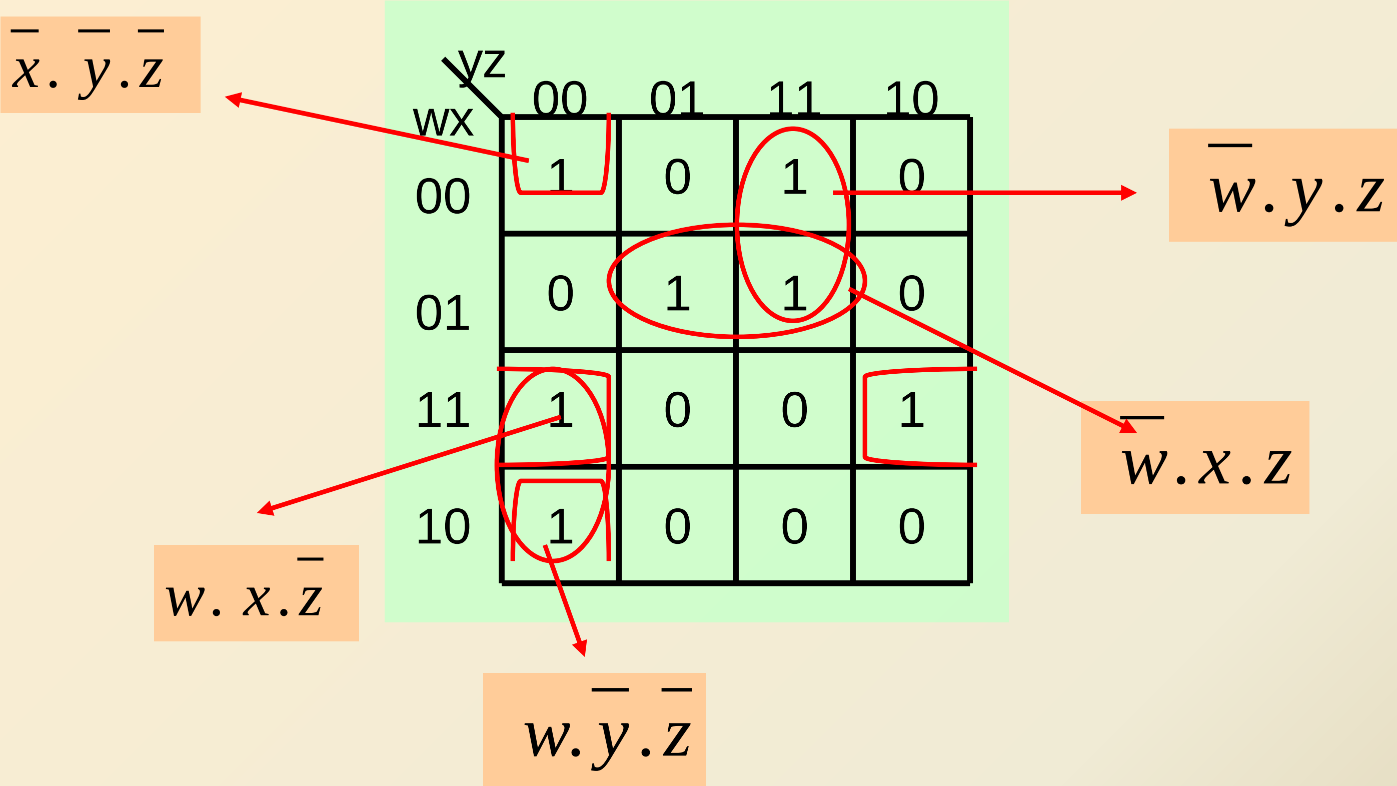


wx \ yz	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	0

$$w.\,x.\bar{y}.\bar{z}+w.\,x.y.\bar{z}=w.\,x.\bar{z}$$

$$w.\,\bar{x}.\bar{y}.\bar{z}+\bar{w}.\,\bar{x}.\bar{y}.\bar{z}=\bar{x}.\,\bar{y}.\bar{z}$$

4-variable minimization



wx \ yz	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	0

$\bar{x}.\bar{y}.\bar{z}$

$\bar{w}.y.z$

$\bar{w}.x.z$

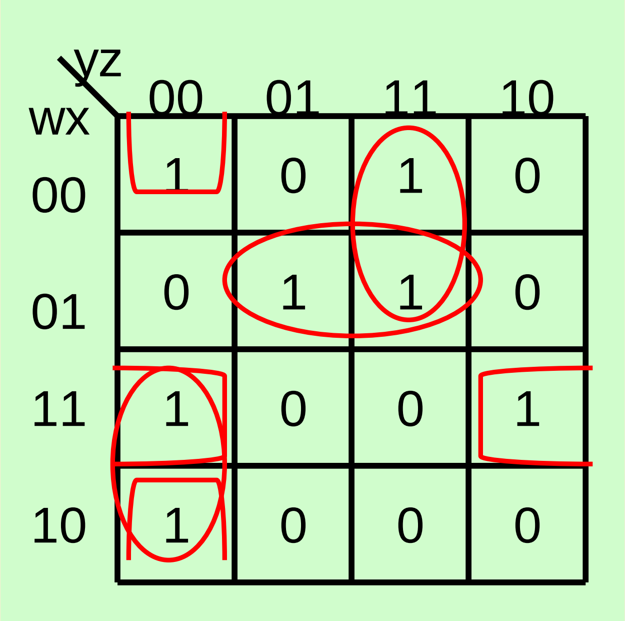
$w.x.\bar{z}$

$w.\bar{y}.\bar{z}$

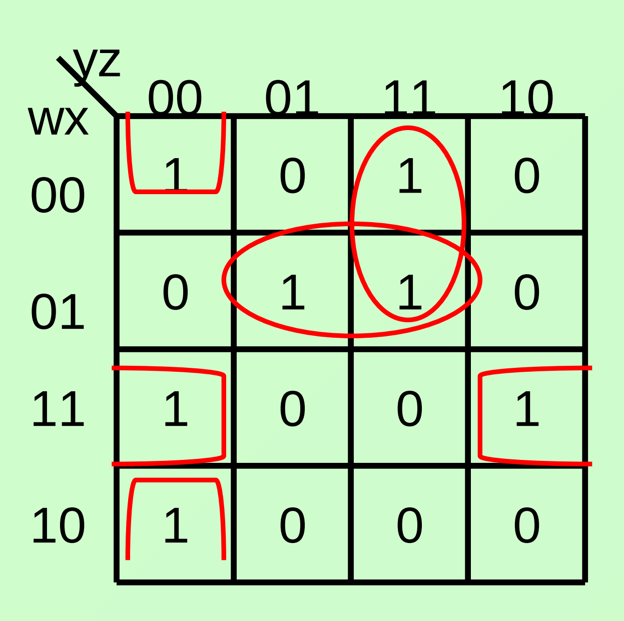
$$f = \bar{w}.y.z + \bar{w}.x.z + w.\bar{y}.\bar{z} + w.x.\bar{z} + \bar{x}.\bar{y}.\bar{z}$$

Is this the best that we can do ?

Cover the 1's with minimum number of terms



wx \ yz	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	0

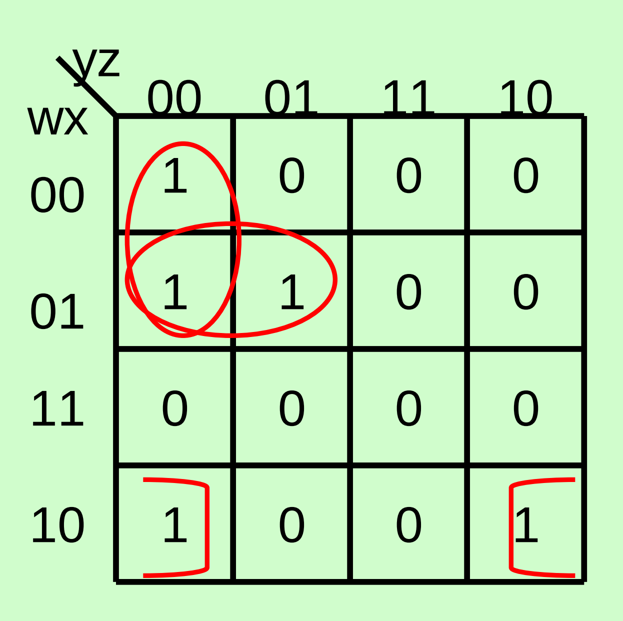


wx \ yz	00	01	11	10
00	1	0	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	0

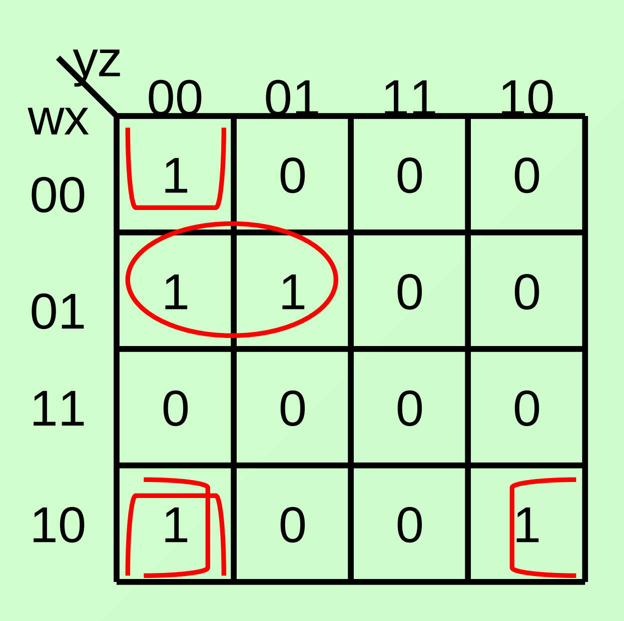
$$f = \overline{w}.y.z + \overline{w}.x.z + w.\overline{y}.\overline{z} + w.x.\overline{z} + \overline{x}.\overline{y}.\overline{z}$$

$$f = \overline{w}.y.z + \overline{w}.x.z + w.x.\overline{z} + \overline{x}.\overline{y}.\overline{z}$$

4-variable minimization



wx \ yz	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	0	0	0	0
10	1	0	0	1

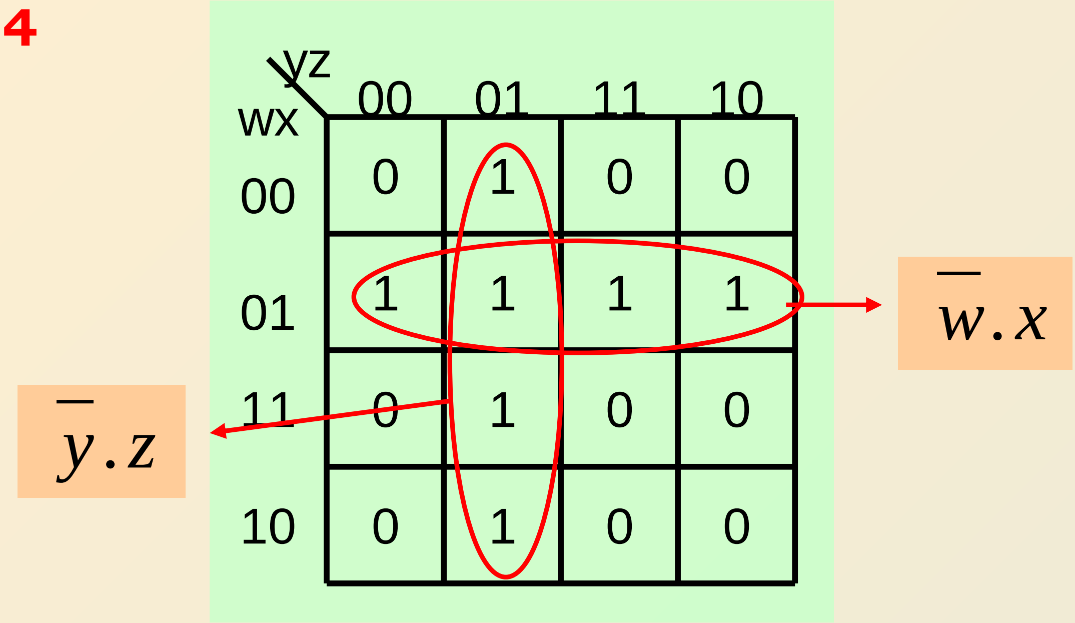


wx \ yz	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	0	0	0	0
10	1	0	0	1

$$f = \overline{w}.x.\overline{y} + w.\overline{x}.\overline{z} + \overline{w}.\overline{y}.\overline{z}$$

$$f = \overline{w}.x.\overline{y} + w.\overline{x}.\overline{z} + \overline{x}.\overline{y}.\overline{z}$$

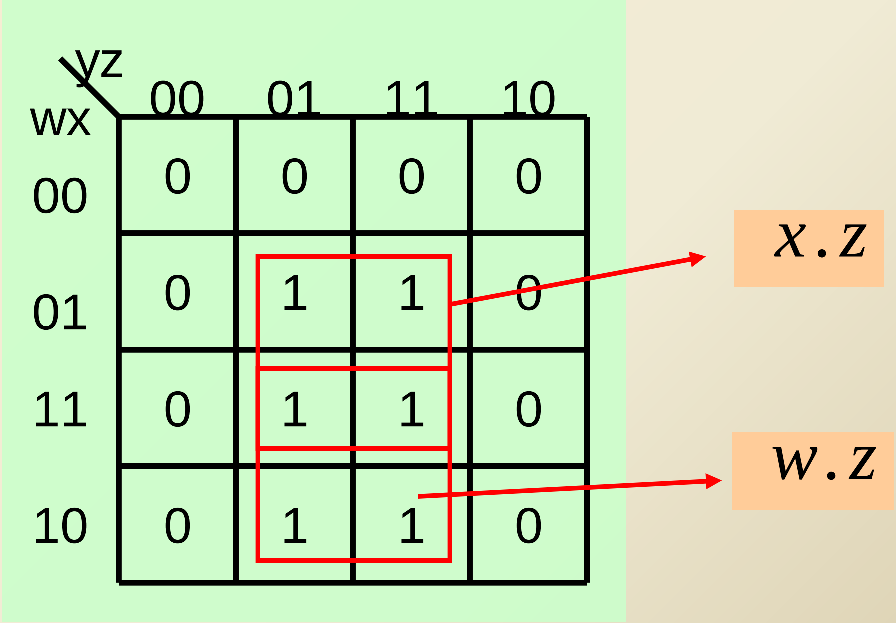
Groups of 4



wx \ yz	00	01	11	10
00	0	1	0	0
01	1	1	1	1
11	0	1	0	0
10	0	1	0	0

$\overline{w}.x$

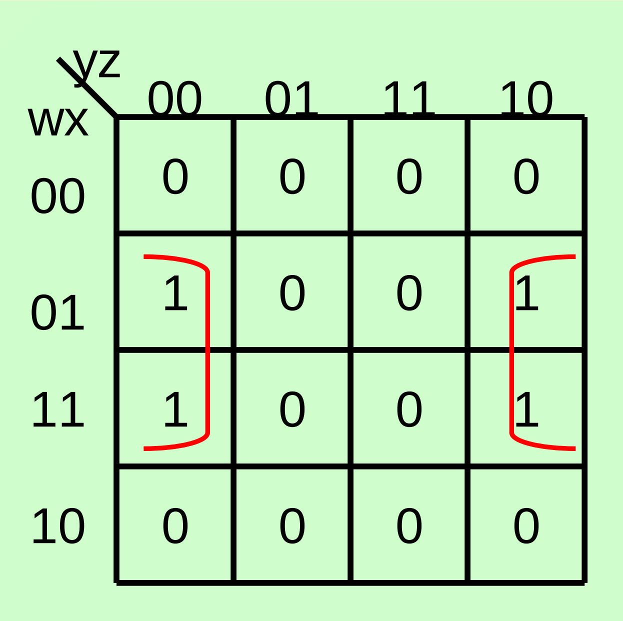
$\overline{y}.z$



wx \ yz	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

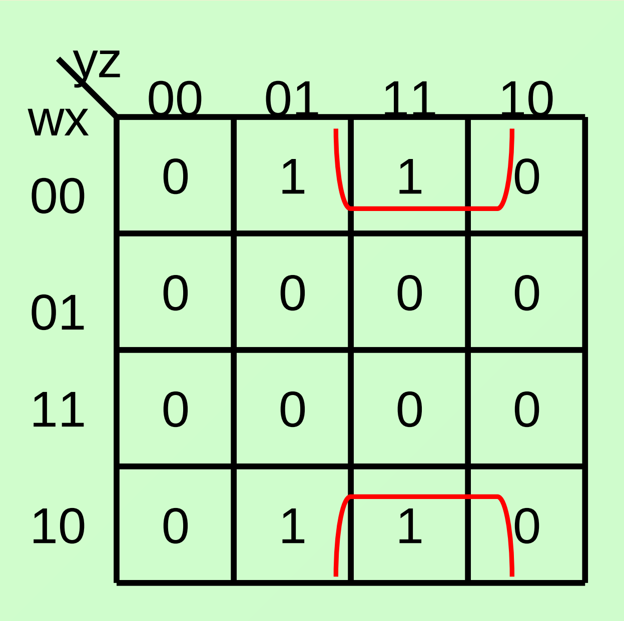
$x.z$

$w.z$



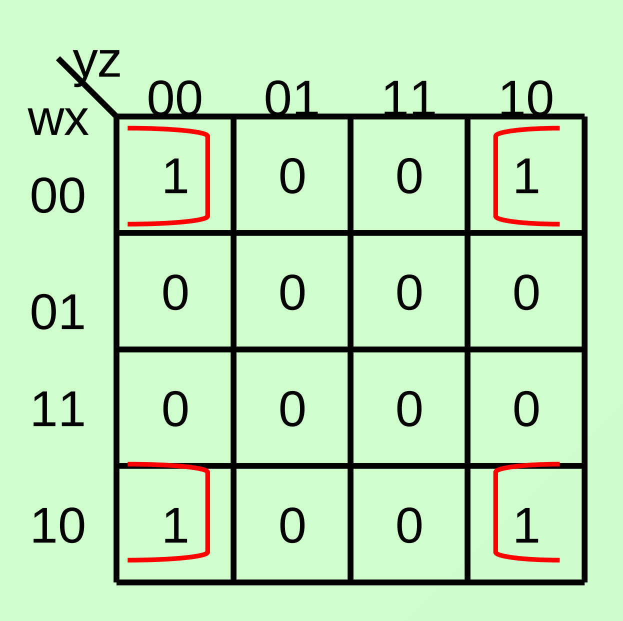
wx \ yz	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

$x.\bar{z}$



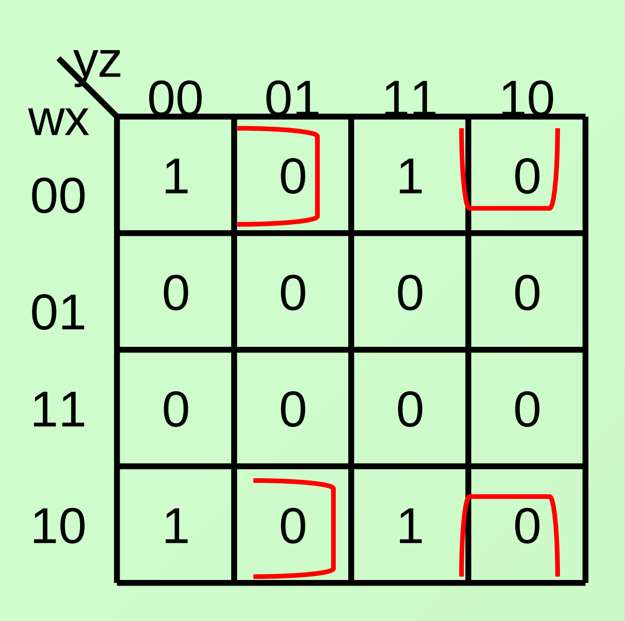
wx \ yz	00	01	11	10
00	0	1	1	0
01	0	0	0	0
11	0	0	0	0
10	0	1	1	0

$\bar{x}.z$



wx \ yz	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

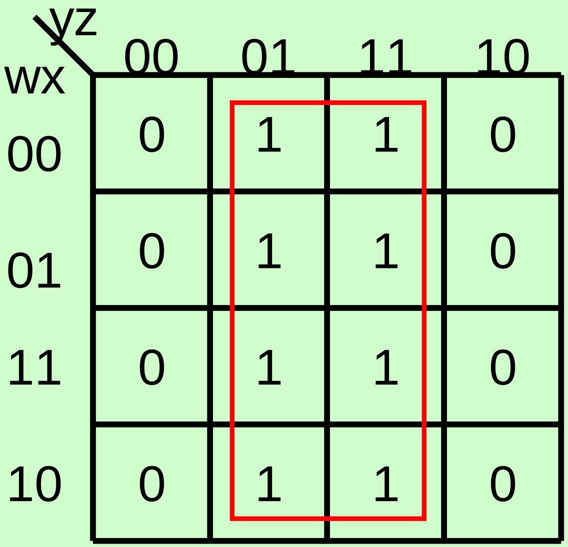
$\bar{x}.\bar{z}$



wx \ yz	00	01	11	10
00	1	0	1	0
01	0	0	0	0
11	0	0	0	0
10	1	0	1	0

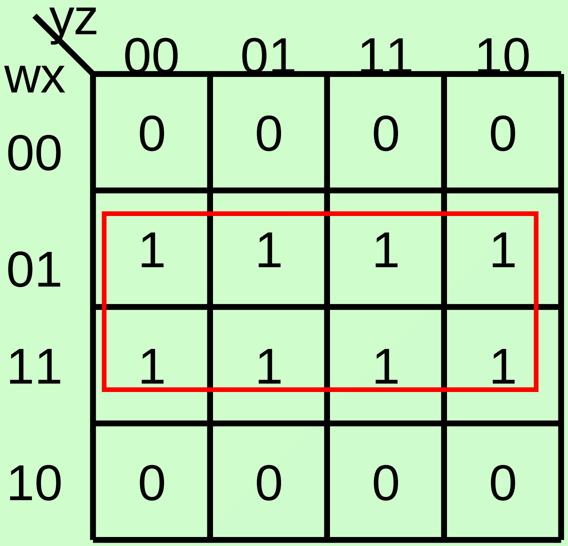
??

Groups of 8



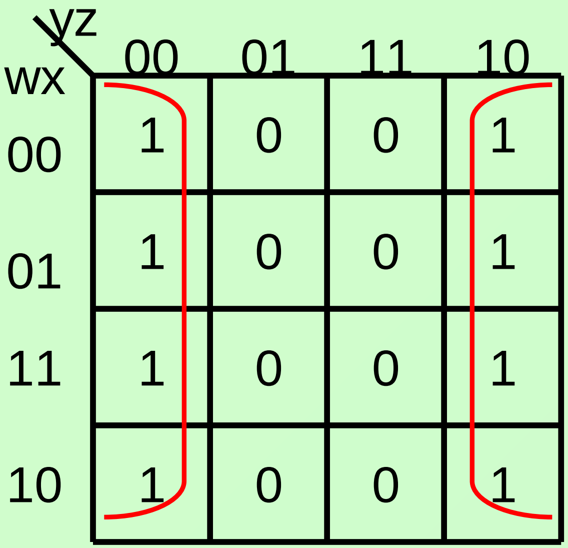
wx \ yz	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

z



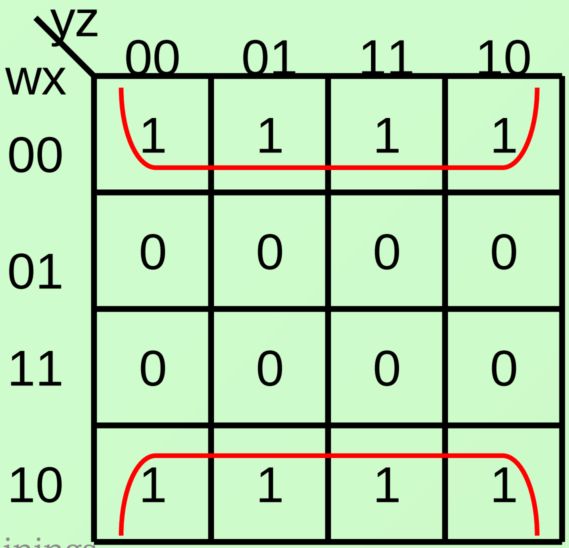
wx \ yz	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

x



wx \ yz	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

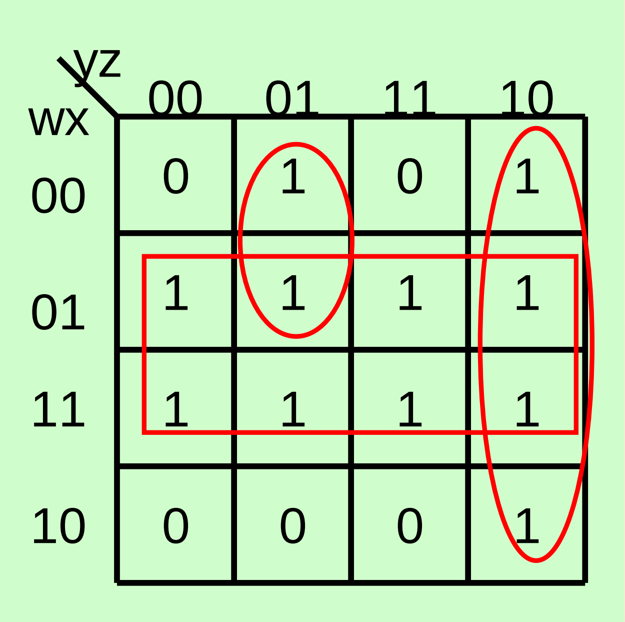
$\overline{z}$



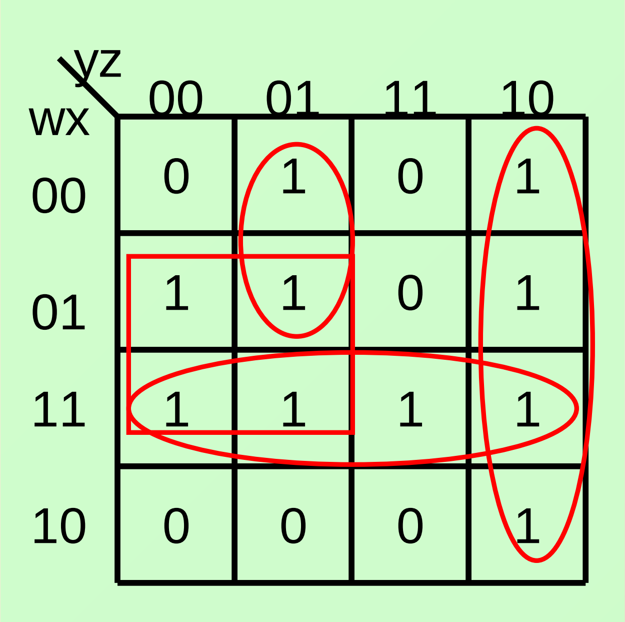
wx \ yz	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

$\overline{x}$

Examples

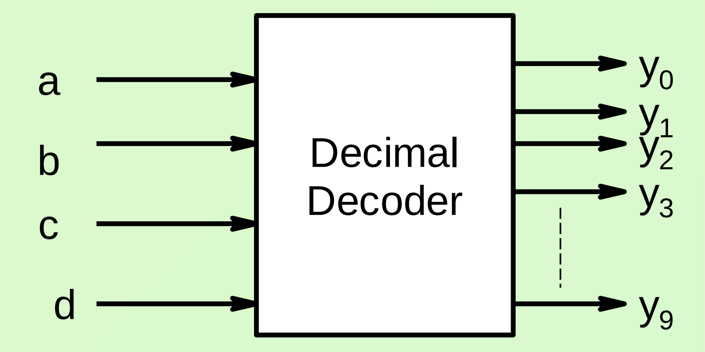


wx \ yz	00	01	11	10
00	0	1	0	1
01	1	1	1	1
11	1	1	1	1
10	0	0	0	1



wx \ yz	00	01	11	10
00	0	1	0	1
01	1	1	0	1
11	1	1	1	1
10	0	0	0	1

Don’t care terms



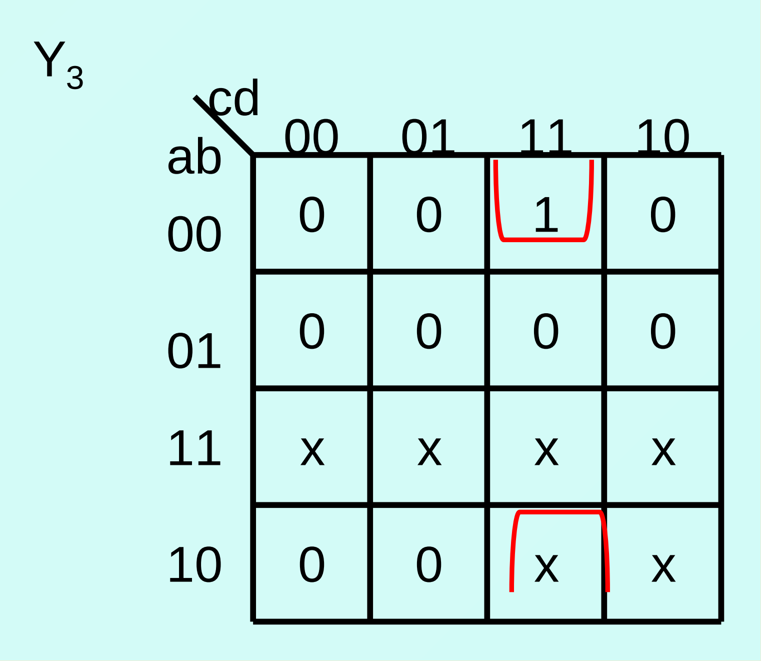
Y_3

cd / ab	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	x	x	x	x
10	0	0	x	x

$$y_3 = \overline{a}.\overline{b}.c.d$$

a	b	c	d	$y_0y_1y_2y_3y_4y_5y_6y_7y_8y_9$
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 0 0 0
0	0	1	0	0 0 1 0 0 0 0 0 0 0
0	0	1	1	0 0 0 1 0 0 0 0 0 0
0	1	0	0	0 0 0 0 1 0 0 0 0 0
0	1	0	1	0 0 0 0 0 1 0 0 0 0
0	1	1	0	0 0 0 0 0 0 1 0 0 0
0	1	1	1	0 0 0 0 0 0 0 1 0 0
1	0	0	0	0 0 0 0 0 0 0 0 1 0
1	0	0	1	0 0 0 0 0 0 0 0 0 1
1	0	1	0	x x x x x x x x x x
1	0	1	1	x x x x x x x x x x
1	1	0	0	x x x x x x x x x x
1	1	0	1	x x x x x x x x x x
1	1	1	0	x x x x x x x x x x
1	1	1	1	x x x x x x x x x x

Don’t care terms can be chosen as 0 or 1.
Depending on the problem, we can choose the don’t care term as 1 and use it to obtain a simpler Boolean expression



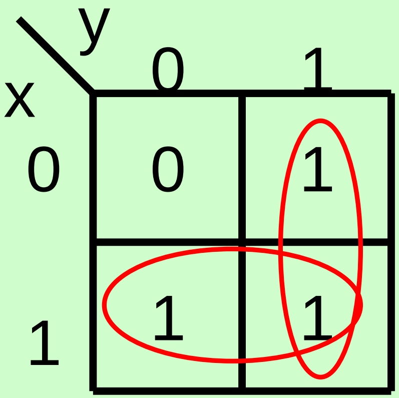
Y_3

ab \ cd	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	x	x	x	x
10	0	0	x	x

$$y_3 = \bar{b}.c.d$$

Don’t care terms should only be included in encirclements if it helps in obtaining a larger grouping or smaller number of groups.

Minimization of Product of Sum Terms using Kmap

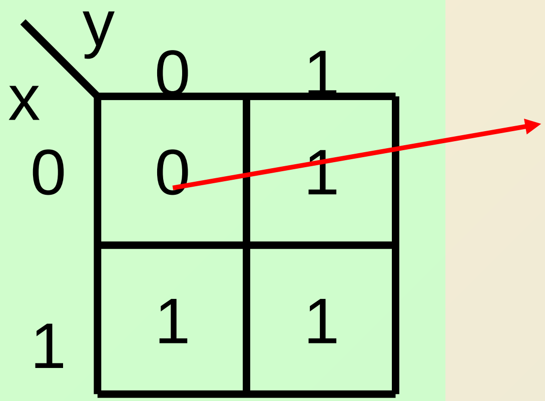


x \ y	0	1
0	0	1
1	1	1

$$f = x + \bar{x}.y + x.y$$

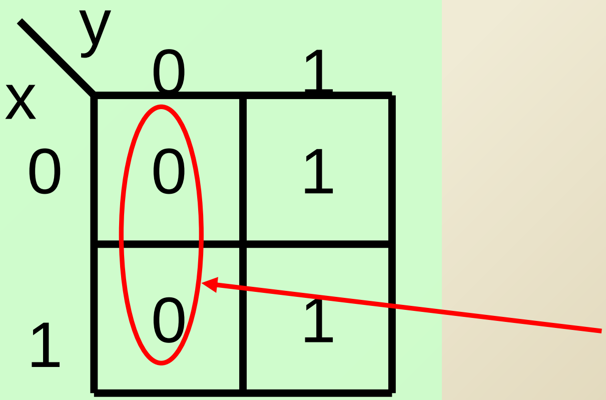
$$= x + (\bar{x} + x).y$$

$$= x + y$$



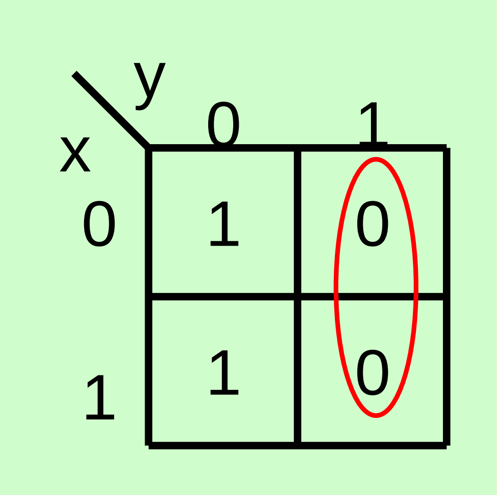
x \ y	0	1
0	0	1
1	1	1

$$f = x + y$$



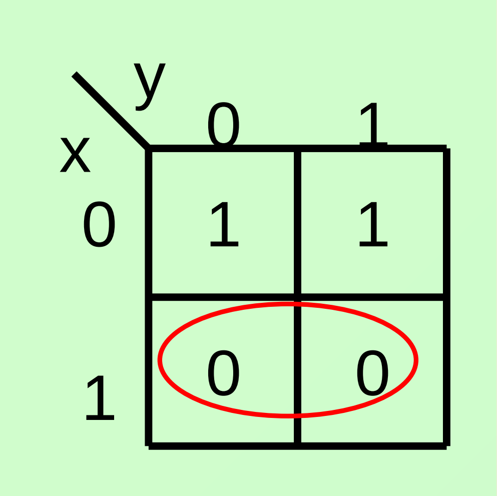
x \ y	0	1
0	0	1
1	0	1

$$f = y$$



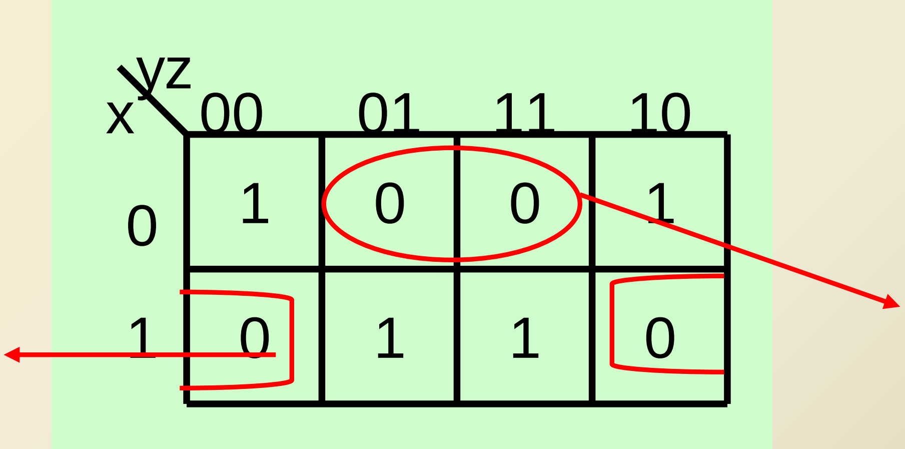
x \ y	0	1
0	1	0
1	1	0

$\Rightarrow f = \bar{y}$



x \ y	0	1
0	1	1
1	0	0

$\Rightarrow f = \bar{x}$



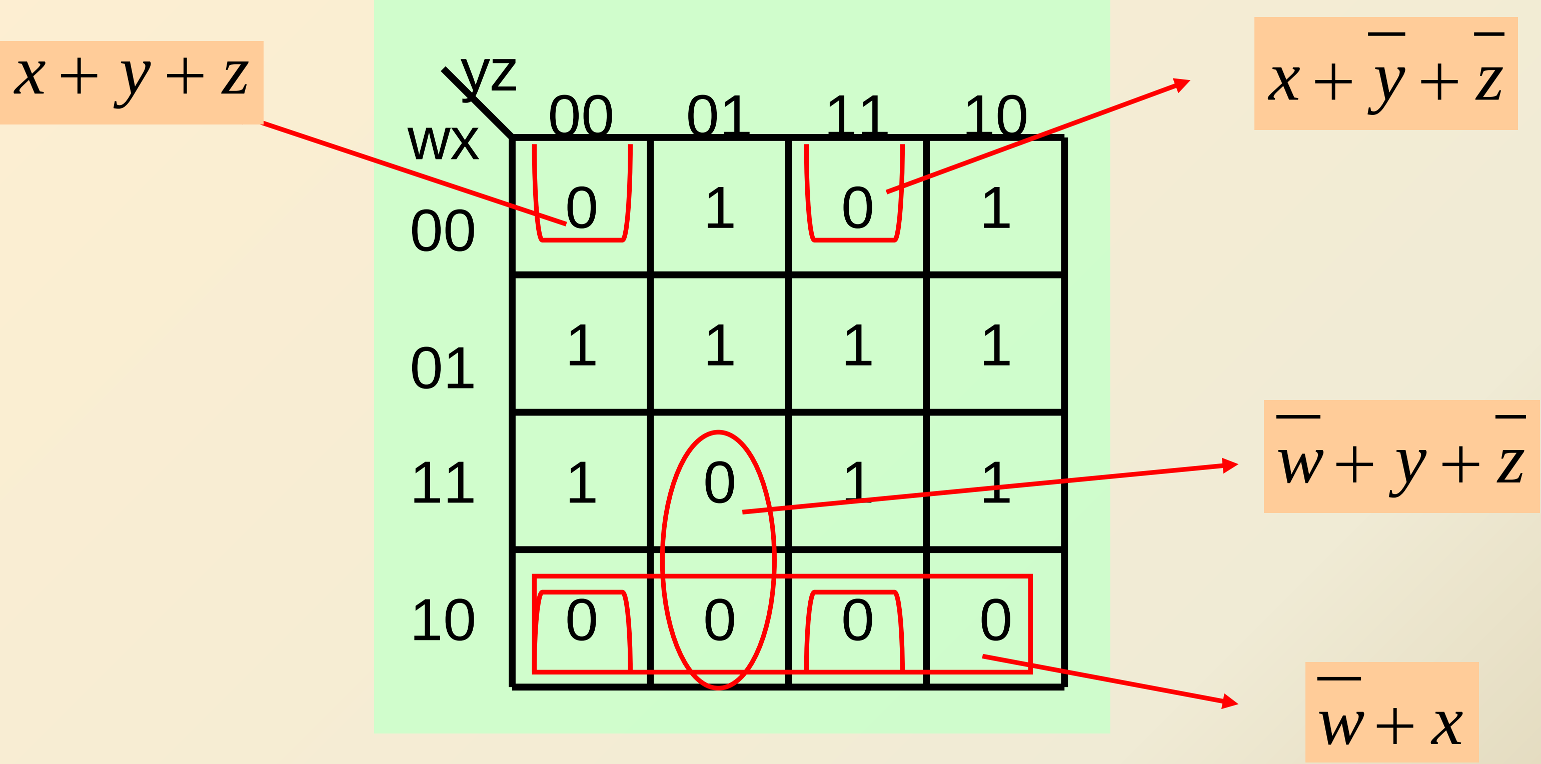
x \ yz	00	01	11	10
0	1	0	0	1
1	0	1	1	0

$\bar{x}.+z$

$x+\bar{z}$

$f = (\bar{x}\ .+z).(x+\bar{z})$

$\Rightarrow f = \bar{x}\ .\bar{z}+x.z$



wx \ yz	00	01	11	10
00	0	1	0	1
01	1	1	1	1
11	1	0	1	1
10	0	0	0	0

$x+y+z$

$x+\overline{y}+\overline{z}$

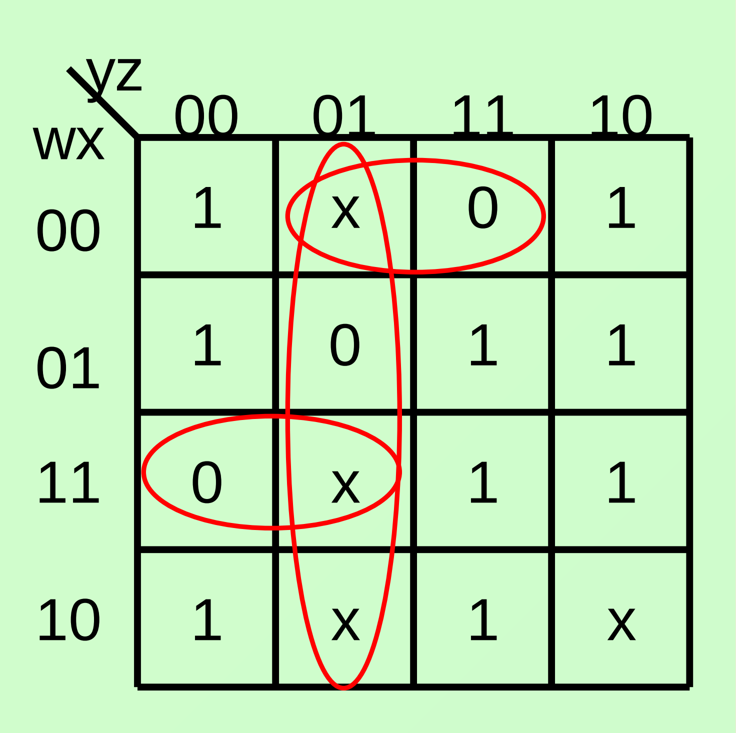
$\overline{w}+y+\overline{z}$

$\overline{w}+x$

$$\mathrm{f}=(x+y+z).(x+\overline{y}+\overline{z}).(\overline{w}+y+\overline{z}).(\overline{w}+x)$$

Example

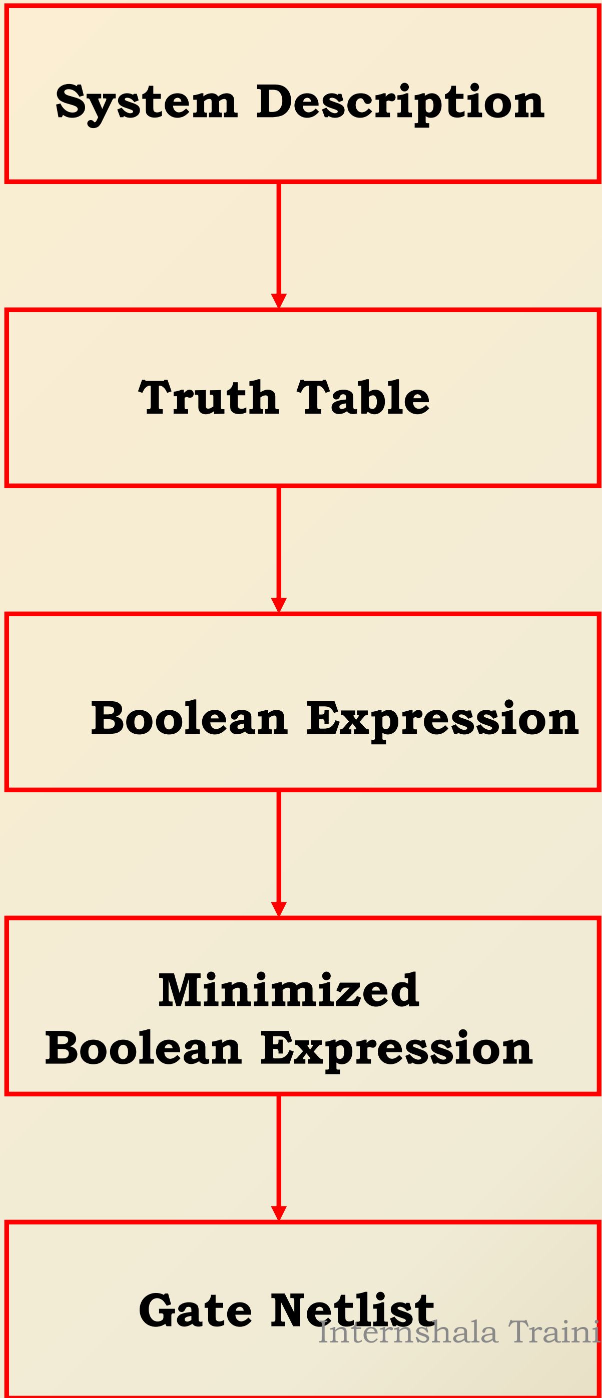
Obtain the minimized PoS by suitably using don’t care terms



wx \ yz	00	01	11	10
00	1	x	0	1
01	1	0	1	1
11	0	x	1	1
10	1	x	1	x

$$f = (x + w + \bar{z}).(\bar{x} + \bar{w} + y).(y + \bar{z})$$

Design Flow

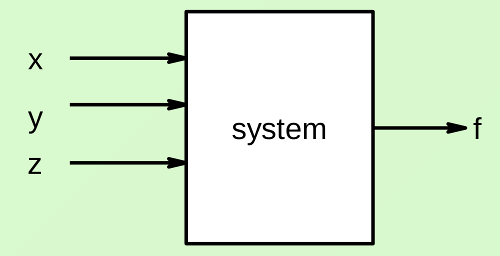

System Description

Truth Table

Boolean Expression

Minimized Boolean Expression

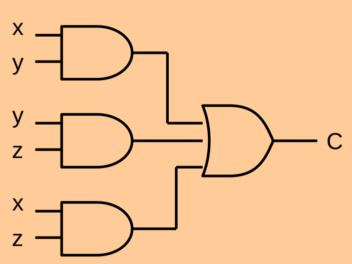
Gate Netlist



x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

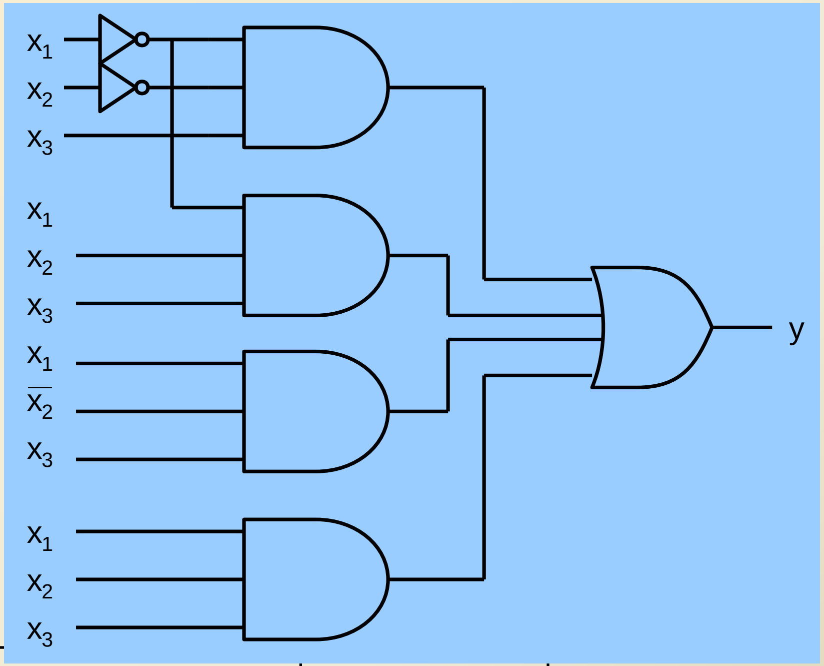
$$\mathrm{f} = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.z + x.y.z$$

$$\Rightarrow \mathrm{f} = \bar{x}\,.\bar{z} + x.z$$



Mapping of Boolean expression to a Network of gates available in the library

$$y=\overline{x_1}\,.\,\overline{x_2}\,.x_3+\overline{x_1}.x_2.x_3+x_1.\overline{x_2}.x_3+x_1.x_2.x_3$$



Library of available Gates	Cost
Inverter	1
Two input NAND	2
Three input NAND	3
AND-OR-Invert $Y=\overline{AB+C}$	3

IMPLEMENTATION USING SPECIFIC GATES

Implementation using only NAND gates

$\overline{x.x} = \overline{x}$

$\overline{x.y}$

$x.y$

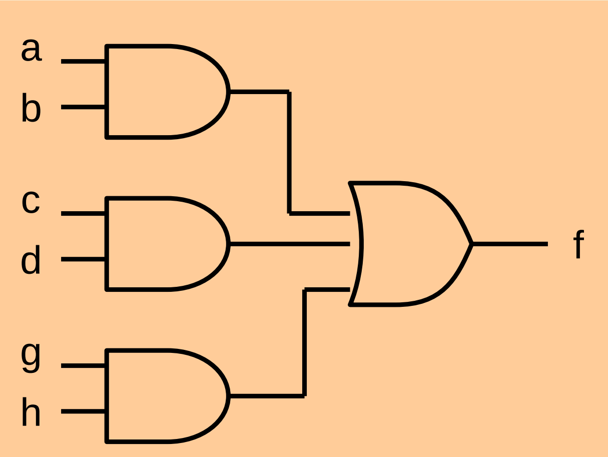
$\overline{x}$

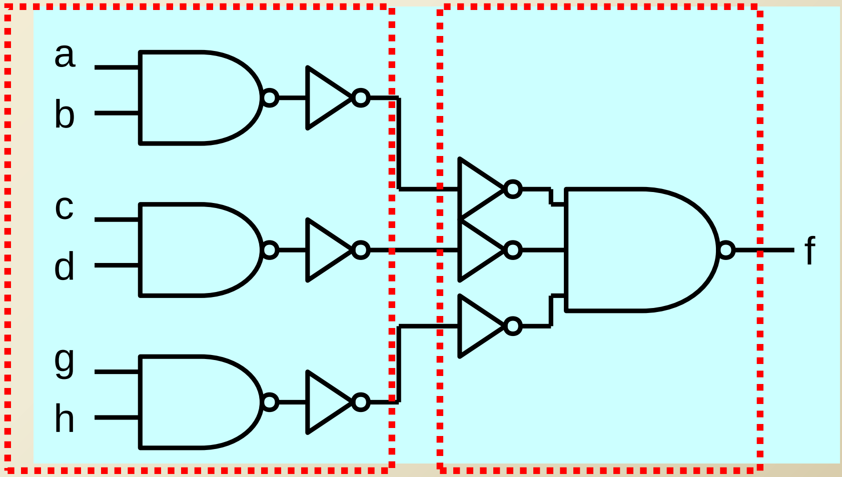
$\overline{y}$

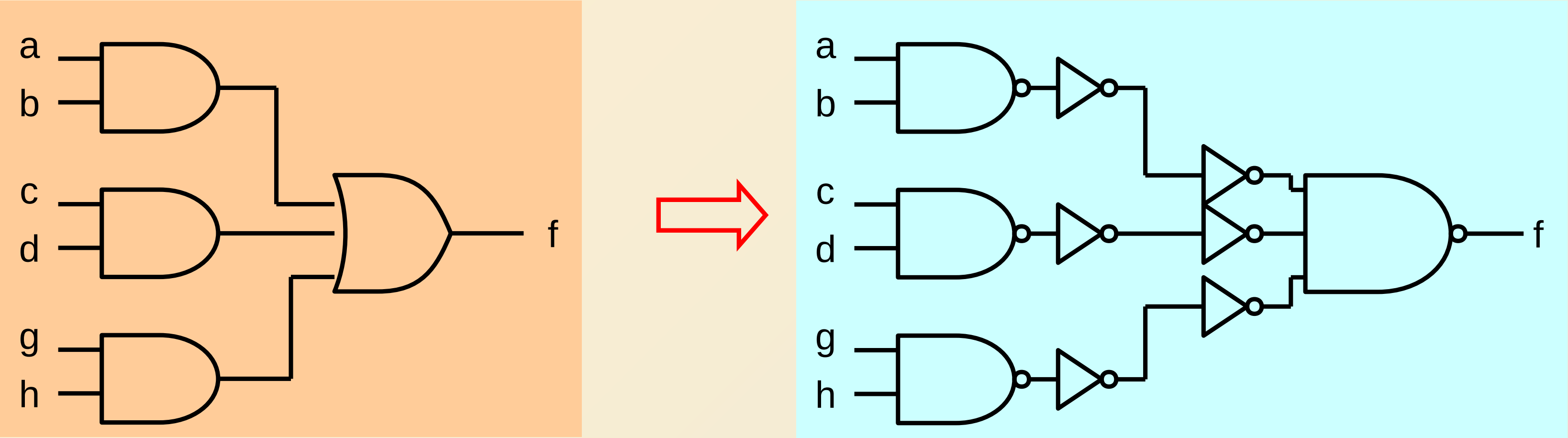
$f = \overline{\overline{x}.\overline{y}} = x + y$

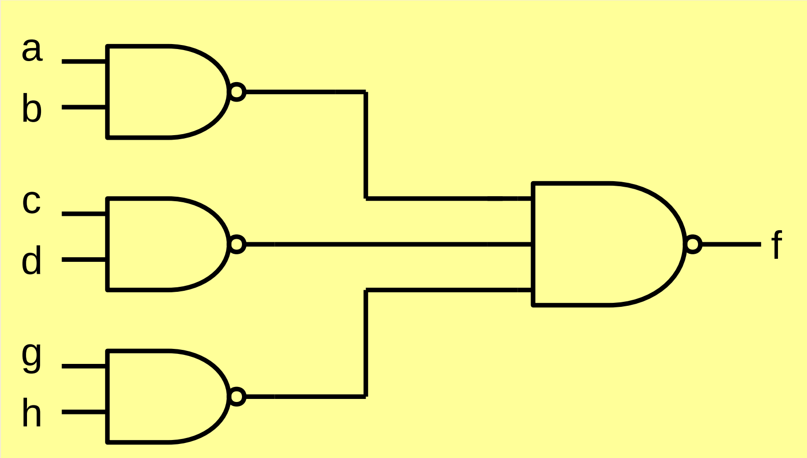
A SoP expression is easily implemented with NAND gates.

$f = a.b + c.d + g.h$





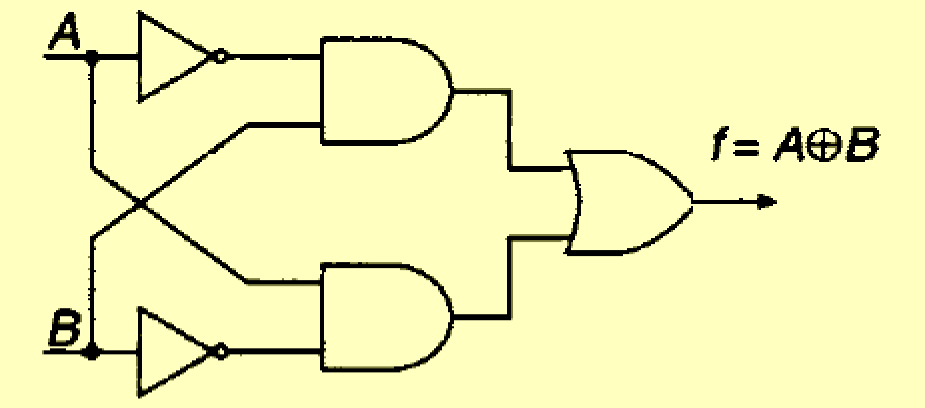


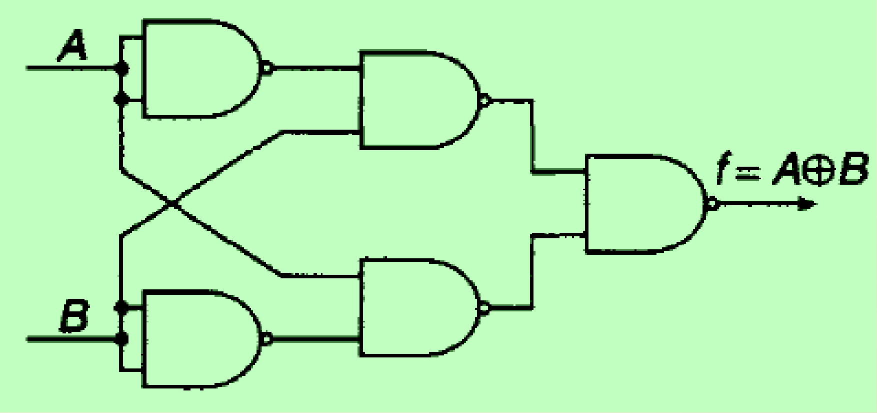


There is a one-to-one mapping between AND-OR network and NAND network

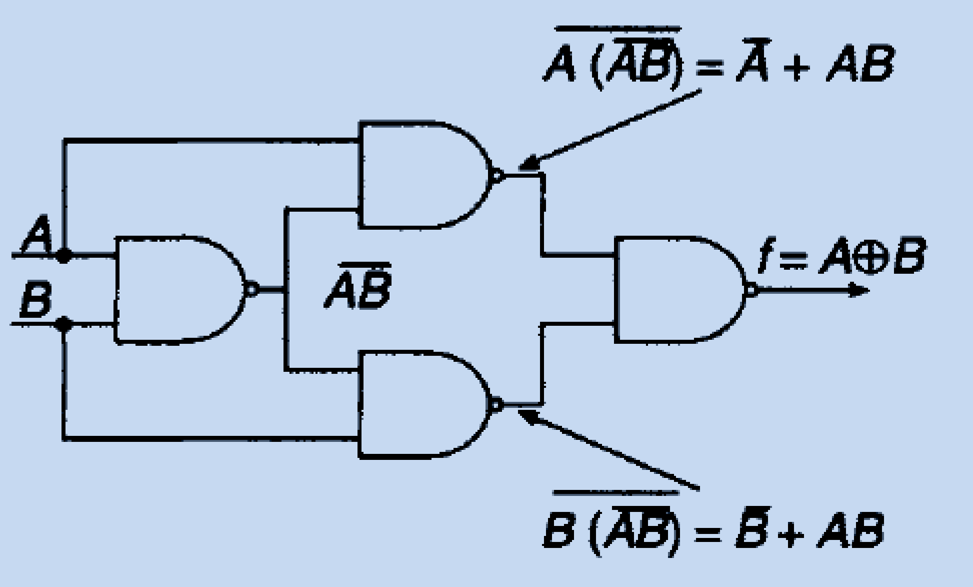
Often there is lot of further optimization that can be done

Consider implementation of XOR gate $f = \overline{A}.B + A.\overline{B}$





$$
\begin{aligned}
f &= \overline{A}.B + B.\overline{B} + A.\overline{B} + A.\overline{A} \\
&= B(\overline{A} + \overline{B}) + A(\overline{A} + \overline{B})
\end{aligned}
$$


$\overline{A(\overline{AB})} = \overline{A} + AB$
A
B
$\overline{AB}$
f= A⊕B
$\overline{B(\overline{AB})} = \overline{B} + AB$

Implementation using only NOR gates

$\overline{x+x}=\bar{x}$

$\overline{x+y}$

$x+y$

$\bar{x}$

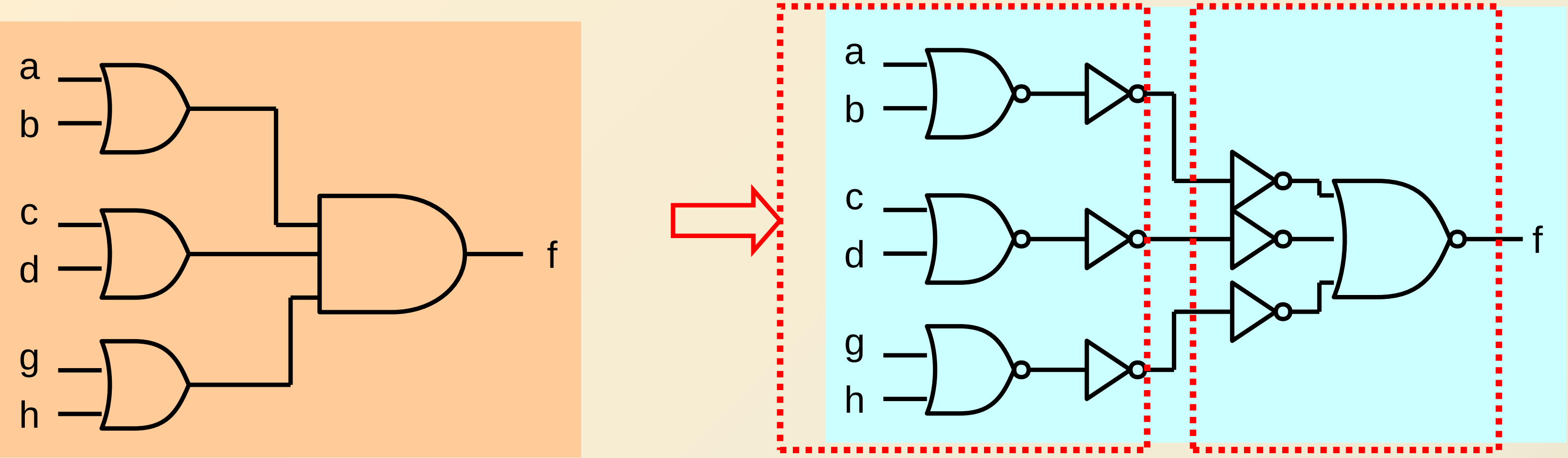
$\bar{y}$

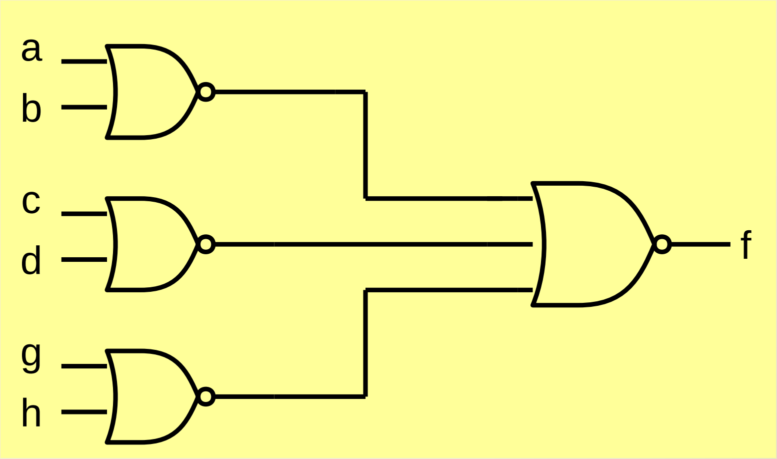
$f=\overline{\bar{x}+\bar{y}}=x.y$

To implement using NOR gates, it is easiest to start with minimized Boolean expression in POS form

$$f=(a+b).(c+d).(g+h)$$

$$f = (a+b).(c+d).(g+h)$$

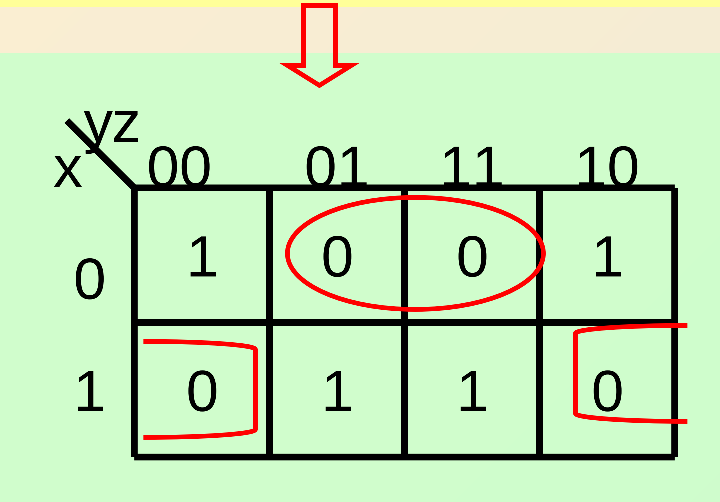




There is a one-to-one mapping between OR-AND network and NOR network

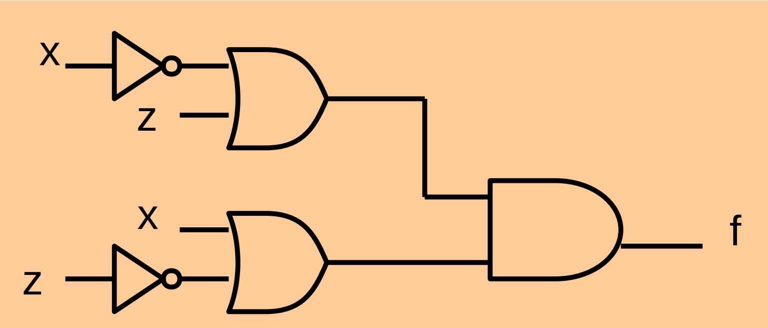
To implement SoP expression using NOR gates, determine first the corresponding PoS expression and then follow the procedure outlined earlier

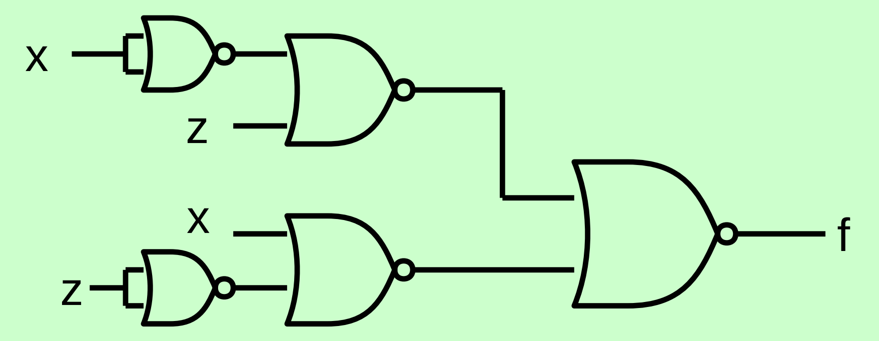
Implement $f(x,y,z)=\bar{x}\,.\bar{z}+x.z$ using NOR gates



x \ yz	00	01	11	10
0	1	0	0	1
1	0	1	1	0

$\Rightarrow\ f=(\bar{x}\,.+z).(x+\bar{z})$





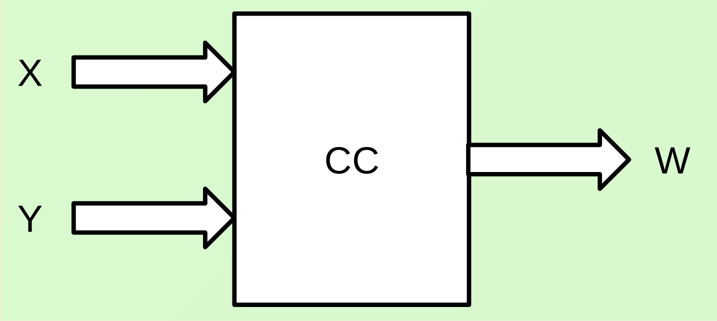
Similarly PoS expression can be implemented as NAND network by first converting it to SoP expression and then following the procedure outlined earlier

COMBINATIONAL CIRCUIT DESIGN

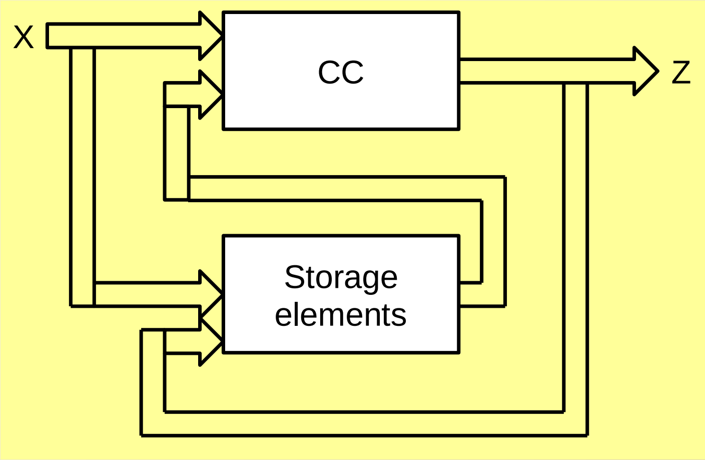
Digital Circuits

Combinational Circuits

Sequential Circuits

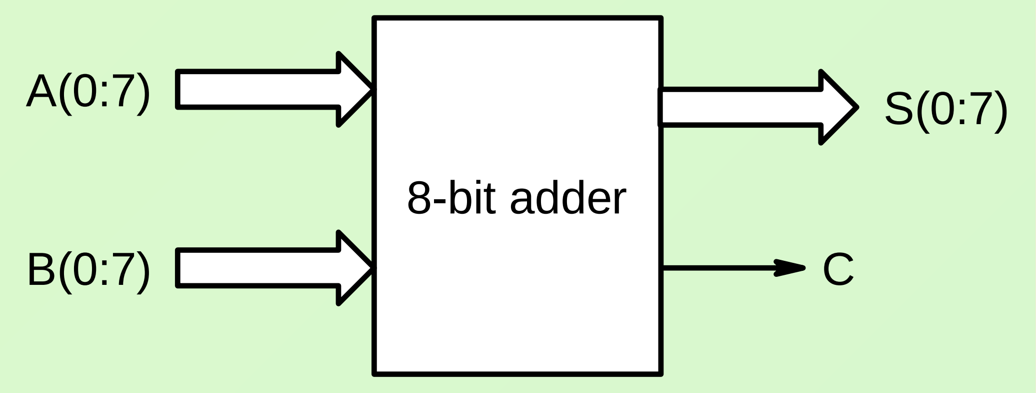


Output is determined by current values of inputs only.



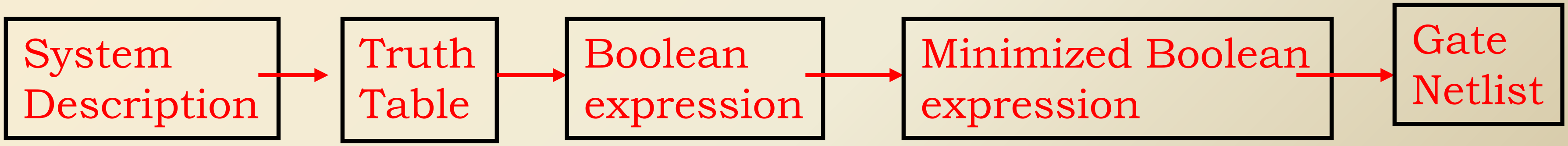
Output is determined in general by current values of inputs and past values of inputs/outputs as well.

Design of Complex Combinational circuits



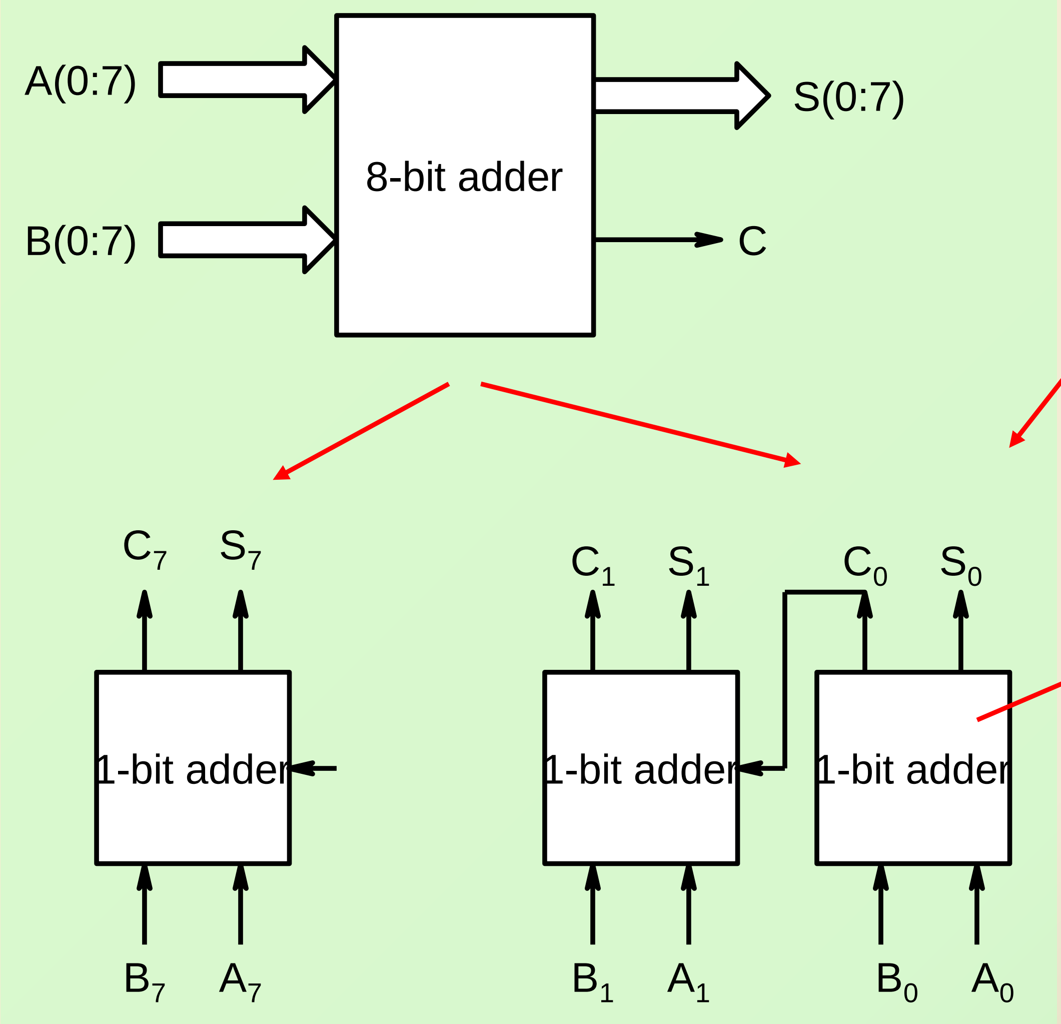
$A_7A_6\ldots A_0$	$B_7B_6\ldots B_0$	$S_7S_6\ldots S_0$	C
00...0	00...0	00...0	0
00...1	00...0	00...1	0
00...0	00...1	00...1	0
⋮	⋮	⋮	⋮

Truth table has 2^{16} entries



This design approach becomes difficult to use

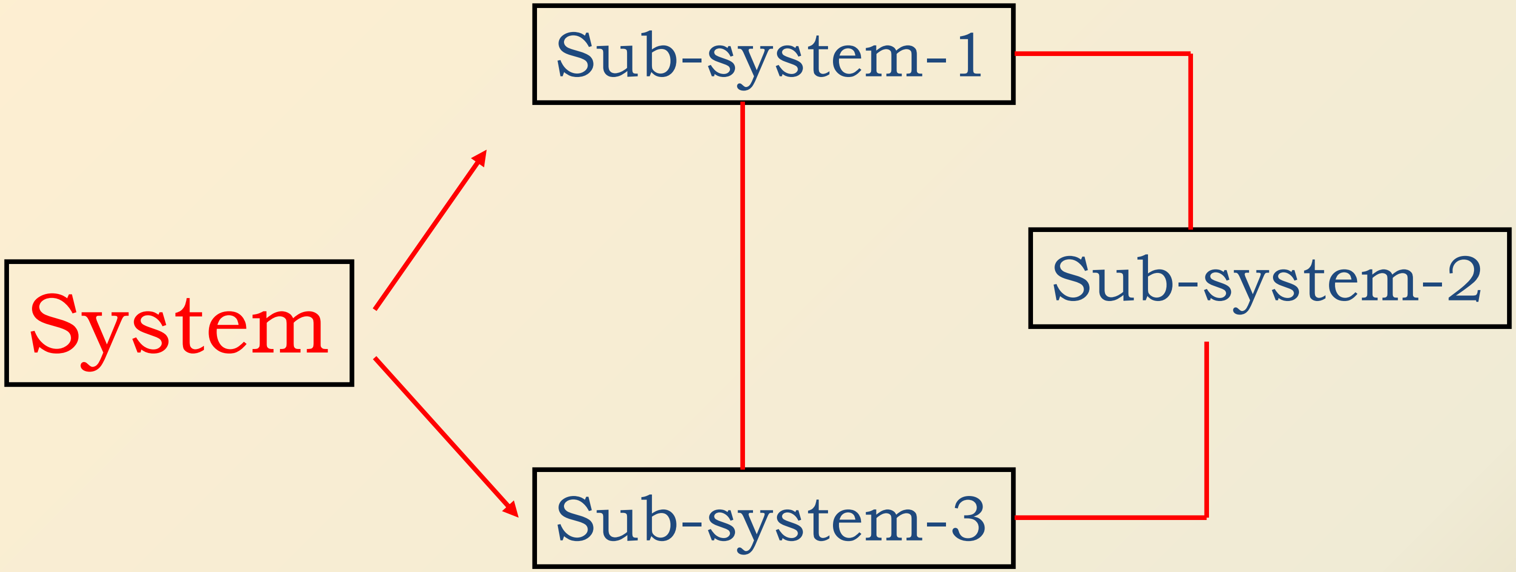
Design system as a network of sub-systems that are of manageable size and can be implemented using the earlier approach of truth table, minimization etc.



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

a	b	c	S	CY
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

General Approach

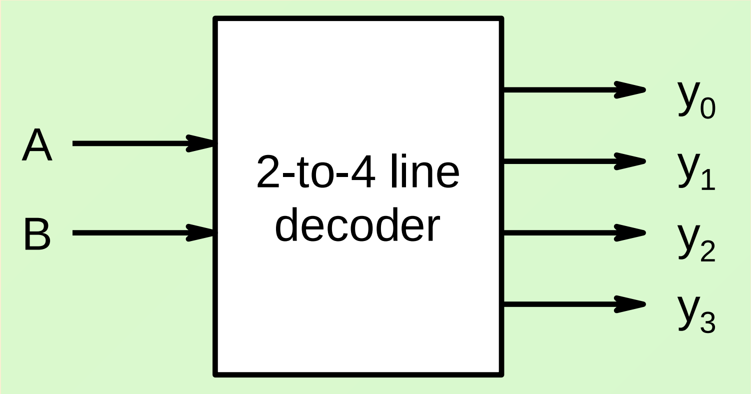


There are certain sub-systems or blocks that are used quite often such as :

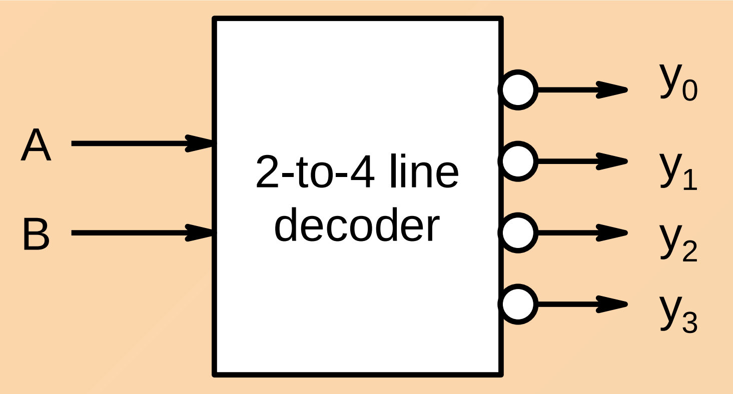
1. **decoders, encoders**
2. **Multiplexers**
3. **Adder/Subtractors, Multipliers**
4. **Comparators**
5. **Parity Generators**
6. **……..**

Decoders

Maps a smaller number of inputs to a larger set of outputs in general



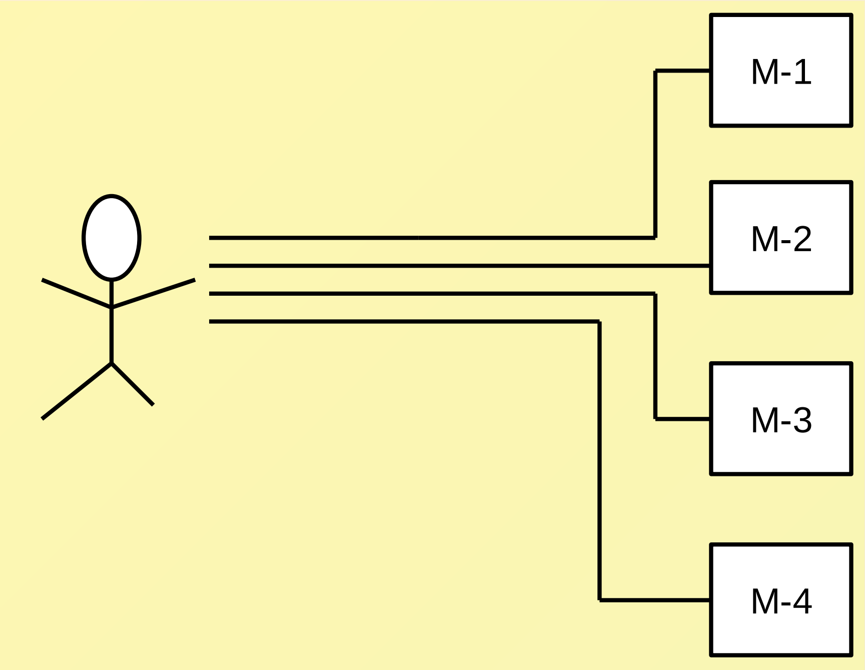
B	A	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

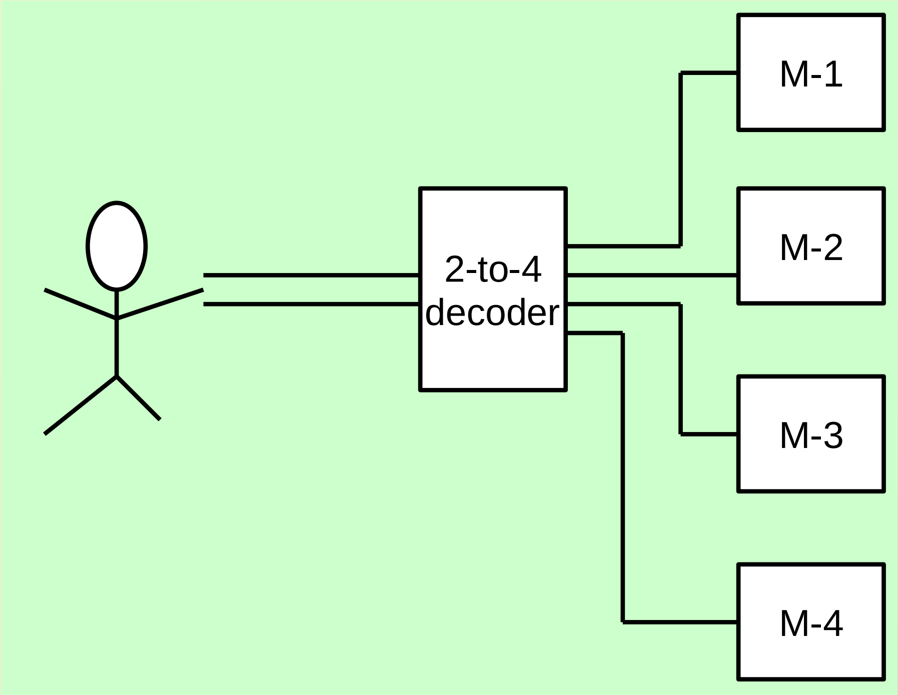


b	a	Y_0	Y_1	Y_2	Y_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

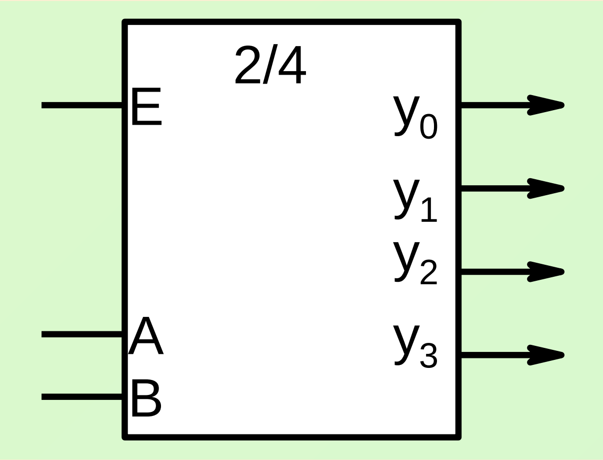
Active Low

Example

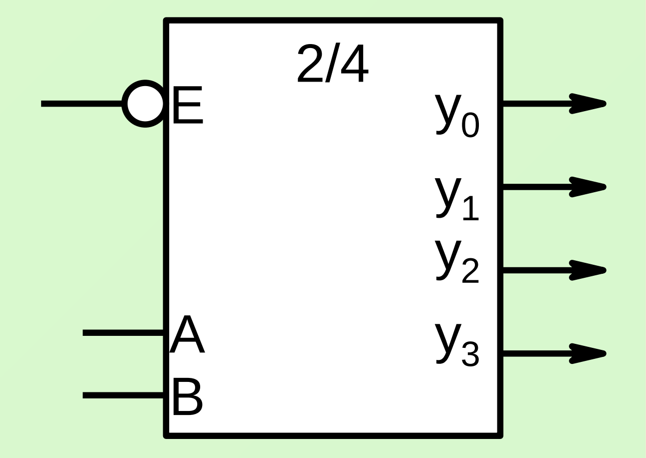




Decoder with Enable Input

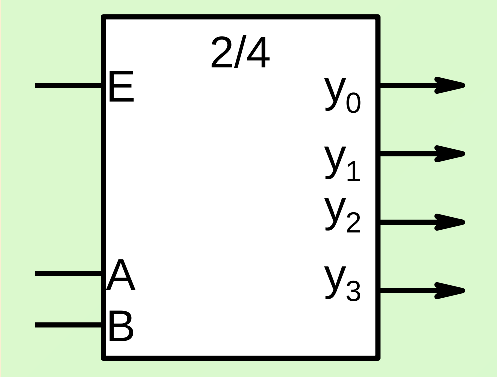


E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



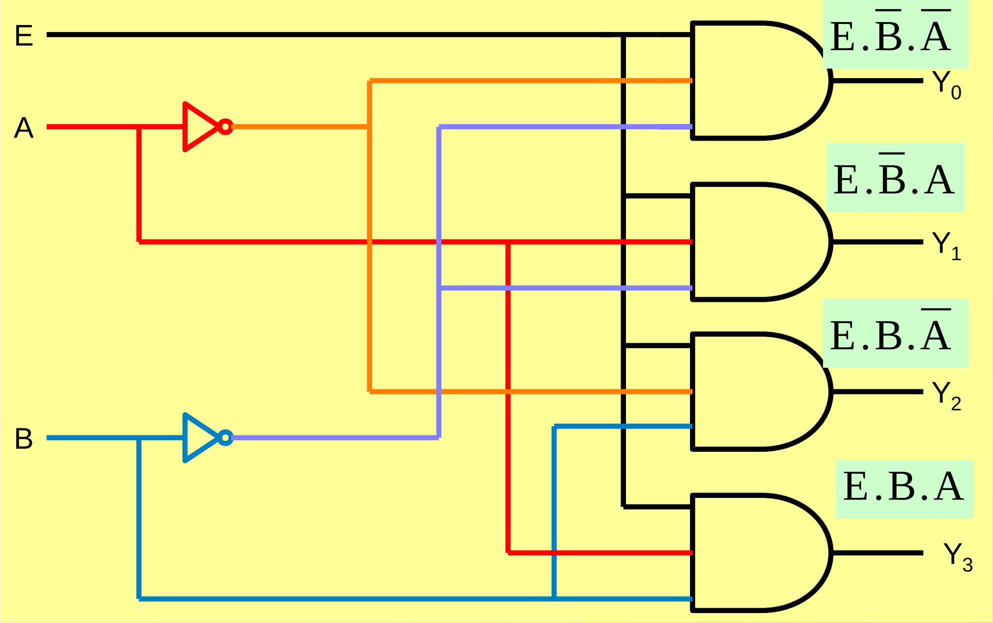
E	B	A	Y_0	Y_1	Y_2	Y_3
1	x	x	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

Decoder: gate Implementation



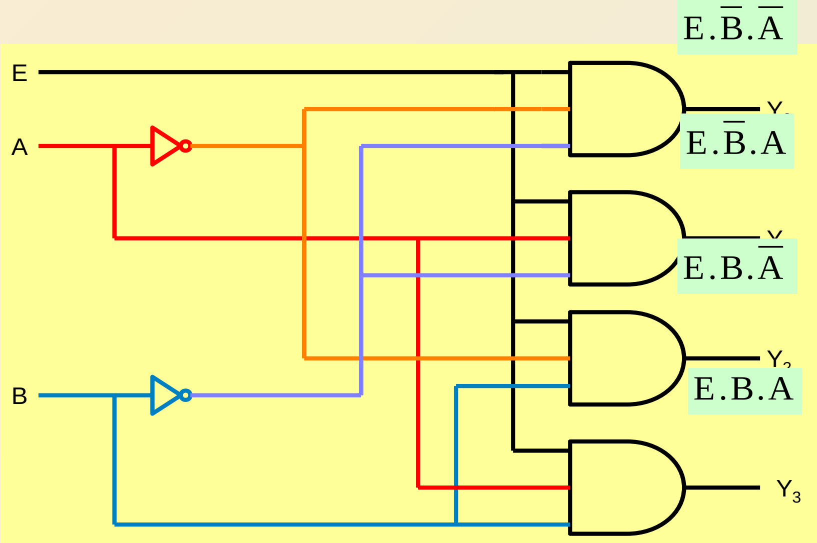
E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

$$Y_0 = E.\overline{B}.\overline{A}\ ; Y_1 = E.\overline{B}.A\ ; Y_2 = E.B.\overline{A}\ ; Y_3 = E.B.A$$



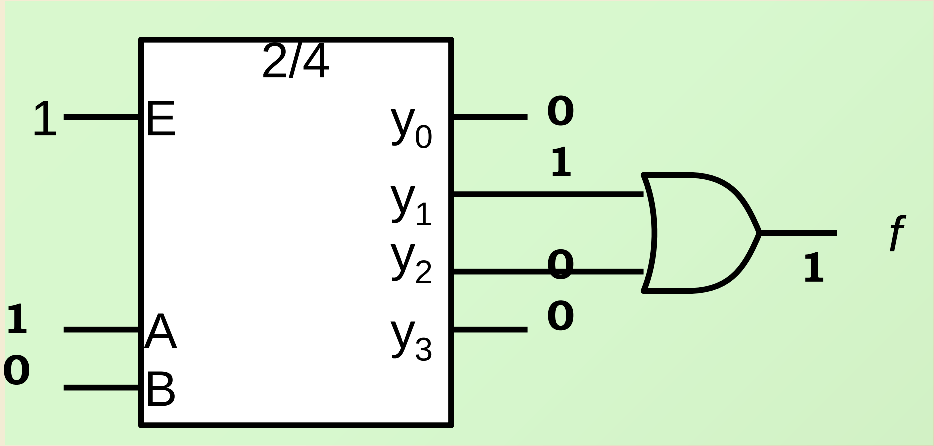
A n to 2^n decoder is a minterm generator

x	y	min term	
0	0	$\overline{x}.\overline{y}$	m0
0	1	$\overline{x}.y$	m1
1	0	$x.\overline{y}$	m2
1	1	$x.y$	m3



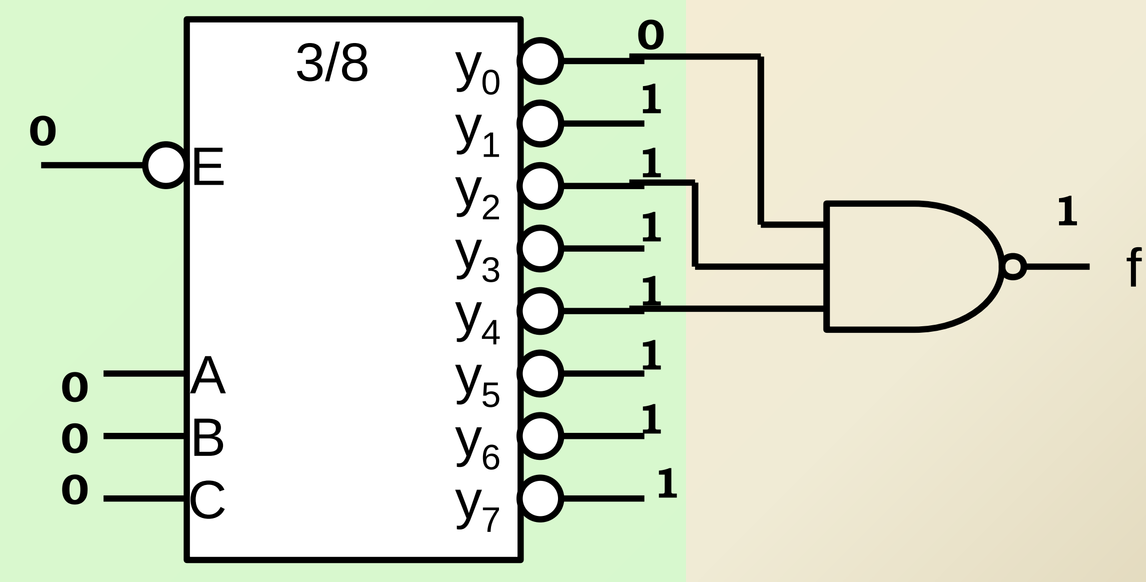
It can be used to implement any combinational circuit

B	A	f_1
0	0	0
0	1	1
1	0	1
1	1	0



Implementation of a 3-variable function with a 3-to-8 decoder

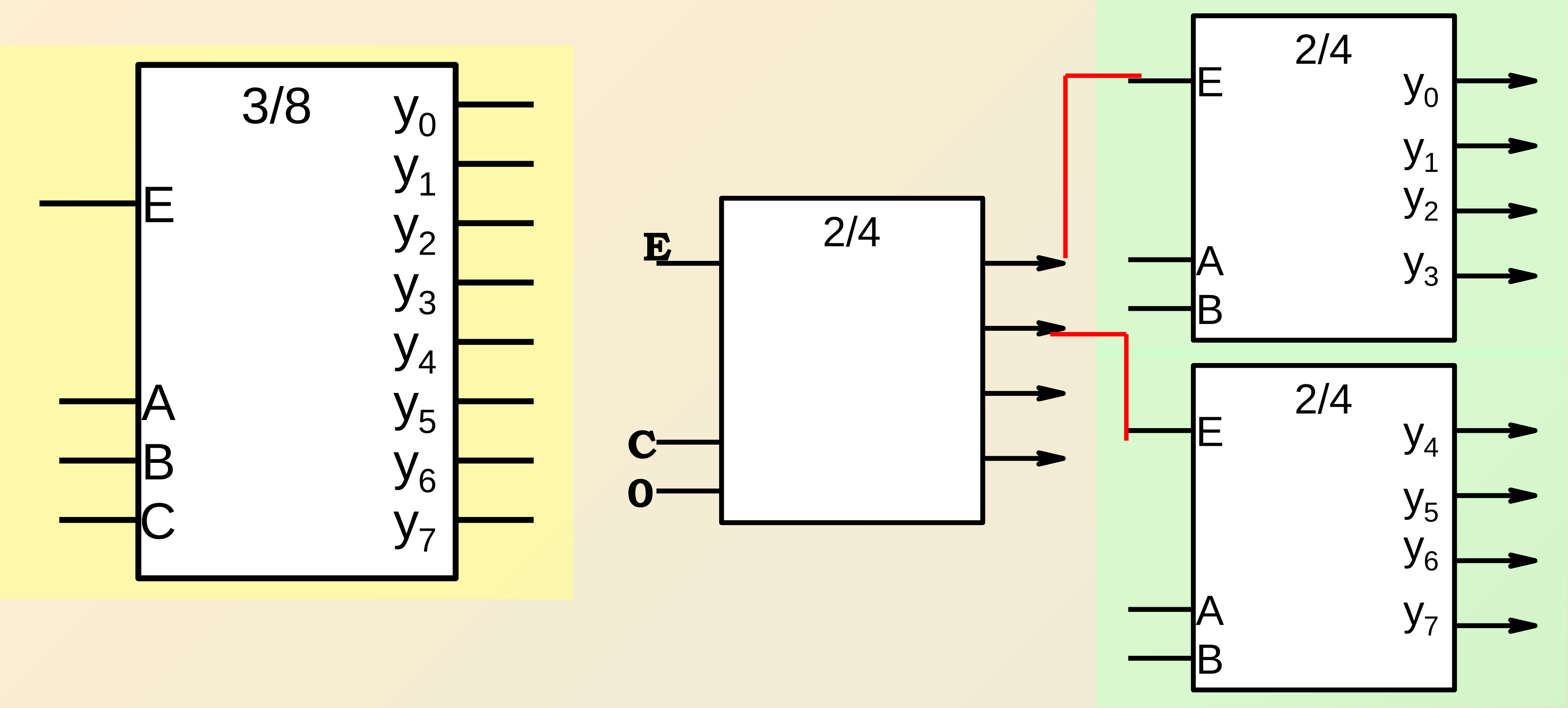
C	B	A	*f*
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



Although it is easy to implement any combinational circuit with this method , it is often very inefficient in terms of gate utilization. Note that this method does not require any minimization.

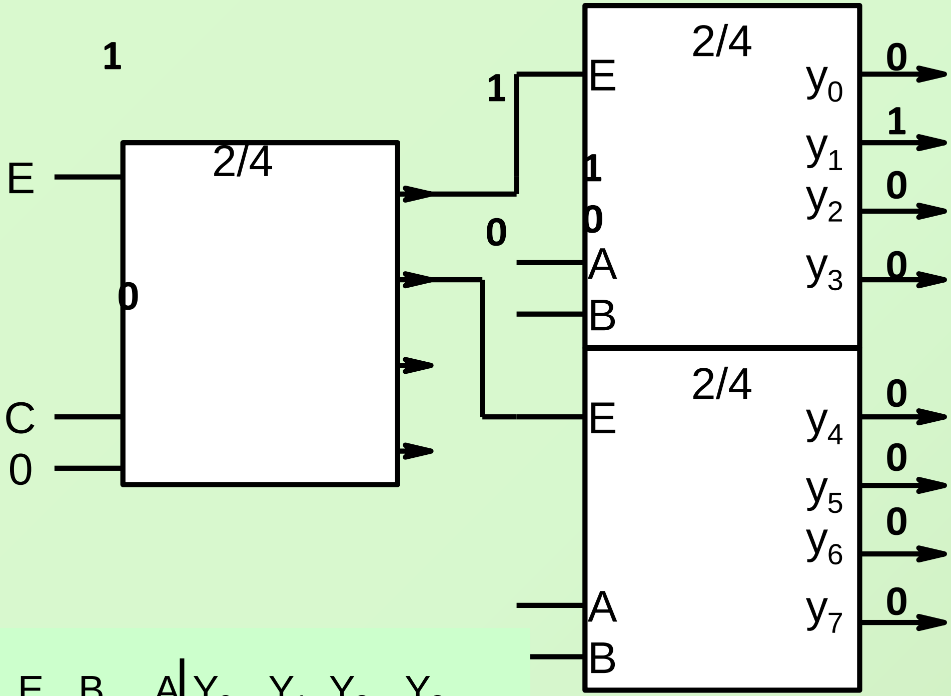
Implementing larger decoders using simpler ones.

3/8 decoder using 2/4 decoders



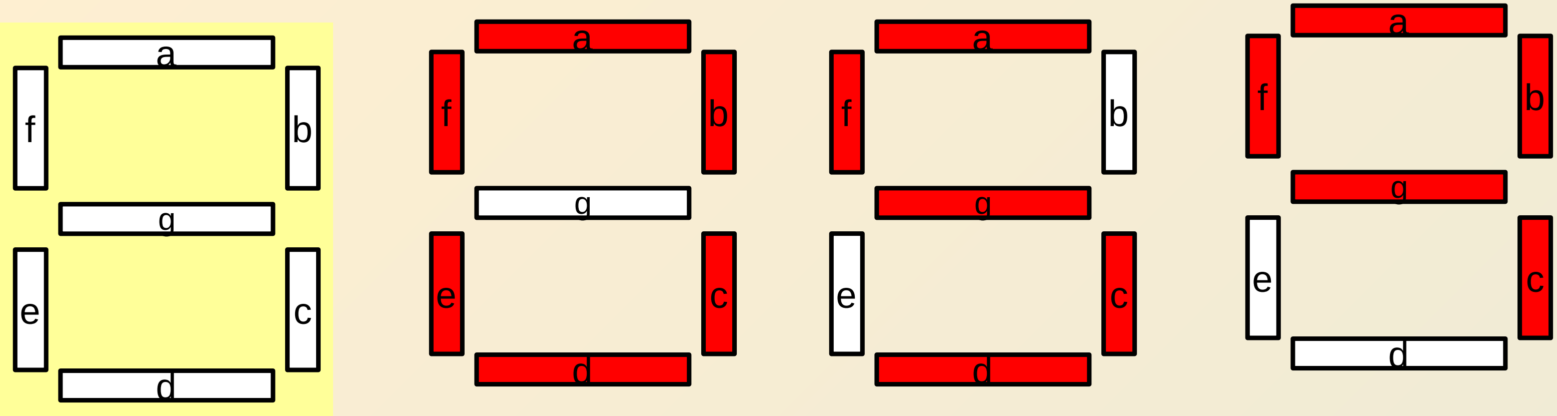
How many 2/4 decoders are required to implement a 4/16 decoder ?

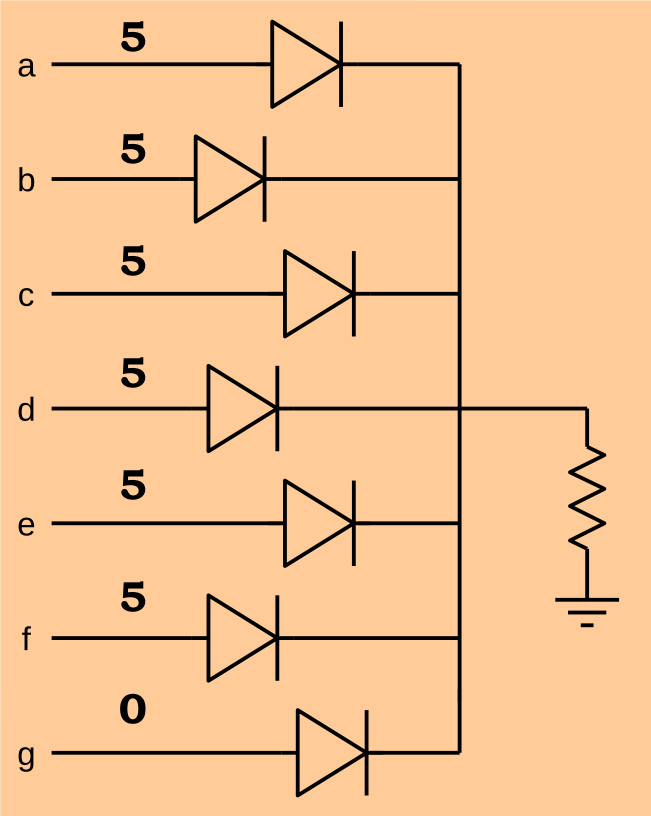
E	C	B	A	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

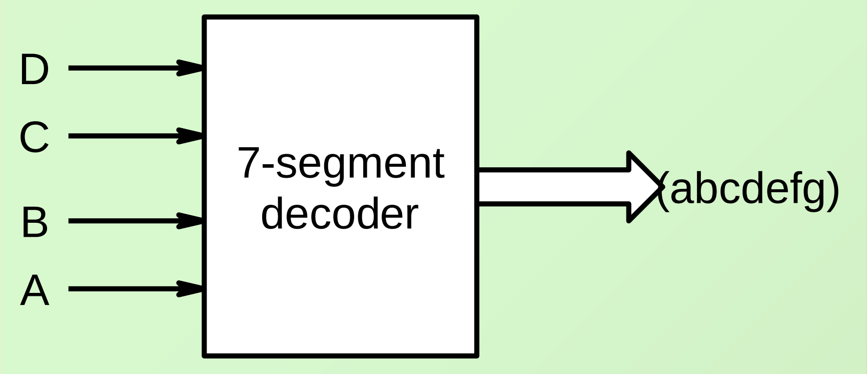


E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

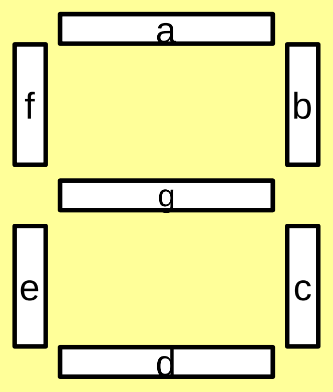
Seven segment decoder

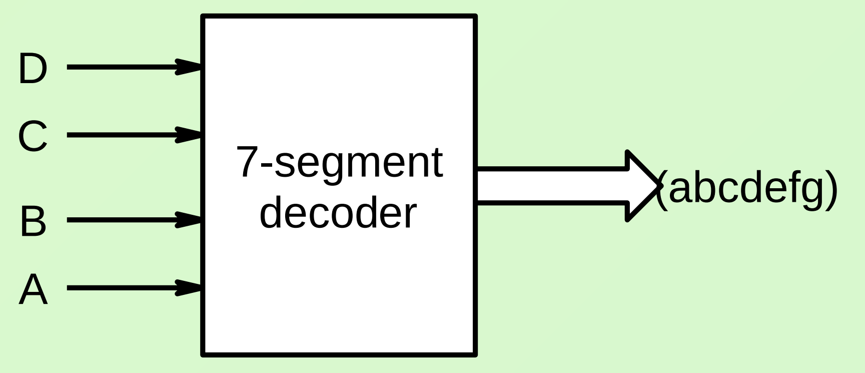


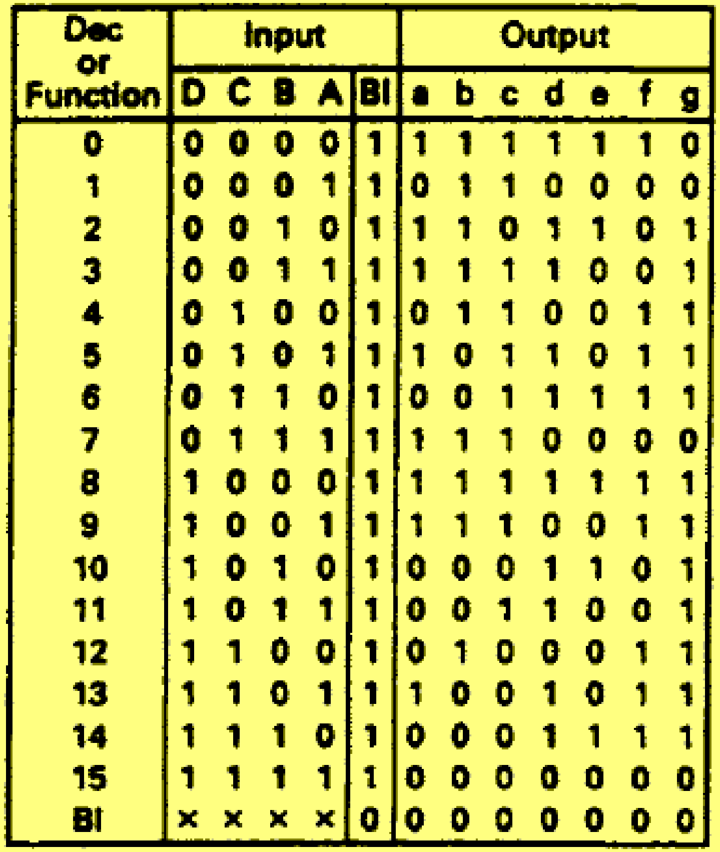




Seven segment decoder

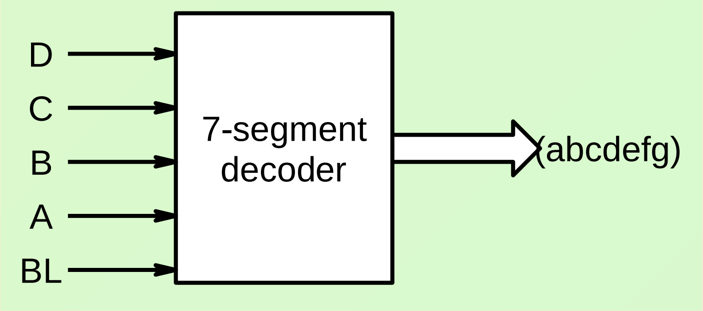


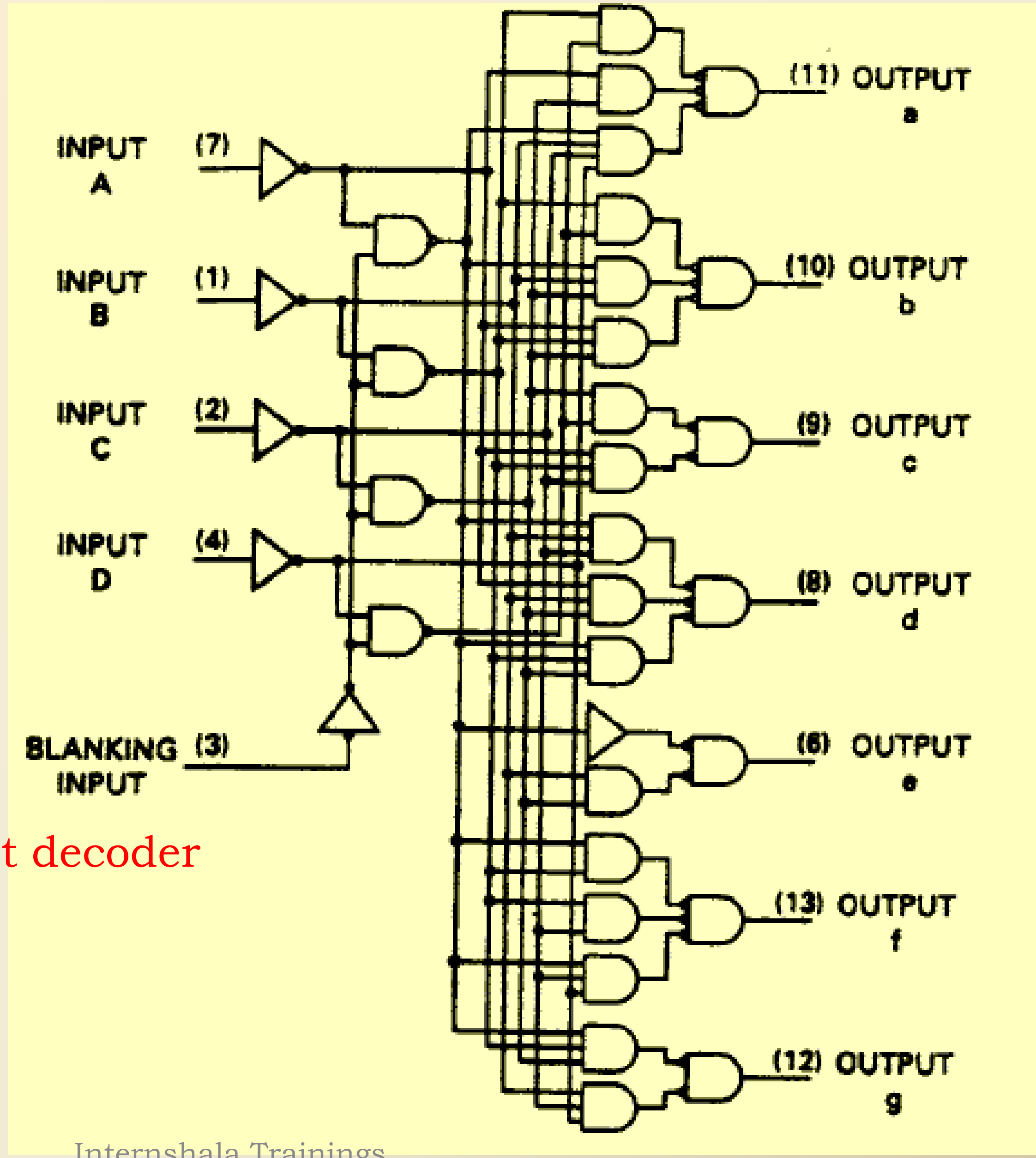




Dec or Function	Input					Output						
	D	C	B	A	BI	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	1	0	0	1
4	0	1	0	0	1	0	1	1	0	0	1	1
5	0	1	0	1	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	0	1	1
10	1	0	1	0	1	0	0	0	1	1	0	1
11	1	0	1	1	1	0	0	1	1	0	0	1
12	1	1	0	0	1	0	1	0	0	0	1	1
13	1	1	0	1	1	1	0	0	1	0	1	1
14	1	1	1	0	1	0	0	0	1	1	1	1
15	1	1	1	1	1	0	0	0	0	0	0	0
BI	x	x	x	x	0	0	0	0	0	0	0	0

DC \ BA	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	0	0
10	1	1	0	0

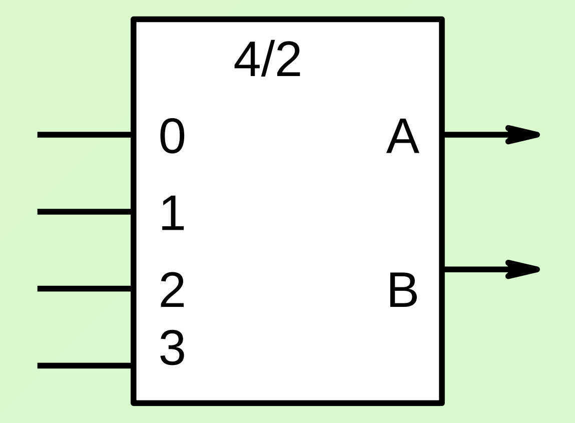




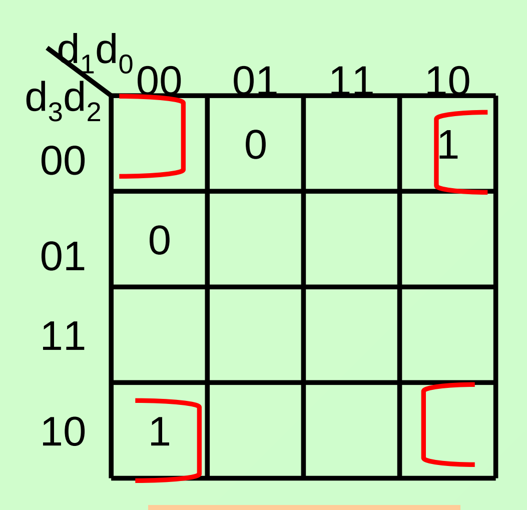
7449 BCD to seven segment decoder

Encoders

An encoder performs the inverse operation of a decoder.

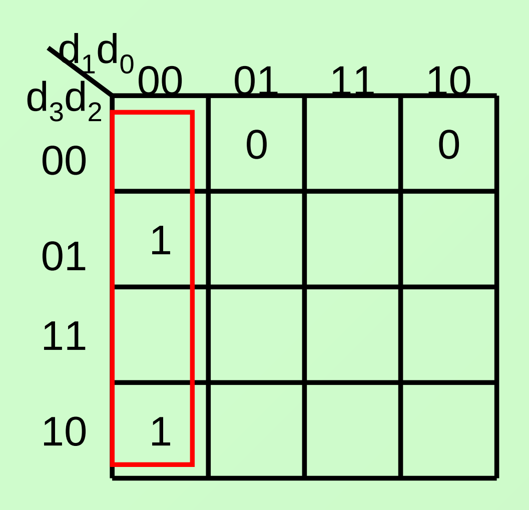


d_3	d_2	d_1	d_0	B	A
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



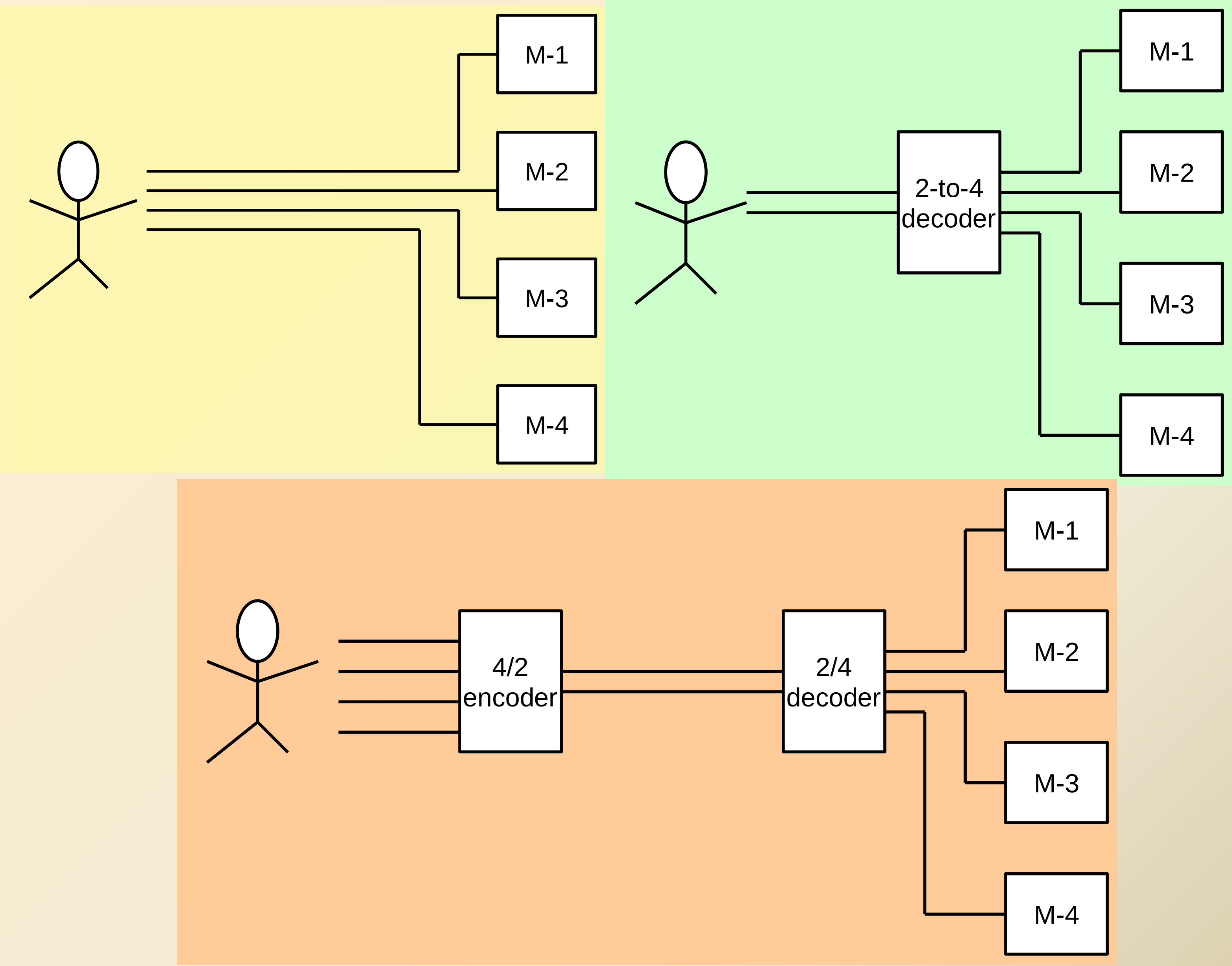
d_3d_2 \ d_1d_0	00	01	11	10
00		0		1
01	0			
11				
10	1			

$$A = \overline{d_2}\ \overline{d_0}$$

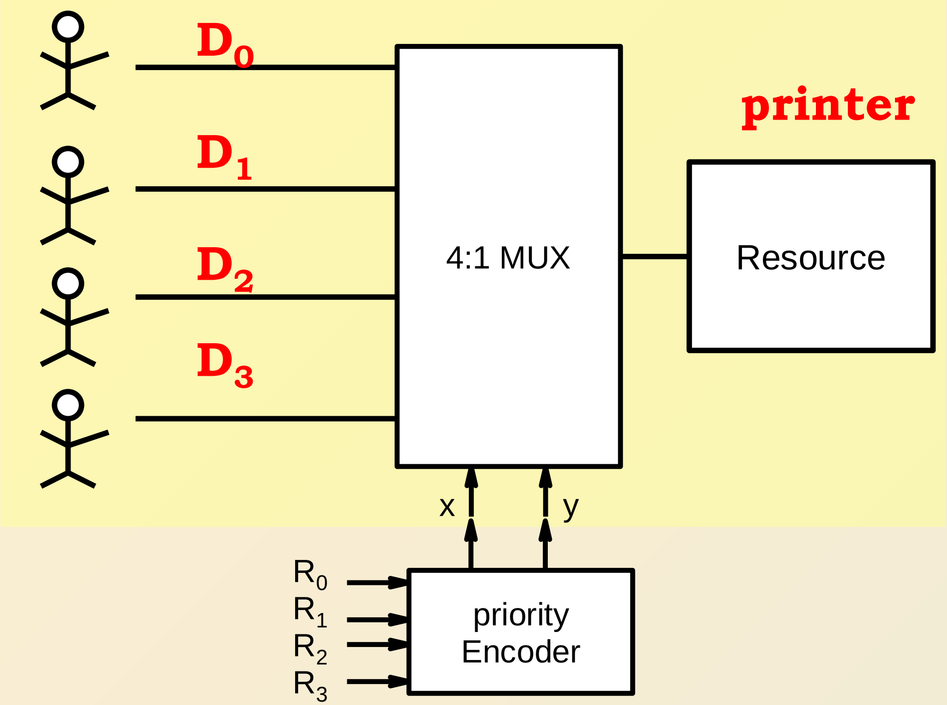


d_3d_2 \ d_1d_0	00	01	11	10
00		0		0
01	1			
11				
10	1			

$$B = \overline{d_1}\ \overline{d_0}$$


M-1
M-2
M-3
M-4
2-to-4
decoder
M-1
M-2
M-3
M-4
4/2
encoder
2/4
decoder
M-1
M-2
M-3
M-4

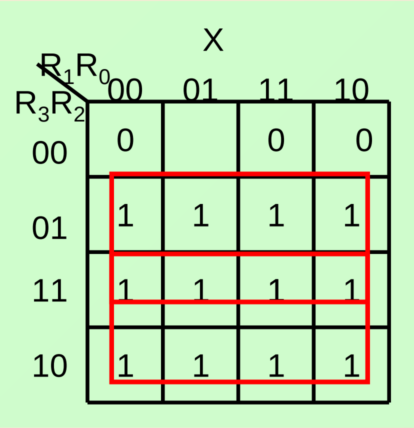
Priority Encoders



Priority is 3,2,1,0 with user 3 having the highest priority

X, Y have to be determined based on this priority order and the requests to use the resource.

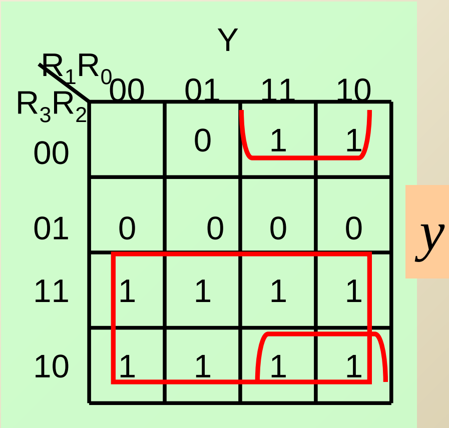
R_0	R_1	R_2	R_3	x	y
0	0	0	0	x	x
1	0	0	0	0	0
x	1	0	0	0	1
x	x	1	0	1	0
x	x	x	1	1	1



X

R_3R_2 \ R_1R_0	00	01	11	10
00	0		0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$$x = R_2 + R_3$$



Y

R_3R_2 \ R_1R_0	00	01	11	10
00		0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$$y = R_1\overline{R_2} + R_3$$

Gray Codes

decimal	Natural Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

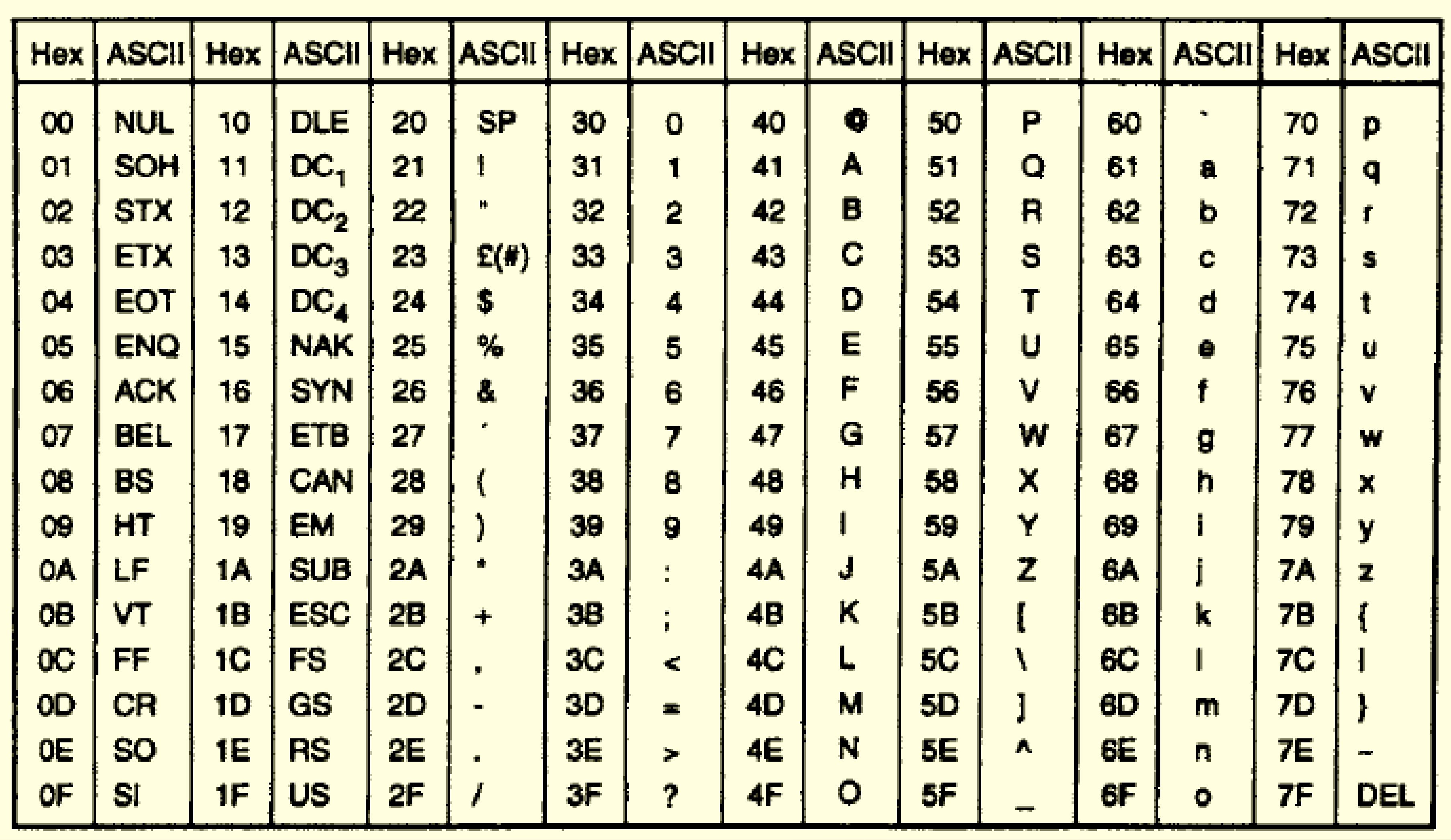
In the case of natural binary code:

0111-1111-1000 0111-0000-1000

In the case of Gray code, no such problem occurs.

1-bit change as one goes from one code word to the next.

ASCII: American Standard Code for information interchange



Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC$_1$	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC$_2$	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC$_3$	23	£(#)	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC$_4$	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

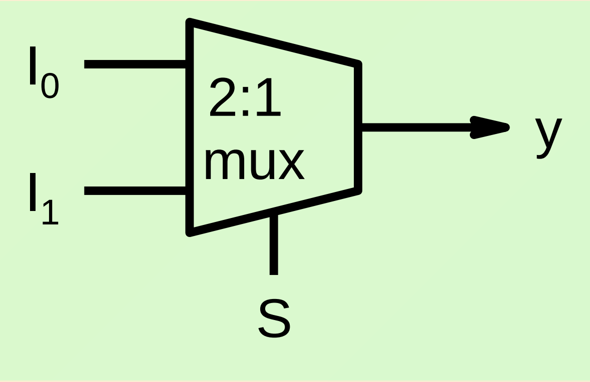
Parity

Extra bits are added to aid in error detection and correction

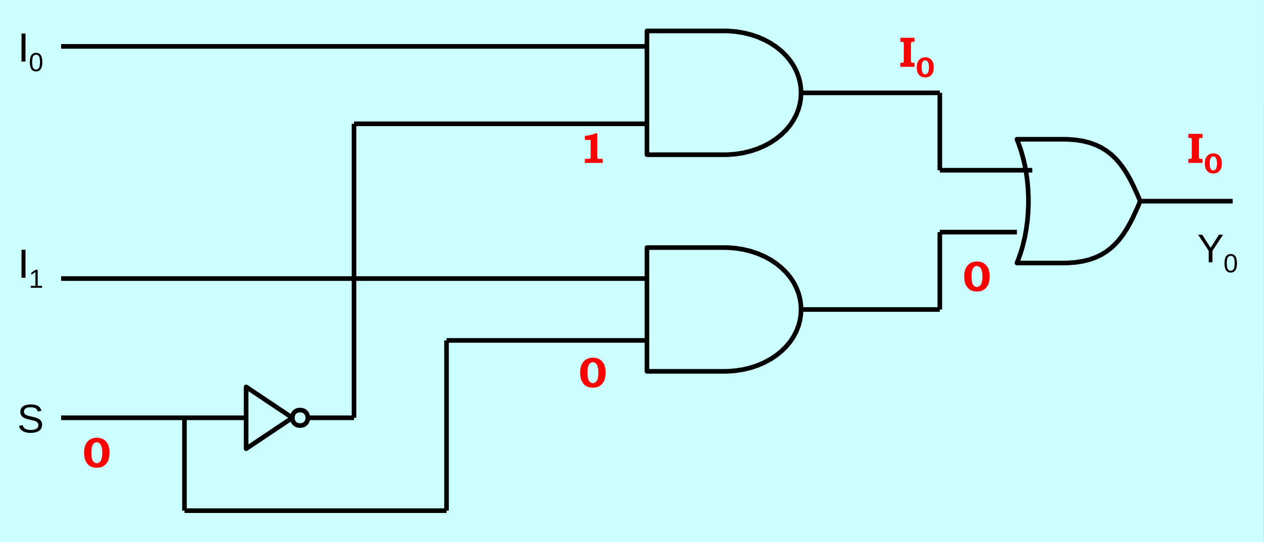
Decimal	Binary	Even parity	Odd parity
0	000	0000	0001
1	001	0011	0010
2	010	0101	0100
3	011	0110	0111
4	100	1001	1000
5	101	1010	1011
6	110	1100	1101
7	111	1111	1110

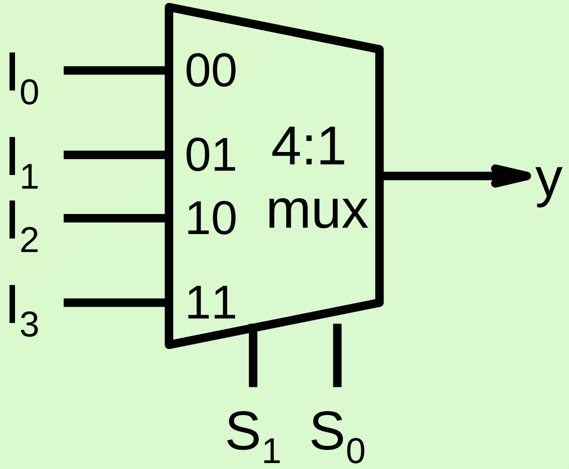
A 1-bit error changes the parity and thus can be detected

Multiplexers

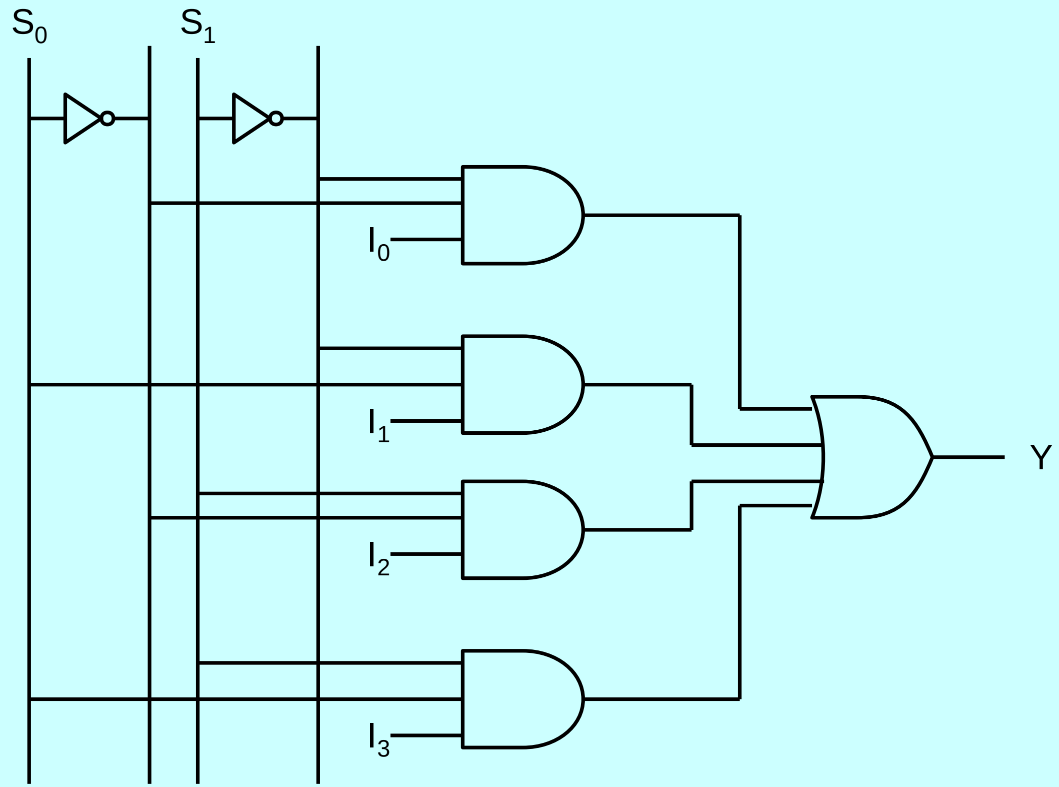


S	y
0	I_0
1	I_1

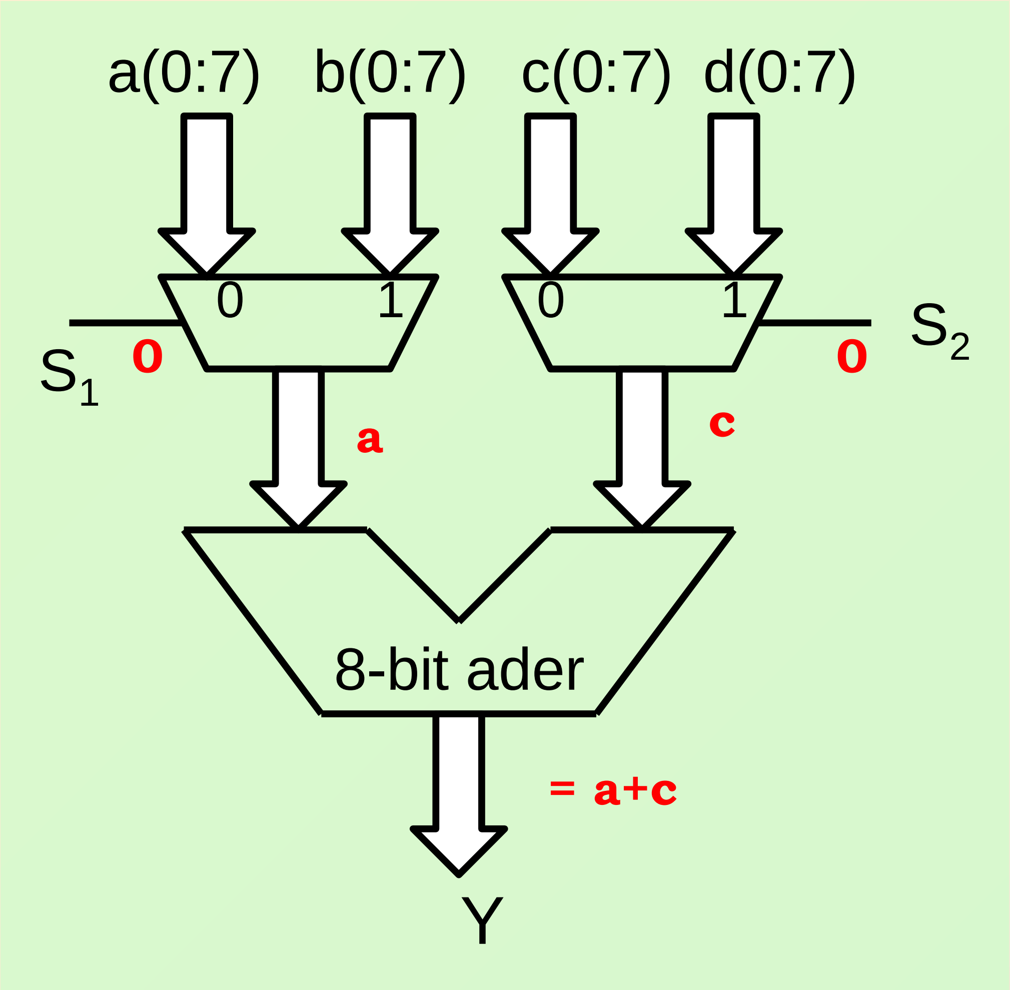




S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



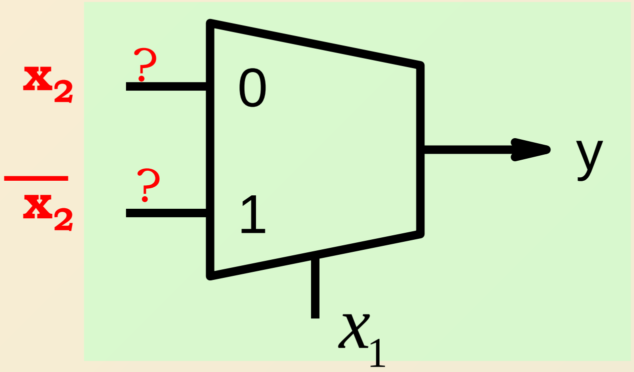
Mux is often used when resources have to be shared



S_1	S_0	y =
0	0	a+c
0	1	a+d
1	0	b+c
1	1	b+d

Implementing Boolean expressions using Multiplexers

$$y = x_1 \overline{x_2} + \overline{x_1} x_2$$



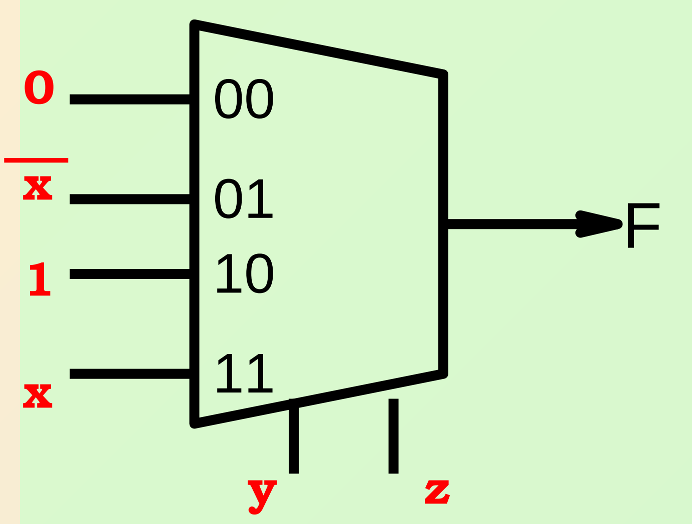
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$y = x_2$ when $x_1 = 0$

$y = \overline{x_2}$ when $x_1 = 1$

$$F(x, y, z) = \sum(1, 2, 6, 7)$$

A 3 variable function can be implemented with a 4:1 mux with 2 select lines



x	y	z	*F*
0	0	0	0
1	0	0	0
0	0	1	1
1	0	1	0
0	1	0	1
1	1	0	1
0	1	1	0
1	1	1	1

F = 0 when yz = 00

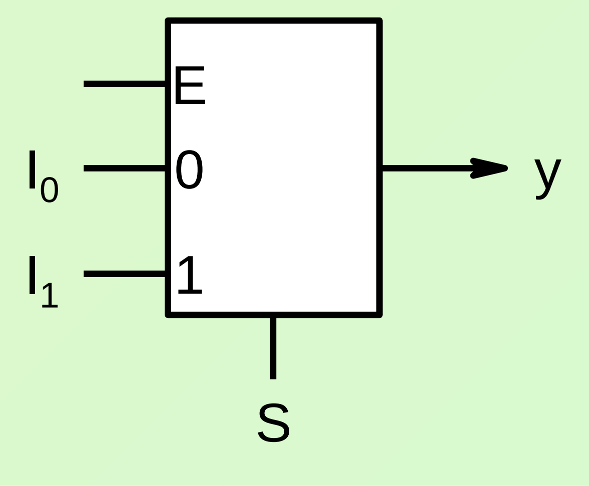
F = $\overline{x}$ when yz =01

F = 1 when yz = 10

F = x when yz = 11

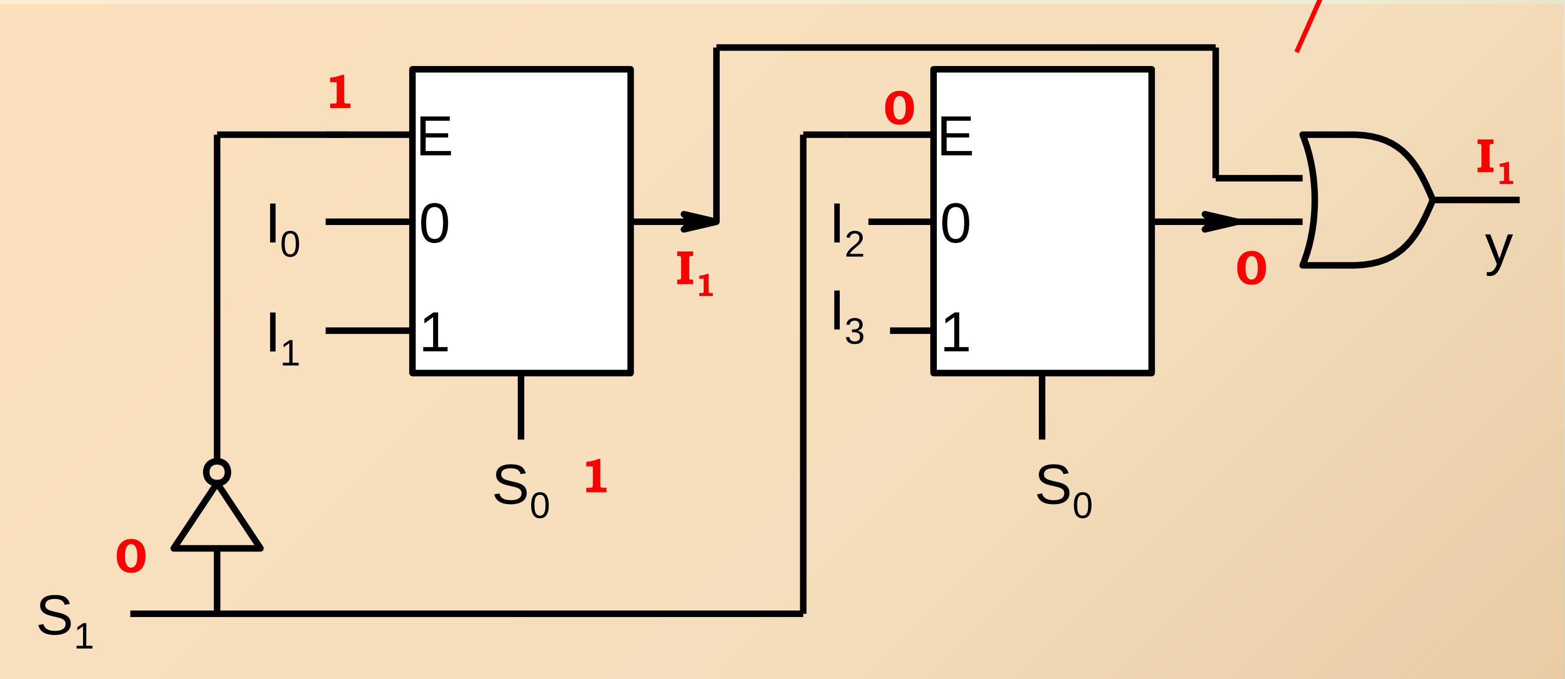
Mux is more efficient way of implementing combinational circuits as compared to decoders.

Mux. expansion

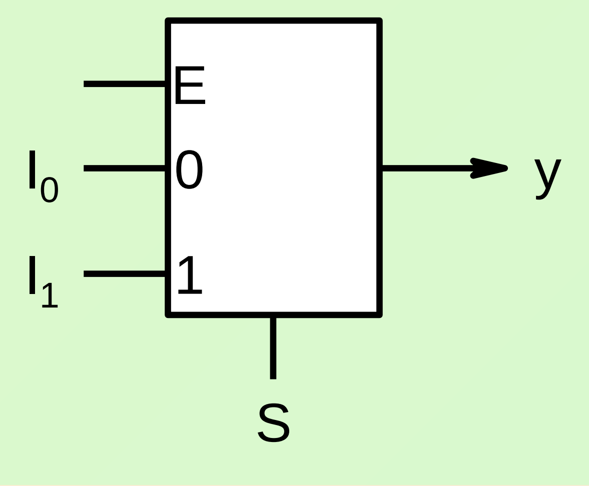


E	S	y
0	x	0
1	0	I_0
1	1	I_1

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

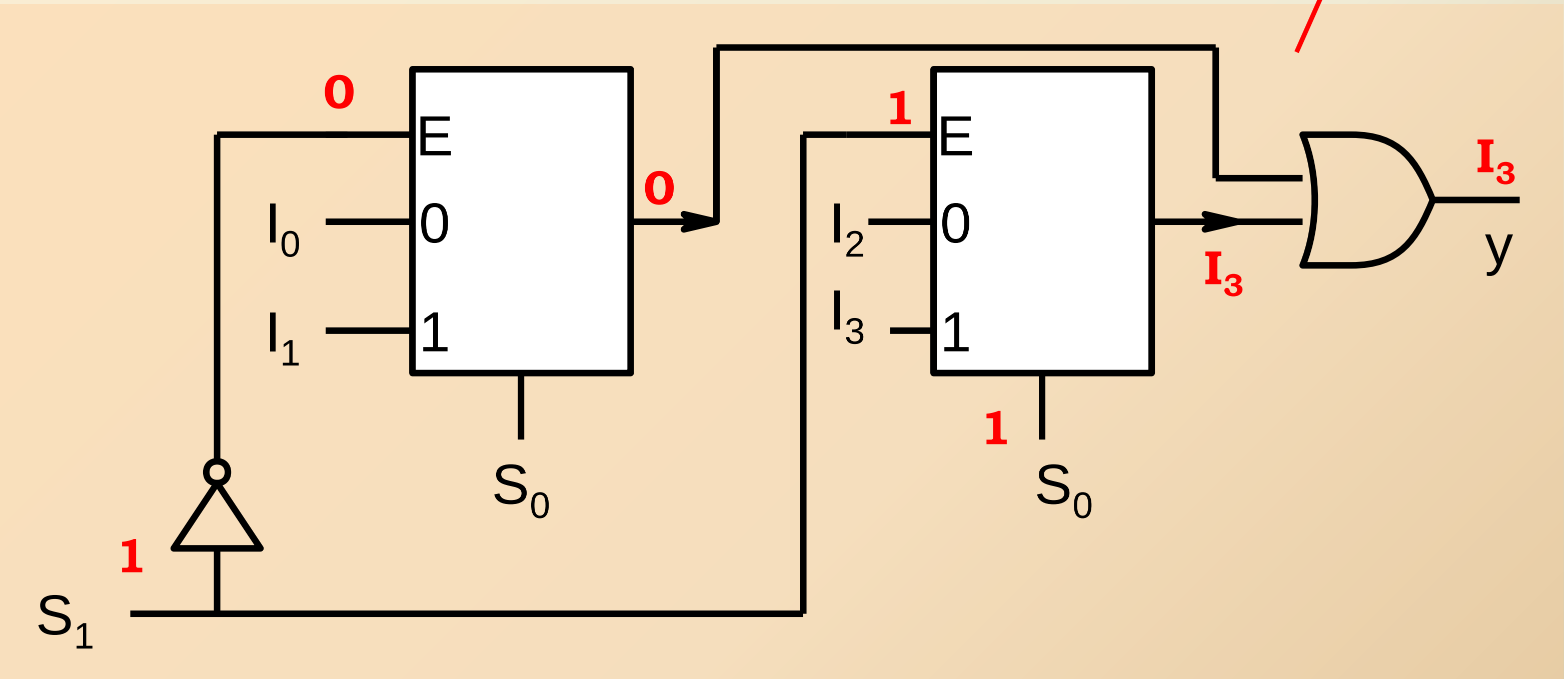


Mux. expansion

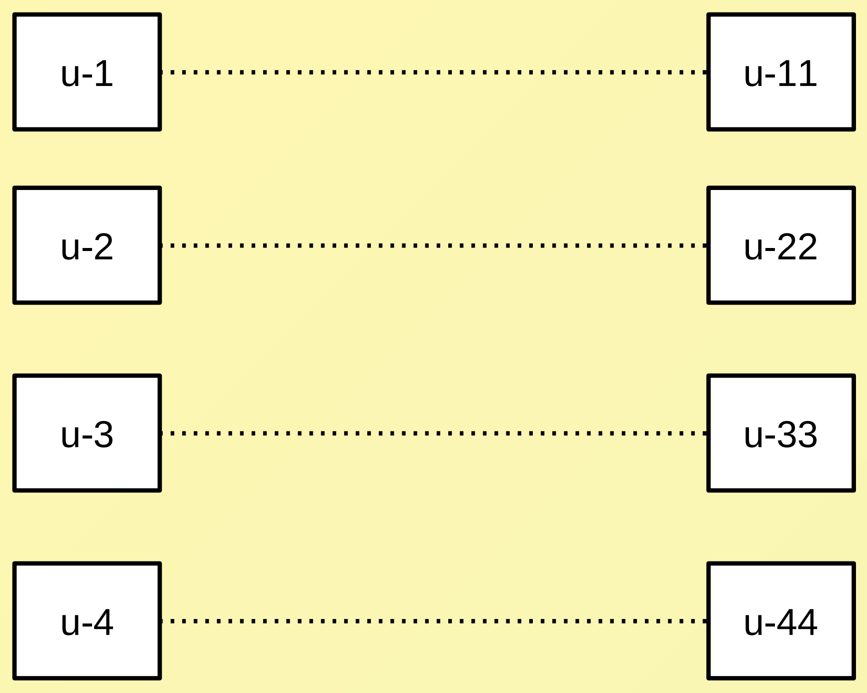


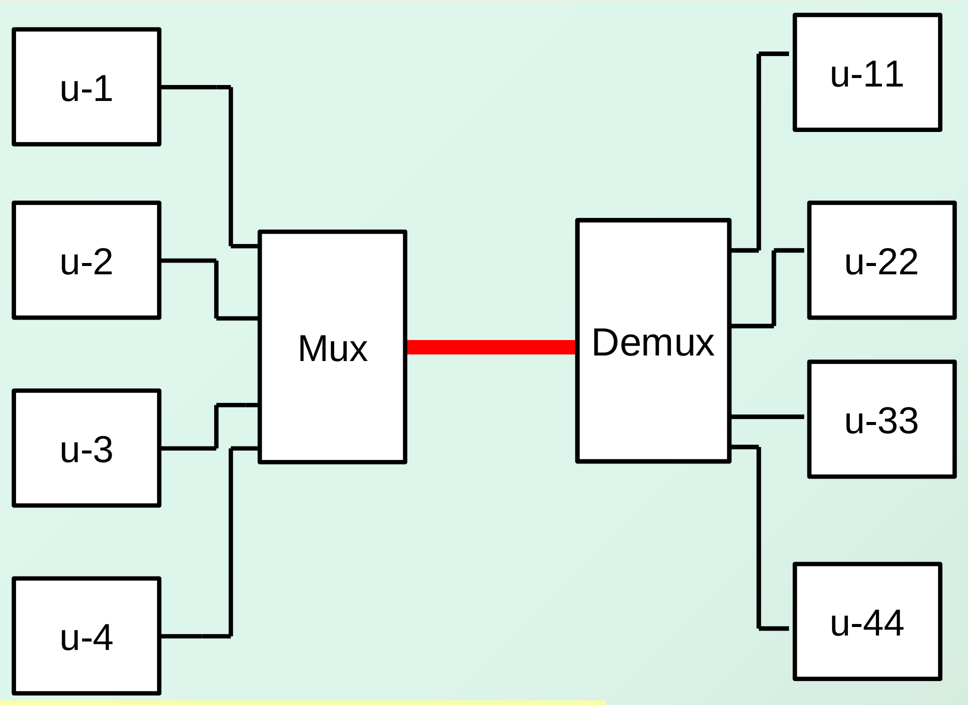
E	S	y
0	x	0
1	0	I_0
1	1	I_1

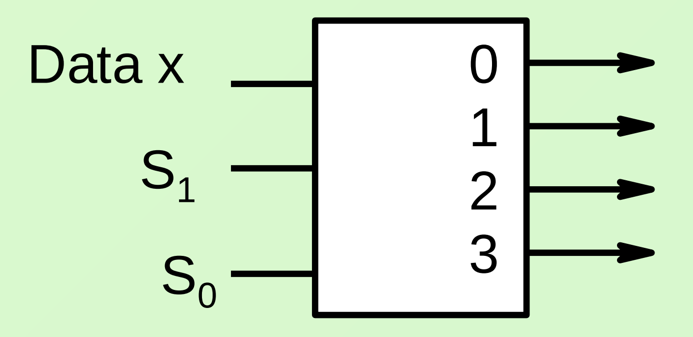
S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



DeMultiplexer

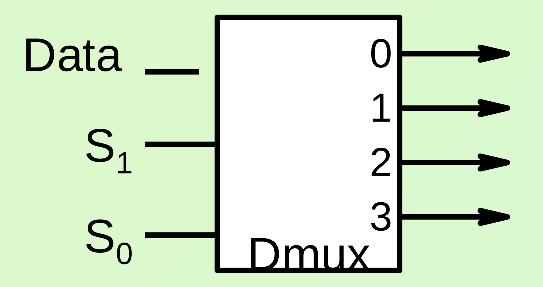




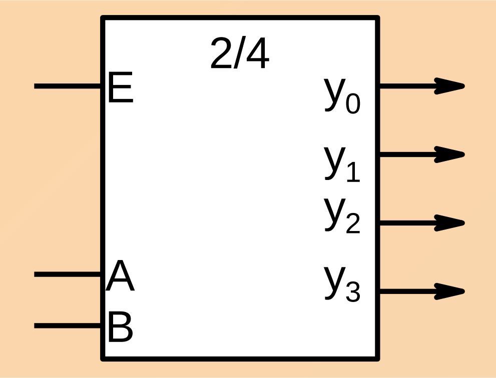


S_1	S_0	y_0	y_1	y_2	y_3
0	0	x	0	0	0
0	1	0	x	0	0
1	0	0	0	x	0
1	1	0	0	0	x

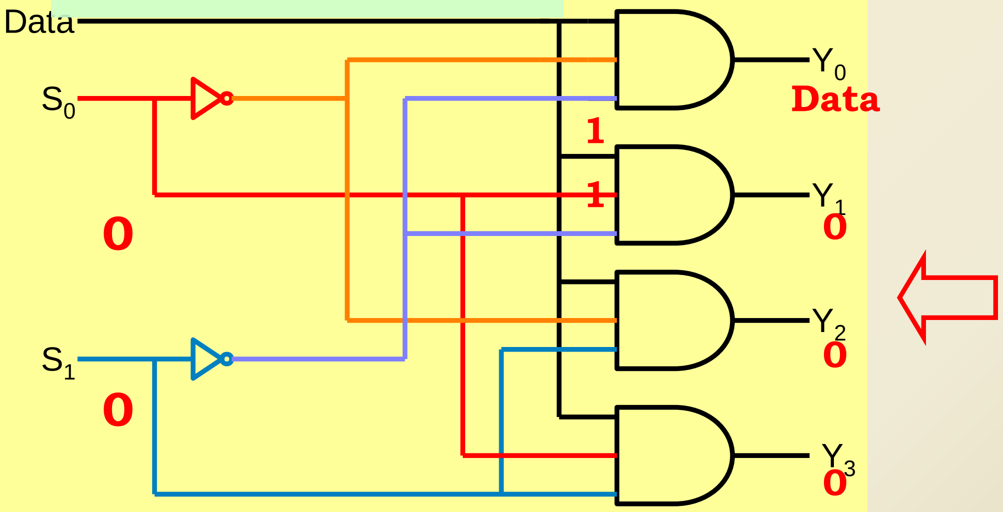
Demultiplexer is very much like a decoder

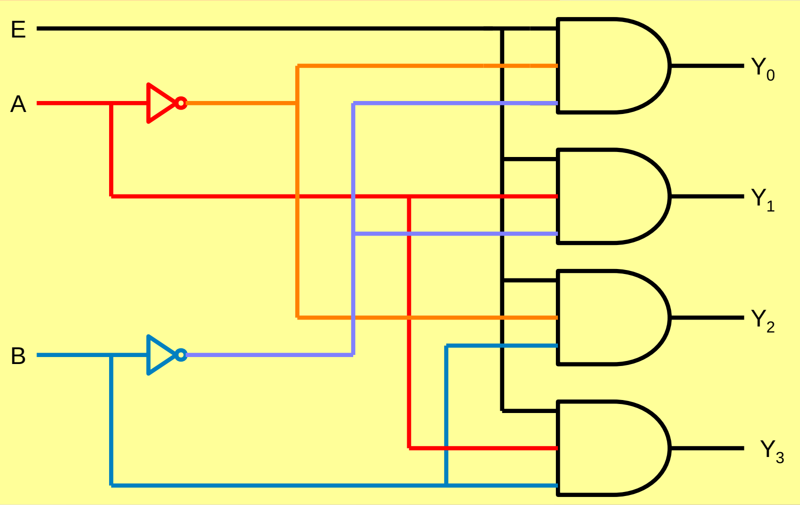


S_1	S_0	y_0	y_1	y_2	y_3
0	0	x	0	0	0
0	1	0	x	0	0
1	0	0	0	x	0
1	1	0	0	0	x

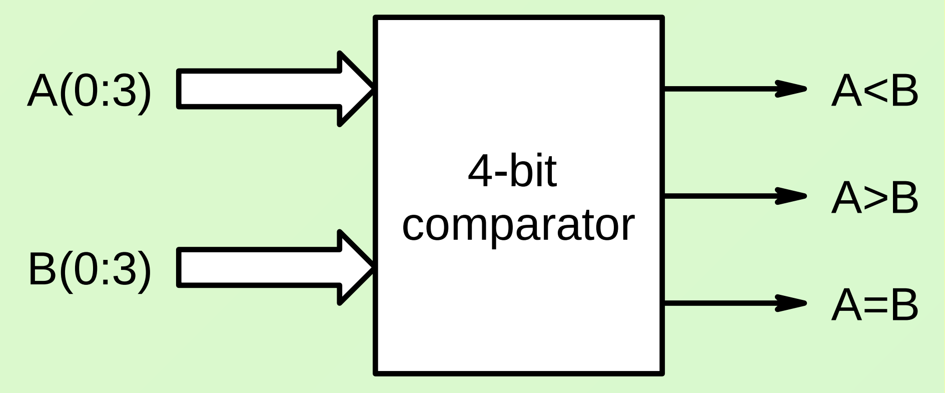


E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

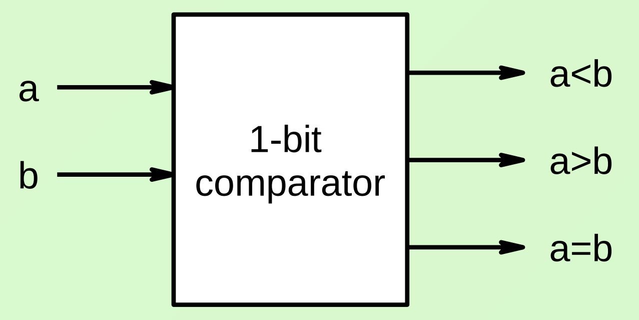


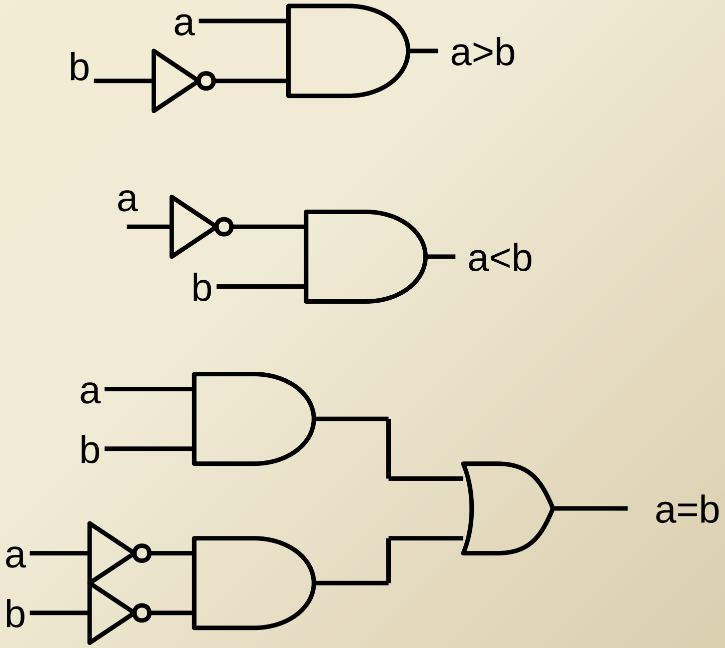


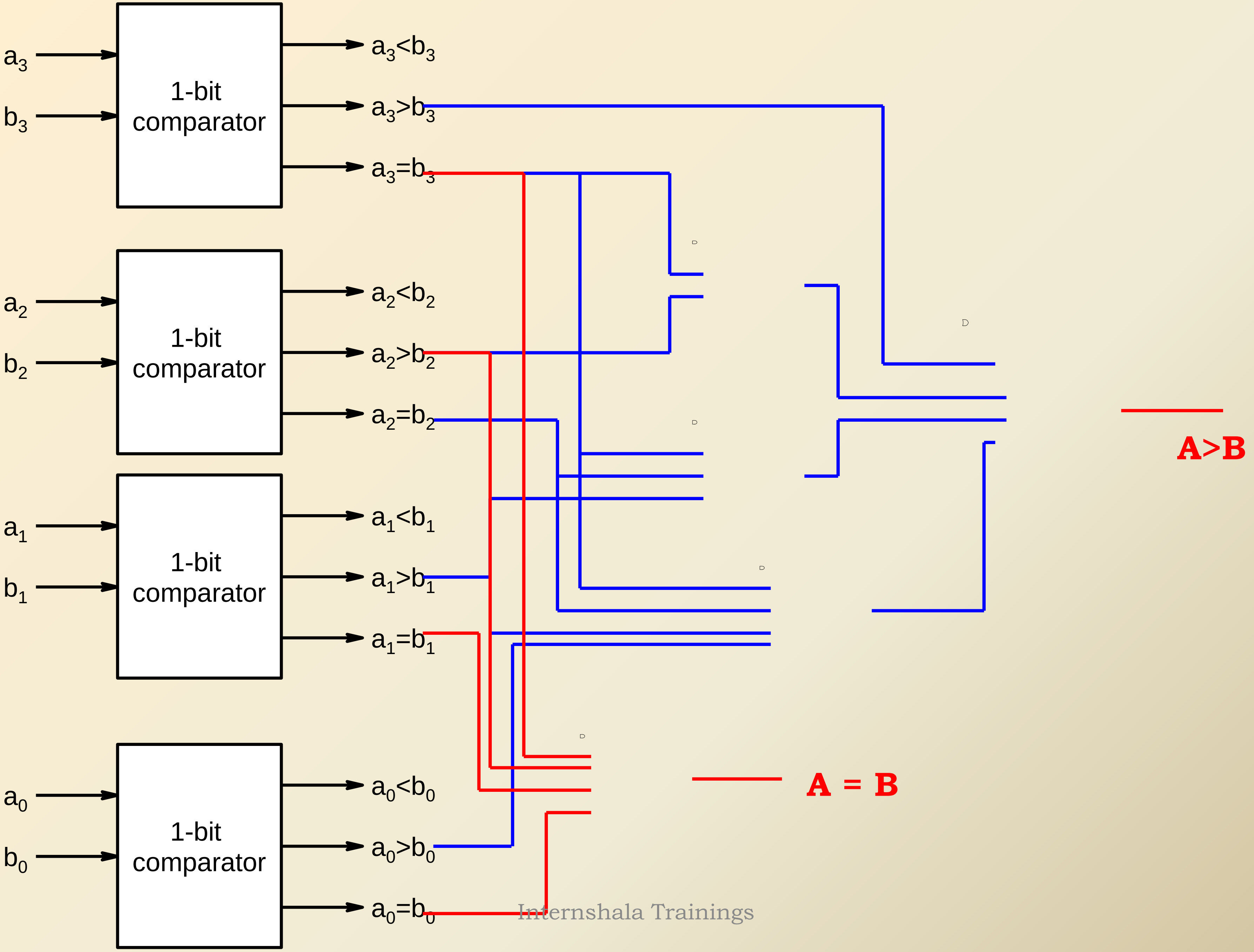
Comparator



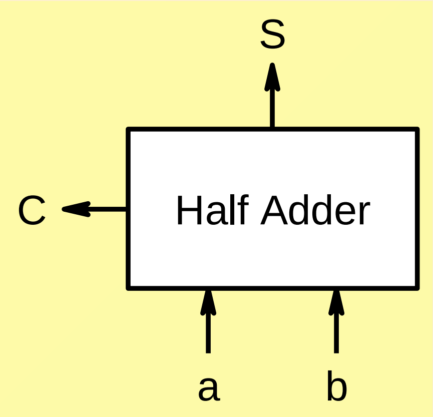
$A_3A_2A_1A_0$	$B_3B_2B_1B_0$	A<B	A>B	A=B
0000	0000	0	0	1
0000	0001	1	0	0
0001	0000	0	1	0
⋮	⋮	⋮	⋮	⋮



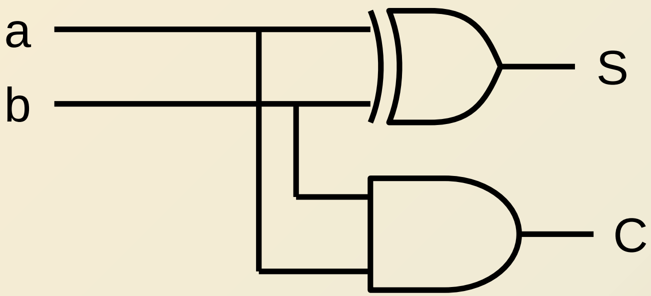



a_3
b_3
1-bit comparator
$a_3<b_3$
$a_3>b_3$
$a_3=b_3$
a_2
b_2
1-bit comparator
$a_2<b_2$
$a_2>b_2$
$a_2=b_2$
a_1
b_1
1-bit comparator
$a_1<b_1$
$a_1>b_1$
$a_1=b_1$
a_0
b_0
1-bit comparator
$a_0<b_0$
$a_0>b_0$
$a_0=b_0$
A>B
A = B

Adder/Subtractor



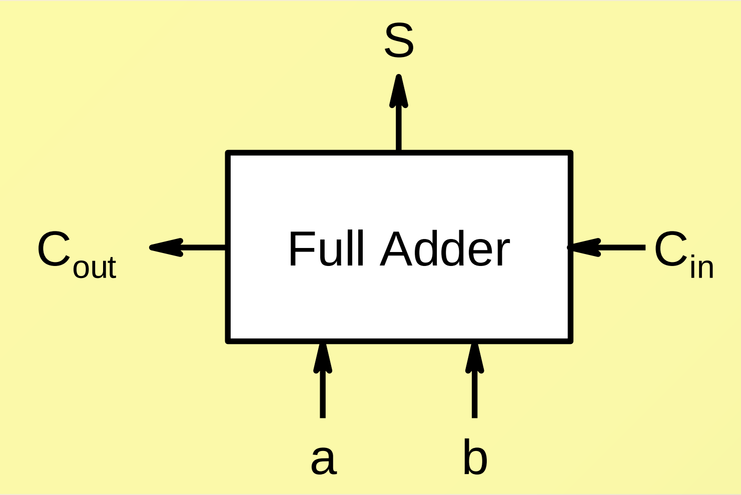
a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S = \bar{a}.b + a.\bar{b}\,; C = a.b$$

a	b	C_{in}	S	C_{out}
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

$$\begin{array}{r} 1 \\ 1\,1\,1 \\ \underline{1\,1\,0} \\ \underline{1\,1\,1\,1} \end{array}$$



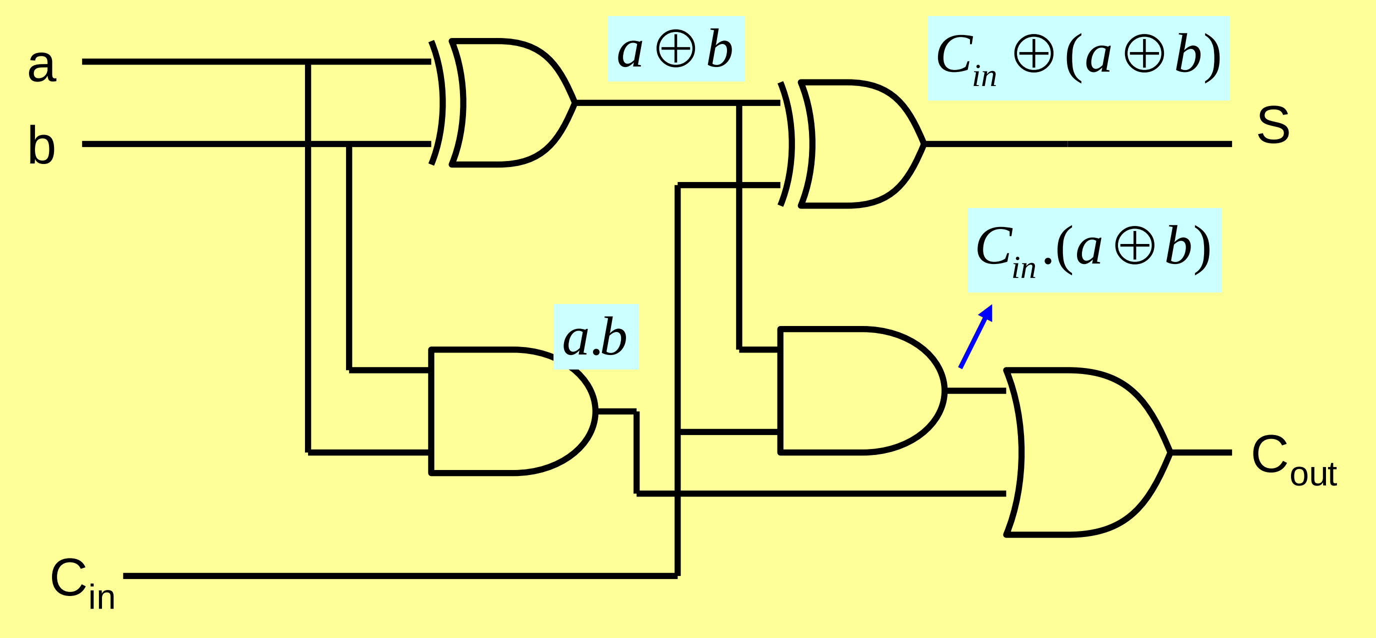
$$S = \bar{a}.\bar{b}.c_{in} + \bar{a}.b.\overline{c_{in}} + a.\bar{b}.\overline{c_{in}} + a.b.c_{in}\,; C_{out} = a.b + a.c_{in} + b.c_{in}$$

$$S = \bar{a}.\bar{b}.c_{in} + \bar{a}.b.\overline{c_{in}} + a.\bar{b}.\overline{c_{in}} + a.b.c_{in}$$

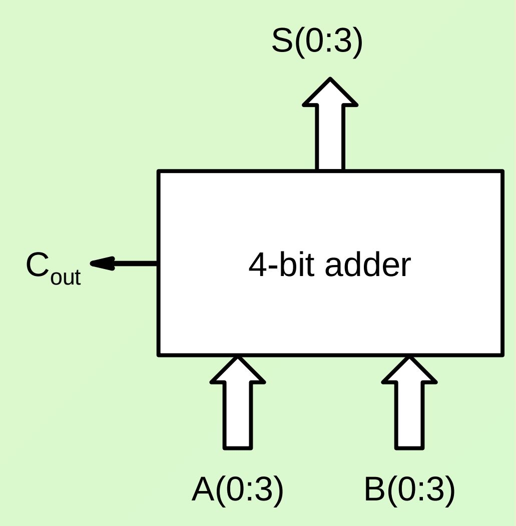
$$S = C_{in} \oplus (a \oplus b)$$

$$C_{out} = a.b + a.C_{in} + b.C_{in}$$

$$C_{out} = C_{in}(a.\bar{b} + \bar{a}.b) + a.b = C_{in}.(a \oplus b) + a.b$$

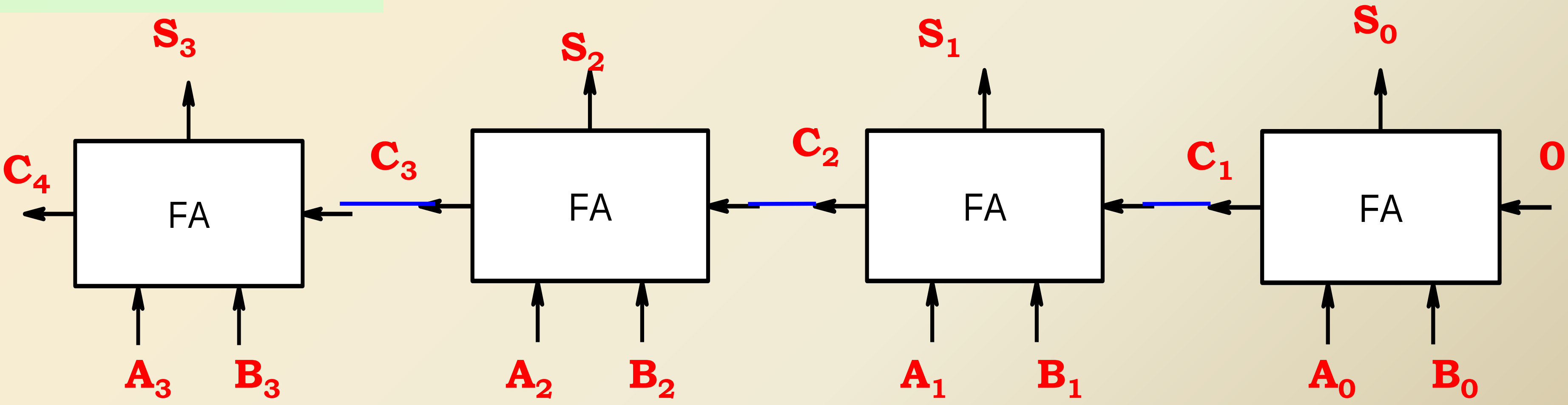


4-bit Adder



$A_3A_2A_1A_0$	$B_3B_2B_1B_0$	$S_3S_2S_1S_0$	C_{out}
0000	0000	0000	1
0000	0001	0001	0
0001	0000	0001	0
⋮	⋮	⋮	⋮

$$\begin{array}{ccccc} & C_3 & C_2 & C_1 & \\ & A_3 & A_2 & A_1 & A_0 \\ & B_3 & B_2 & B_1 & B_0 \\ \hline C_4 & S_3 & S_2 & S_1 & S_0 \\ \hline \end{array}$$



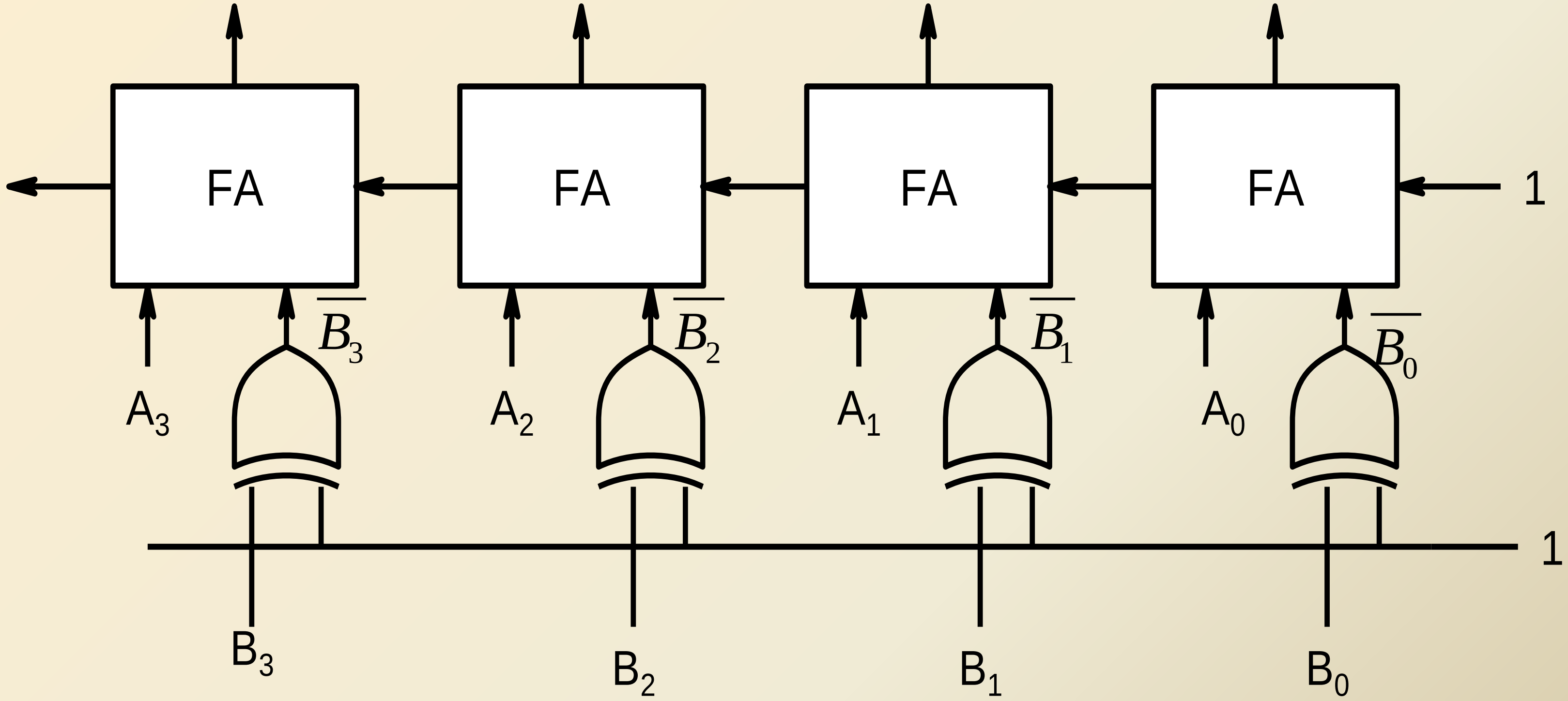
Ripple Carry Adder (20 gate circuit)

Subtraction

A - B = A + 2's complement of B

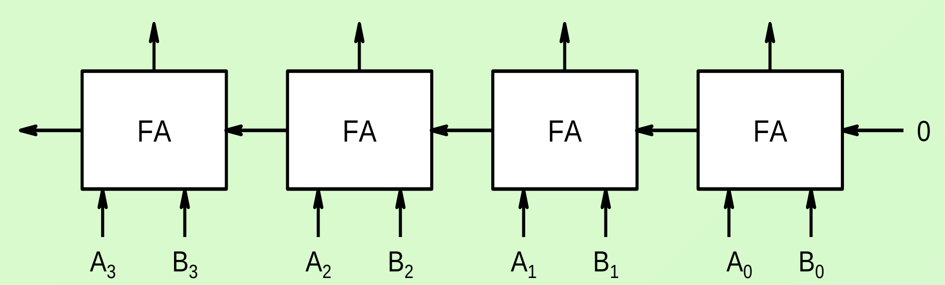
A - B = A + 1's complement of B+1

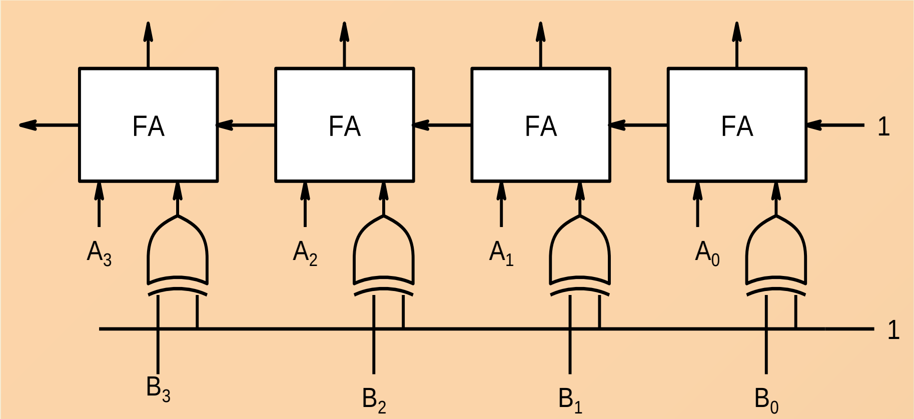
$$\boxed{A - B = A \overline{+ B}+1}$$

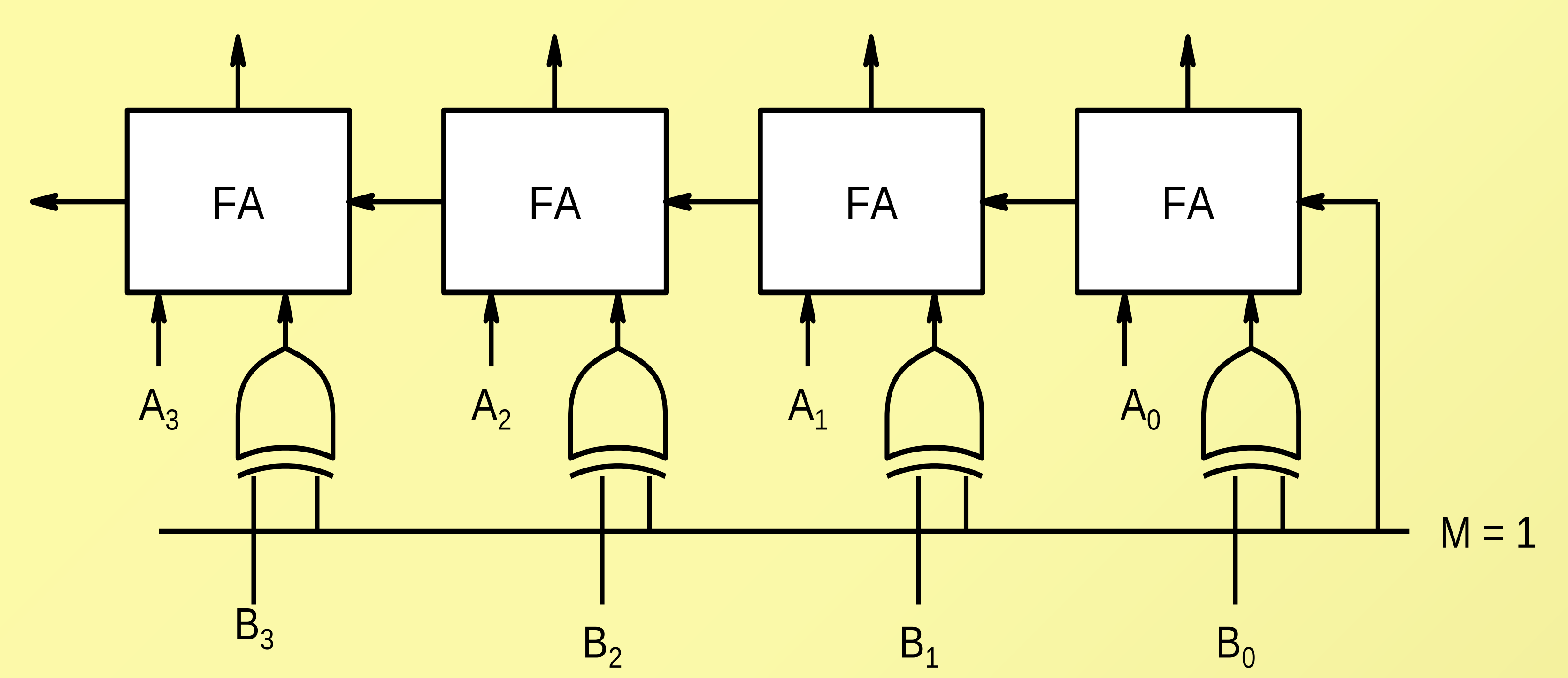


One needs add a circuit for predicting errors resulting from overflow

Adder/Subtractor







Multiplier

B_1 B_0 → multiplicand

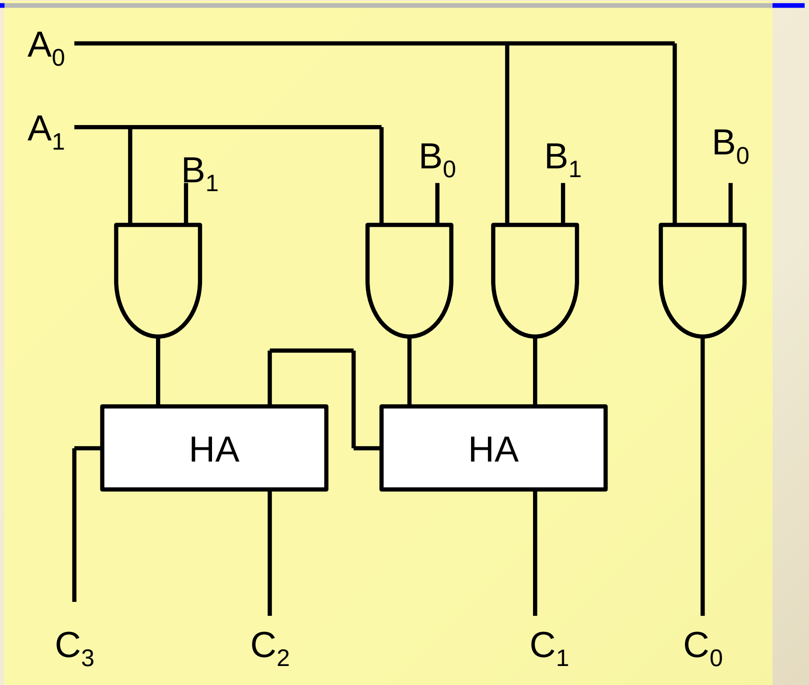
A_1 A_0 → multiplier

A_0B_1 A_0B_0

A_1B_1 A_1B_0

→ Partial products

C3 C2 C_1 C_0

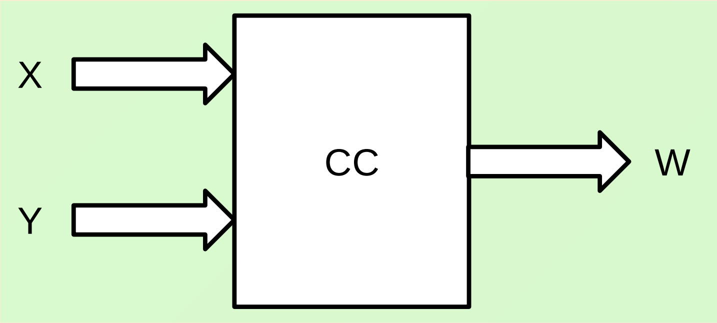


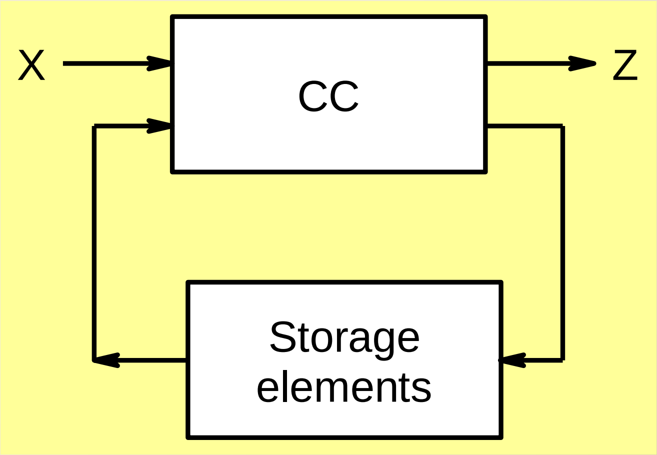
SEQUENTIAL CIRCUITS

Digital Circuits

Combinational Circuits

Sequential Circuits

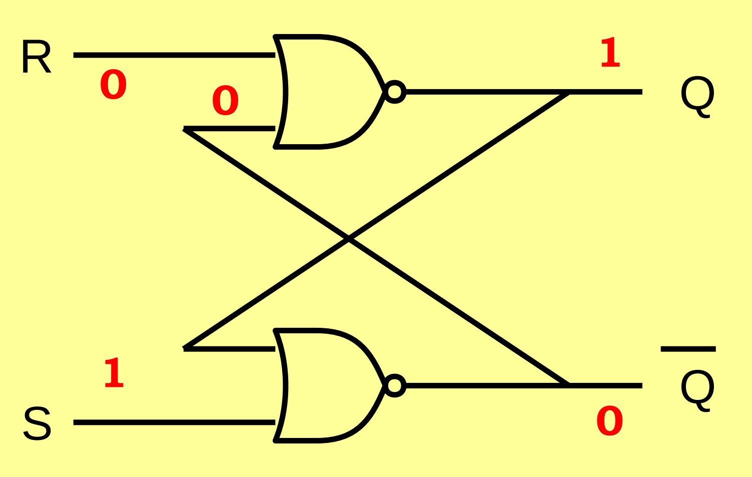




Output is determined by current values of inputs only.

Output is determined in general by current values of inputs and past values of inputs/outputs as well.

NOR SR Latch

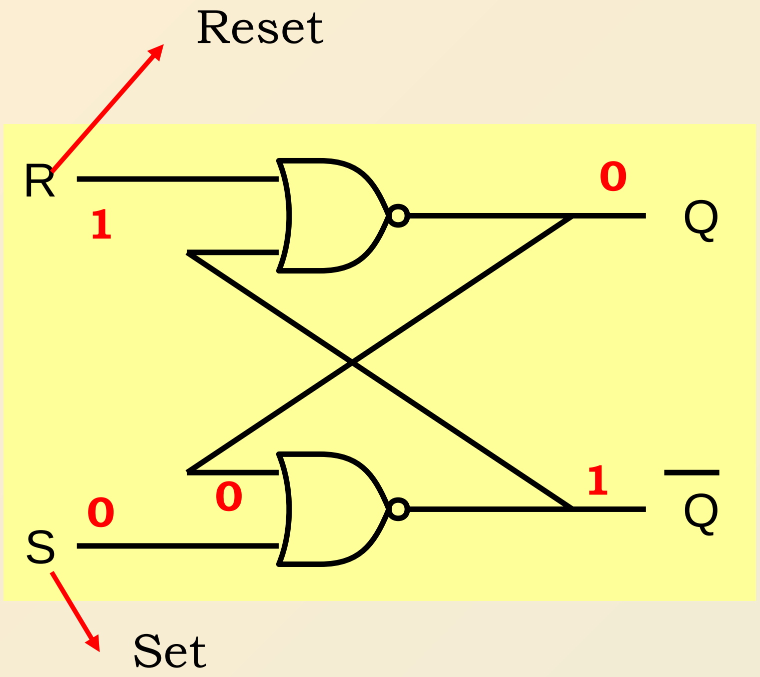


$Q = 1; \overline{Q} = 0$ Set State

$Q = 0; \overline{Q} = 1$ Re set State

S	R	Q	$\overline{Q}$	State
1	0	1	0	SET

NOR SR Latch

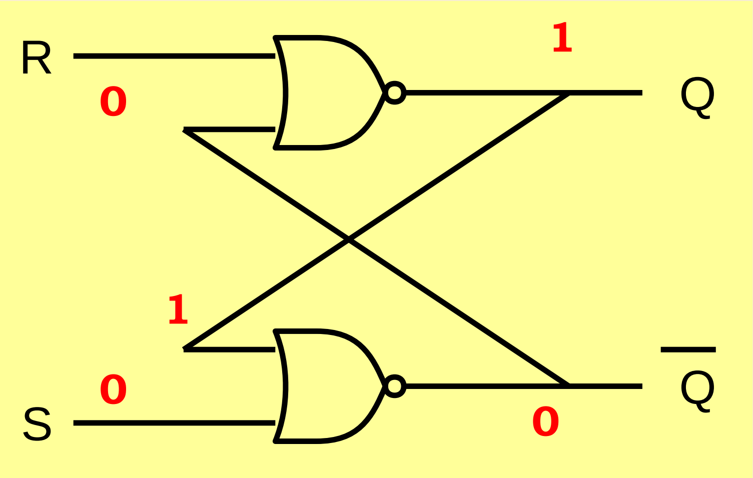


$Q = 1; \overline{Q} = 0$ Set State

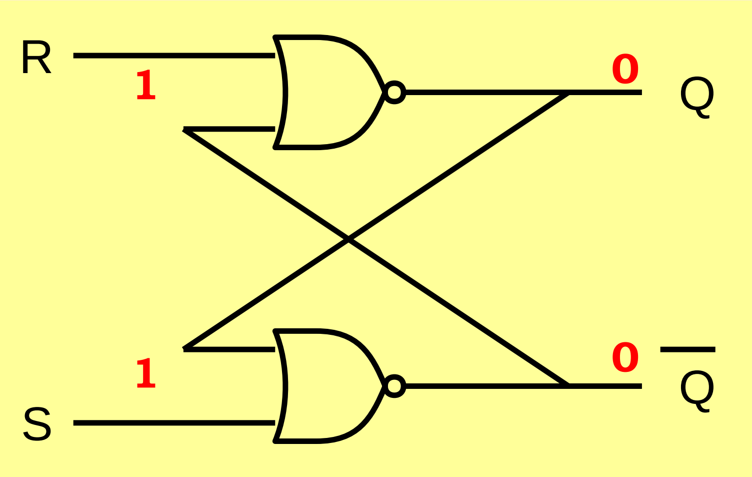
$Q = 0; \overline{Q} = 1$ Re set State

S	R	Q	$\overline{Q}$	State
1	0	1	0	SET
0	1	0	1	RESET

HOLD State

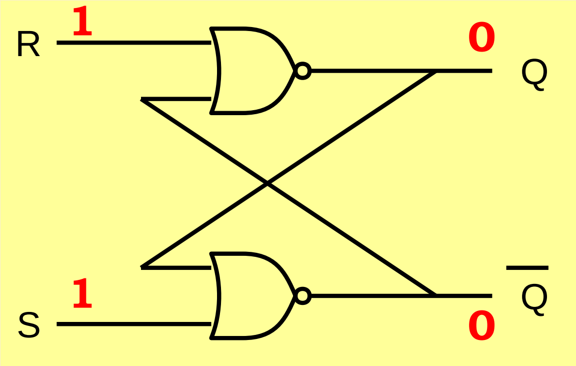


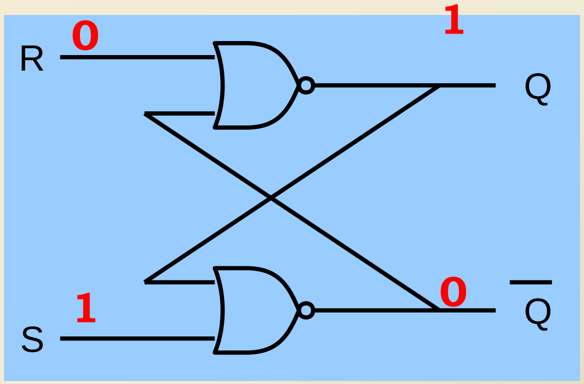
S	R	Q	$\overline{Q}$	State
1	0	1	0	SET
0	1	0	1	RESET
0	0	Q	$\overline{Q}$	HOLD
1	1	0	0	INVALID

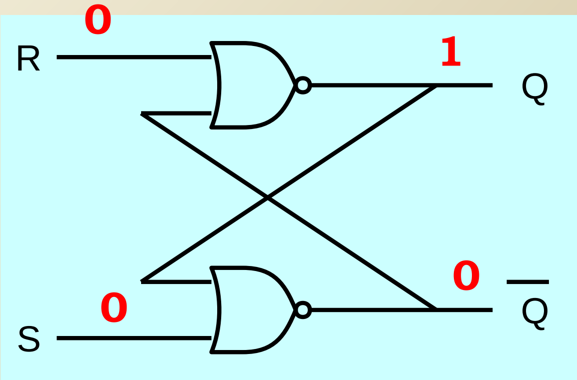


Both the outputs are well defined and 0. the first problem is that we do not get complementary output.

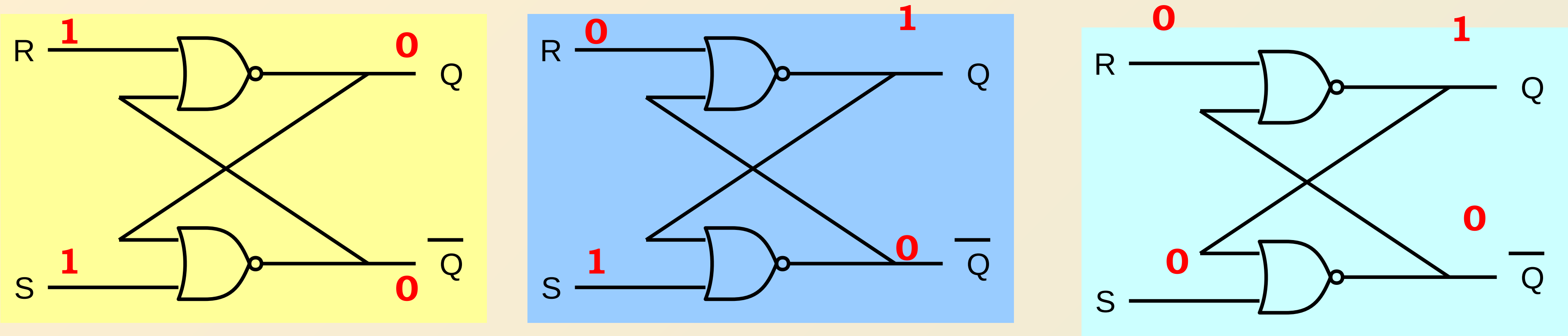
A more serious problem occurs when we switch the latch to the hold state by changing RS from 11 → 00 . Suppose the inputs do not change simultaneously and we get the situation 11 → 01^* → 00





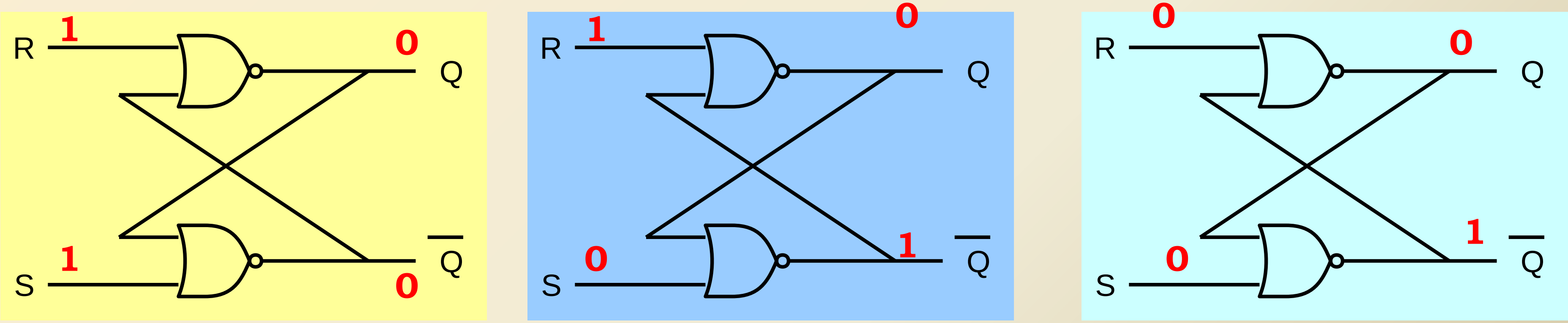


Q = 1



Q = 1

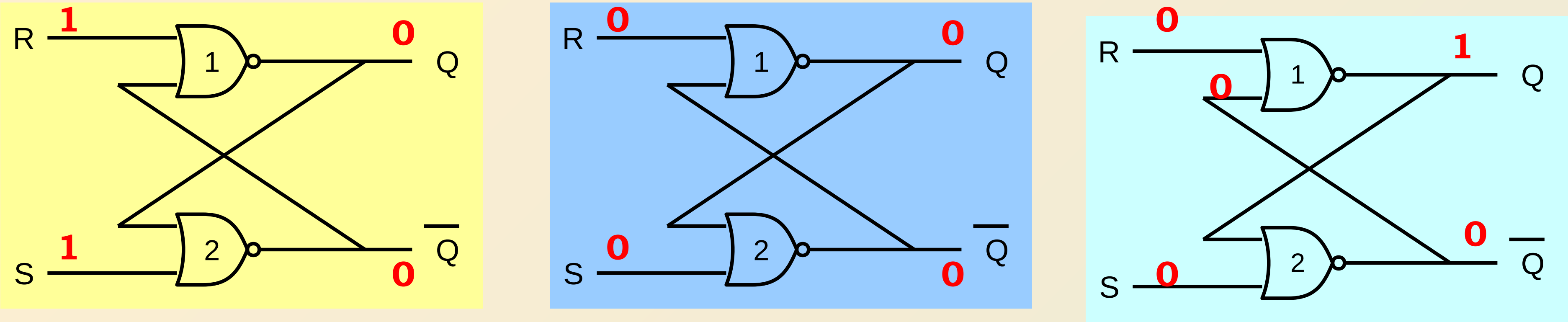
Suppose the inputs change as RS = 11 → 10* → 00



Q = 0

So although output is well defined when we apply RS = 11, it becomes unpredictable once we switch the latch to hold state by applying RS = 00. That is why RS = 11 is not used as an input combination.

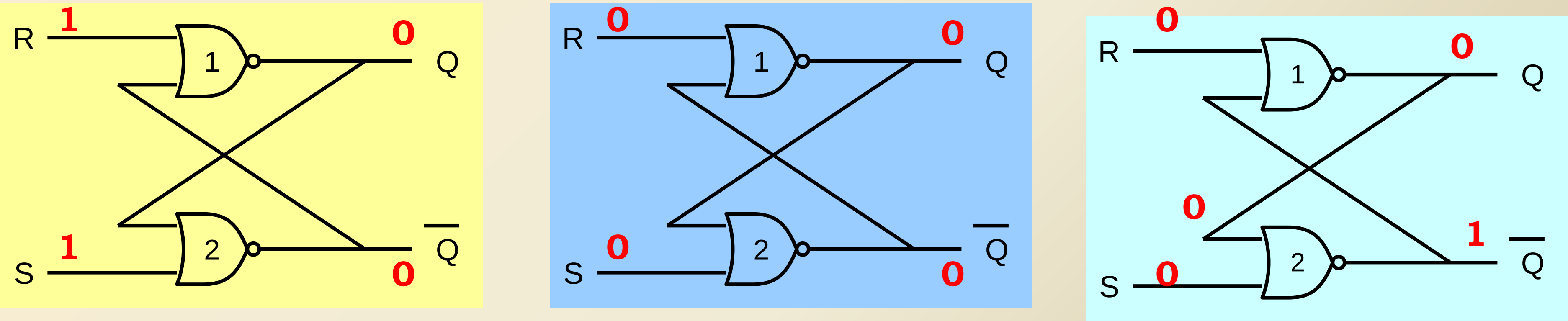
The error can occur also due to unequal gate delays.



Suppose gate-1 is faster

Q = 1

On the other hand suppose that gate-2 is faster.

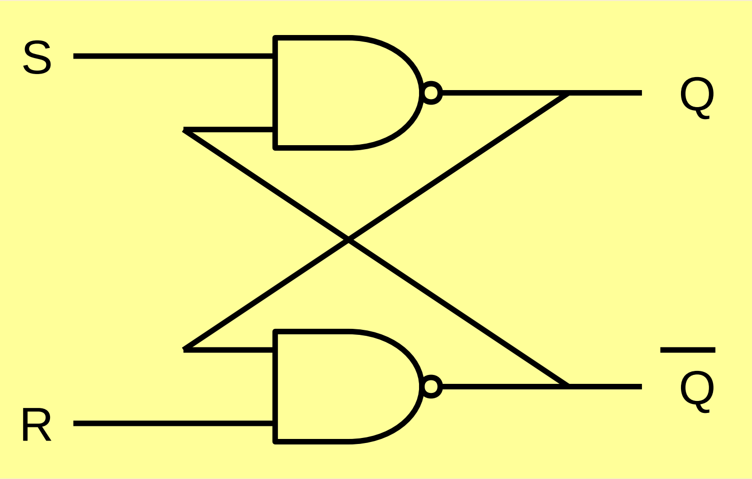


Again the output is unpredictable in general

Q = 0

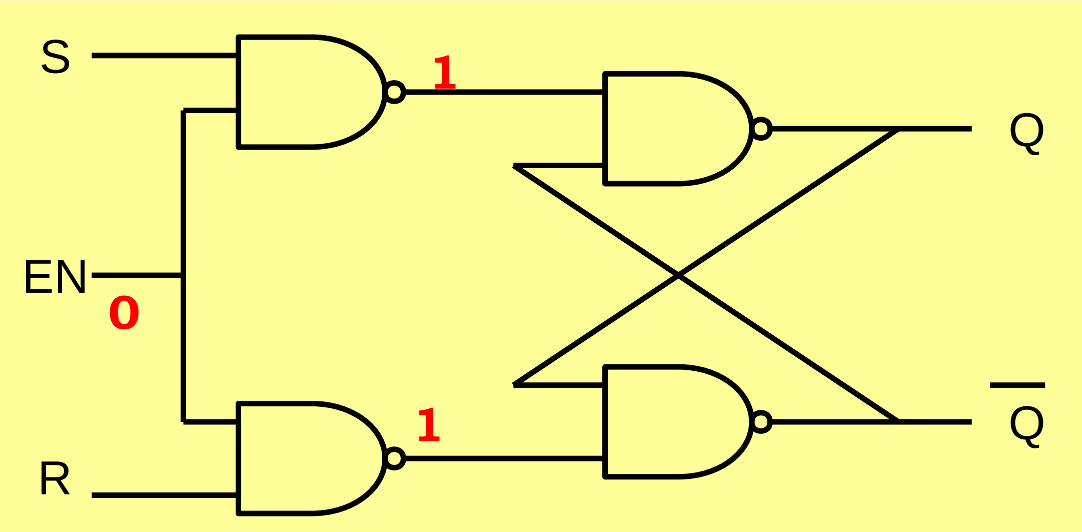
Internshala Trainings

NAND Latch

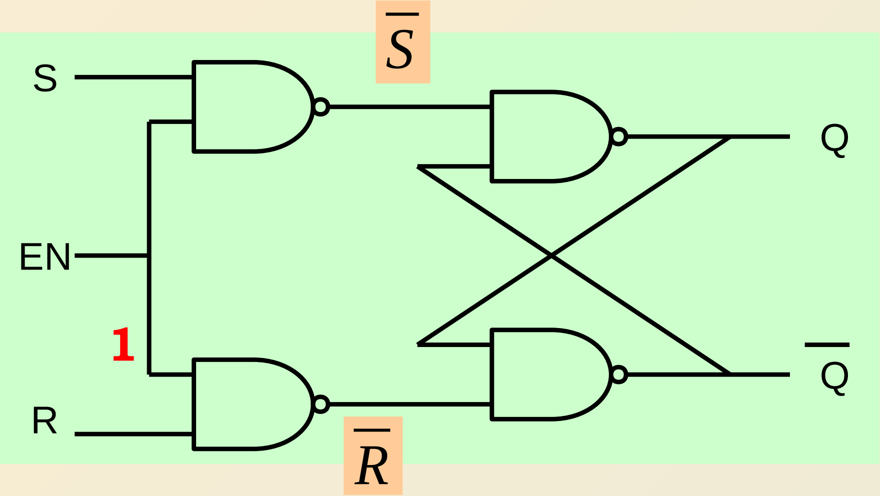


S	R	Q	$\overline{Q}$	State
0	1	1	0	SET
1	0	0	1	RESET
1	1	Q	$\overline{Q}$	HOLD
0	0	1	1	INVALID

RS NAND Latch with Enable

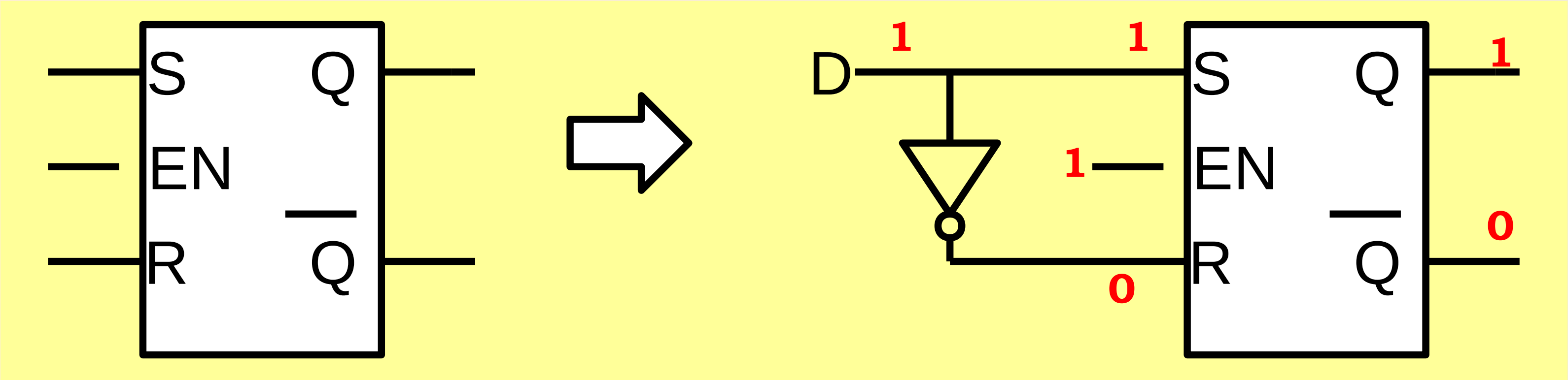


Hold State

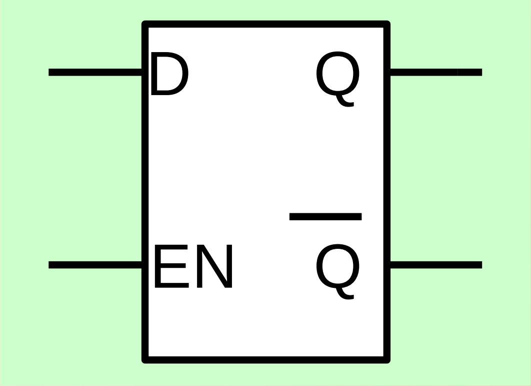


Enable	S R	Q $\overline{Q}$	State
0	x x	Q $\overline{Q}$	Hold
1	1 0	1 0	Set
1	0 1	0 1	Reset
1	0 0	Q $\overline{Q}$	Hold
1	1 1	0 0	Invalid

D latch

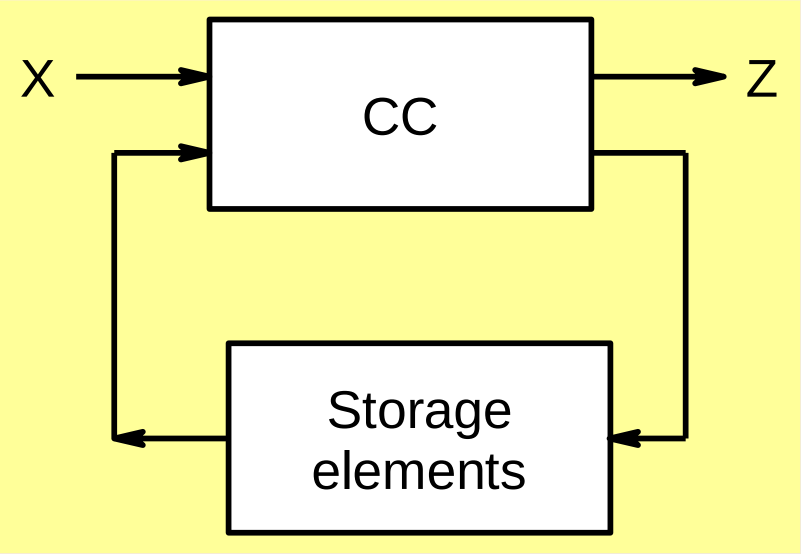


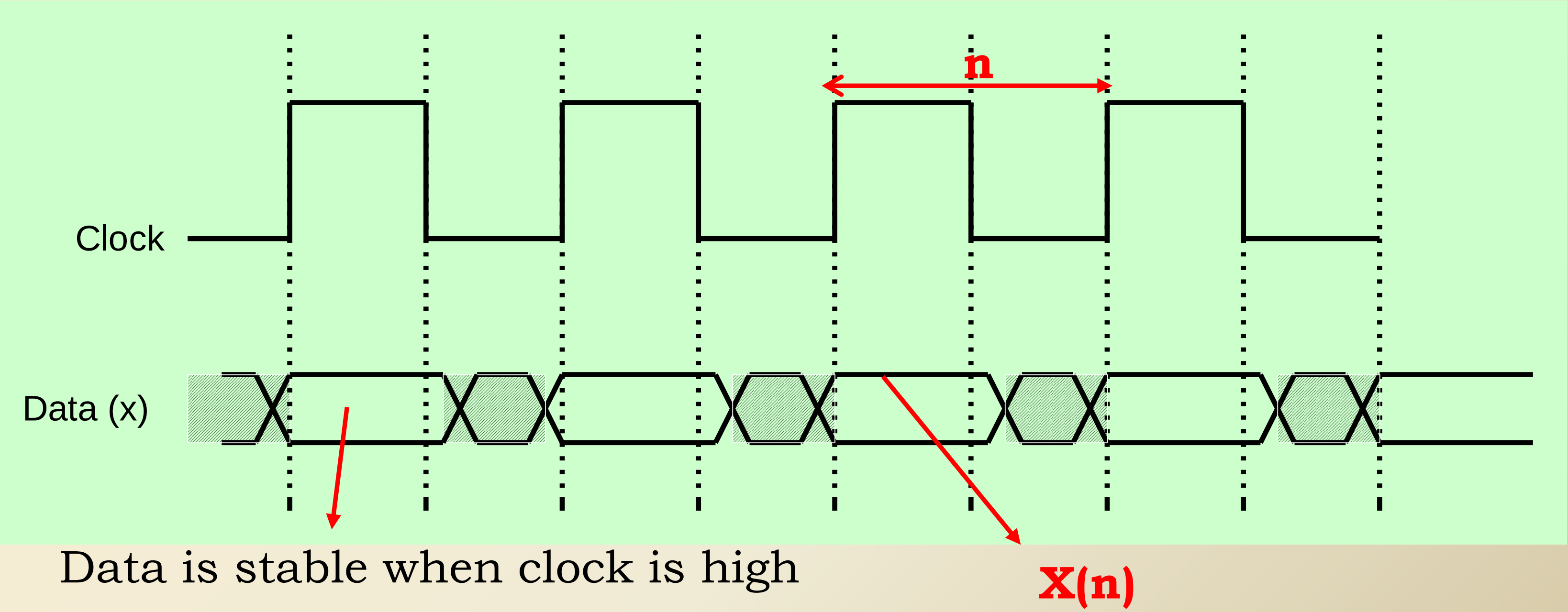
Enable	S R	Q $\overline{Q}$	State
0	x x	Q $\overline{Q}$	Hold
1	1 0	1 0	Set
1	0 1	0 1	Reset
1	0 0	Q $\overline{Q}$	Hold
1	1 1	0 0	Invalid



If EN = 1 then Q = D otherwise the latch is in Hold state

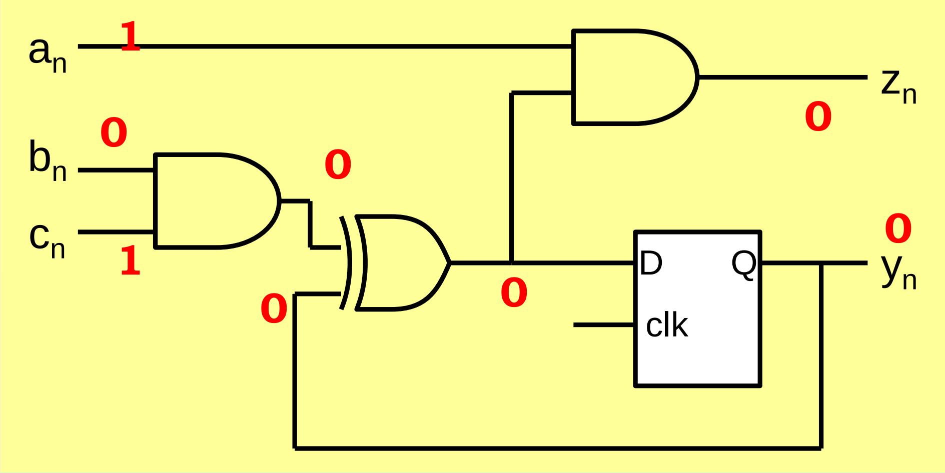
Synchronous Sequential Circuits

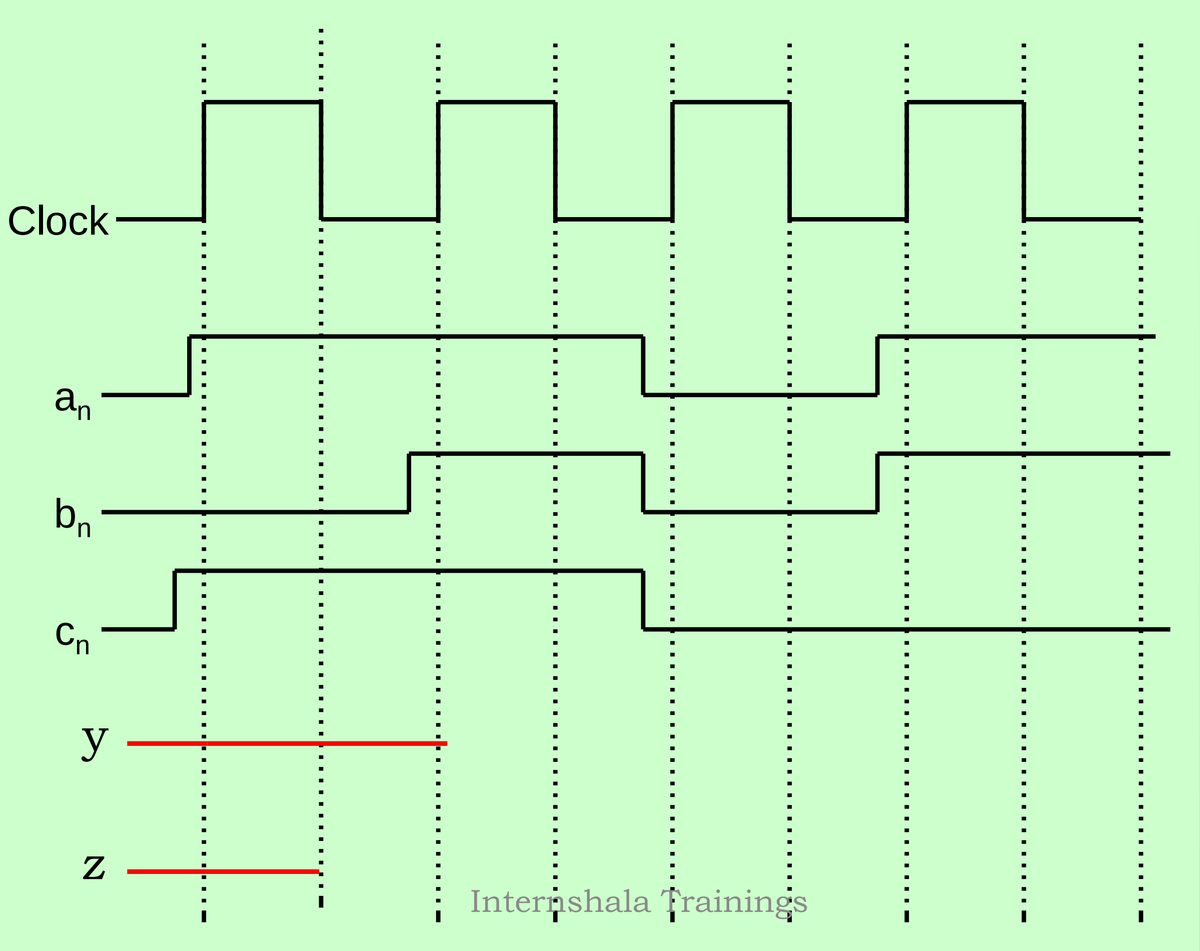




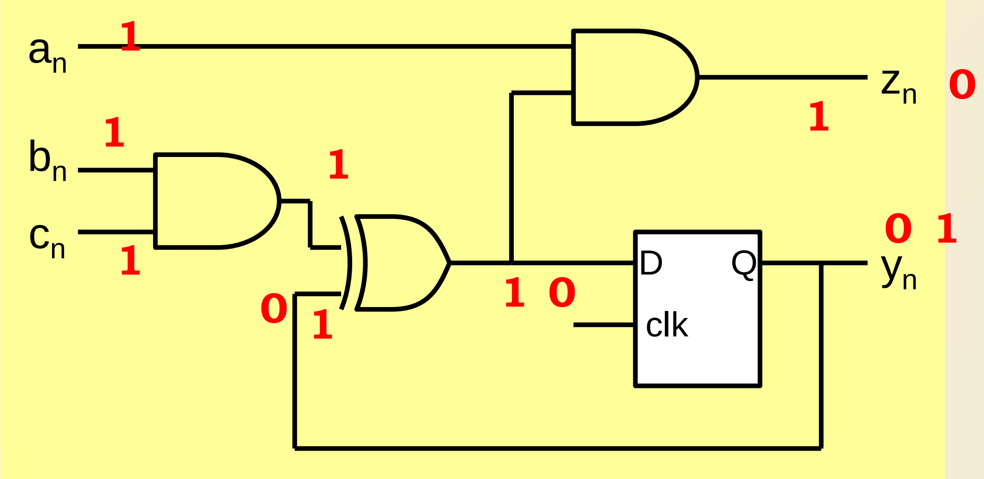
Data is stable when clock is high

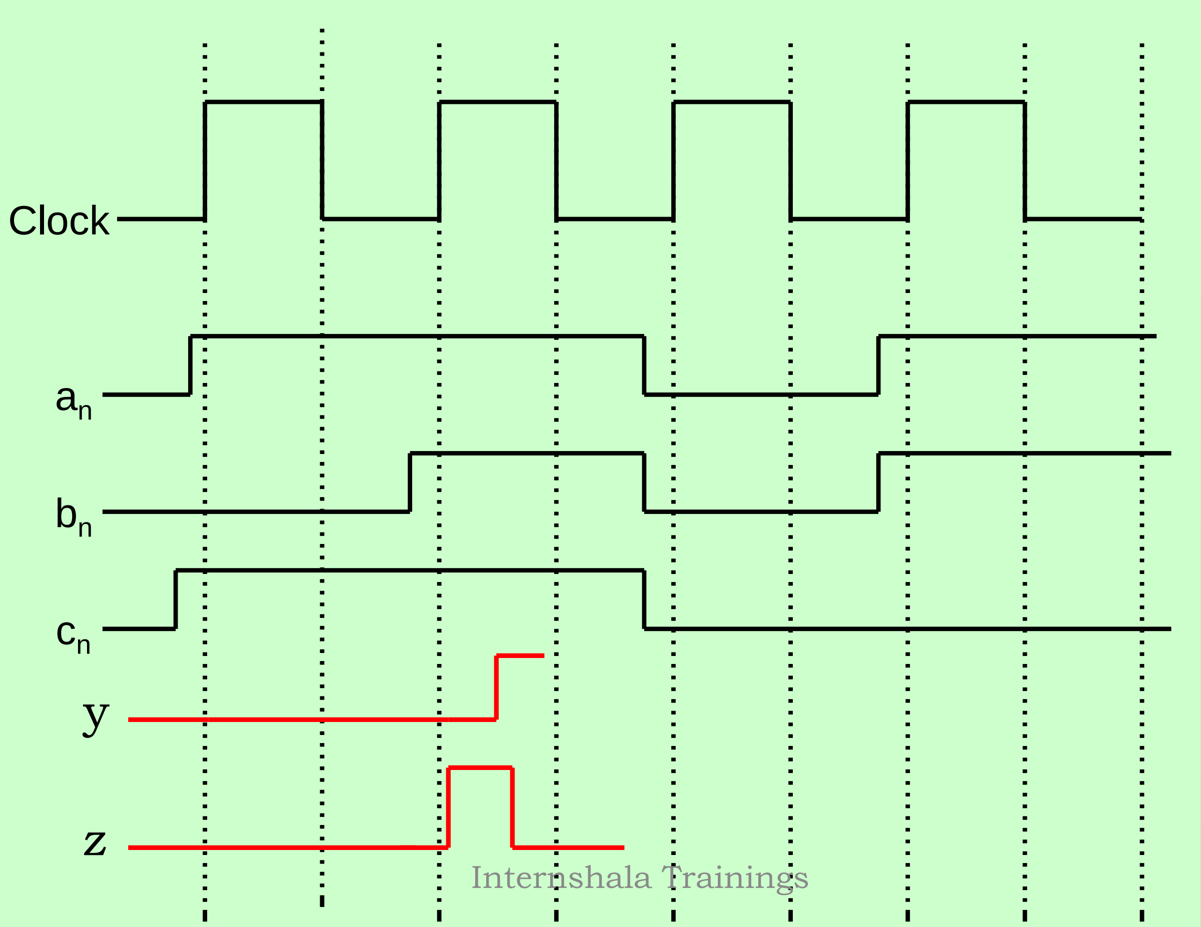
Example



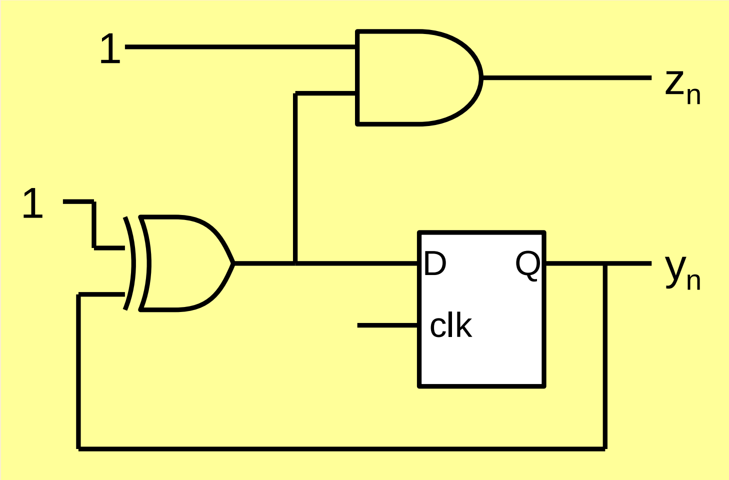


Example

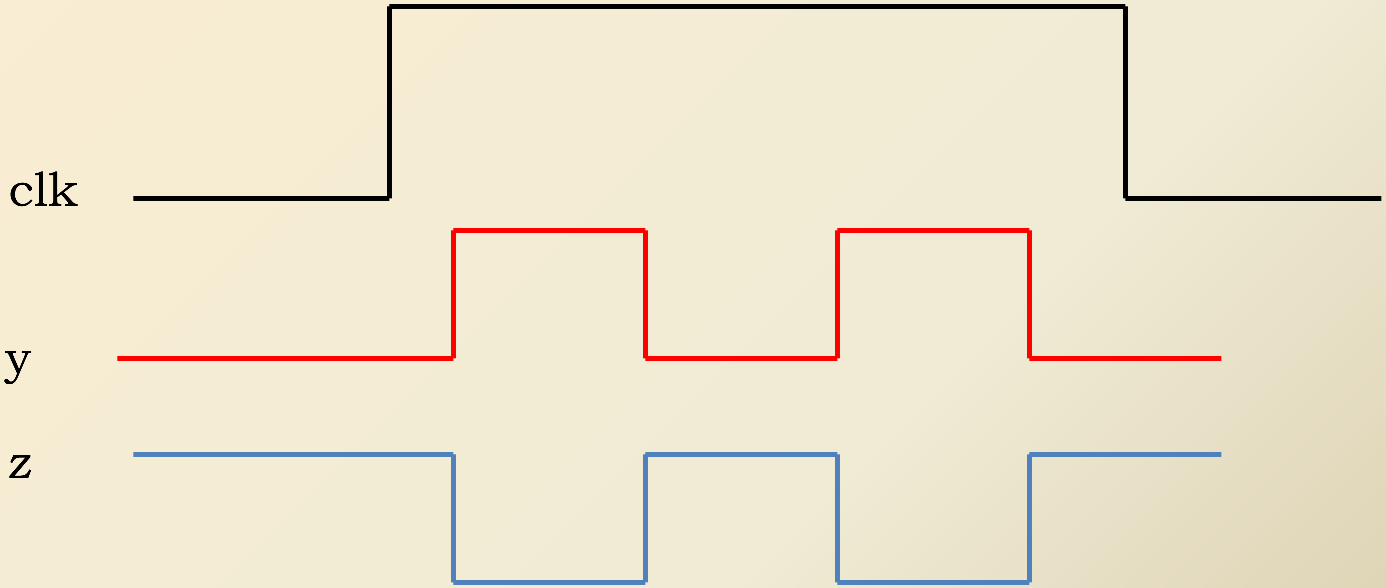




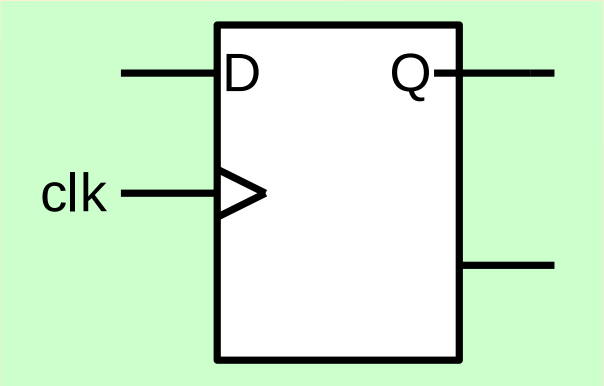
Problem with Latch

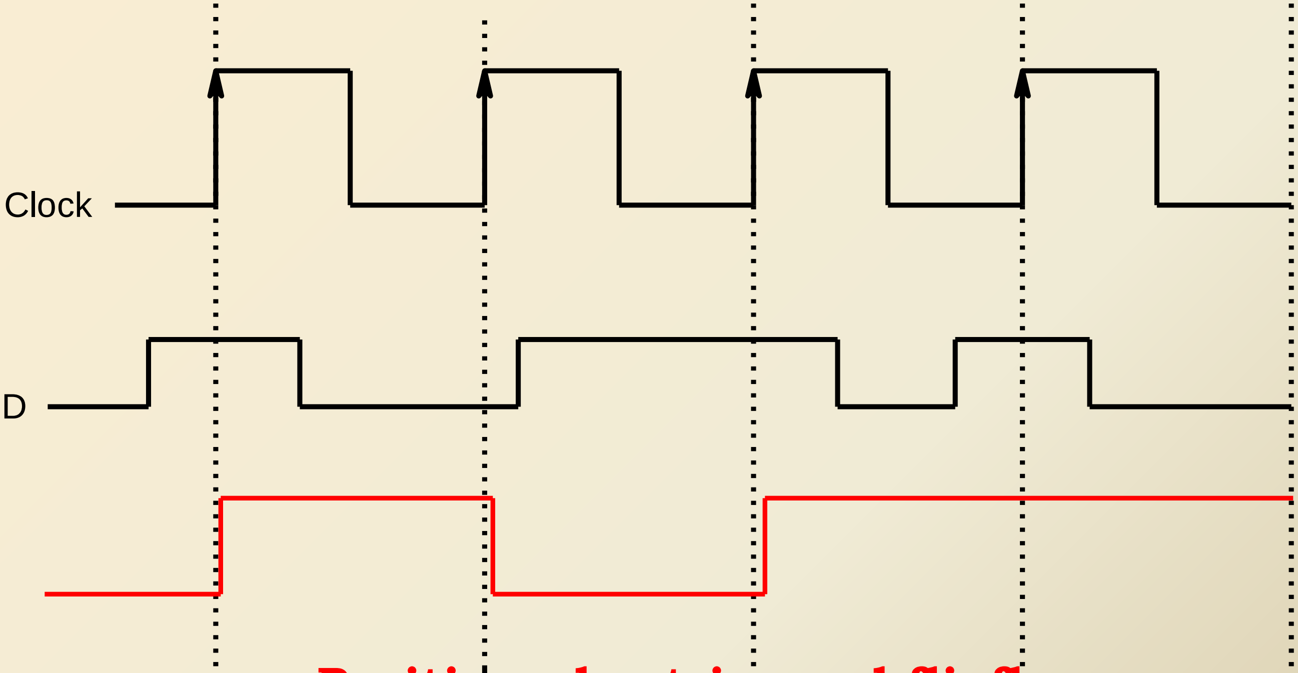


Circuits are designed with the idea there would be single change in output or memory state in single clock cycle.



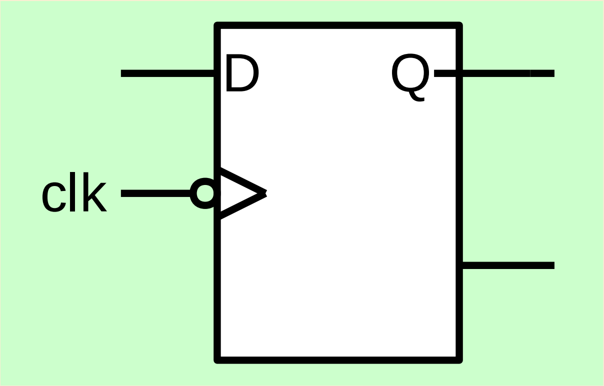
Edge Triggered Latch or Flip-flop

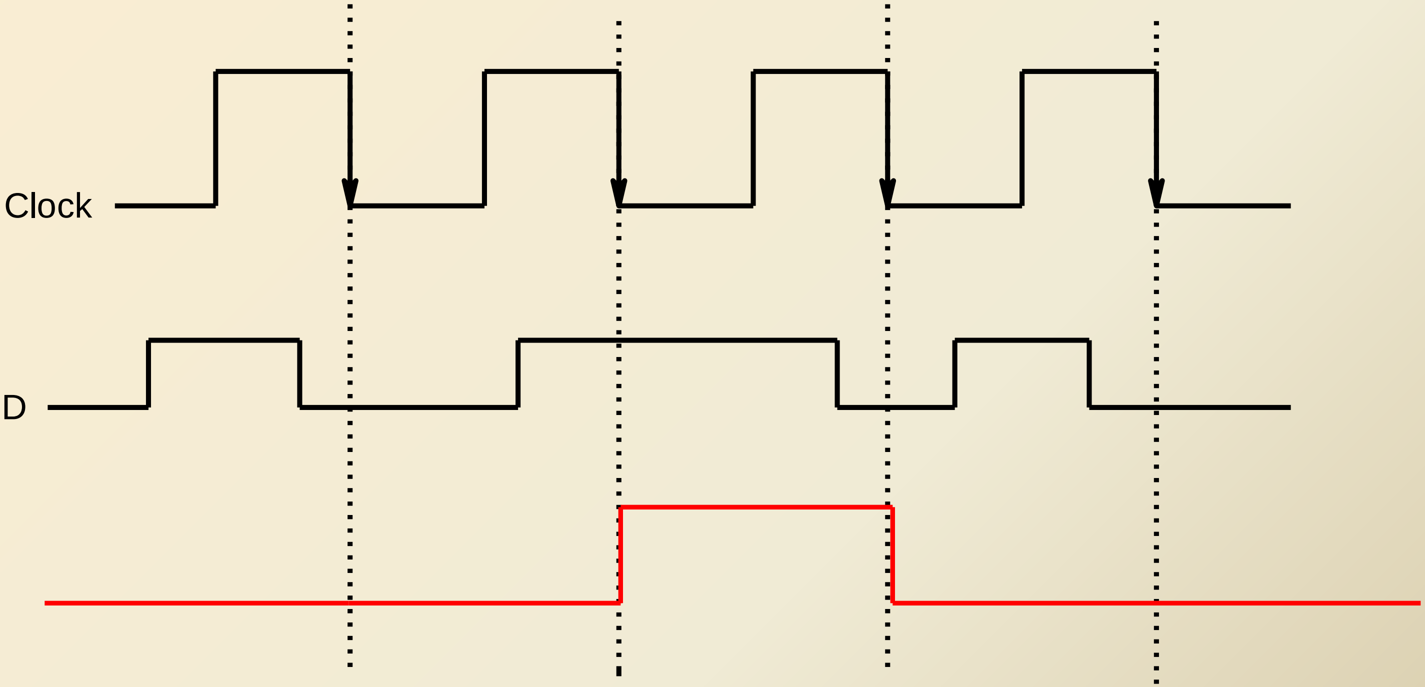




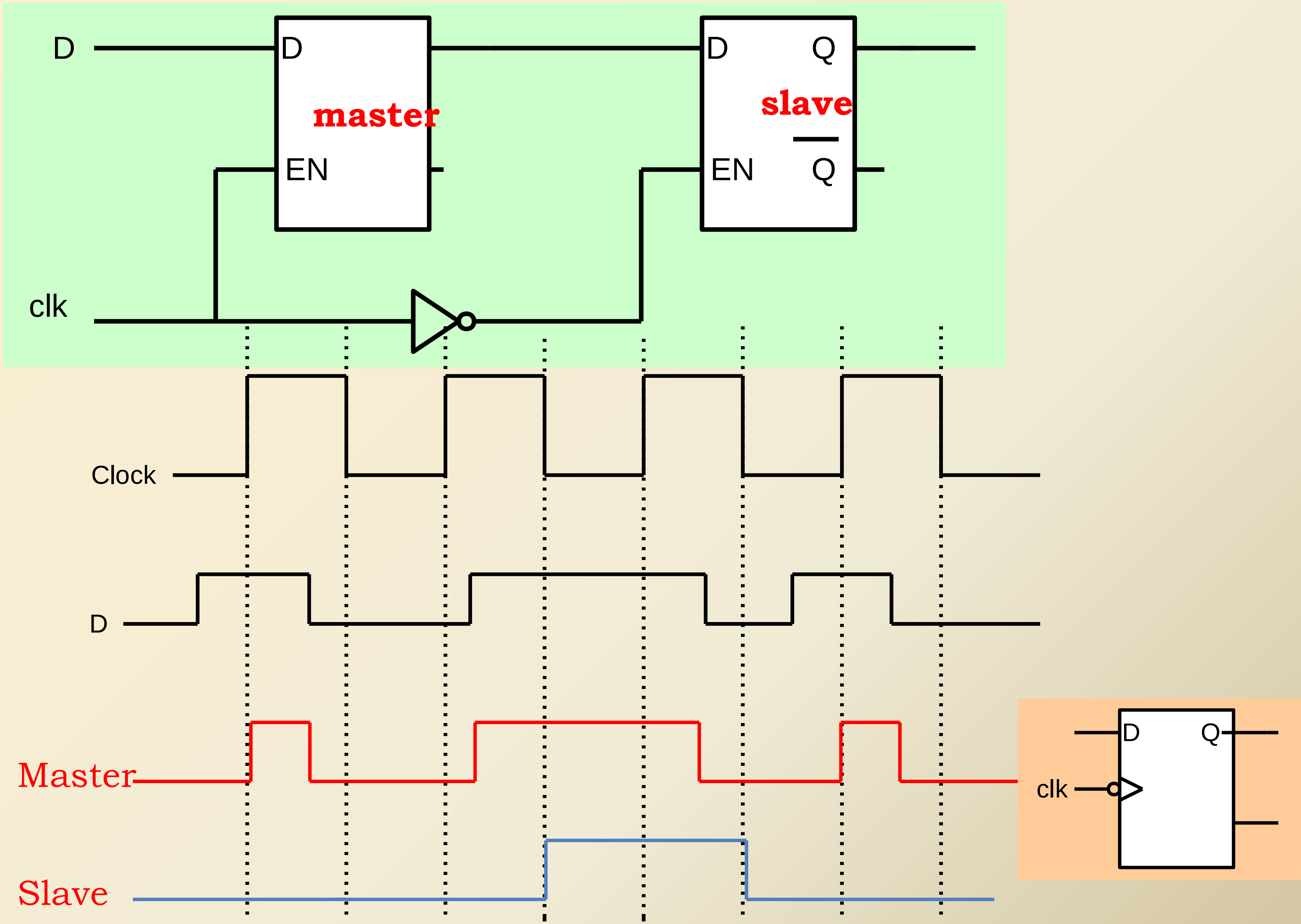
Positive edge triggered flipflop

Negative Edge Triggered Latch or Flip-flop

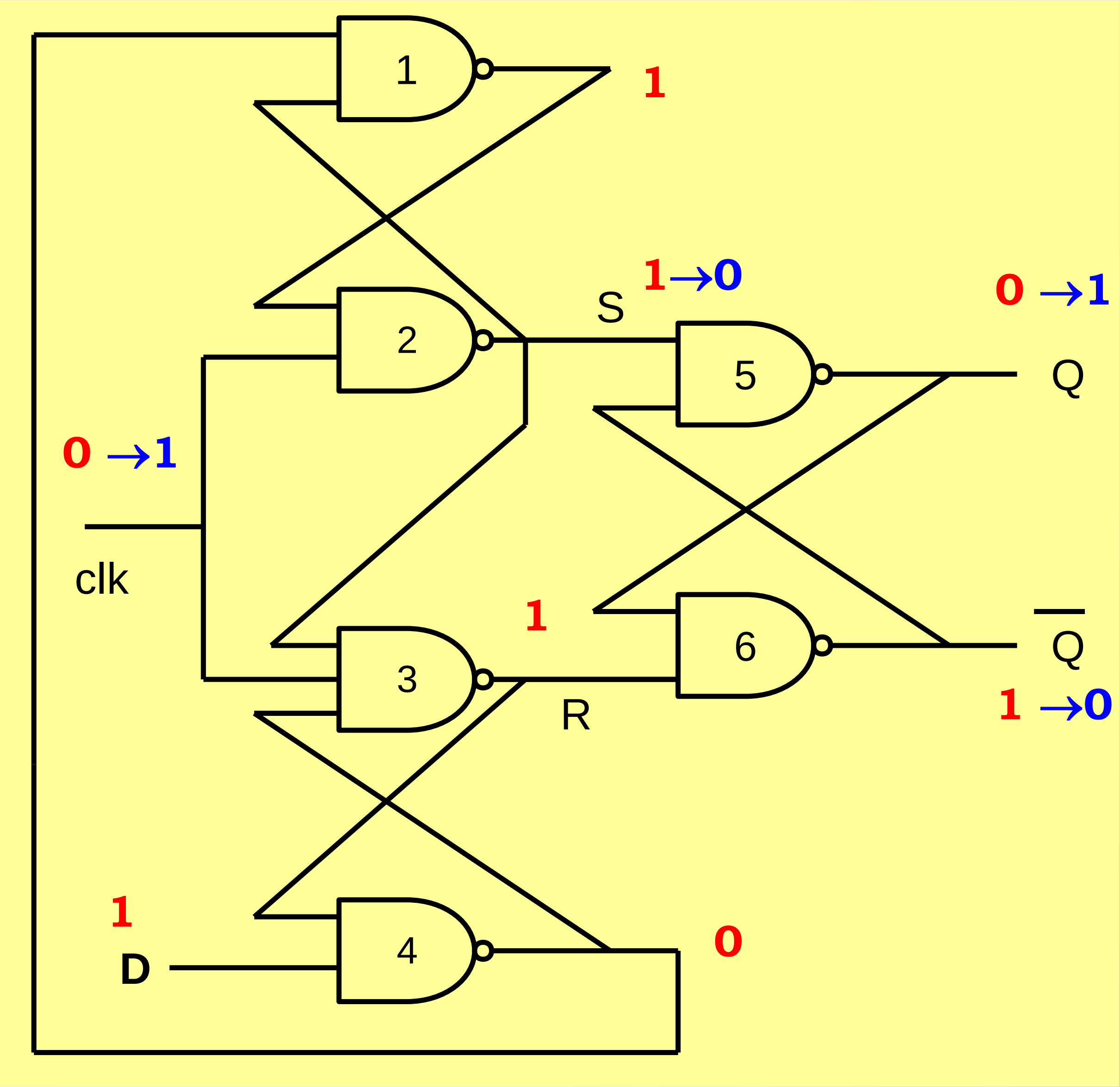




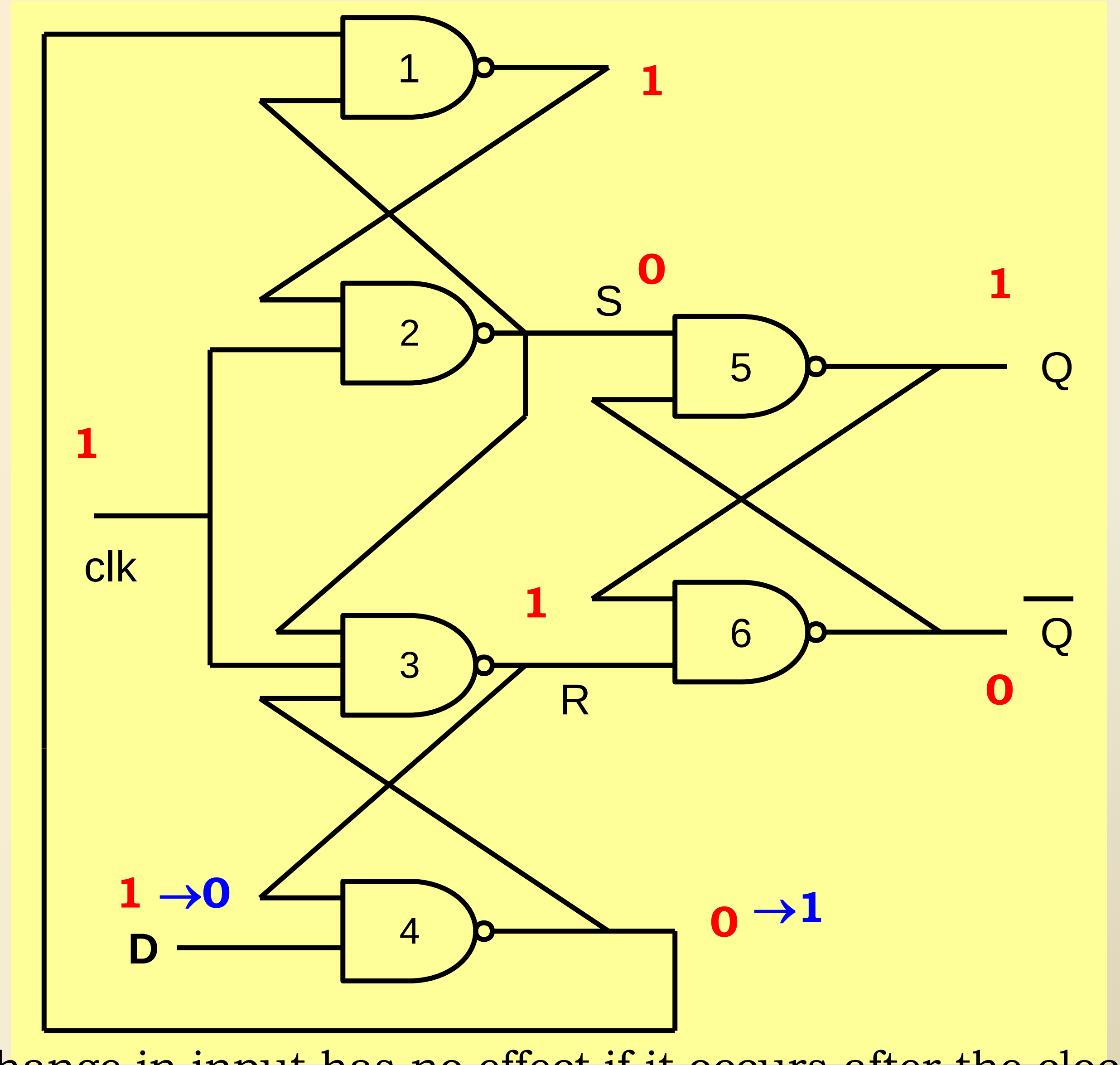
Master-Slave D Flip-flop



Positive edge triggered Flip-flop

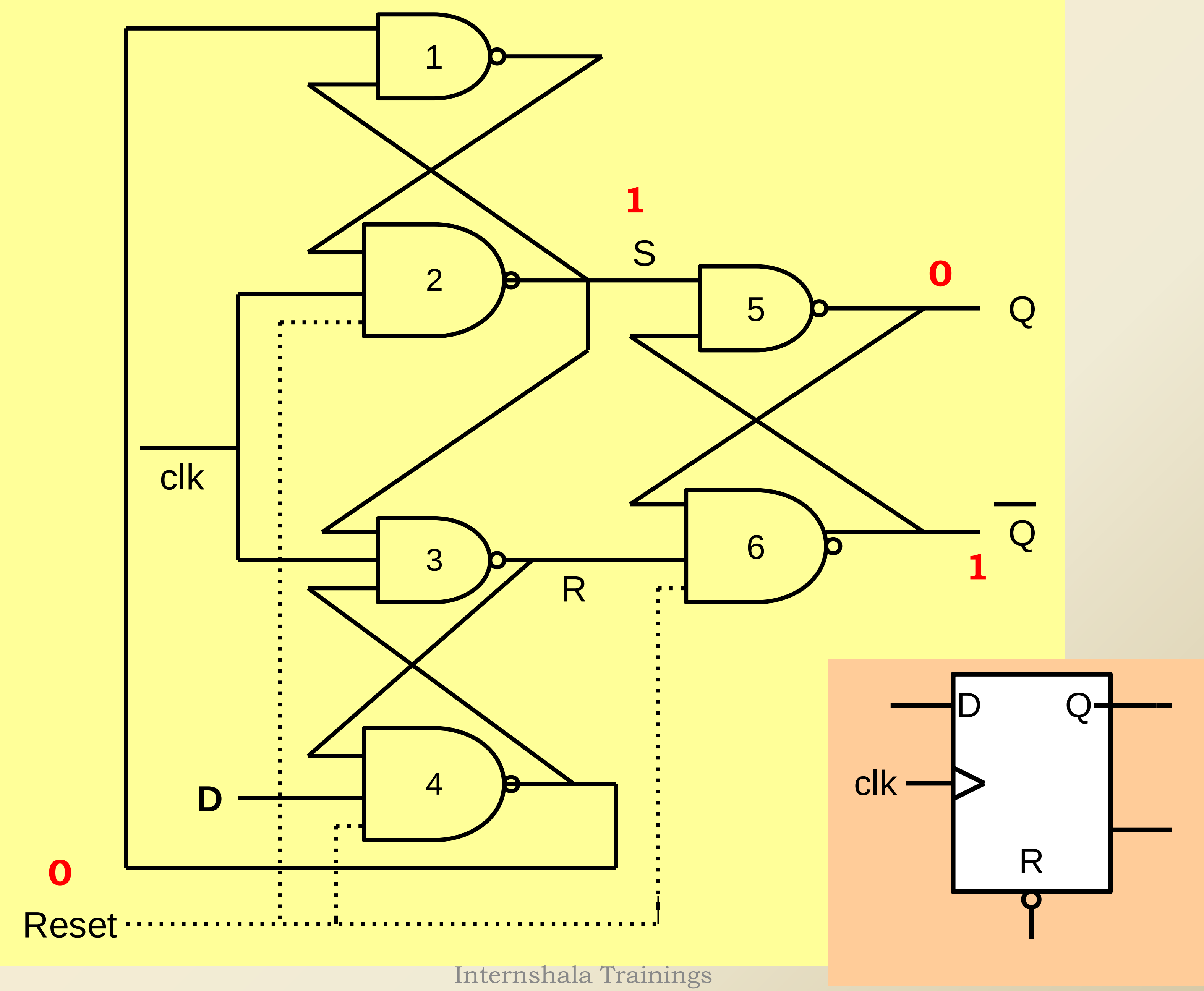


Positive edge triggered Flip-flop



A change in input has no effect if it occurs after the clock edge

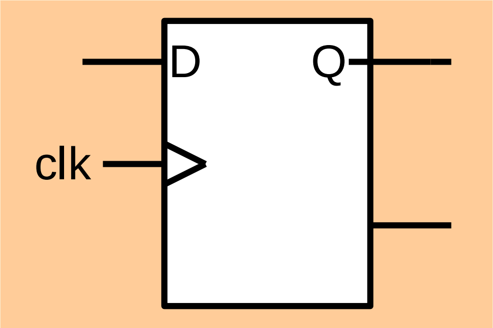
Positive edge Triggered Flip-flop with Asynchronous Reset



Internshala Trainings

Characteristic table

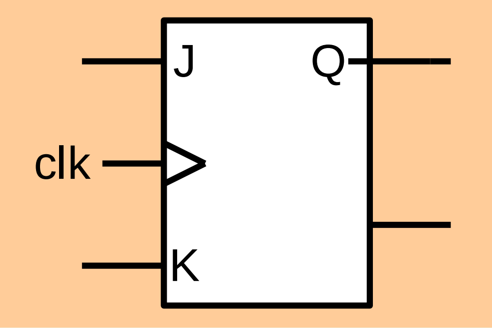
Given a input and the present state of the flip-flop, what is the next state of the flip-flop



Inputs (D)	Q(t+1)
0	0
1	1

$$Q(t+1)=D$$

JK Flip-flop

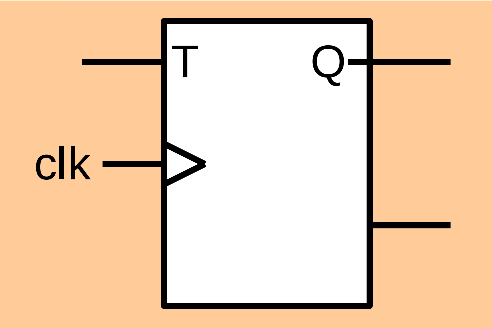


Inputs J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

$$Q(t+1)=\overline{Q(t)}.J+Q(t).\overline{K}$$

→**Characteristic equation**

Toggle or T Flip-flop



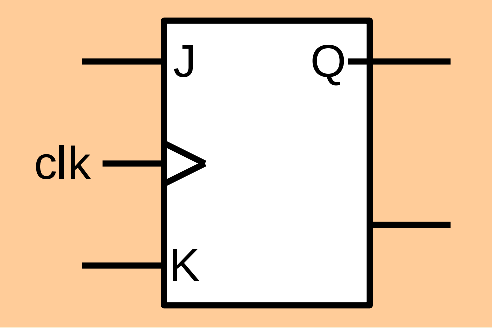
Inputs (T)	Q(t+1)
0	Q(t)
1	$\overline{Q(t)}$

$$Q(t+1) = \overline{Q(t)}.T + Q(t).\overline{T}$$

Excitation Table What inputs are required to effect a particular state change

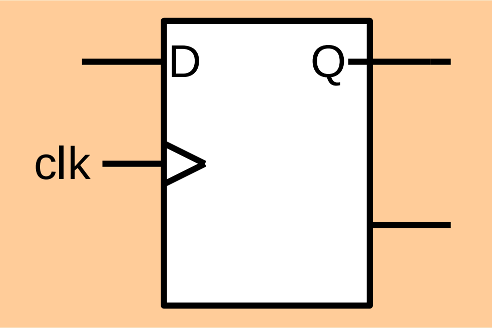
Q(t)	Q(t+1)	Inputs T
0	0	0
0	1	1
1	0	1
1	1	0

Excitation Table



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

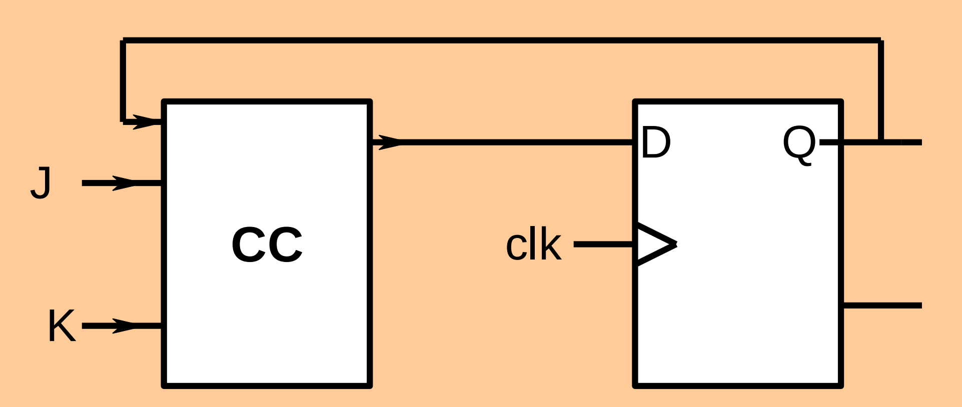
		Inputs	
Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0



D	Q(t+1)
0	0
1	1

		Inputs
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

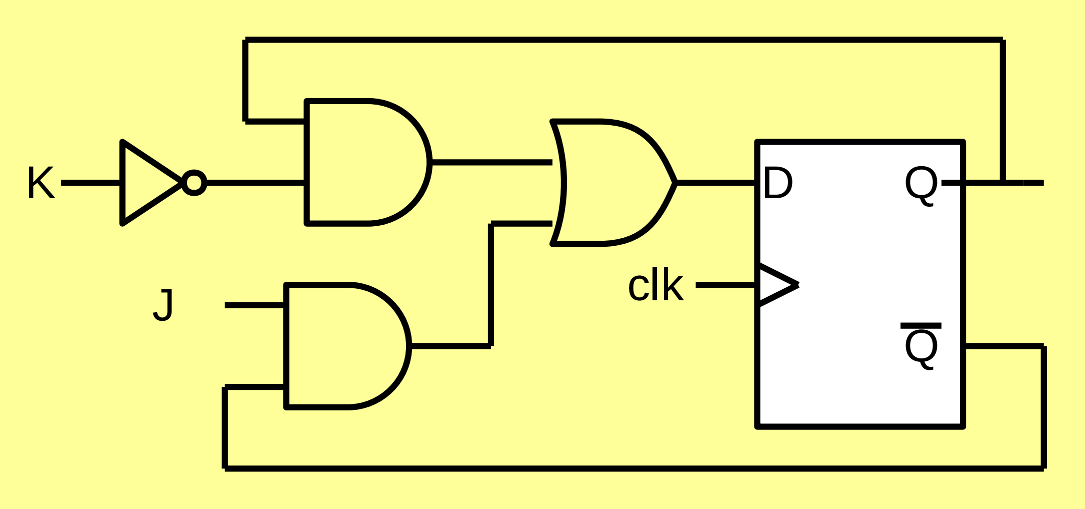
Convert a D FF to JK FF



J	K	Q(t+1)	D
0	0	Q(t)	**Q(t)**
0	1	0	**0**
1	0	1	**1**
1	1	$\overline{Q(t)}$	$\overline{\mathbf{Q(t)}}$

Q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$D = \overline{Q}.J + Q.\overline{K}$$


K
J
clk
D
Q
$\overline{Q}$