

Binary Representation

- The basis of all digital data is binary representation.
- Binary - means 'two'
 - 1, 0
 - True, False
 - Hot, Cold
 - On, Off
- We must be able to handle more than just values for real world problems
 - 1, 0, 56
 - True, False, Maybe
 - Hot, Cold, LukeWarm, Cool
 - On, Off, Leaky

Number Systems

- To talk about binary data, we must first talk about number systems
- The decimal number system (base 10) you should be familiar with!
 - A digit in base 10 ranges from 0 to 9.
 - A digit in base 2 ranges from 0 to 1 (binary number system). A digit in base 2 is also called a 'bit'.
 - A digit in base R can range from 0 to R-1
 - A digit in Base 16 can range from 0 to 16-1 (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). Use letters A-F to represent values 10 to 15. Base 16 is also called Hexadecimal or just 'Hex'.

Positional Notation

Value of number is determined by multiplying each digit by a weight and then summing. The weight of each digit is a POWER of the BASE and is determined by position.

$$\begin{aligned} 953.78 &= 9 * 10^2 + 5 * 10^1 + 3 * 10^0 + 7 * 10^{-1} + 8 * 10^{-2} \\ &= 900 + 50 + 3 + .7 + .08 = 953.78 \end{aligned}$$

$$\begin{aligned} 0b1011.11 &= 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2} \\ &= 8 + 0 + 2 + 1 + 0.5 + 0.25 \\ &= 11.75 \end{aligned}$$

$$\begin{aligned} 0xA2F &= 10*16^2 + 2*16^1 + 15*16^0 \\ &= 10 * 256 + 2 * 16 + 15 * 1 \\ &= 2560 + 32 + 15 = 2607 \end{aligned}$$

Base 10, Base 2, Base 16

The textbook uses subscripts to represent different bases (ie. $A2F_{16}$, 953.78_{10} , 1011.11_2)

I will use special symbols to represent the different bases. The default base will be decimal, no special symbol for base 10.

The '0x' will be used for base 16 (0xA2F)

The '0b' will be used for base 2 (0b10101111)

If ALL numbers on a page are the same base (ie, all in base 16 or base 2 or whatever) then no symbols will be used and a statement will be present that will state the base (ie, all numbers on this page are in base 16).

Common Powers

$$2^{-3} = 0.125$$

$$2^{-2} = 0.25$$

$$2^{-1} = 0.5$$

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

$$2^8 = 256$$

$$2^9 = 512$$

$$2^{10} = 1024$$

$$2^{11} = 2048$$

$$2^{12} = 4096$$

$$16^0 = 1 = 2^0$$

$$16^1 = 16 = 2^4$$

$$16^2 = 256 = 2^8$$

$$16^3 = 4096 = 2^{12}$$

$$2^{10} = 1024 = 1 \text{ Ki (kilobinary)}$$

$$2^{20} = 1048576 = 1 \text{ Mi (1 megabinary)} = 1024 \text{ K} = 2^{10} * 2^{10}$$

$$2^{30} = 1073741824 = 1 \text{ Gi (1 gigabinary)}$$

Conversion of Any Base to Decimal

Converting from ANY base to decimal is done by multiplying each digit by its weight and summing.

Binary to Decimal

$$\begin{aligned}0b1011.11 &= 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 + 1*2^{-1} + 1*2^{-2} \\ &= 8 + 0 + 2 + 1 + 0.5 + 0.25 \\ &= 11.75\end{aligned}$$

Hex to Decimal

$$\begin{aligned}0xA2F &= 10*16^2 + 2*16^1 + 15*16^0 \\ &= 10 * 256 + 2 * 16 + 15 * 1 \\ &= 2560 + 32 + 15 = 2607\end{aligned}$$

Conversion of Decimal Integer To ANY Base

Divide Number N by base R until quotient is 0. **Remainder** at EACH step is a digit in base R, from Least Significant digit to Most significant digit.

Convert 53 to binary

$$53/2 = 26, \text{ rem} = 1 \longleftarrow \text{Least Significant Digit}$$

$$26/2 = 13, \text{ rem} = 0$$

$$13/2 = 6, \text{ rem} = 1$$

$$6/2 = 3, \text{ rem} = 0$$

$$3/2 = 1, \text{ rem} = 1$$

$$1/2 = 0, \text{ rem} = 1 \longleftarrow \text{Most Significant Digit}$$


$$53 = 0b\ 110101$$

$$= 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$$

$$= 32 + 16 + 0 + 4 + 0 + 1 = 53$$

Least Significant Digit Most Significant Digit

53 = 0b 110101



Most Significant Digit
(has weight of 2^5 or
32). For base 2, also
called Most Significant
Bit (MSb). Always
LEFTMOST digit.

Least Significant Digit
(has weight of 2^0 or 1).
For base 2, also called
Least Significant Bit
(LSb). Always
RIGHTMOST digit.

More Conversions

Convert 53 to Hex

$$53/16 = 3, \text{ rem} = 5$$

$$3/16 = 0, \text{ rem} = 3$$

$$53 = 0x35$$

$$= 3 * 16^1 + 5 * 16^0$$

$$= 48 + 5 = 53$$

More Conversions

Convert 435 to Hex

$$435/16 = 27, \text{ rem} = 3$$

$$27/16 = 1, \text{ rem} = 11 = 0xB$$

$$1/16 = 0, \text{ rem} = 1$$

$$435 = 0x1B3$$

To check:

$$0x1B3 = 1*16^2 + 11*16^1 + 3*16^0 = 256+176+3 = 435.$$

Hex (base 16) to Binary Conversion

Each Hex digit represents 4 bits. To convert a Hex number to Binary, simply convert each Hex digit to its four bit value.

Hex Digits to binary:

0x0 = 0b 0000

0x1 = 0b 0001

0x2 = 0b 0010

0x3 = 0b 0011

0x4 = 0b 0100

0x5 = 0b 0101

0x6 = 0b 0110

0x7 = 0b 0111

0x8 = 0b 1000

Hex Digits to binary (cont):

0x9 = 0b 1001

0xA = 0b 1010

0xB = 0b 1011

0xC = 0b 1100

0xD = 0b 1101

0xE = 0b 1110

0xF = 0b 1111

Hex to Binary, Binary to Hex

$0xA2F = 0b\ 1010\ 0010\ 1111$

$0x345 = 0b\ 0011\ 0100\ 0101$

Binary to Hex is just the opposite, create groups of 4 bits starting with least significant bits. If last group does not have 4 bits, then pad with zeros for unsigned numbers.

$0b\ 1010001 = 0b\ 0101\ 0001 = 0x51$

↑
Padded with a zero

A Trick!

If faced with a large binary number that has to be converted to decimal, I first convert the binary number to HEX, then convert the HEX to decimal. Less work!

$$\begin{aligned} 0b\ 110111110011 &= 0b\ 1101\ 1111\ 0011 \\ &= \quad D \quad F \quad 3 \\ &= 13 * 16^2 + 15 * 16^1 + 3 * 16^0 \\ &= 13 * 256 + 15 * 16 + 3 * 1 \\ &= 3328 + 240 + 3 \\ &= 3571 \end{aligned}$$

Binary Numbers Again

Recall that N binary digits (N bits) can represent unsigned integers from 0 to $2^N - 1$.

4 bits = 0 to 15

8 bits = 0 to 255

16 bits = 0 to 65535

Besides simply representation, we would like to also do arithmetic operations on numbers in binary form.

Principle operations are addition and subtraction.

Binary Arithmetic, Subtraction

The rules for binary arithmetic are:

$$0 + 0 = 0, \text{ carry} = 0$$

$$1 + 0 = 1, \text{ carry} = 0$$

$$0 + 1 = 1, \text{ carry} = 0$$

$$1 + 1 = 0, \text{ carry} = 1$$

The rules for binary subtraction are:

$$0 - 0 = 0, \text{ borrow} = 0$$

$$1 - 0 = 1, \text{ borrow} = 0$$

$$0 - 1 = 1, \text{ borrow} = 1$$

$$1 - 1 = 0, \text{ borrow} = 0$$

Borrows, Carries from digits to left of current of digit.

Binary subtraction, addition works just the same as decimal addition, subtraction.

Binary, Decimal addition

Decimal

34
+ 17

51
from LSD to MSD:
7+4 = 1; with carry out of 1
to next column

1 (carry) + 3 + 1 = 5.
answer = 51.

Binary

0b 101011
+ 0b 000001

101100
From LSb to MSb:
1+1 = 0, carry of 1
1 (carry)+1+0 = 0, carry of 1
1 (carry)+0 + 0 = 1, no carry
1 +0 = 1
0 + 0 = 0
1 + 0 = 1
answer = 0b 101100

Hex Addition

$$\begin{array}{r} 0x3A \\ + 0x28 \\ \hline 0x62 \end{array}$$

A+8 = 2; with carry out of
1 to next column

1 (carry) + 3 + 2 = 6.
answer = 0x62

Decimal check.

$$\begin{aligned} 0x3A &= 3 * 16 + 10 \\ &= 58 \end{aligned}$$

$$\begin{aligned} 0x28 &= 2 * 16 + 8 \\ &= 40 \end{aligned}$$

$$58 + 40 = 98$$

$$\begin{aligned} 0x62 &= 6 * 16 + 2 \\ &= 96 + 2 = 98!! \end{aligned}$$

Hex addition again

Why is $0xA + 0x8 = 2$ with a carry out of 1?

The carry out has a weight equal to the BASE (in this case 16). The digit that gets left is the excess (BASE - sum).

$$Ah + 8h = 10 + 8 = 18.$$

18 is GREATER than 16 (BASE), so need a carry out!

Excess is $18 - \text{BASE} = 18 - 16 = 2$, so '2' is digit.

Exactly the same thing happens in Decimal.

$$5 + 7 = 2, \text{ carry of } 1.$$

$$5 + 7 = 12, \text{ this is greater than } 10!.$$

So excess is $12 - 10 = 2$, carry of 1.

Subtraction

Decimal

$$\begin{array}{r} 900 \\ - 001 \\ \hline 899 \end{array}$$

0-1 = 9; with borrow of 1
from next column
0 - 1 (borrow) - 0 = 9, with
borrow of 1
9 - 1 (borrow) - 0 = 8.
Answer = 899.

Binary

$$\begin{array}{r} 0b\ 100 \\ - 0b\ 001 \\ \hline 011 \end{array}$$

0-1 = 1; with borrow of 1
from next column
0 - 1 (borrow) - 0 = 1, with
borrow of 1
1 - 1 (borrow) - 0 = 0.
Answer = % 011.

Hex Subtraction

$$\begin{array}{r} 0x34 \\ - 0x27 \\ \hline 0x0D \end{array}$$

4-7 = D; with borrow of 1
from next column

3 - 1 (borrow) - 2 = 0.
answer = 0x0D.

Decimal check.

$$\begin{aligned} 0x34 &= 3 * 16 + 4 \\ &= 52 \end{aligned}$$

$$\begin{aligned} 0x27 &= 2 * 16 + 7 \\ &= 39 \end{aligned}$$

$$52 - 39 = 13$$

$$0x0D = 13 !!$$

Hex subtraction again

Why is $0x4 - 0x7 = 0xD$ with a borrow of 1?

The borrow has a weight equal to the BASE (in this case 16).

$$\text{BORROW} + 0x4 - 0x7 = 16 + 4 - 7 = 20 - 7 = 13 = 0xD.$$

$0xD$ is the result of the subtraction with the borrow.

Exactly the same thing happens in decimal.

$$3 - 8 = 5 \text{ with borrow of } 1$$

$$\text{borrow} + 3 - 8 = 10 + 3 - 8 = 13 - 8 = 5.$$

Fixed Precision

With paper and pencil, I can write a number with as many digits as I want:

1,027,80,032,034,532,002,391,030,300,209,399,302,992,092,920

A microprocessor or computing system usually uses **FIXED PRECISION** for integers; they limit the numbers to a fixed number of bits:

0x AF4500239DEFA231	64 bit number, 16 hex digits
0x 9DEFA231	32 bit number, 8 hex digits
0x A231	16 bit number, 4 hex digits
0x 31	8 bit number, 2 hex digits

High end microprocessors use 64 or 32 bit precision; low end microprocessors use 16 or 8 bit precision.

Unsigned Overflow

In this class I will use 8 bit precision most of the time, 16 bit occasionally.

Overflow occurs when I add or subtract two numbers, and the correct result is a number that is outside of the range of allowable numbers for that precision. I can have both unsigned and signed overflow (more on signed numbers later)

8 bits -- unsigned integers 0 to $2^8 - 1$ or 0 to 255.

16 bits -- unsigned integers 0 to $2^{16} - 1$ or 0 to 65535

N bit – unsigned numbers 0 to $2^N - 1$

Unsigned Overflow Example

Assume 8 bit precision; i.e.. I can't store any more than 8 bits for each number.

Lets add $255 + 1 = 256$. The number 256 is OUTSIDE the range of 0 to 255! What happens during the addition?

$$\begin{array}{r} 255 = 0x FF \\ + 1 = 0x 01 \\ \hline 256 \neq 0x00 \end{array}$$

$0xF + 1 = 0$, carry out

$0xF + 1$ (carry) $+ 0 = 0$, carry out

Carry out of MSB falls off end, No place to put it!!!

Final answer is WRONG because could not store carry out.

Unsigned Overflow

A carry out of the Most Significant Digit (MSD) or Most Significant Bit (MSB) is an **OVERFLOW** indicator for addition of **UNSIGNED** numbers.

The correct result has overflowed the number range for that precision, and thus the result is incorrect.

If we could **STORE** the carry out of the MSD, then the answer would be correct. But we are assuming it is discarded because of fixed precision, so the bits we have left are the incorrect answer.

Codes for Characters

Also need to represent Characters as digital data.

The **ASCII** code (American Standard Code for Information Interchange) is a 7-bit code for Character data. Typically 8 bits are actually used with the 8th bit being zero or used for error detection (parity checking).
8 bits = 1 **Byte**.

`'A'` = % 01000001 = 0x41

`'&'` = % 00100110 = 0x26

7 bits can only represent 2^7 different values (128). This enough to represent the Latin alphabet (A-Z, a-z, 0-9, punctuation marks, some symbols like \$), but what about other symbols or other languages?

ASCII

*American Standard
Code for Information
Interchange*

		Most Significant Digit							
		0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
0x0	NUL	DLE	SPC	0	@	P	`	p	
0x1	SOH	DC1	!	1	A	Q	a	q	
0x2	STX	DC2	"	2	B	R	b	r	
0x3	ETX	DC3	#	3	C	S	c	s	
0x4	EOT	DC4	\$	4	D	T	d	t	
0x5	ENQ	NAK	%	5	E	U	e	u	
0x6	ACK	SYN	&	6	F	V	f	v	
0x7	BEL	ETB	'	7	G	W	g	w	
0x8	BS	CAN	(8	H	X	h	x	
0x9	TAB	EM)	9	I	Y	i	y	
0xA	LF	SUB	*	:	J	Z	j	z	
0xB	VT	ESC	+	;	K	[k	{	
0xC	FF	FS	,	<	L	\	l		
0xD	CR	GS	-	=	M]	m	}	
0xE	SO	RS	.	>	N	^	n	~	
0xF	SI	US	/	?	O	_	o	DEL	

UNICODE

UNICODE is a 16-bit code for representing alphanumeric data. With 16 bits, can represent 2^{16} or **65536** different symbols. 16 bits = 2 **Bytes** per character (the extended version uses 32-bits per character, or 4 bytes, for 4,294,967,296 different symbols).

0x0041-005A A-Z

0x0061-4007A a-z

Some other alphabet/symbol ranges

0x3400-3d2d Korean Hangul Symbols

0x3040-318F Hiranga, Katakana, Bopomofo, Hangul

0x4E00-9FFF Han (Chinese, Japanese, Korean)

UNICODE used by Web browsers, Java, most software these days.

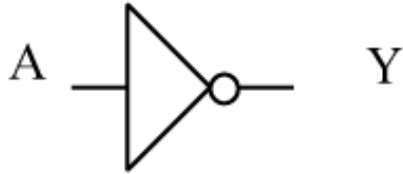
Number System Practice

What should you practice?

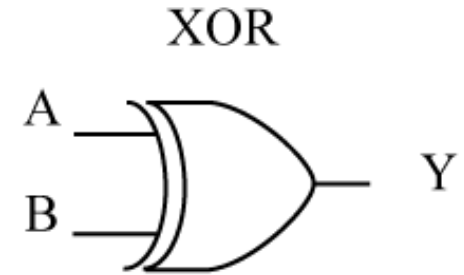
- Hex to decimal, decimal to hex conversion
- Hex to binary, binary to hex conversion
- Hex addition, subtraction

Basic Logic Gates

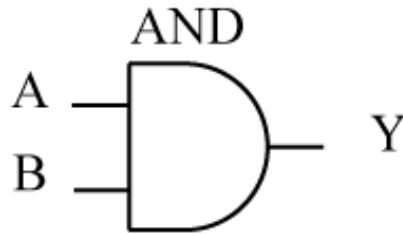
A	Y
0	1
1	0



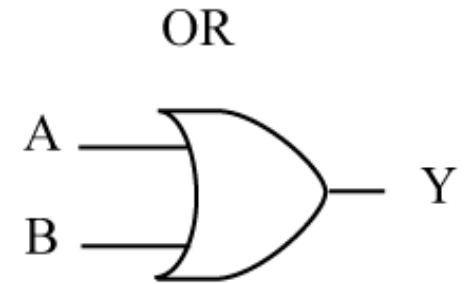
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



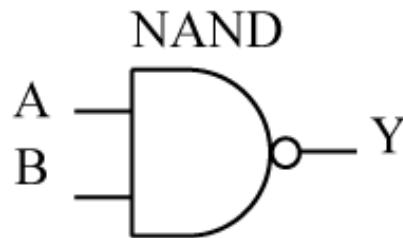
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



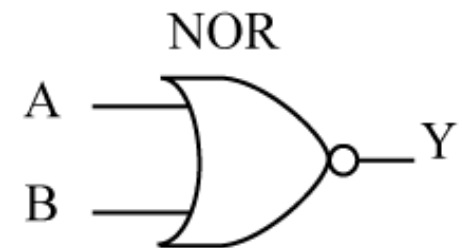
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

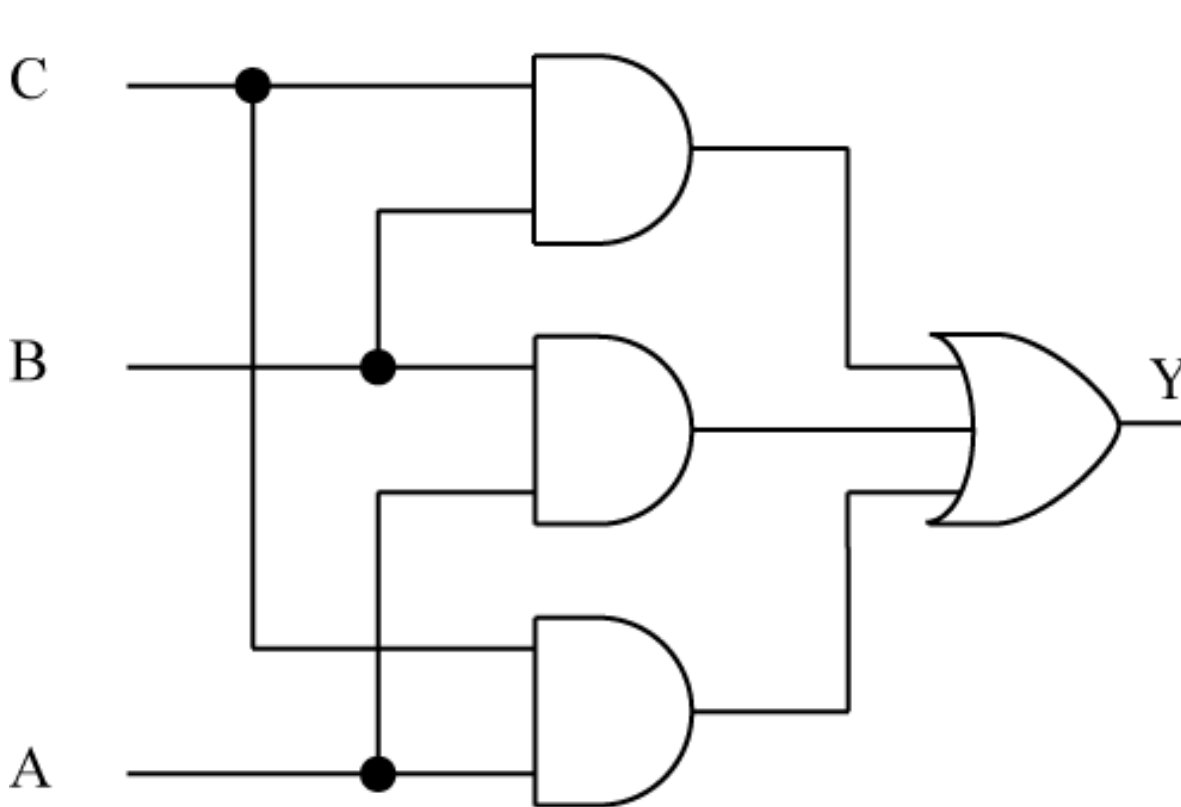


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



Majority Gate (and-or) form

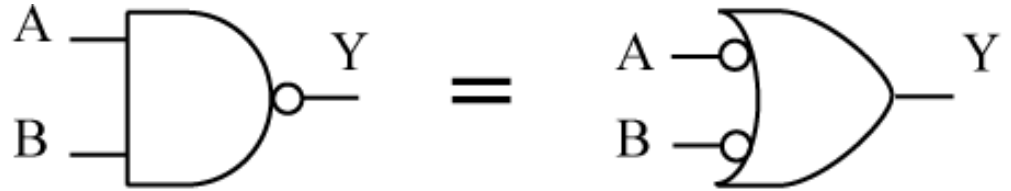
Majority $Y = (A \& B) \mid (B \& C) \mid (A \& C)$



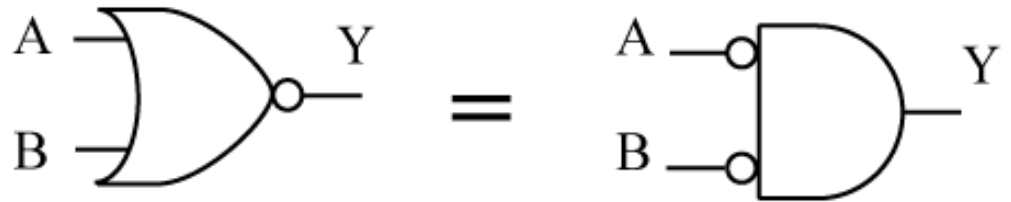
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

DeMorgan's Law

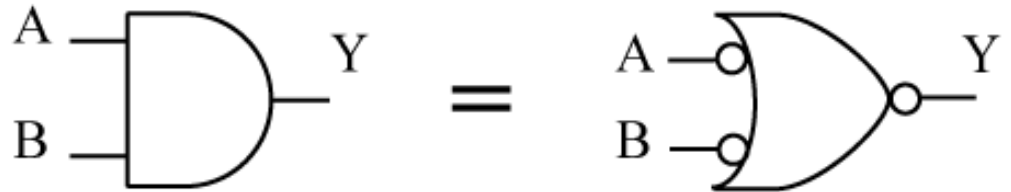
(1) $\sim(A \& B) = (\sim A) | (\sim B)$



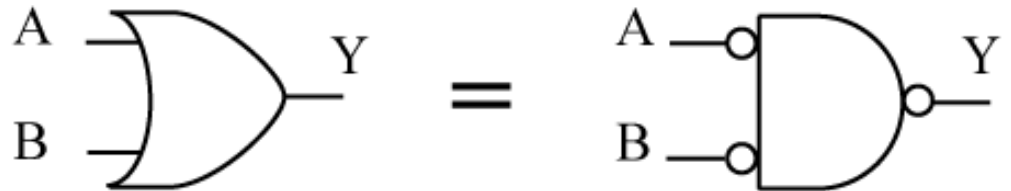
(2) $\sim(A | B) = (\sim A) \& (\sim B)$



(3) $A \& B = \sim((\sim A) | (\sim B))$



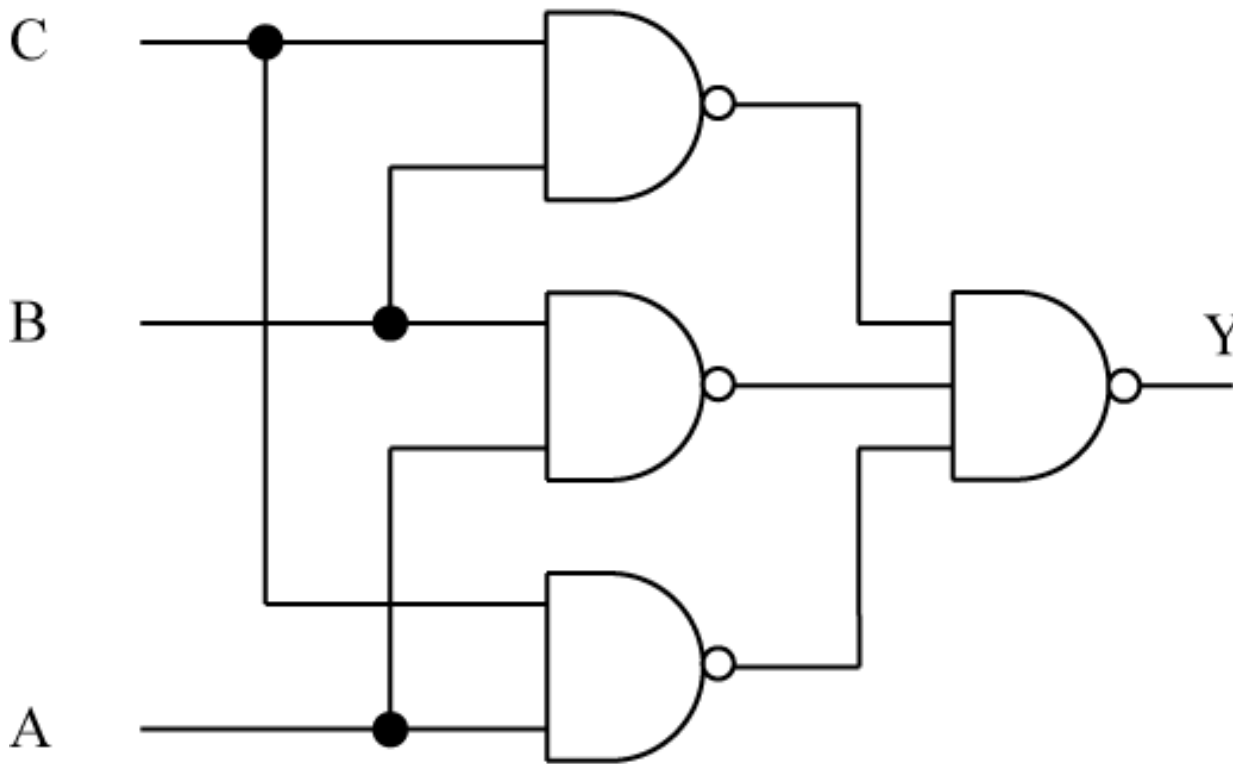
(4) $A | B = \sim((\sim A) \& (\sim B))$



Majority Gate (nand-nand) form

Majority

$$\begin{aligned} Y &= \sim(\sim(A \& B) \& \sim(B \& C) \& \sim(A \& C)) \\ &= (\sim(\sim(A \& B)) \mid (\sim(\sim(B \& C))) \mid (\sim(\sim(A \& C)))) \\ &= (A \& B) \mid (B \& C) \mid (A \& C) \end{aligned}$$

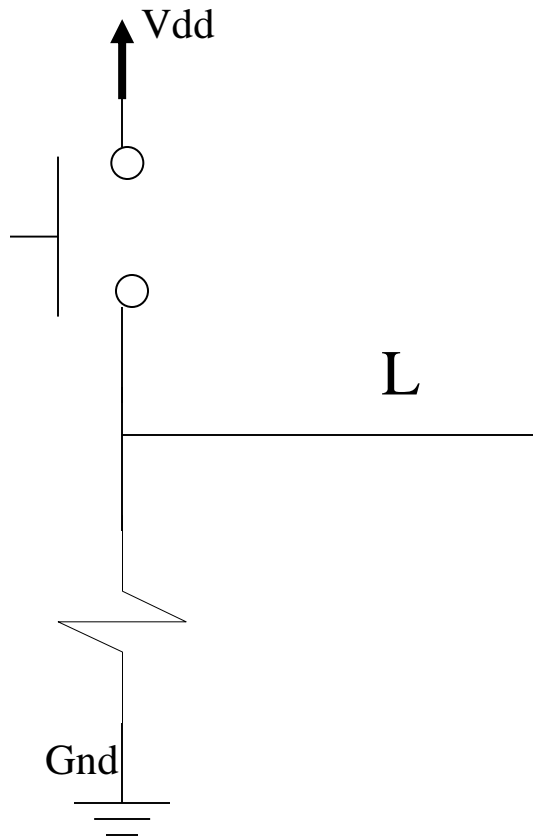


Representing '1' and '0'

- In the electrical world, two ways of representing '0' and '1' are (these are not the only ways):
 - Presence or absence of electrical current
 - Different Voltage levels
- Different voltage levels are the most common
 - Usually 0v for logic '0', some non-zero voltage for logic '1' (i.e. > 3 volts)
- Can interface external sources to digital systems in many ways
 - Switches, buttons, other human controlled input devices
 - Transducers (change a physical quantity like temperature into a digital quantity).

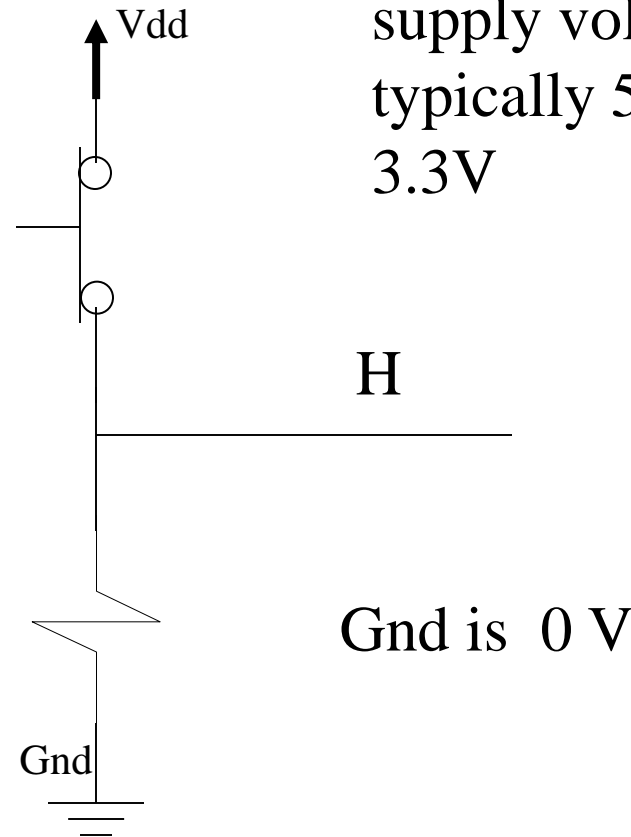
Switch Inputs

High True switch



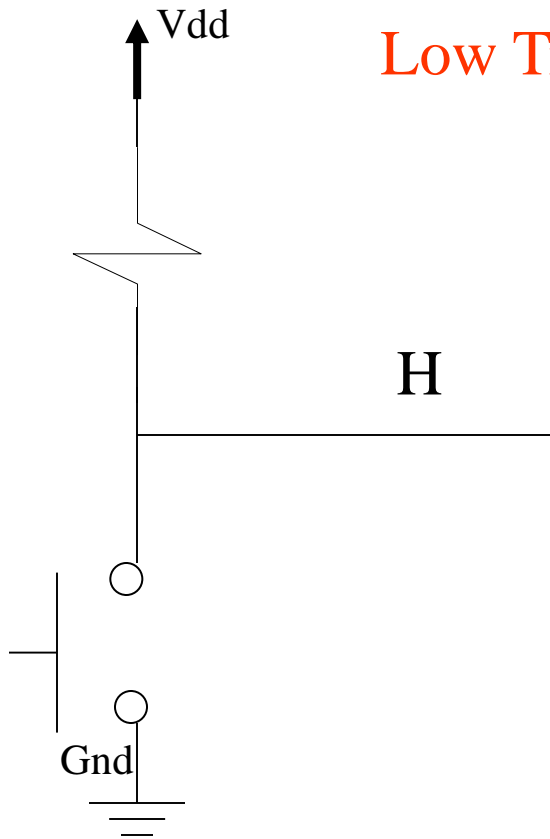
Switch open
(**negated**), output is L

V_{dd} is power supply voltage, typically 5V or 3.3V

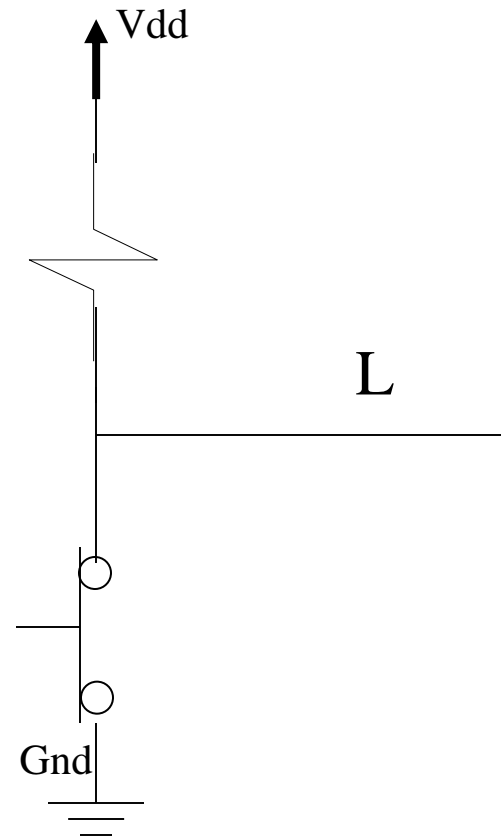


Switch closed (**asserted**), output is H

Examples of high, low signals

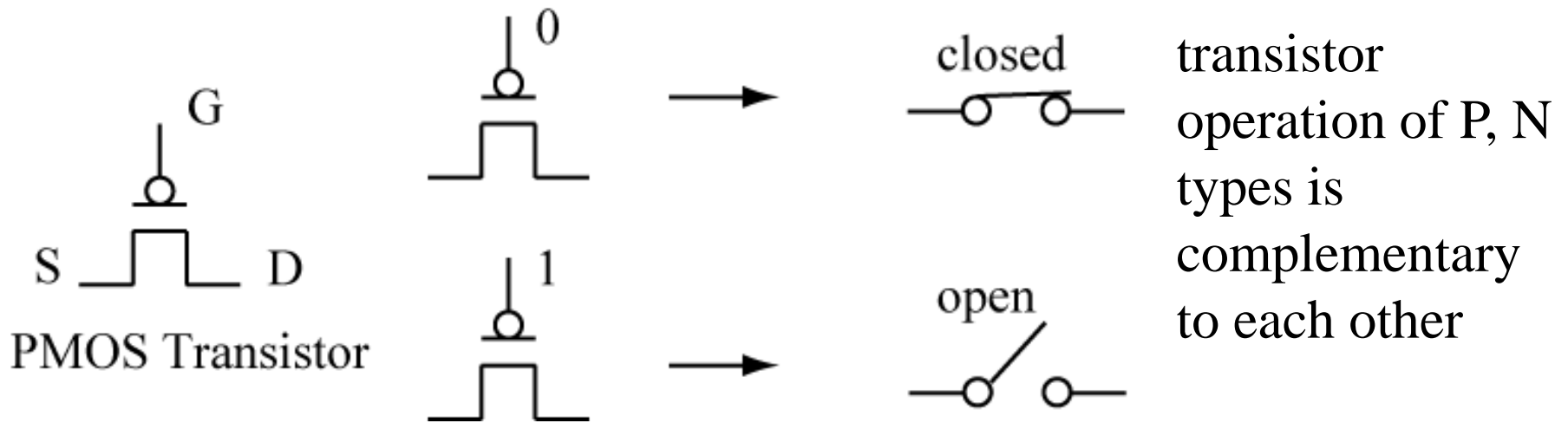
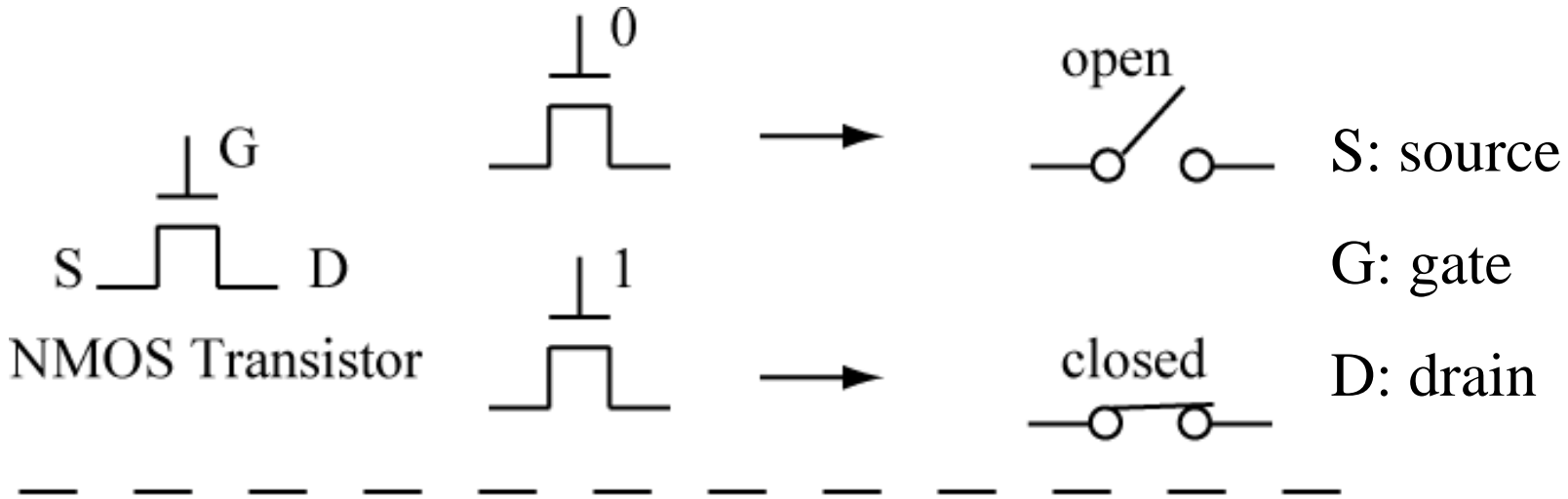


Switch open (negated),
output is H

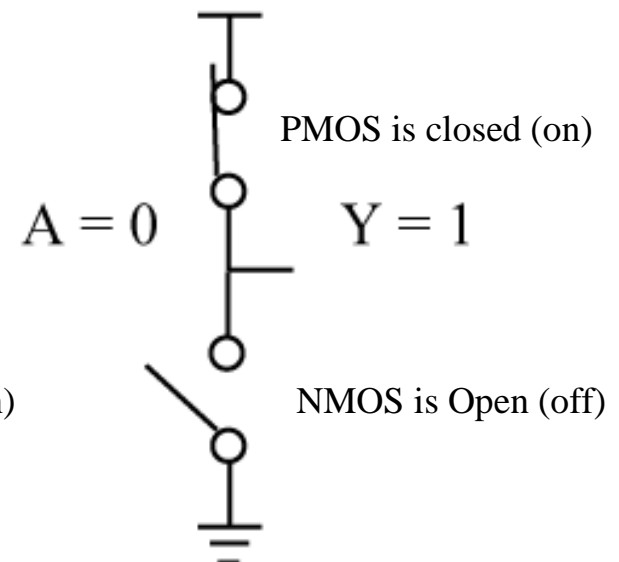
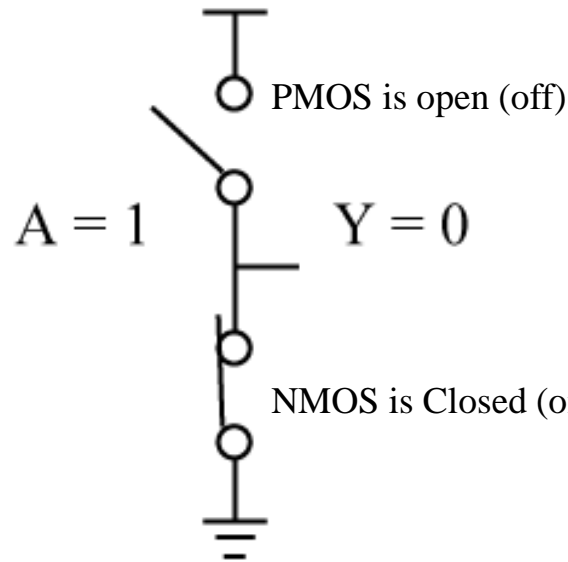
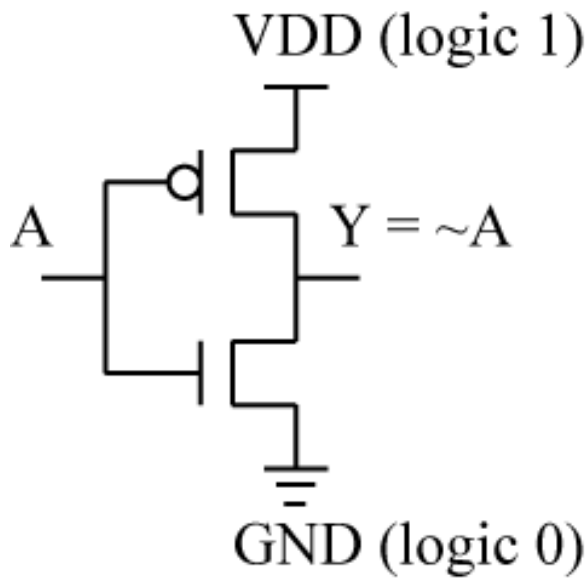
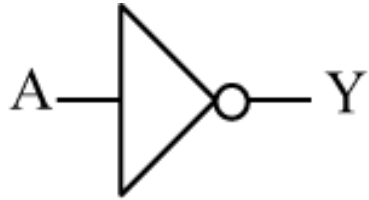


Switch closed (asserted),
output is L

CMOS transistors (P, N)

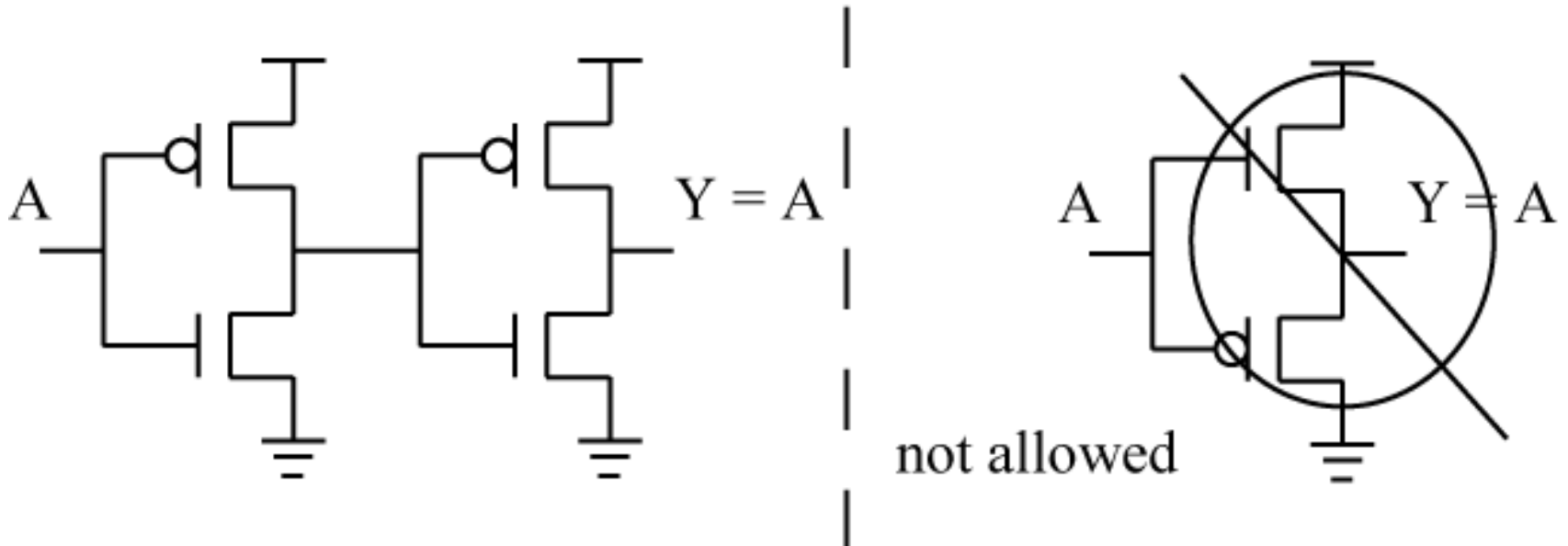
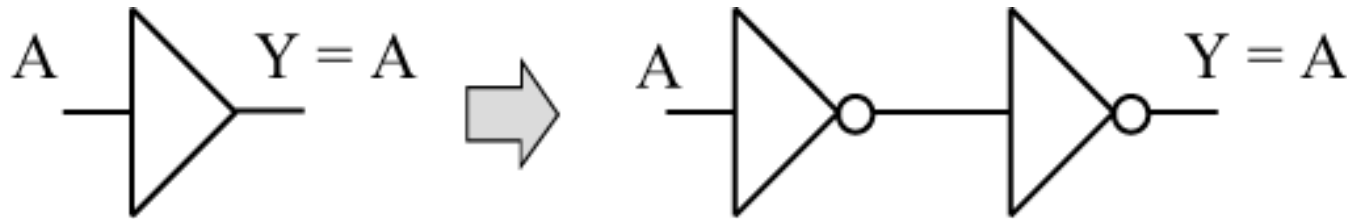


Inverter gate - takes 2 transistors



Copyright 2005. Thomson/Delmar Learning. All rights reserved.

Buffer - takes 4 transistors

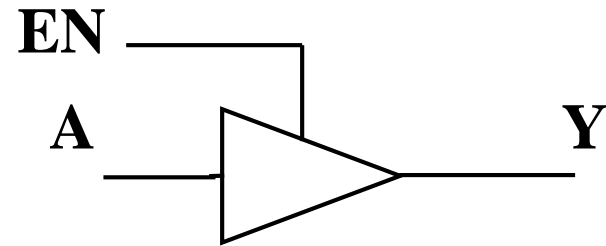
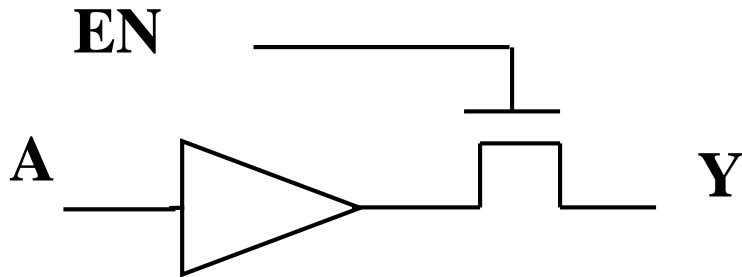


Copyright 2005. Thomson/Delmar Learning. All rights reserved.

In digital logic, NMOS must be connected to ground, PMOS to V_{DD} .

Tri-State Buffer

There is another way to drive a line or bus from multiple sources. Use a TRISTATE buffer.



When $EN = 1$, then $Y = A$.

When $EN = 0$, then $Y = ??????$

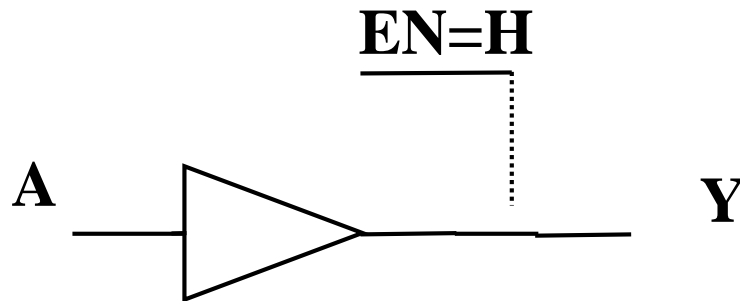
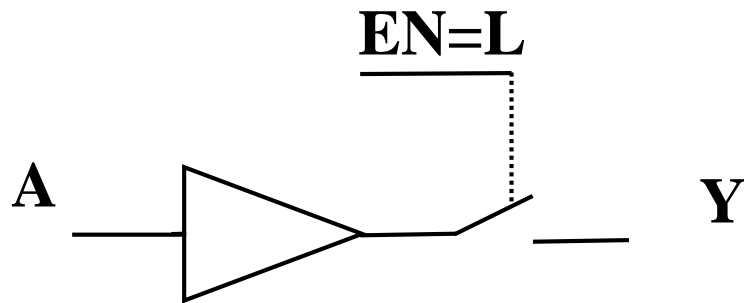
Y is *undriven*, this is called the **high impedance** state.

Designate high impedance by a 'Z'.

When $EN = 0$, then $Y = 'Z'$ (high impedance)

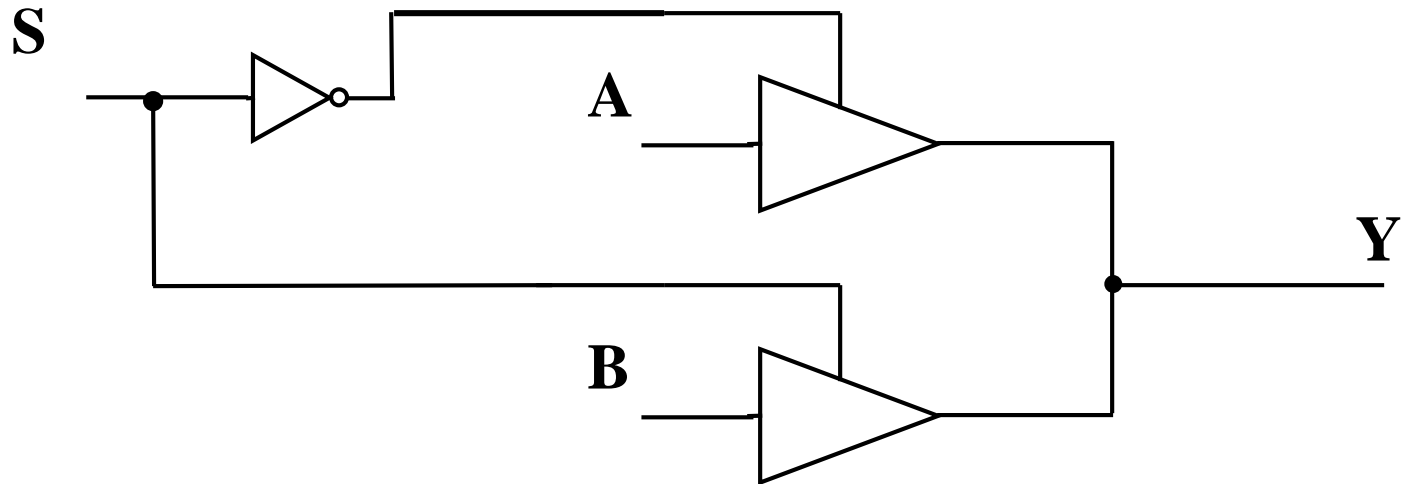
Tri-State Buffer

There is another way to drive a line or bus from multiple sources. Use a TRISTATE buffer.



Using Tri-State Buffers (cont)

Only A or B is enabled at a time.

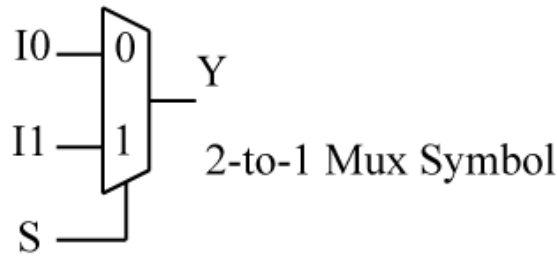


Implements 2/1 Mux function

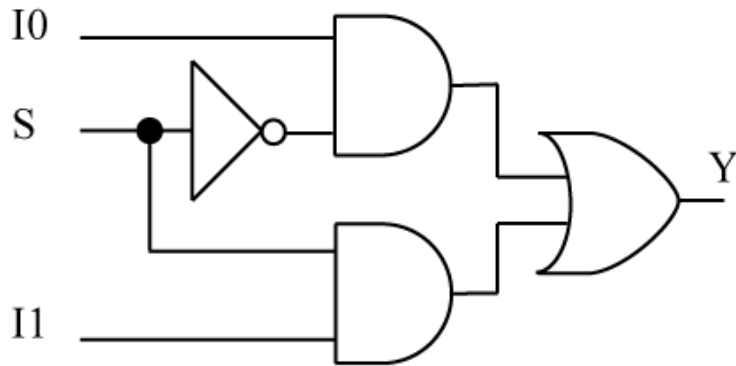
If $S=0$ then $Y = A$

If $S=1$ then $Y = B$

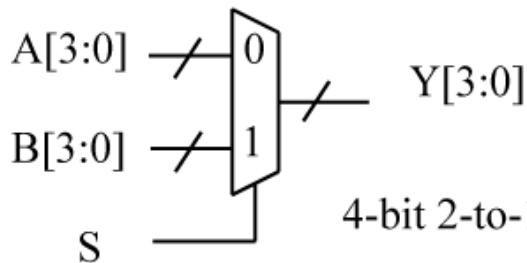
Combinational Building Blocks, Mux



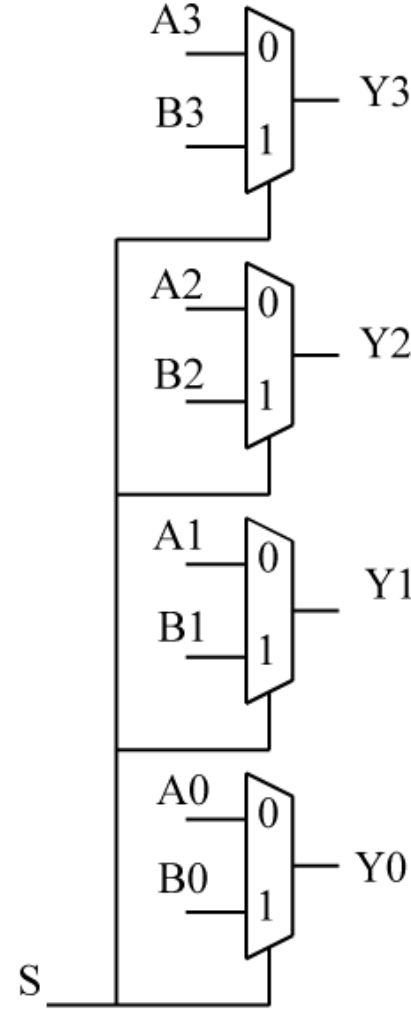
2-to-1 Mux Symbol



Gate level 2-to-1 Mux



4-bit 2-to-1 Mux

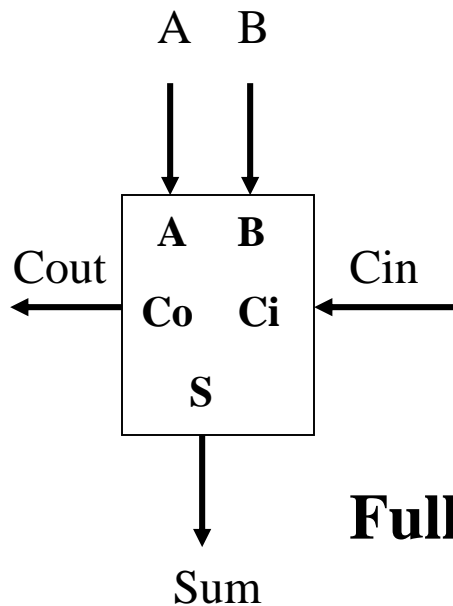


Expanded 4-bit 2-to-1 Mux

Binary Adder

$$\text{Sum (A,B,C)} = A \text{ xor } B \text{ xor } C \quad \text{Cout} = AB + AC + BC$$

These equations look familiar. These define a *Binary Full Adder* :

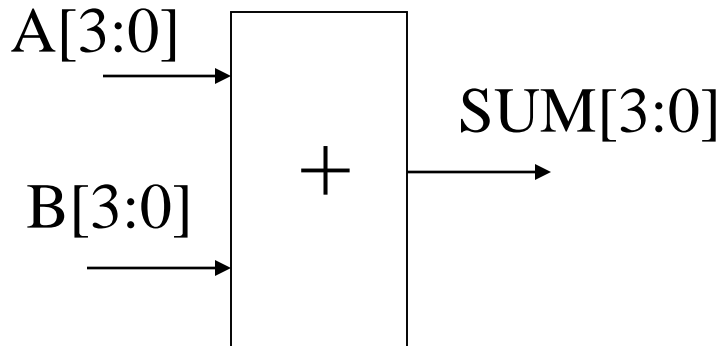
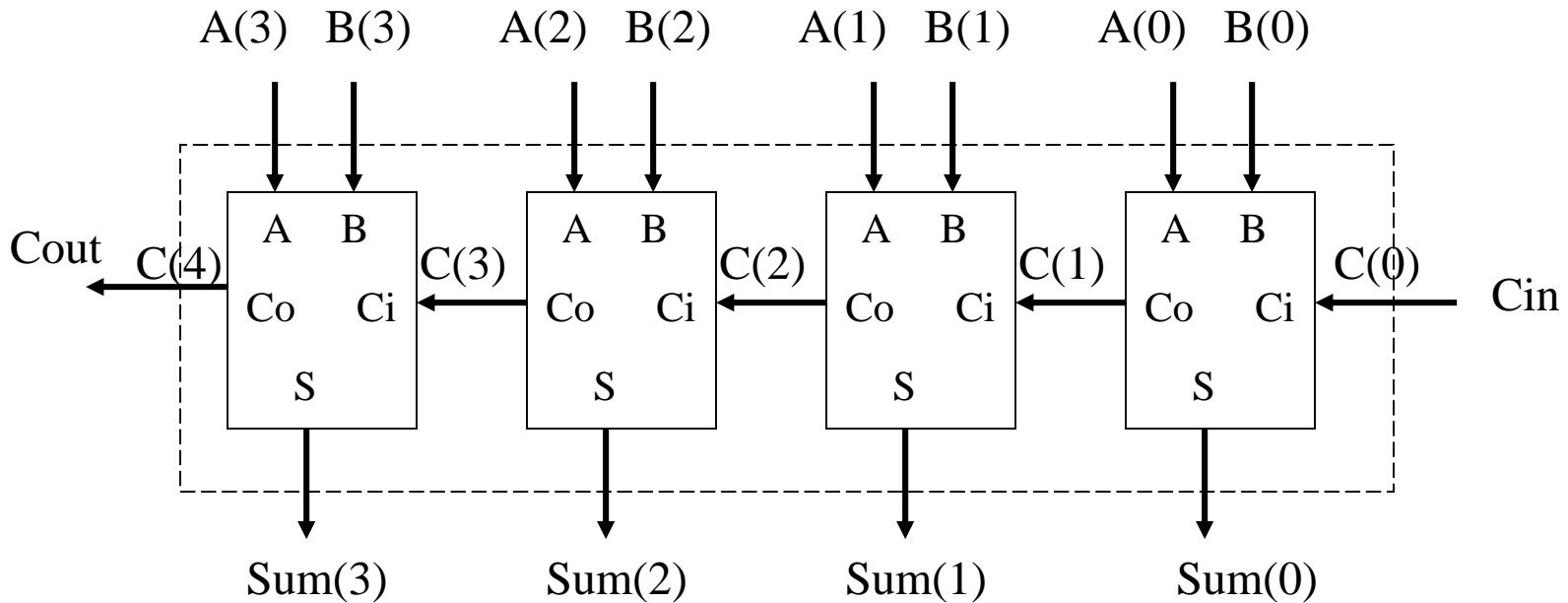


$$\text{Sum} = A \text{ xor } B \text{ xor } \text{Cin}$$

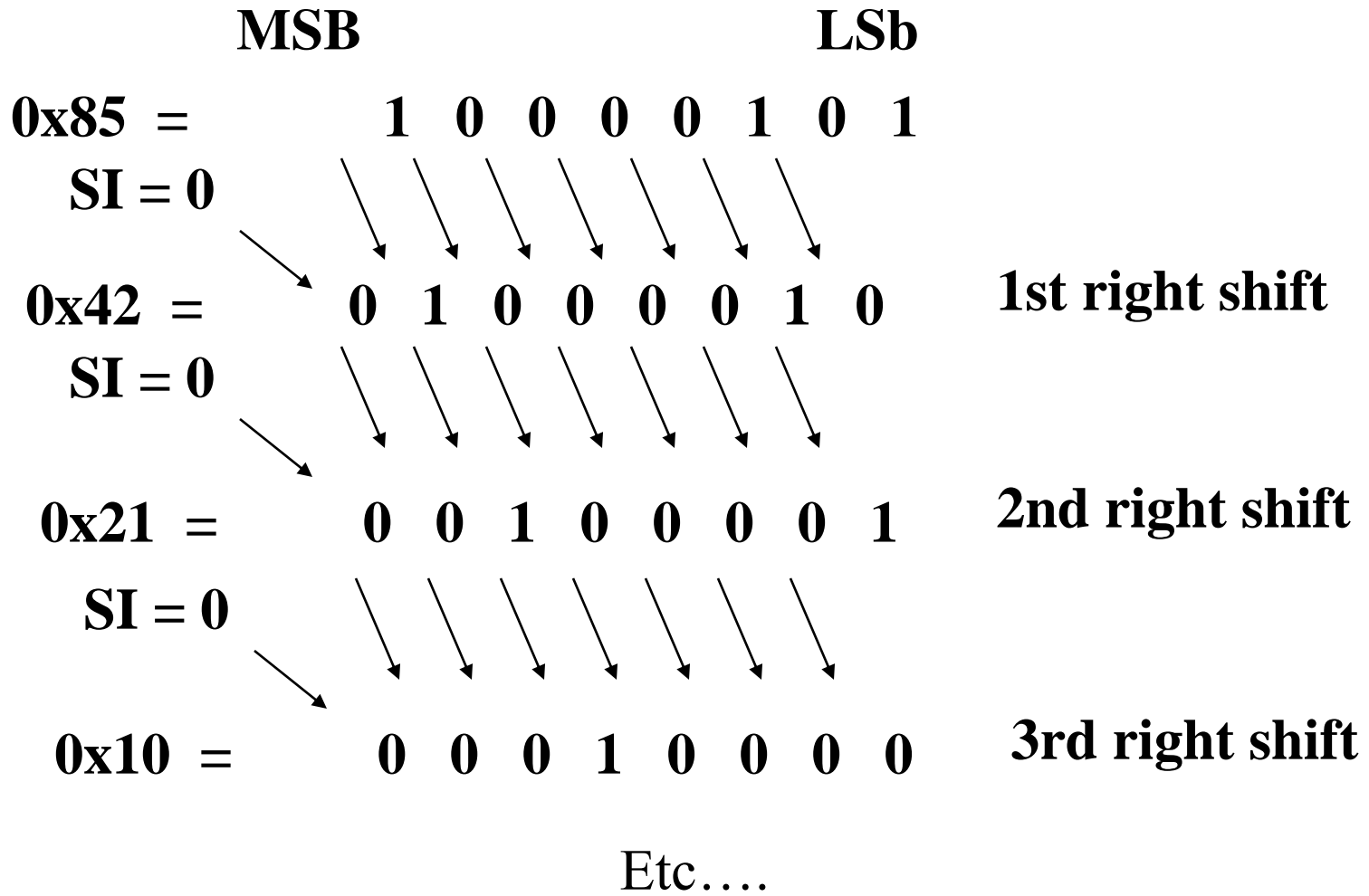
$$\begin{aligned} \text{Cout} &= AB + \text{Cin } A + \text{Cin } B \\ &= AB + \text{Cin } (A + B) \end{aligned}$$

Full Adder (FA)

4 Bit Ripple Carry Adder

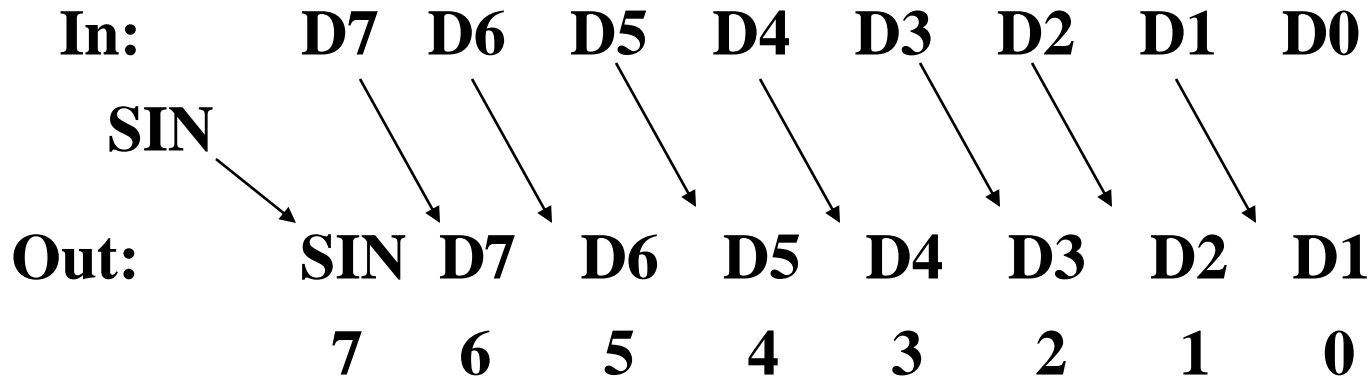


Understanding the shift operation

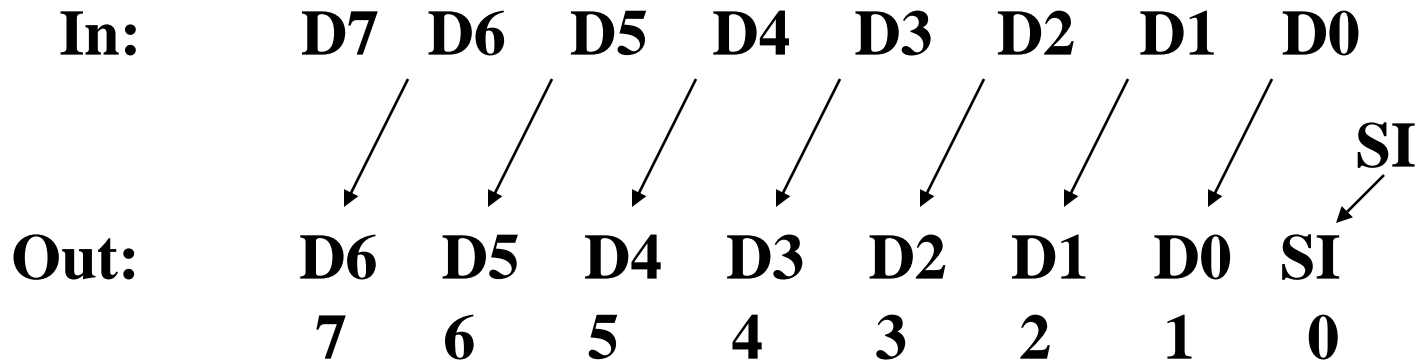


Right Shift vs. Left Shift

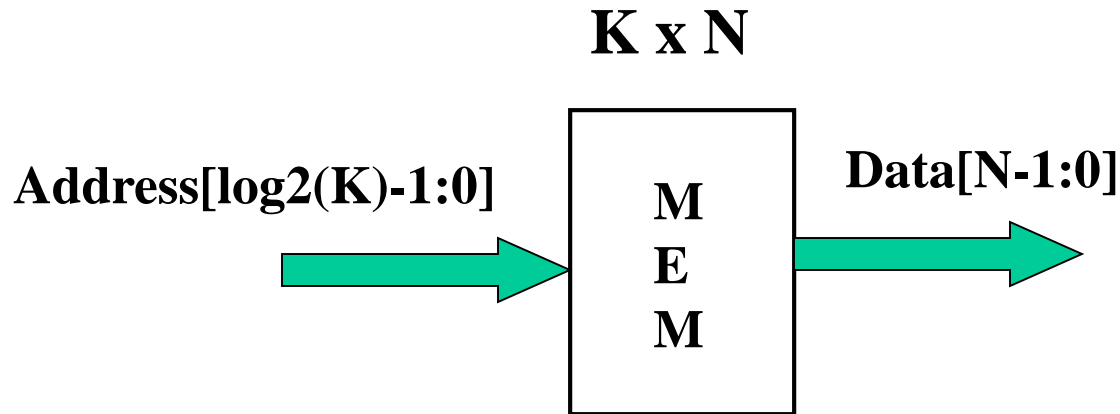
A **right shift** is MSB to LSB (divide by 2)



A **left shift** is LSB to MSB (multiply by 2)



Recall Basic Memory Definition



Example:

16 x 8

(16 locations requires
 $\log_2(16) = 4$ address lines,
each location stores 8 bits.

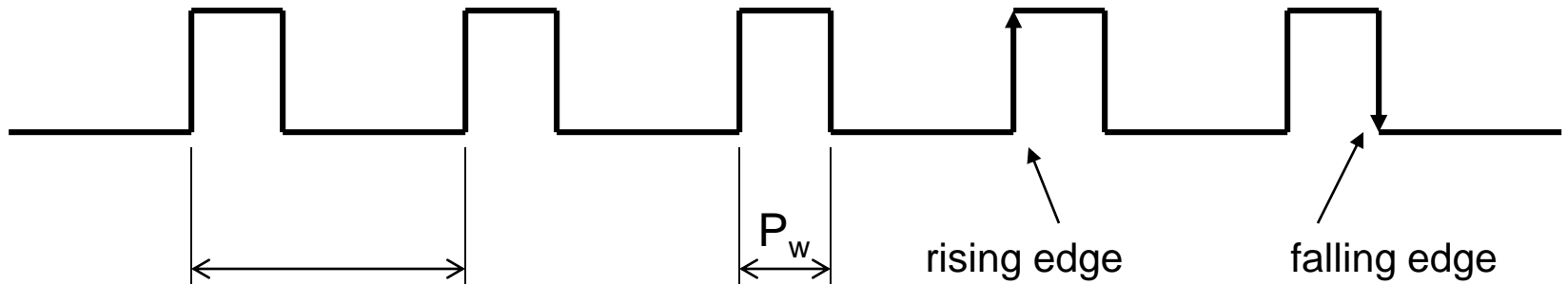
Address bus: A[3:0]

Data bus: D[7:0]

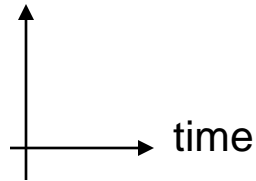
K locations, N bits per location

Address bus has $\log_2(K)$ address lines, data bus has N data lines.

Clock Signal Review



voltage



τ - period (in seconds)

P_w - pulse width (in seconds)

f - frequency pulse width (in Hertz)

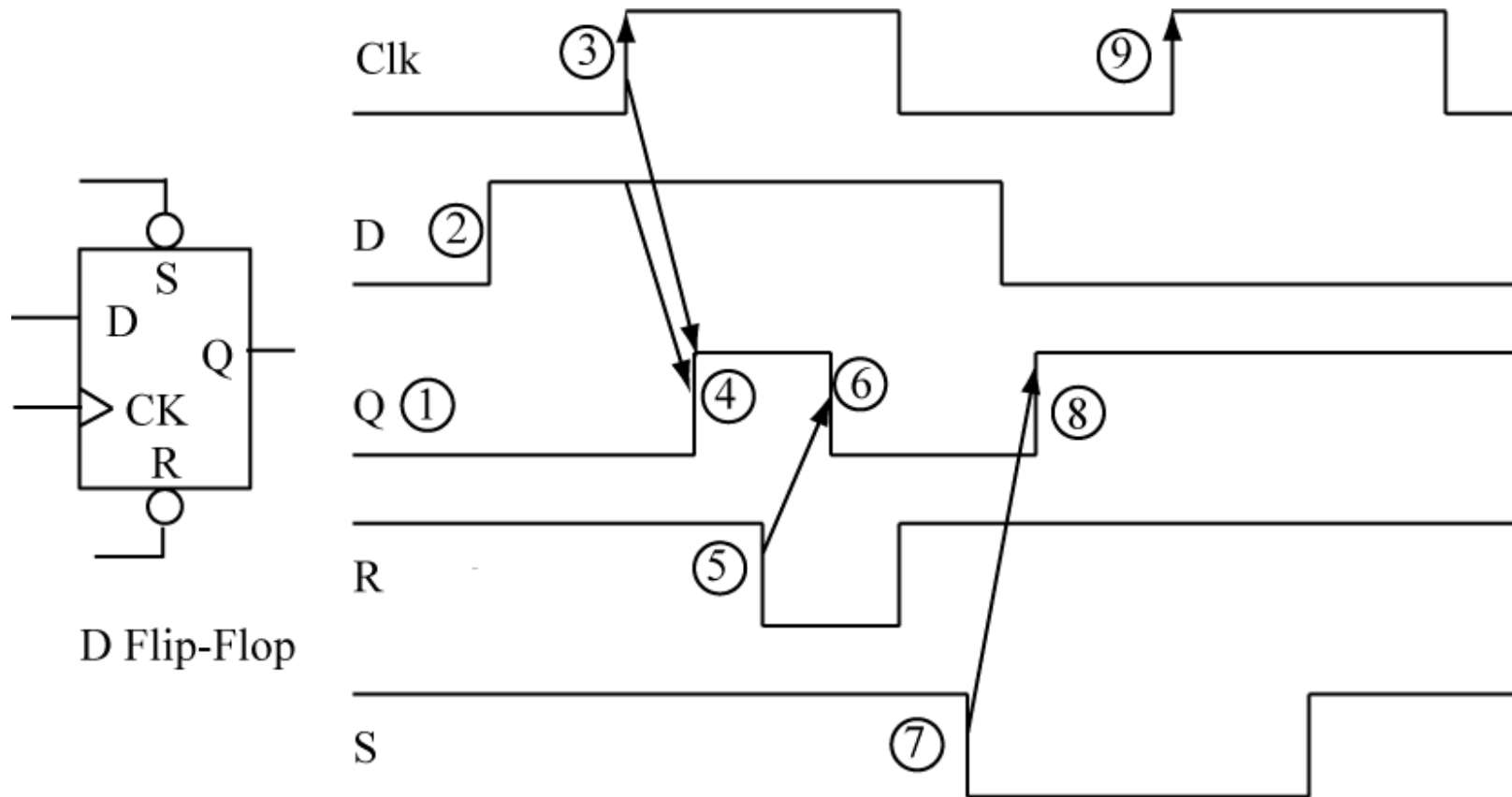
$$f = 1/\tau$$

duty cycle - ratio of pulse width to period (in %)

$$\text{duty cycle} = P_w / \tau$$

millisecond (ms) 10^{-3}	Kilohertz (KHz) 10^3
microsecond (μs) 10^{-6}	Megahertz (MHz) 10^6
nanosecond (ns) 10^{-9}	Gigahertz (GHz) 10^9

Storage Element: The D Flip-Flop



D: data input

CK: clock input

S : set input (asynchronous, low true)

R: reset input (asynchronous, low true)