Notes1 ECE 2500 Digital Logic
To the First Exam
©2026 ***Prof. Dean R. Johnson***

Lecture Topics:

- The Digital World
- Boolean Algebra
- Logic Gates & Circuits
- minterms and K-maps
- Maxterms and K-maps

THE DIGITAL WORLD

Major Application of Digital Logic: the design of processor chips in computers and mobile devices.

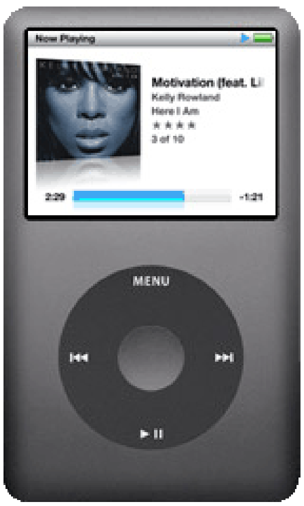
- **Classic iPod (4th generation 2004)**
  From: electronics.howstuffworks.com/ipod3.htm
  

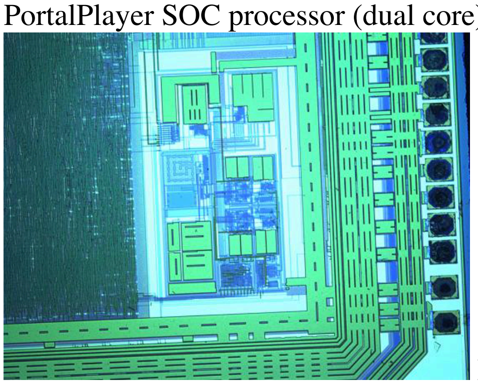
  Photo credit: apple.com
    - Display (320 x 240 pixel LCD)
      From: electronics.howstuffworks.com/lcd2.htm
    - Click Wheel (capacitive sensing controller)
      From: electronics.howstuffworks.com/ipod4.htm
    - PortalPlayer SOC processor (dual core)
      
      Photo credit: microblog.routed.net
    - Memory (SDRAM 32 MB)
    - Hard drive (30 GB)
- **iPhone** 17 Pro differences (2025)
  From: https://en.wikipedia.org/wiki/List_of_iOS_devices


Image credit: apple.com

- 6 GB memory
- 128-1012 GB solid-state drive
- Touch HDR display (2556 x 1179 pixel color OLED)
- 64 bit Apple A19 Pro SOC processor
  - Hex-core CPU
  - Hex-core GPU
  - 16-core NPU

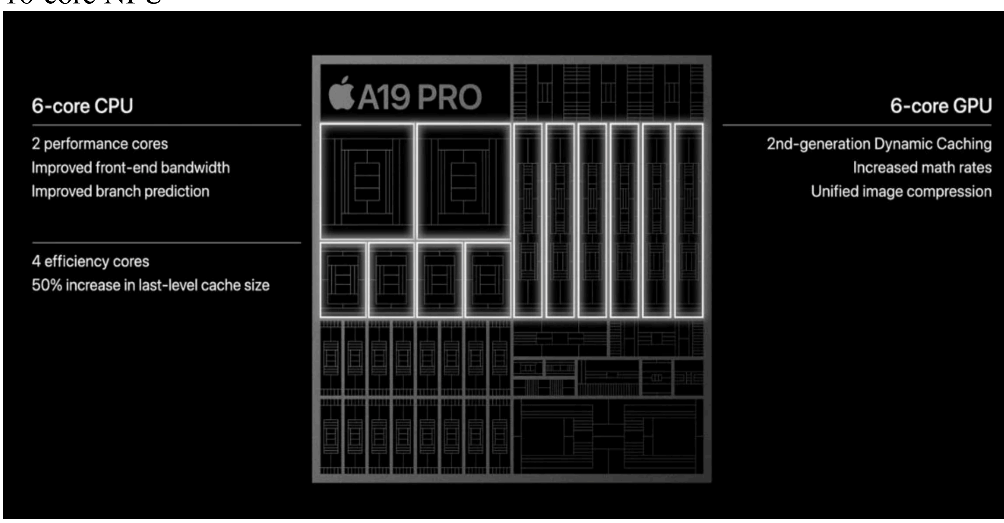
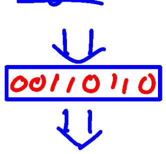


Image Credit: Apple

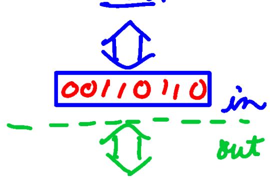
Digital Logic Components: these are the digital building blocks that will be studied in this course.

- **Register** (holds various forms of digital data)
- **Port** (a register interfacing data to/from the outside world)

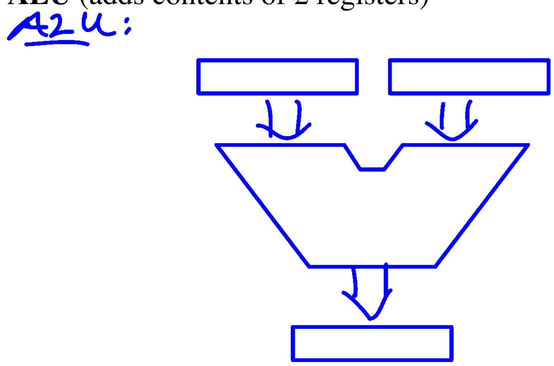
Register:



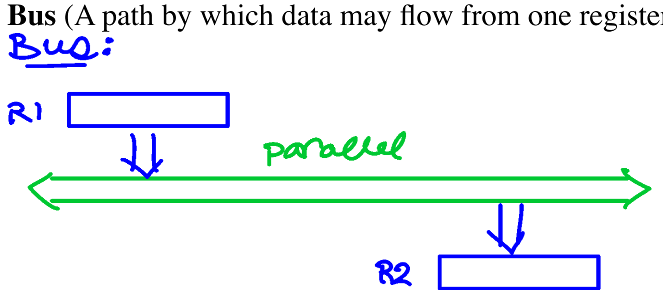
Port:



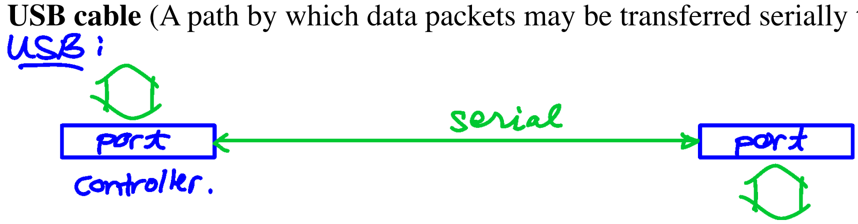
- **ALU** (adds contents of 2 registers)



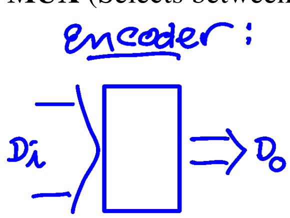
- **Bus** (A path by which data may flow from one register to another in parallel)

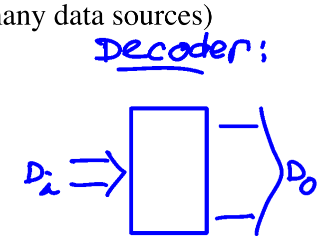


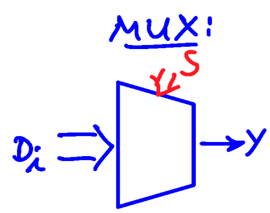
- **USB cable** (A path by which data packets may be transferred serially to ports from a hub)



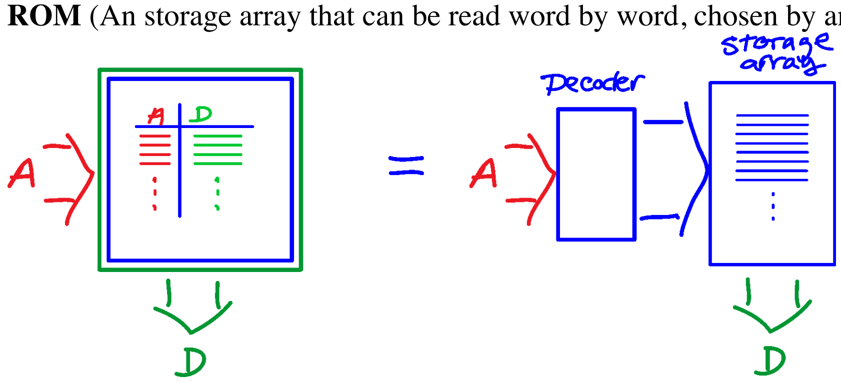
- **Encoder** (Encodes or compresses data)
- **Decoder** (Decodes or expands data. Also used to make memory location selections)
- **MUX** (Selects between many data sources)



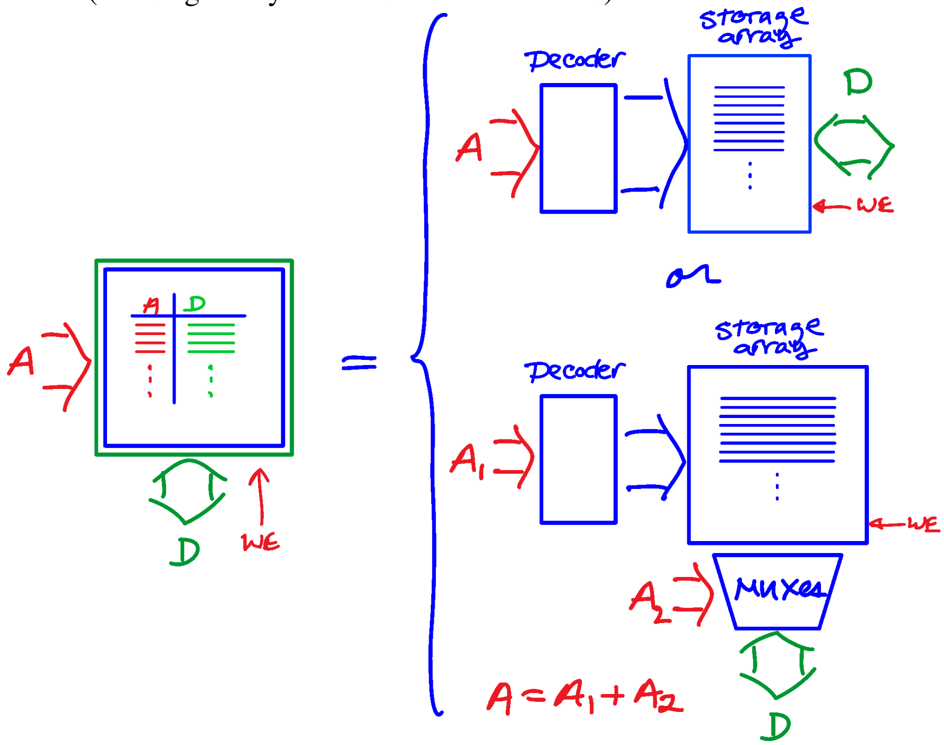




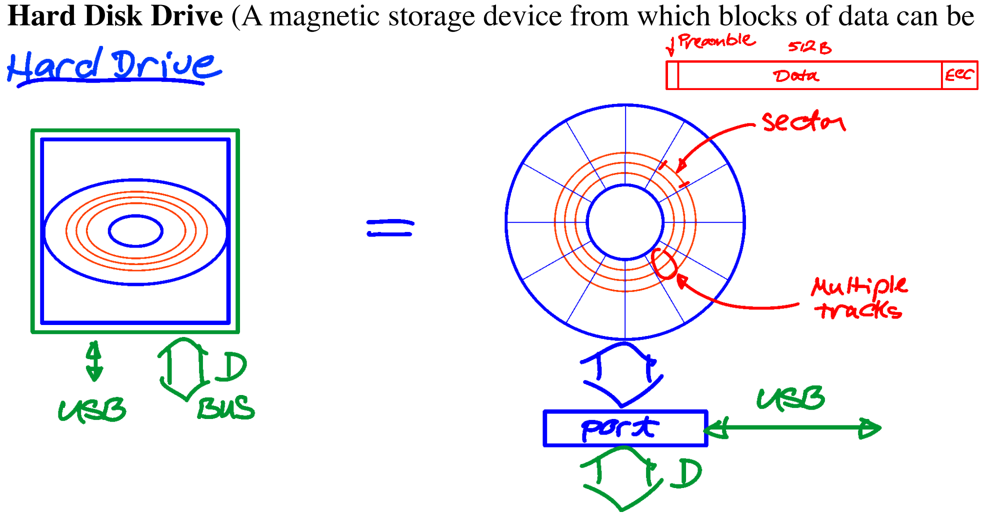
- **ROM** (An storage array that can be read word by word, chosen by an address)



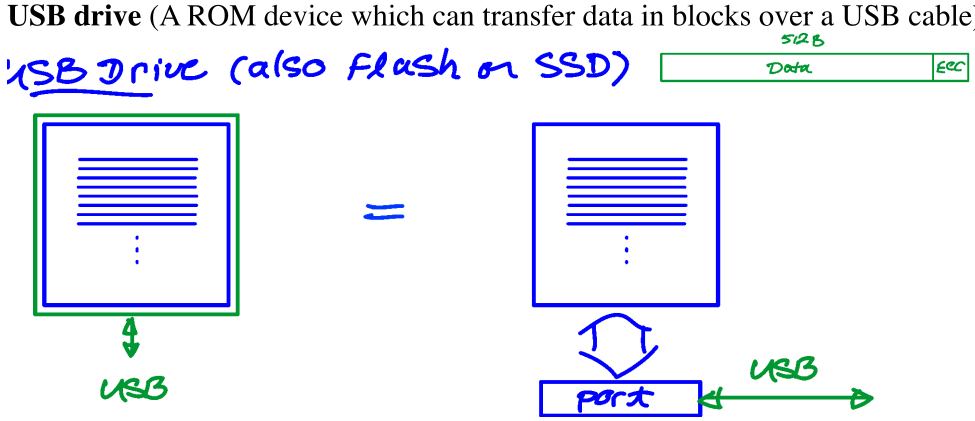
- **RAM** (A storage array that also can be written to)



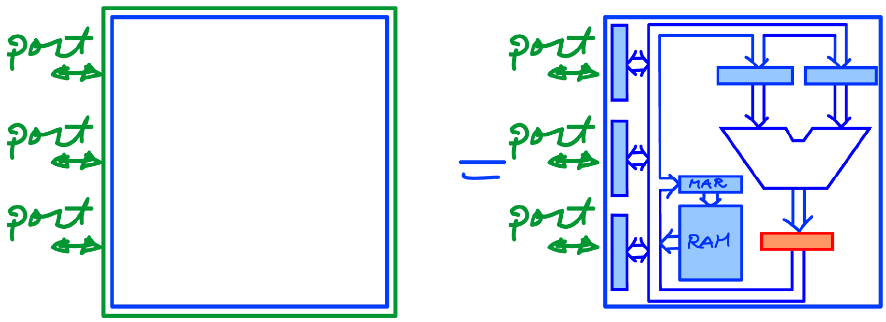
- **Hard Disk Drive** (A magnetic storage device from which blocks of data can be stored and read



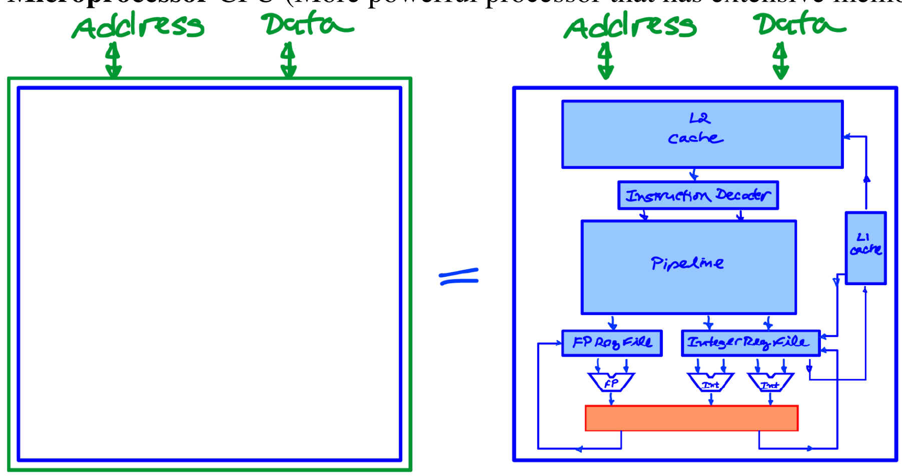
- **USB drive** (A ROM device which can transfer data in blocks over a USB cable)



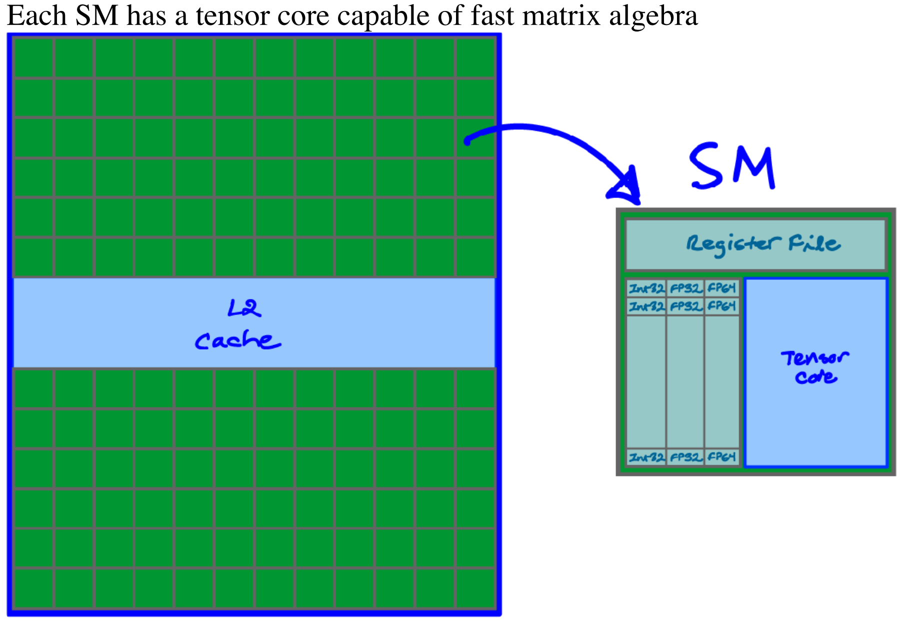
- **Microcontroller** MCU (A processing device consisting of an ALU, registers, ports and RAM)



- **Microprocessor** CPU (More powerful processor that has extensive memory and multiple ALUs)



- **Graphic Processing Unit** GPU
  - The H100 consists of 144 Streaming Multiprocessors (SM)
  - Each SM has a tensor core capable of fast matrix algebra



  - A GPT AI Large Language Model (LLM) application requires:
    - 64-256 GPUs for inference
    - 25,000 GPUs for training

Digital Data Types

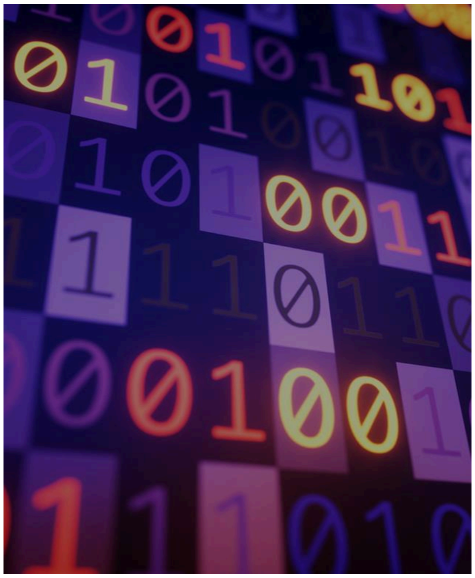
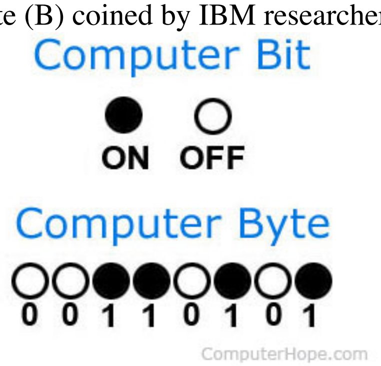
- **Numeric**
  
  Graphic credit: techspirited.com
  - Beginnings:
    - bit (b) defined by Claude Shannon as "basic information digit" (1948)
    - byte (B) coined by IBM researcher Werner Buchholtz (1964)
      

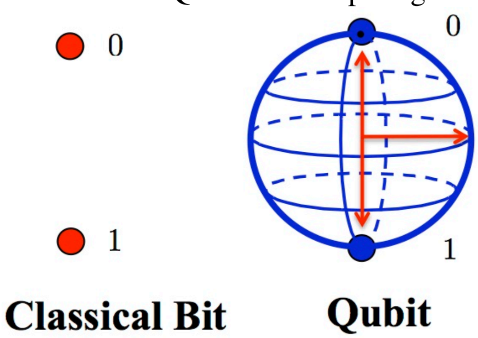
      Image credit: ComputerHope.com
    - A new bit used in Quantum Computing is the Qubit:
      

      Image credit: IBTimes UK
  - Integers in a byte (8 bits)

- Total unsigned (0 -> 255, 256 total members)

1 byte	decimal	hex	1 byte	decimal	hex	1 byte	decimal	hex
00000000	= 0	00	00001000	= 8	08	00010000	= 16	10
00000001	= 1	01	00001001	= 9	09	⋮	⋮	
00000010	= 2	02	00001010	= 10	0A	01100011	= 99	63
00000011	= 3	03	00001011	= 11	0B	01100100	= 100	64
00000100	= 4	04	00001100	= 12	0C	⋮	⋮	
						11001000	= 200	C8
						⋮	⋮	
00000101	= 5	05	00001101	= 13	0D	11111101	= 253	FD
00000110	= 6	06	00001110	= 14	0E	11111110	= 254	FE
00000111	= 7	07	00001111	= 15	0F	11111111	= 255	FF

- *Example*: Hexadecimal and decimal

  7 6 5 4 3 2 1 0

  $\boxed{11001000} = 1\times 2^7 = 128$
  $+1\times 2^6 = 64$
  $+1\times 2^3 = 8$

  C 8

  $\boxed{0xC8 = 200}$

- **C**omparison of decimal, binary, octal and hex:

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Red 1 and 2 = "Carry"
A,B,C,D,E,F = extra hex digits

Important number conversions to remember:

$(10)_{10} = (1010)_2 = (A)_{16}$
$(11)_{10} = (1011)_2 = (B)_{16}$

- Fractionals in a byte
  - *Example*:

    1 2 3 4 5 6 7 8

    $\boxed{.11001000} = 1\times 2^{-1} = .5$
    $+1\times 2^{-2} = .25$
    $+1\times 2^{-5} = .03125$

    . C 8

    $0x.C8 = .78125$

  - Integer conversions between binary, octal and hex
    - Octal: group in3 bits
    - Hex: group in 4 bits
    - *Example #1:*

      Convert $(010101100)_2$ to base 8 and 16

      Add zeros

      2 5 4 → $(254)_8$

      $000(010\,101\,100)_2$

      0 A C → $(AC)_{16}$

- *Example #2:*
  Convert (110.110) $_2$ to base 8 and 16

  Add zeros

  $0(110.110)_2 0$

  Add zeros

  $\overbrace{110}^{6}.\overbrace{110}^{6} \rightarrow (6,6)_8$

  $\underbrace{0110}_{6}.\underbrace{1100}_{C} \rightarrow (6,C)_{16}$

- Juxtapositional notation:
  - Integer, radix point and fraction

    N = number

    $= (a_{n-1}\ a_{n-2} \ldots a_2\ a_1\ a_0\ .\ a_{-1}\ a_{-2}\ a_{-3} \ldots a_{-m})_r$

    Integer | Fraction

    radix point

    a_i = digits

    r = radix

    $0 \le a_i < r$, always holds.

  - *Examples:* (radix = base)

    $(353.12)_{r=10}$

    $= 3 \times 10^2 +$
    $5 \times 10^1 +$
    $3 \times 10^0 +$
    $1 \times 10^{-1} +$
    2×10^{-2}

    $(1010.01)_{r=2}$

    $= 1 \times 2^3 +$
    $0 \times 2^2 +$
    $1 \times 2^1 +$
    $0 \times 2^0 +$
    $0 \times 2^{-1} +$
    1×2^{-2}

    General Polynomial:

    N = number

    $$= \sum_{i=-m}^{n-1} a_i r^i$$

    $N = N_I + N_F$

    $= a_{n-1} r^{n-1} +$
    $a_{n-2} r^{n-2} +$
    $a_{n-3} r^{n-3} +$
    $\ldots +$
    $a_2 r^2 +$
    $a_1 r^1 +$
    $a_0 r^0 +$

    $a_{-1} r^{-1} +$
    $a_{-2} r^{-2} +$
    $a_{-3} r^{-3} +$
    $\ldots +$
    $a_{-m} r^{-m}$

- **Non-numeric**
  - Characters
    - ASCII: 1 B for each of 2^8 = 256 English, control and special characters (Latin-1)

From: http://www.asciitable.com/

character 1 byte

ASCII: "A" = $\boxed{0100\ 0001}$ = 41

"B" = $\boxed{0100\ 0010}$ = 42

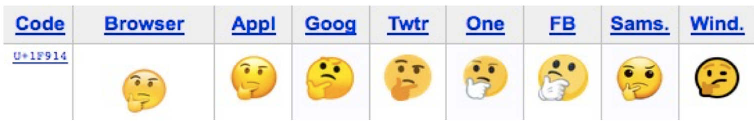
"C" = $\boxed{0100\ 0011}$ = 43

⋮

"ECE 2500" = 45 43 45 20 32 35 30 30
E C E sp 2 5 0 0

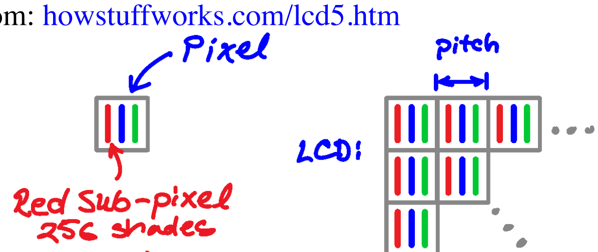
= 8 bytes

- UNICODE: 2^{24} ~17 million characters with code points spread over 2 or 3B (UTF-16 or 24). Handles international characters & emoticons



From: https://unicode.org/emoji/charts/full-emoji-list.html

- Color Codes
From: howstuffworks.com/lcd5.htm

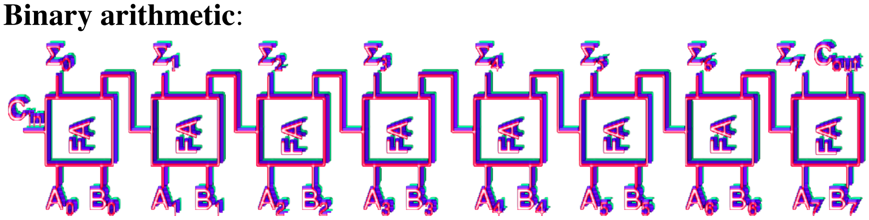


RGB CODES:

R	G	B	
10011001 (9 9)	10011001 (9 9)	10011001 (9 9)	= Gray
11001100 (C C)	0000 0000 (0 0)	0000 0000 (0 0)	= Red
0000 0000 (0 0)	1111 1111 (F F)	0000 0000 (0 0)	= Green

- More html examples: immigration-usa.com/html_colors.html
- 24 bit color -> 2^{24} ~ 17 million colors
  - Red 1 B => 256 shades
  - Green 1 B => 256 shades
  - Blue 1 B => 256 shades

Binary arithmetic:



- Bit by bit addition is done right to left, with **carry bits**

- *Examples:* Adding

$$\begin{array}{r} 3 \\ +4 \\ \hline 7 \end{array} \quad \begin{array}{r} 00011 \\ +00100 \\ \hline 00111 \end{array} \qquad \begin{array}{r} 5 \\ +5 \\ \hline 10 \end{array} \quad \begin{array}{r} 00101 \\ +00101 \\ \hline 01010 \end{array} \qquad \begin{array}{r} 5 \\ +7 \\ \hline 12 \end{array} \quad \begin{array}{r} 00101 \\ +00111 \\ \hline 01100 \end{array}$$

- Subtraction can be done by employing **borrow bits**, or more simply, by adding something called a 2's complement.
  - *Examples:*

$$\begin{array}{r} 10 \\ -6 \\ \hline 4 \end{array} \quad \begin{array}{r} 01010 \\ -00110 \\ \hline 00100 \end{array} \qquad \begin{array}{r} 10 \\ +(-6) \\ \hline 4 \end{array} \quad \begin{array}{rl} 01010 & \\ +11010 & (2\text{'s C}) \\ \hline 1\,\boxed{00100} & \end{array}$$

↑ overflow

- The concept of a *complement* of a decimal number:

$-N$ 10's C

$-1 \rightarrow 9$
$-2 \rightarrow 8$
$-3 \rightarrow 7$
$-4 \rightarrow 6$
$-5 \rightarrow 5$
$-6 \rightarrow 4$
$-7 \rightarrow 3$
$-8 \rightarrow 2$
$-9 \rightarrow 1$

$$\begin{array}{r} 10 \\ -6 \\ \hline 4 \end{array} \rightarrow \begin{array}{r} 10 \\ +4 \\ \hline 1\boxed{4} \end{array}$$

$$\begin{array}{r} 8 \\ -6 \\ \hline 2 \end{array} \rightarrow \begin{array}{r} 8 \\ +4 \\ \hline 1\boxed{2} \end{array}$$

$$\begin{array}{r} 7 \\ -9 \\ \hline -2 \end{array} \rightarrow \begin{array}{r} 7 \\ +1 \\ \hline \boxed{8} \end{array} \rightarrow -2$$

- 2's complement procedure:
  - Reverse all the bits of ***N***
  - Add ***1*** to the result. This is ***N****.
  - The sign of ***N**** (as well as *N*) is shown by the most significant bit: 0 = "+"; 1 = "-"
  - *Examples:*

sign bit = '+'

$$\begin{array}{rrl} N = 6 = & \boxed{0}0110 & \\ & 11001 & \text{step\#1} \\ & +1 & \text{step\#2} \\ \hline -N = N^* = -6 = & \boxed{1}1010 & \end{array}$$

↑ sign bit = '-'

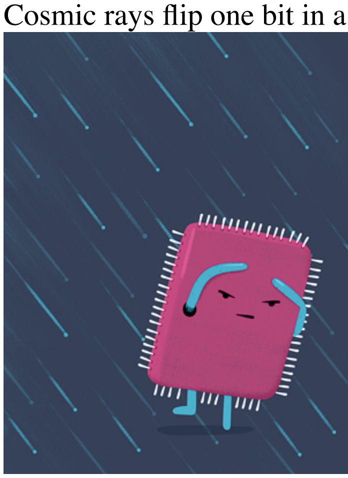
$$\begin{array}{rrl} N^* = & \boxed{1}1110 & \text{what is } -N? \\ & 00001 & \text{step\#1} \\ & +1 & \text{step\#2} \\ \hline & \boxed{0}0010 = N & \therefore -N = -2 \end{array}$$

  - All the 2's complement numbers that fit into a byte.
    - 127 positive numbers *N* (sign bit = 0)
    - 128 negative numbers *N** (sign bit = 1)

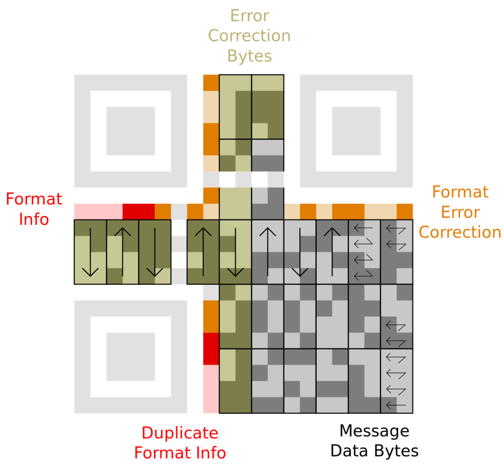
- Zero (not shown)

N	decimal	$N^* = -N$	decimal
00000001	= 1	11111111	= -1
00000010	= 2	11111110	= -2
00000011	= 3	11111101	= -3
00000100	= 4	11111100	= -4
00000101	= 5	11111011	= -5
00000110	= 6	11111010	= -6
⋮	⋮	⋮	⋮
01111110	= 126	10000010	= -126
01111111	= 127	10000001	= -127
x	= 128	10000000	= -128

Error Correction Codes (ECC):

- Provides self-correction of errors that occur in the data when transporting data
  - Scratched disk alter recorded data:
    
    From: hardwaresecrets.com/
  - Cosmic rays flip one bit in a 4GB chip everyday:
    
    From: spectrum.ieee.org/

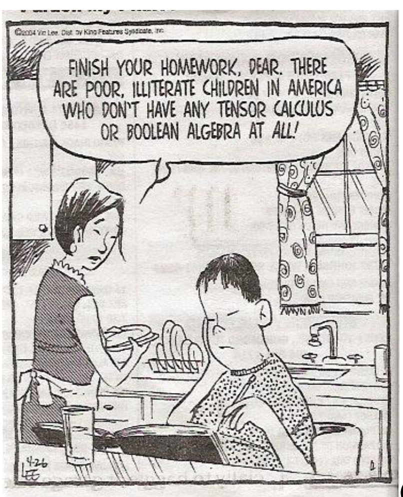
- Defaced QR code:
   From: hwww.i-programmer.info
- Reed-Solomon ECC code in QR:

  en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders

Quiz #1 & selected solutions

BOOLEAN ALGEBRA



(Photo credit: Vic Lee, King Features Syndicate)

Some Preliminaries...

Binary numbers can also be used to represent truth or **logic values**.

Logic defined: the process of classifying information.

Binary logic (or more commonly, *digital logic*) is the process of classifying information into two distinct classes, e.g.

(TRUE, FALSE) = truth values
(Yes, No)
(CLOSE, OPEN) = relay positions
blown, intact = fuse state
(ON, OFF) = switch positions
(1, 0) = binary numbers, or (Logic 1, Logic 0)

Logic design is based upon the **three logic operators**

Binary Logic Operations (Variables)

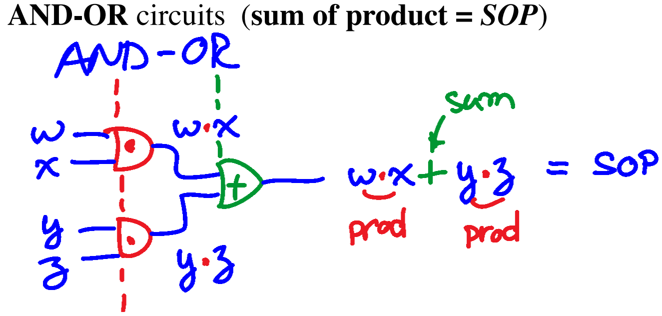
- **AND**: $z = x \bullet y$
- **OR**: $z = x + y$
- **NOT**: $z = x'$

Binary Logic Operations

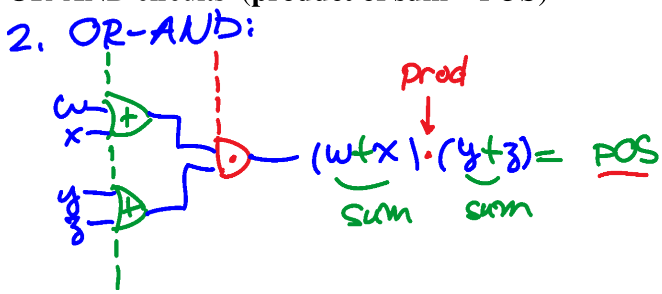
OR	**XOR**	**AND**
0 + 0 = 0	**0 ⊕ 0 = 0**	**0 • 0 = 0**
0 + 1 = 1	**0 ⊕ 1 = 1**	**0 • 1 = 0**
1 + 0 = 1	**1 ⊕ 0 = 1**	**1 • 0 = 0**
1 + 1 = 1	**1 ⊕ 1 = 0**	**1 • 1 = 1**

Two Level Logic Circuits with AND/OR/XOR gates:
From: computer.howstuffworks.com/boolean1.htm

- **AND-OR** circuits (**sum of product = *SOP***)



- **OR-AND** circuits (**product of sum = *POS***)



These circuits can also be described algebraically with the use of an algebra system for logic variables called…

Boolean Algebra

- **Fundamental properties of Boolean Algebra**: Each ***x, y*** and ***z*** are elements of ***B = {0,1}***
  1. ***Identities***: (P3, P4) (Dual)
     $x+0 = x$ $\qquad$ $x \bullet 1 = x$
     $x+1 = 1$ $\qquad$ $x \bullet 0 = 0$
     Also Idempotency: (P6)
     $x+x = x$ $\qquad$ $x \bullet x = x$
  2. ***Commutativity***: (P1)
     $x+y = y+x$ $\qquad$ $x \bullet y = y \bullet x$
  3. ***Associativity***: (P2)

$x+(y+z) = (x+y)+z$ $\quad$ $x\bullet(y\bullet z) = (x\bullet y)\bullet z$

4. ***Distributivity***: (P8)
   $x+(y\bullet z) = (x+y)\bullet(x+z)$ $\quad$ $x\bullet(y+z) = x\bullet y+x\bullet z$
5. ***Existence of the complement***: (P5)
   There exists an element x', called NOT x, such that
   $x+x' = 1$ $\quad$ $x\bullet x' = 0$
6. ***Involution***: (P7) $\quad$ $(x')' = x$
7. ***Absorption***: (P12)
   $x+xy = x$ $\quad$ $x(x+y) = x$
8. ***Adjacency***: (P9)
   $xy+xy' = x$ $\quad$ $(x+y)\bullet(x+y') = x$
9. ***DeMorgan's Law***: (P11)
   $(x+y+z)' = x'y'z'$ $\quad$ $(x\bullet y\bullet z)' = x'+y'+z'$

- **Duality**: Left and right hand properties above are *duals*
  - A dual may be derived by interchanging
    - 1 and 0
    - • (AND) and + (OR)
  - *Examples*:

$x+0=x$ $\quad$ $x\cdot(x+y)=x$

$x\cdot 1=x$ $\quad$ $x+xy=x$

Boolean Functions and Logic Circuits

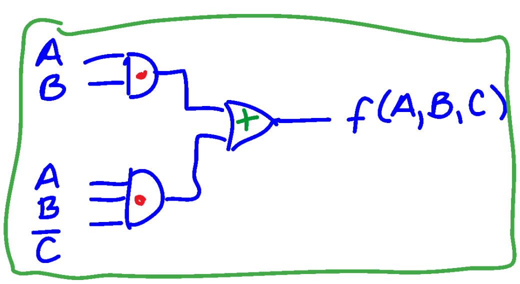
- Boolean function f
  - $f(A, B, C)$ is an algebraic expression of A, B, C
  - A, B, C are Boolean variables
- Boolean functions are implemented by logic circuits
- Boolean functions may be simplified, resulting in simpler logic circuits
- Circuits and functions may be verified by constructing a truth table
- *Example #1:*
  - Derive a logic circuit from a Boolean function:

$$f(A,B,C) = A\cdot B \overset{\text{sum}}{+} A\cdot B\cdot\overline{C} = \text{SOP}$$

products

$= \text{sum of products}$

Circuit 1

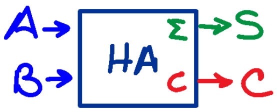


- *Example #2:*

- Derive a Boolean function for a half adder:

$$\begin{array}{r} A \\ +\,B \\ \hline C\;S \end{array} \qquad \begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$$

S, C

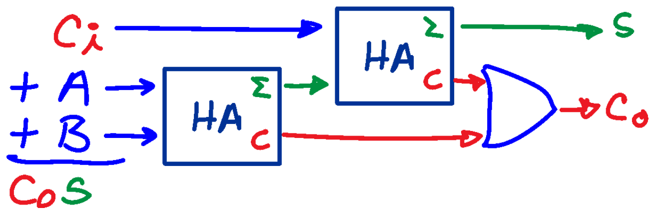


$C = AB$

$S = \bar{A}B + A\bar{B}$

$= A \oplus B$

- Make a full adder from two half adders:



- *Example#3*
  - Simplify the Boolean function of *Example#1* by pattern matching terms with the Boolean properties above:

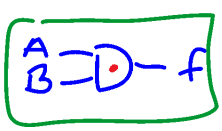
Boolean Algebra
(pattern matching)

$f = AB + AB\bar{C}$ ✓

$\underbrace{x + xy = x}_{\text{pattern}}$

Let $\begin{cases} x = AB \\ y = \bar{C} \end{cases}$

$\therefore f = \underline{\underline{AB}}$



Circuit ②

  - Compare before and after circuits with a truth table:

Truth Table:

ABC	② $A \cdot B$	minterm $AB\bar{C}$	① $f = AB + AB\bar{C}$
000	0	0	0+0=0
001	0	0	0+0=0
010	0	0	0+0=0
011	0	0	0+0=0
100	0	0	0+0=0
101	0	0	0+0=0
110	1	1	1+1=1
111	1	0	1+0=1

← same →

- *Example #4:* More simplifications

$$f = A\bar{B}\bar{C} + \bar{B}$$

$$A\bar{B}\bar{C} + \bar{B}$$

$$\underbrace{xy + x}_{\text{pattern}} = x$$

$$\text{Let} \begin{cases} x = \bar{B} \\ y = A\bar{C} \end{cases}$$

$$\therefore f = x = \bar{B} \checkmark$$

$$f = ABC + A\bar{B}C$$

$$ABC + A\bar{B}C$$

$$\underbrace{xy + x\bar{y}}_{\text{pattern}} = x$$

$$\text{Let} \begin{cases} x = AC \\ y = \bar{B} \end{cases}$$

$$\therefore f = x = AC \checkmark$$

- *Example #5:* DeMorgan's Law

$$F = \bar{A}B + C + \bar{D}$$

$$\bar{F} = ?\ \overline{\underbrace{\bar{A}B}_{x} + \underbrace{C}_{y} + \underbrace{\bar{D}}_{z}}$$

pattern:

$$\overline{x+y+z} = \bar{x}\cdot\bar{y}\cdot\bar{z}$$

$$\bar{F} = \overline{\bar{A}B} \cdot \bar{C} \cdot \bar{\bar{D}}$$

$xy \rightarrow$ pattern: $\overline{x\cdot y} = \bar{x}+\bar{y}$

$$= (\bar{\bar{A}}+\bar{B})\cdot \bar{C} \cdot D$$

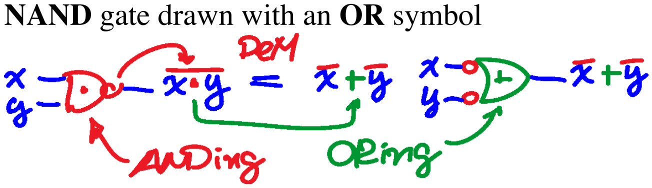
$$= (A+\bar{B})\,\bar{C}\cdot D$$

Quiz #2 & selected solutions

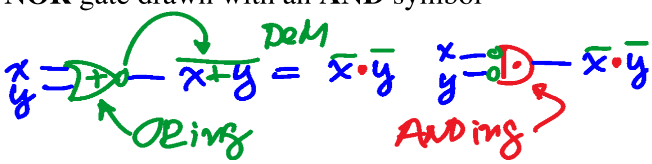
LOGIC GATES AND CIRCUITS

DeMorgan's Laws Shows Equivalent Graphical Symbols for Logic Gates: Examples are given to describe

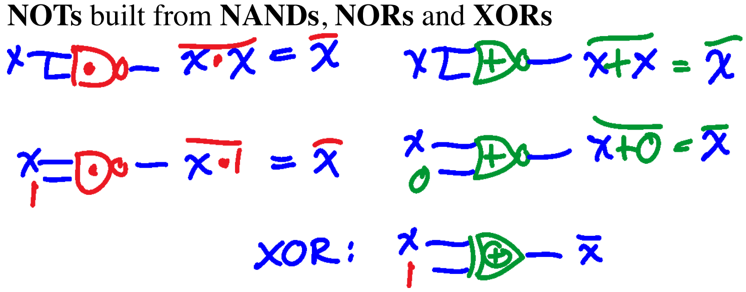
- **NAND** gate drawn with an **OR** symbol



- **NOR** gate drawn with an **AND** symbol

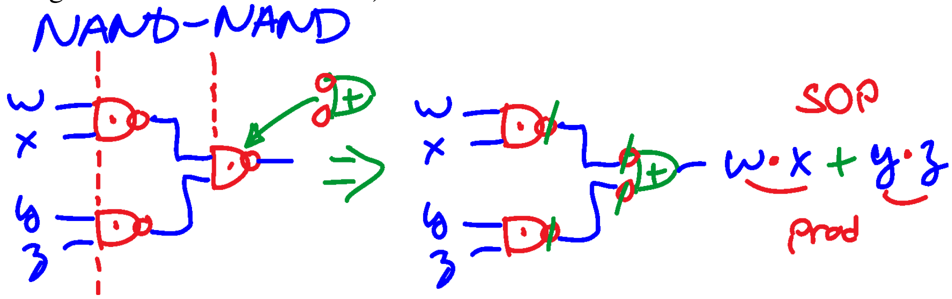


- **NOTs** built from **NANDs**, **NORs** and **XORs**

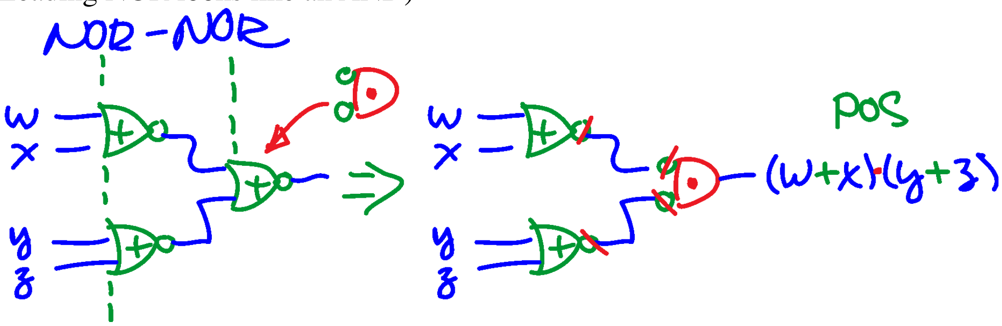


Two Level Logic Circuits with Other Gates: Examples are given to describe

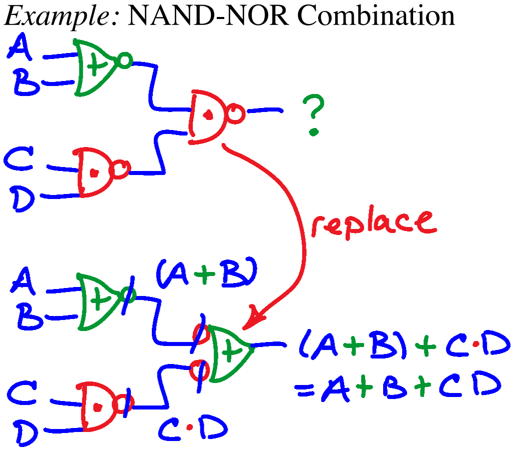
- **NAND-NAND** circuits = **AND-OR** circuits; makes ***SOP*** functions
  (Leading NAND looks like an OR)



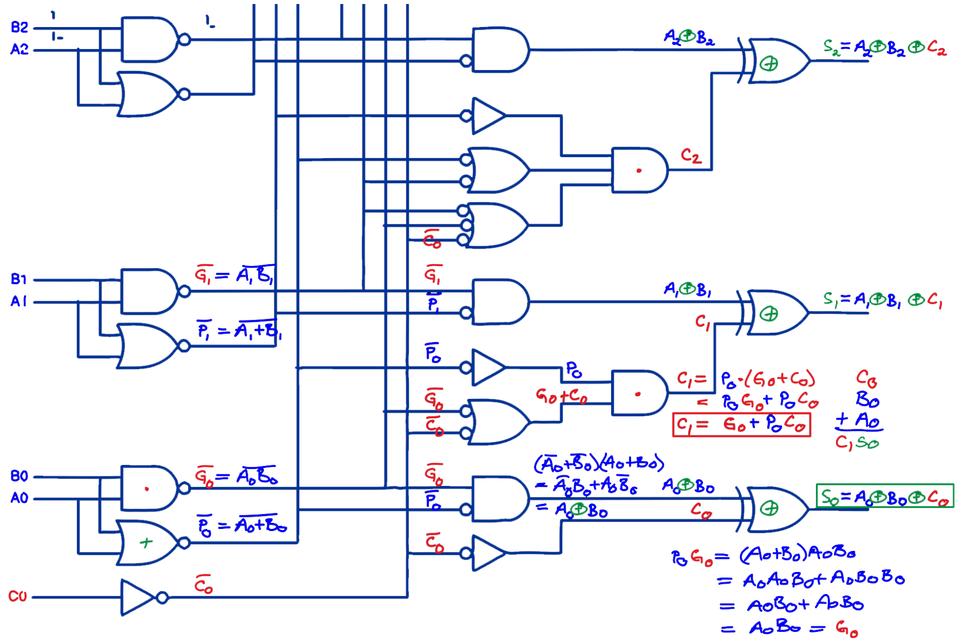
- **NOR-NOR** circuits = **OR-AND** circuits; makes ***POS*** functions
  (Leading NOR looks like an AND)



- *Example:* NAND-NOR Combination

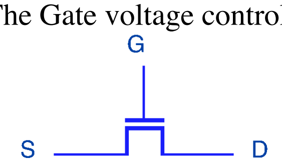


- *Example:* Carry-lookahead adder logic
  - Most heavily designed circuit in the history of electronics
  - NOT, NAND, NOR, XOR combination
  - Gate fronts and backs match so bubbles cancel



CMOS Implementation of Logic Gates
Examples are shown to implement **NAND**, **NOR**, and **NOT** gates from elementary **NMOS** and **PMOS** transistors.

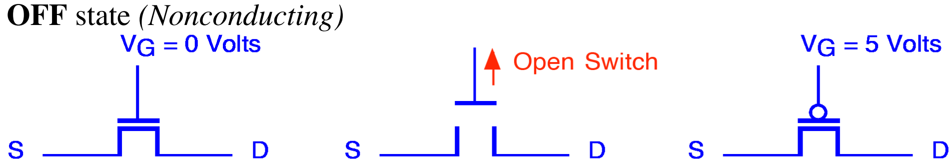
- **CMOS** transistors = NMOS plus PMOS
- Current flows between the Source and the Drain
- The Gate voltage controls the conduction value between the Source and Drain:



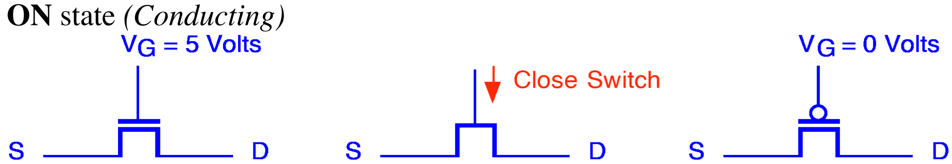
G = Gate
S = Source
D = Drain
(S and D are interchangeable)

The NMOS transistor may be configured to operate in one of three different states, as determined by the **voltage** at the gate terminal V_G:

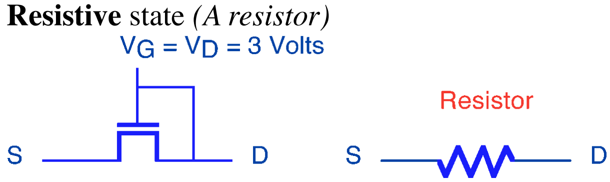
- Three states of a **NMOS** and **PMOS** transistors:
  - **OFF** state *(Nonconducting)*



  - **ON** state *(Conducting)*

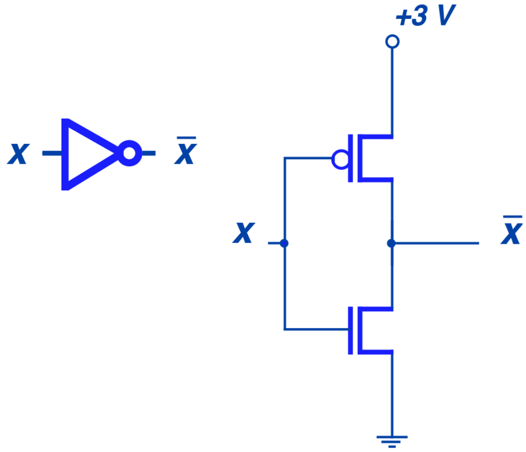


  - **Resistive** state *(A resistor)*

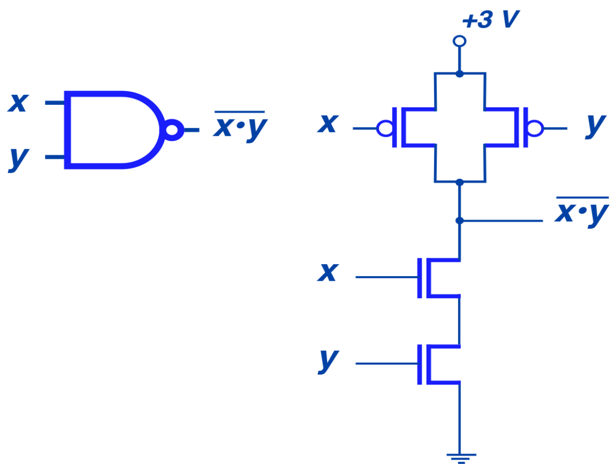


- **NAND**, **NOR** and **NOT** gates can be constructed from two to four transistors.

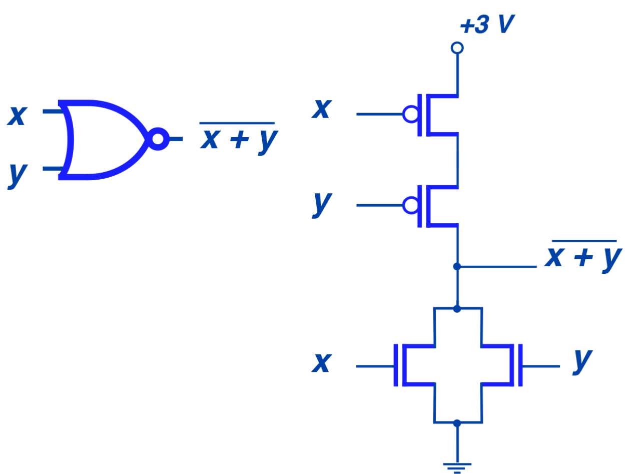
    - **NOT**



    - **NAND**

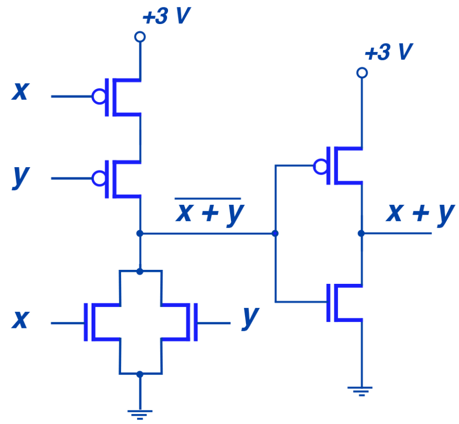


    - **NOR**



- **AND** and **OR** gates
    - Require at least six **CMOS** transistors.

- *Example:* OR gate = NOR plus NOT

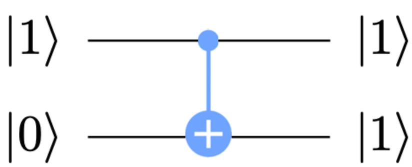


- Integrated circuit layout for **NOT**, **NAND** and **NOR** gates, using CMOS.
- Zoom down inside an IC to see gates!

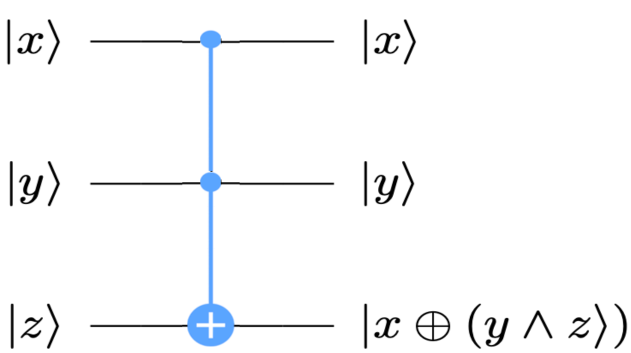
Quantum Computing Implementation of Logic Gates
Taken from: mcharemza_quant_circ.pdf

- **Pauli X (NOT) gate**
  |0> State becomes |1> State

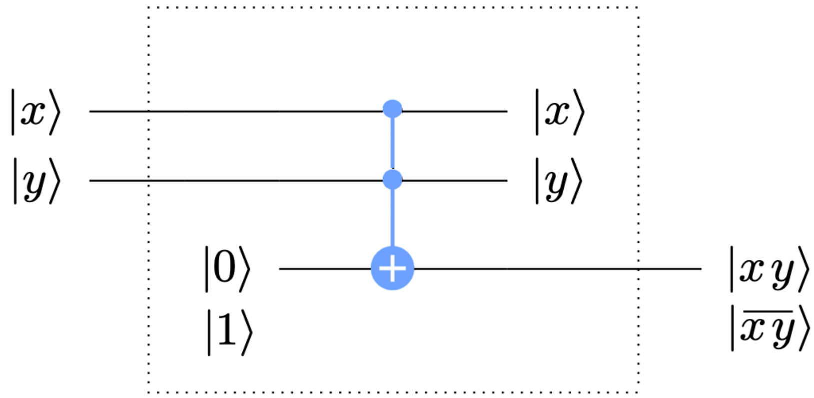
- **CNOT gate** (controlled NOT)
  |0> State becomes |1> State if the Control is |1>



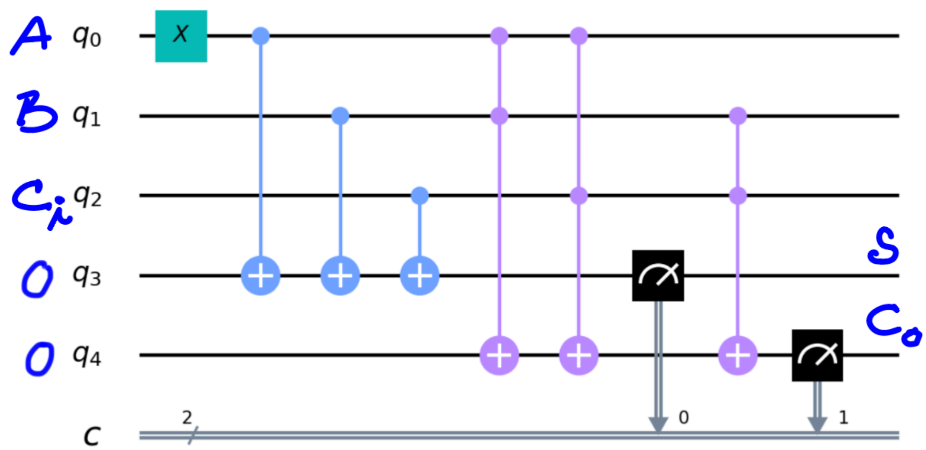
- **Toffoli gate** (CCNOT)



- AND/NAND gates
  0 ⊕ xy = xy, 1 ⊕ xy = (xy)'

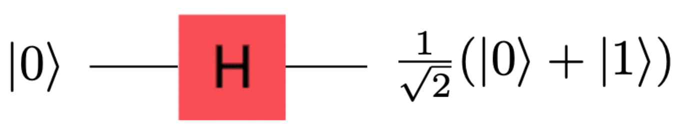


- Example Quantum Full Adder Circuit

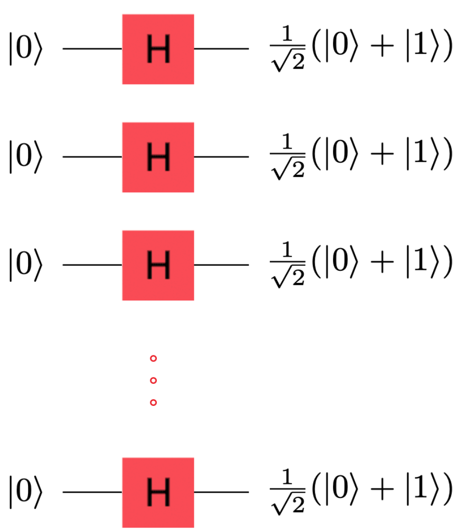


(Image: thequantuminsider.com)

- **H gate**
  |0> State becomes 50% superimposed with |1> State



Example random number generator:



Quiz #3 & selected solutions

MINTERMS

Suppose we want to expand a certain ***SOP*** function f_1 into canonical form whereby each resulting product term contains a literal of every independent variable of f_1.

$$f_1(A,B,C) = \bar{A}\cdot B + B\cdot\bar{C} + ABC = \text{SOP}$$
$$= \bar{A}B(C+\bar{C}) + B\bar{C}(A+\bar{A}) + ABC$$
$$f_1(A,B,C) = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC = \text{canonical SOP}$$

minterms

The product terms above are called **minterms**, the properties of which are now be presented.

Minterm Properties and Notation

- A **minterm** is a product term which produces a single **1** in a truth table

row	ABC	m_2 $\bar{A}B\bar{C}$ (010)	m_3 $\bar{A}BC$ (011)	m_6 $AB\bar{C}$ (110)	m_7 ABC (111)	f_1	$\bar{f_1}$
0	000	0	0	0	0	0	1
1	001	0	0	0	0	0	1
2	010	1	0	0	0	1	0
3	011	0	1	0	0	1	0
4	100	0	0	0	0	0	1
5	101	0	0	0	0	0	1
6	110	0	0	1	0	1	0
7	111	0	0	0	1	1	0

- The minterm which yields a **1** in row i is denoted as minterm m_i
- Express f in terms of minterms:
  - Compose a minterm list for f. (An atomic list)
  - *Example:*

$$f_1 = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC$$
$$= m_2 + m_3 + m_6 + m_7$$
$$= \Sigma m(2,3,6,7)$$

- Additional properties of minterm lists
  - $f = \Sigma m(\text{row\#s where } f = 1)$
  - $f' = \Sigma m(\text{row\#s where } f = 0)$
  - $\Sigma m(\text{all row\#s}) = \mathbf{1}$
- Minterm index i
  - Obtained by determining the row code for which $m_i = \mathbf{1}$
  - *Example:*

$\bar{A}B\bar{C} = 1$ where ?

$\bar{A}\downarrow B\downarrow \bar{C}\downarrow$: 010 in row 2

$\therefore \bar{A}B\bar{C} = m_2$

Quiz #4 & selected solutions (also includes ***Maxterms*** discussed below)

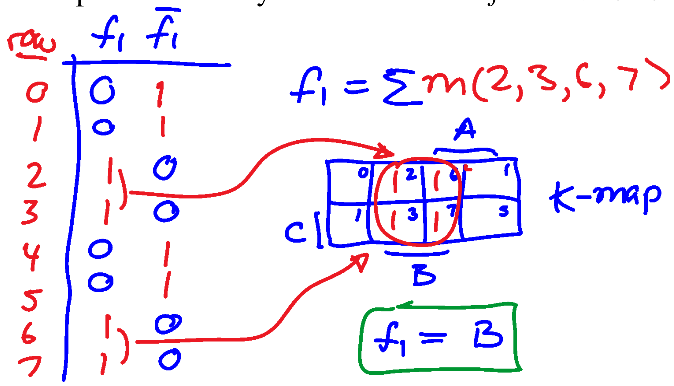
K-MAPS

Karnaugh Maps

What is a K-map? It is a graphical tool that quickly finds minimal algebraic forms of Boolean functions. The ***SOP*** forms are discussed here; ***POS*** forms are described in a later section.

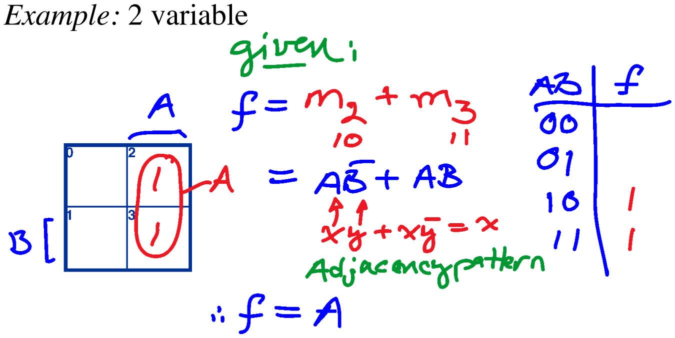
Karnaugh Map Properties

- Each cell in a K-map for a function *f* corresponds to a row of the truth table describing *f*
- **Cell *i*** is a place mark for minterm m_i.
- K-map labels identify the *coincidence of literals* to combine adjacent minterm.

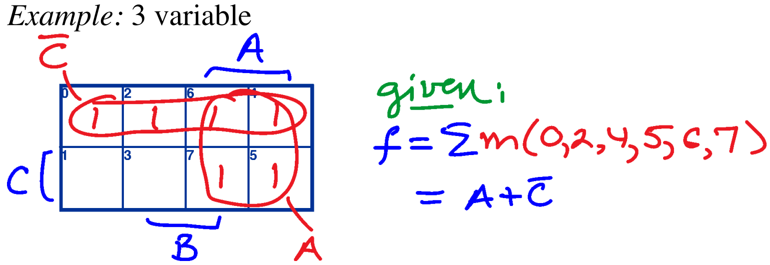


Sizes of K-maps

- *Example:* 2 variable

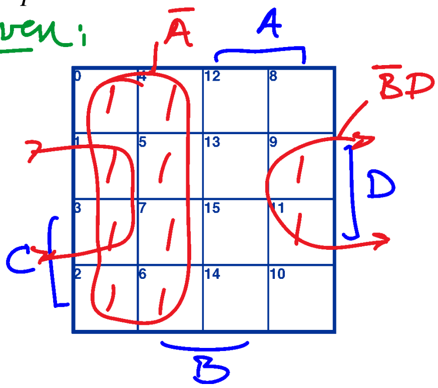


- *Example:* 3 variable



- *Example:* 4 variable

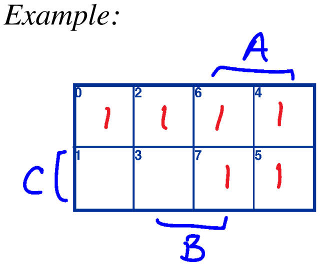
given;



$f = \bar{A} + \bar{B} \cdot D$

Procedure for Plotting *SOP* Functions on a K-map

- **Determine the minterms m_i** contained in f (found by observing the rows where $f = 1$ in the truth table).
- **Plot the 1's of the function** to be minimized on the K-map.
    - For each minterm m_i in f, enter a **1** in **cell i.**
    - *Example:*

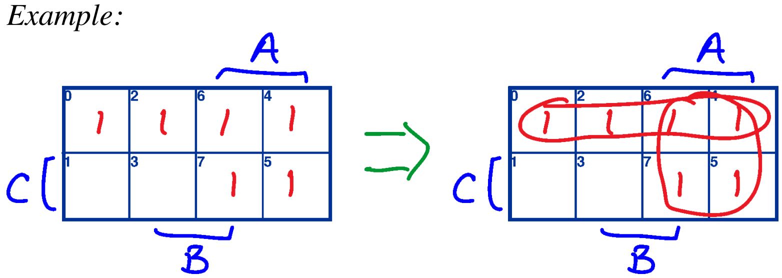


given;
$f = \Sigma m(0,2,4,5,6,7)$

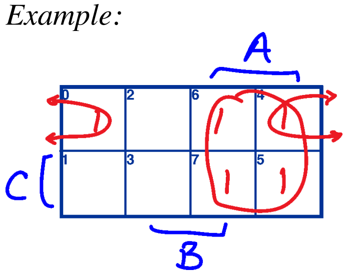
- For each *don't care* contained in f, enter a d (or x) in the associated K-map cell (see example later).

Procedure for *reading* minimal *SOP* expressions from K-maps

- Draw **loops** around **adjacent 1**-entries (cells with **1**'s) in largest groups possible.
    - Group size **in a power of two** (e.g. 1 cell, 2 cells, 4 cells, 8 cells, etc.)
    - *Example:*

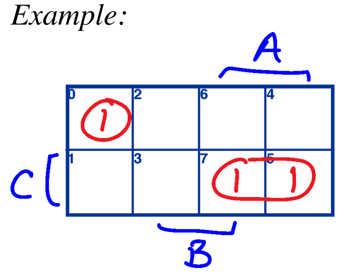


- Cells over the left and right edges, or the upper and lower edges are also *defined* to be adjacent.
    - *Example:*

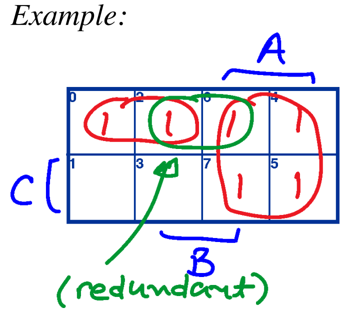


- **1**-entries not adjacent to other **1**-entries are circled as groups of one.

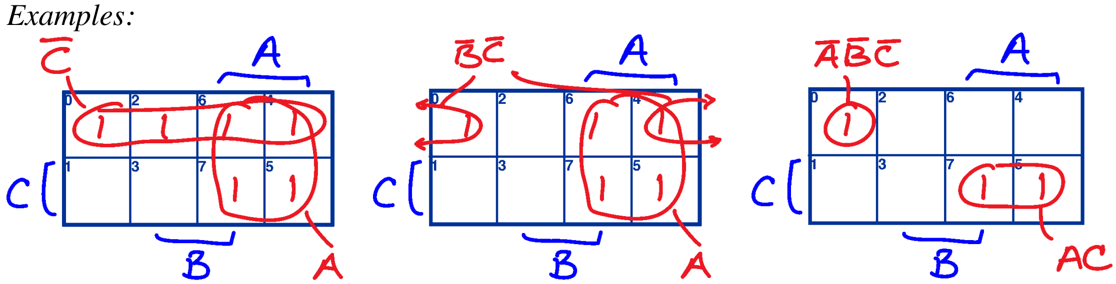
- *Example:*



- Discard **redundant groupings**
  - Discard those entries covered entirely by other groups
  - *Example:*



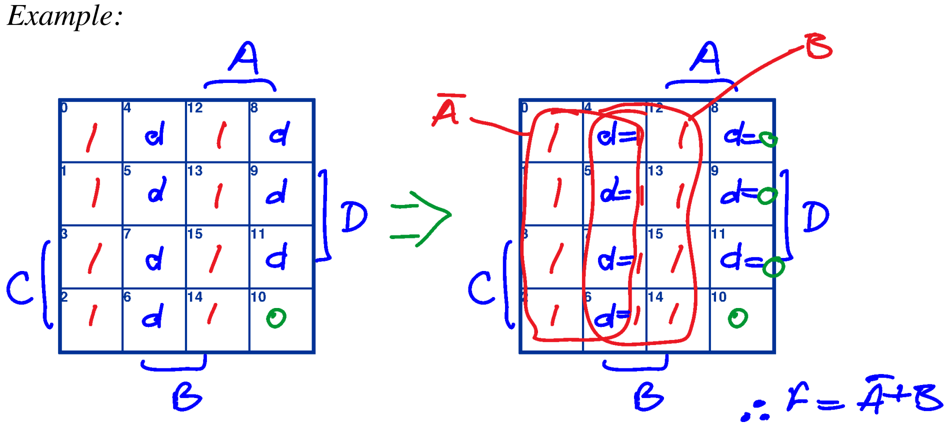
- For each group (as seen in the several examples above),
  - **Read** off the **coincident literals**, by exploiting K-map labels
  - **AND** those literals together to form products formed by the groups
  - **OR** the resulting products to create a SOP expression
  - *Examples:*



$\therefore f = A + \bar{C}$ $\therefore f = A + \bar{B}\bar{C}$ $f = \bar{A}\bar{B}\bar{C} + AC$

Map Simplification Resulting from Don't Cares

- Don’t care = ***d*** (or ***X***) = {0,1} (either a 0 or a 1)
- Group **1**-entries as before, but
  - Also include any ***d***-entries which serve to increase the size of the group of **1's**
  - Treat unused ***d***-entries as **0**-entries
  - Must have at least one **1**-entry in all groups
  - *Example:*



- Never group cells consisting **entirely** of don't care entries. This results in a redundant group.

Quiz #5 & selected solutions (includes ***POS*** usages)

MAXTERMS & K-MAPS

Maxterm Properties and Notation

Now we want to consider expanding a certain ***POS*** function f_2 into canonical form whereby each resulting sum term contains a literal of every independent variable of f_2.

product

$$f_2(A,B,C) = \underbrace{(A+B)}_{\text{sum}} \cdot \underbrace{(\bar{A}+B+C)}_{\text{sum}} \cdot \underbrace{(\bar{A}+B+\bar{C})}_{\text{sum}} = POS$$

$$= (A+B+C)\cdot(A+B+\bar{C})\cdot(\bar{A}+B+C)\cdot(\bar{A}+B+\bar{C})$$

Maxterms = canonical POS

The sum terms above are called **Maxterms**, the properties of which now follow.

- A **Maxterm** is a sum term which produces a single **0** in a truth table

row	ABC ($n=3$)	M_0: $(A+B+C)$ (000)	M_1: $(A+B+\bar{C})$ (001)	M_4: $(\bar{A}+B+C)$ (100)	M_5: $(\bar{A}+B+\bar{C})$ (101)	f_2	$\bar{f_2}$
0	000	0	1	1	1	0	1
1	001	1	0	1	1	0	1
2	010	1	1	1	1	1	0
3	011	1	1	1	1	1	0
4	100	1	1	0	1	0	1
5	101	1	1	1	0	0	1
6	110	1	1	1	1	1	0
7	111	1	1	1	1	1	0

- Maxterm which yields a ***0*** in row i is denoted as maxterm M_i
- Expressing f in terms of Maxterms:
  - Compose a Maxterm list for f. (An atomic list)
  - *Example:*

$$f_2 = (A+B+C)\cdot(A+B+\bar{C})\cdot(\bar{A}+B+C)\cdot(\bar{A}+B+\bar{C})$$

$$= M_0 \cdot M_1 \cdot M_4 \cdot M_5$$

$$= \Pi M(0,1,4,5)$$

- Additional properties of Maxterm lists
  - $f = \Pi M(\text{row\#s where } f = 0)$
  - $f' = \Pi M(\text{row\#s where } f = 1)$
  - $\Pi M(\text{all row\#s}) = \mathbf{0}$

- Maxterm index i
  - Obtained by determining the row code for which $M_i = \mathbf{1}$

- *Example:*

  $\bar{A}+B+\bar{C} = 0$ where ?

  $(1\ 0\ 1)$ in row 5

  5

  $\therefore\ \bar{A}+B+\bar{C} = M_5$

Other Properties of minterms and Maxterms

- $f = \Sigma m(row\#s) = \Pi M(opposite\ row\#s)$
  $f = \Pi M(row\#s) = \Sigma m(opposite\ row\#s)$
- If $f = \Sigma m(row\#s)$ then $f\,' = \Sigma m(opposite\ row\#s)$
  If $f = \Pi M(row\#s)$ then $f\,' = \Pi M(opposite\ row\#s)$
  - *Examples:*

    For our 2 tables earlier:

    $f_1 = \Sigma m(2,3,6,7) = \Pi M(0,1,4,5) = f_2$

    $\bar{f}_1 = \Sigma m(0,1,4,5) = \Pi M(2,3,6,7) = \bar{f}_2$
- $m_i' = M_i$
  $M_i' = m_i$
  - *Examples:*

    $m_i + M_i = m_i + \bar{m}_i = 1$

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

    $m_i \cdot \bar{M}_i = m_i \cdot m_i = m_i$

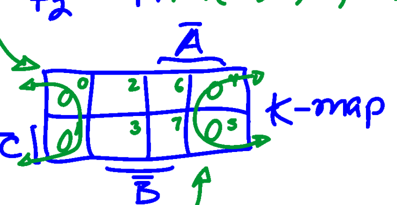
    $(x \cdot x = x)$

K-Map *POS* Properties

- **Cell *i*** is a place mark for Maxterm M_i.
- K-map labels identify the *coincidence of literals* to combine adjacent Maxterms.

row	f_2	$\bar{f}_2$
0	0	1
1	0	1
2	1	0
3	1	0
4	0	1
5	0	1
6	1	0
7	1	0

$f_2 = \Pi M(0,1,4,5)$

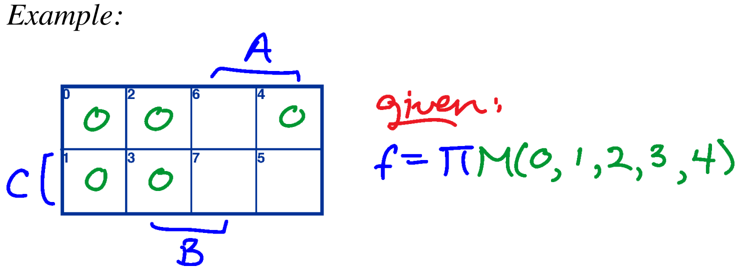


$\boxed{f_2 = B}$

Procedure for plotting and reading minimal POS expressions from K-maps

- **Plot the 0's of the function** to be minimized on a K-map.
  - For each maxterm M_i in f, enter a **0** in **cell *i*.**

- *Example:*



- Draw **loops** around **adjacent 0**-entries.
- For each group
  - **Read** off the <u>complement</u> of the **coincident literals** covering the group
  - **OR** those literals together to form sums
  - **AND** the resulting sums to create a product
  - *Example:*