Putting Hands on Logic Design

By Hamza Elkotb

Pre-version 1.0.0.0
Special Version for FCAI HU Students

Special thanks to Dr. Ahmed Hisham for his awesome teaching for Logic Design and bringing us to a high level, which enabled us, -thanks to Allah-, to transfer what we learned to our colleagues and perhaps to other students.

{ وَمَا أُوتِيتُم مِّنَ الْعِلْمِ إِلَّا قَلِيلًا }

(85) الإسراء

Putting Hands on Logic Design

(Digital Electronics)

This e-book, is a pre-release version

Authored by HamzaDev & IEEE802

❝ *Computers are composed of nothing more than logic gates stretched out to the horizon in a vast numerical irrigation system.* ❞

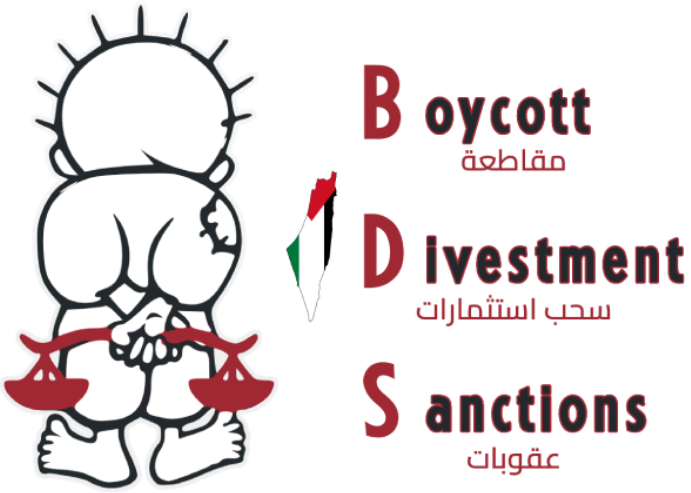
History of the Integrated Circuits

NOTE: If you notice any mistakes or there is something you didn't understand well, just mention **HamzaDev** in any group!

https://www.youtube.com/@CSCoreDecoded

من النهر الى البحر
تحريرها كلها مؤكد

لا تنسوا اخواننا في فلسطين والسودان وسوريا ولبنان من دعائكم



We will never forget!

We will never forgive crimes against our people in Palestine!

Our enemy is one!

The aggression continues!

Keep boycotting!

Introduction To Digital Logic Design

#Authors `: HamzaDev & IEEE802`

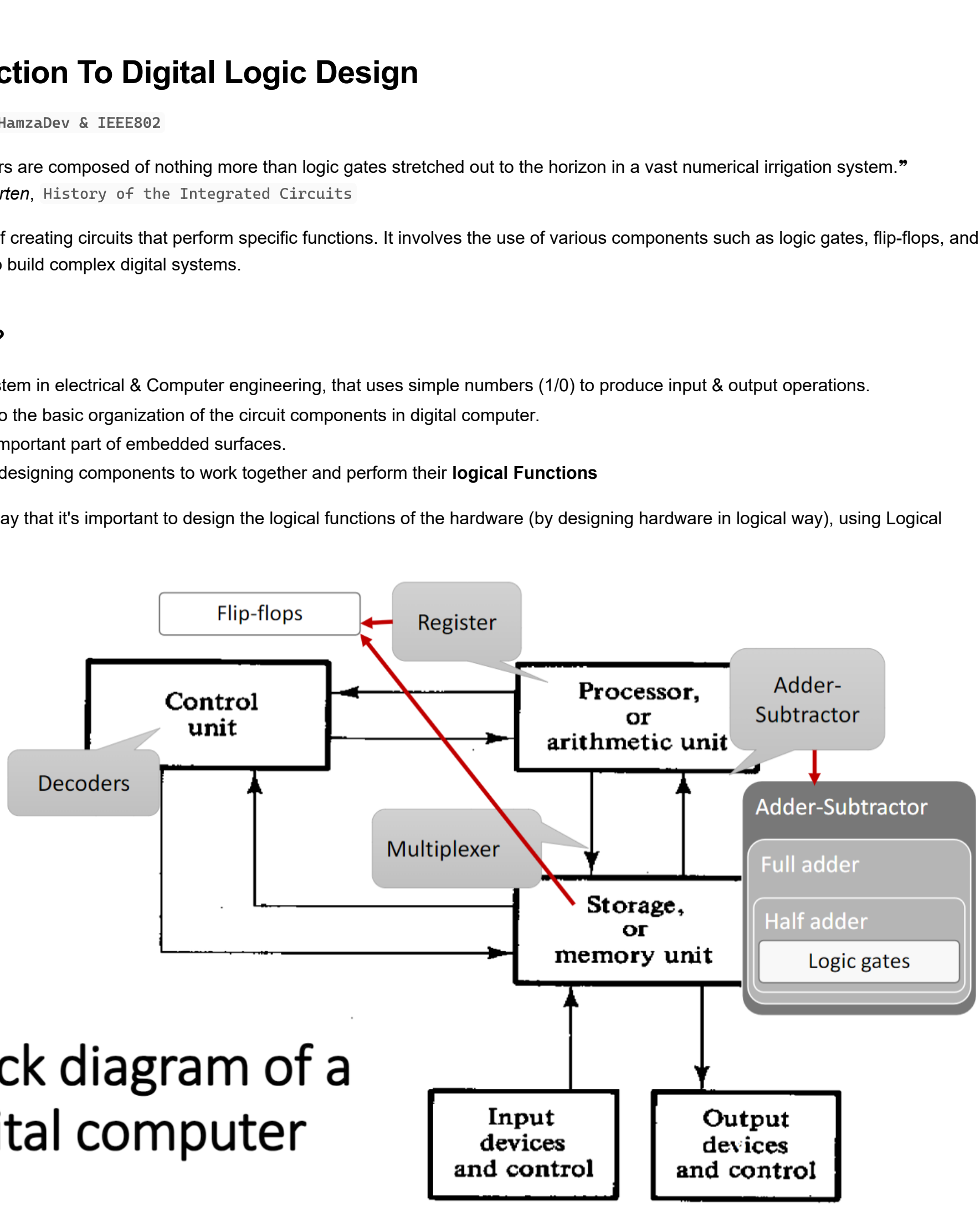
> **“**Computers are composed of nothing more than logic gates stretched out to the horizon in a vast numerical irrigation system.**”**
> *Stan Augarten*, `History of the Integrated Circuits`

The process of creating circuits that perform specific functions. It involves the use of various components such as logic gates, flip-flops, and multiplexers to build complex digital systems.

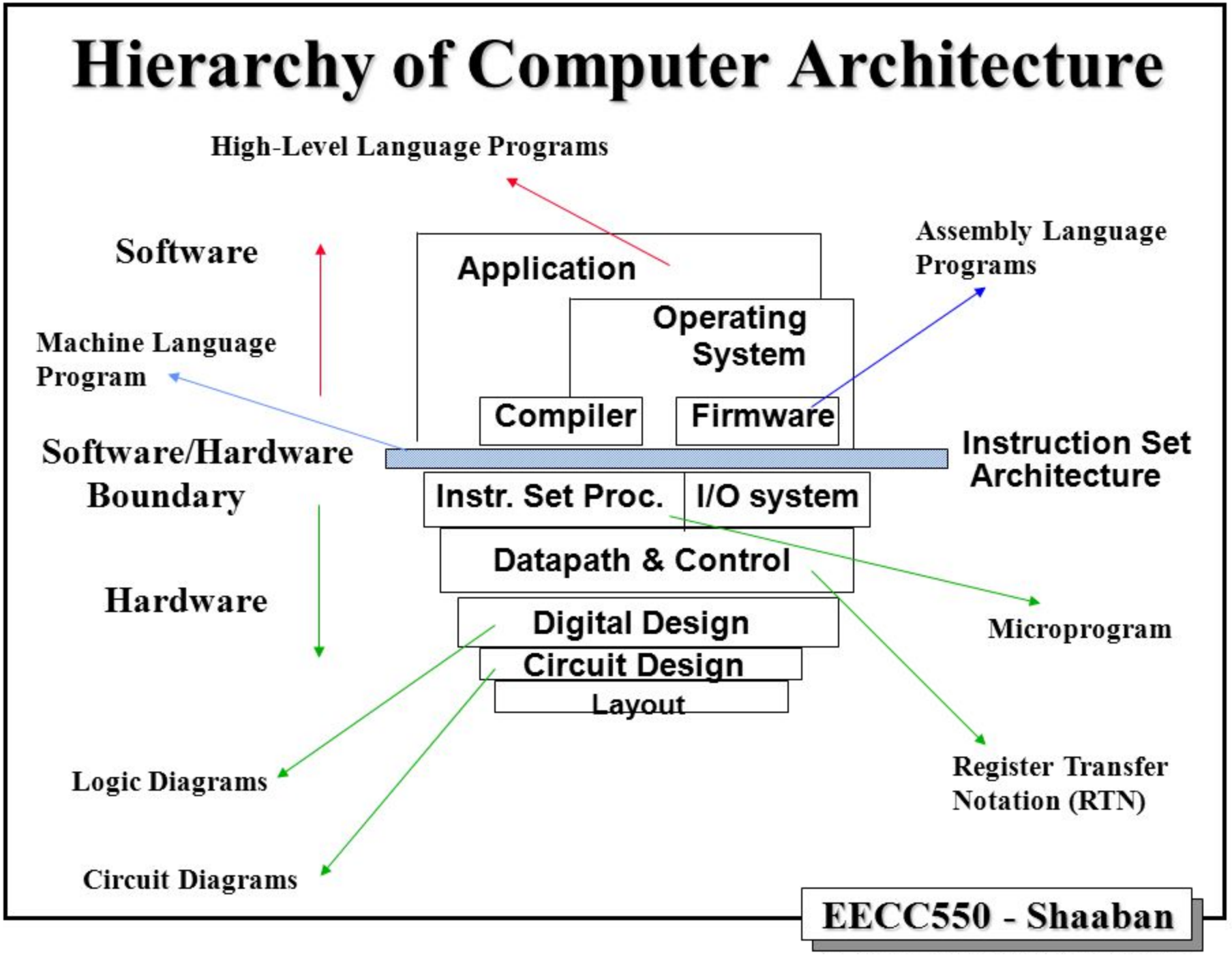
What is it?

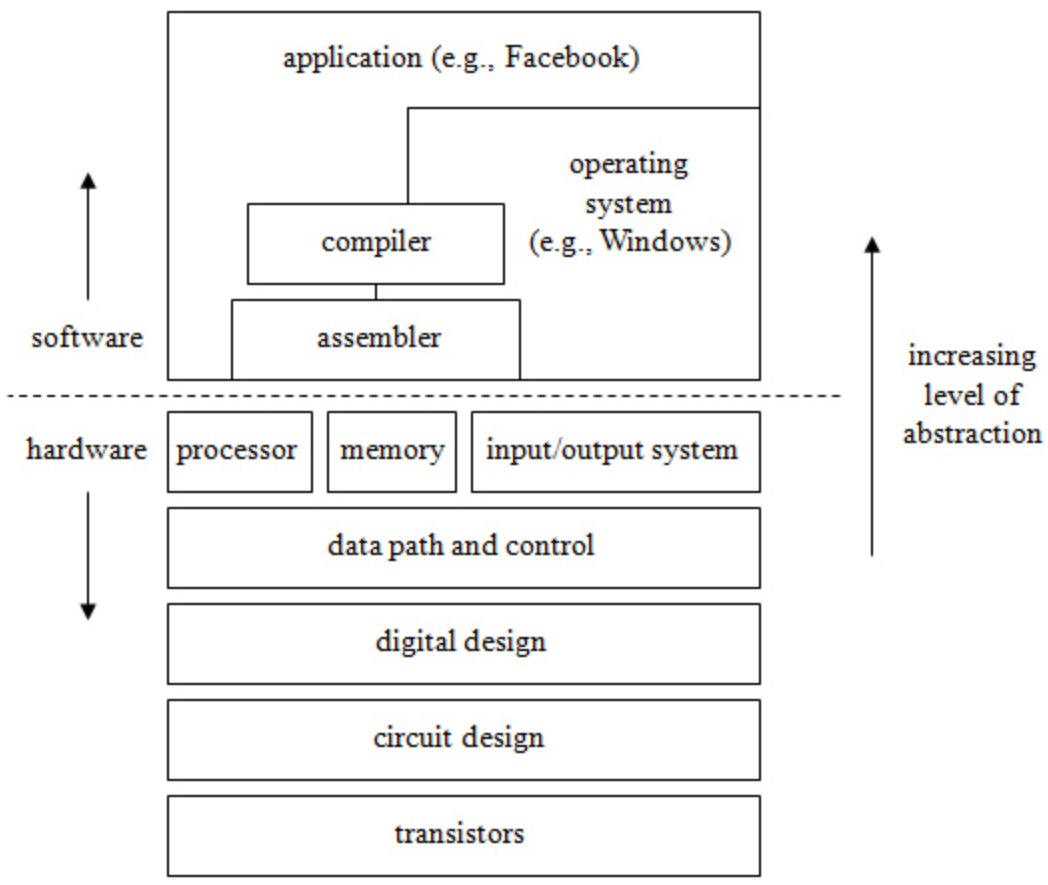
- It is a system in electrical & Computer engineering, that uses simple numbers (1/0) to produce input & output operations.
- It refers to the basic organization of the circuit components in digital computer.
- It forms important part of embedded surfaces.
- Involves designing components to work together and perform their **logical Functions**

SO: you can say that it's important to design the logical functions of the hardware (by designing hardware in logical way), using Logical system (1,0).



Hierarchy of Comp Arch





Digital Systems

Systems uses:

- signals that have 2 distinct values.
- Circuit elements that have 2 stable states.

Applications

- Develop hardware (designing circuit boards, microprocessors, RAM, ROM ...etc)

How it Benefits me as CS Student

- Understand Computer Architecture
- Understand how software instructions are executed by hardware

Subjects based on it

- Comp Arch
- OS
- DSP (Digital Signal Processing)

Search:

- [ ] digital computer
- [ ] Analog Computer
- [ ] fundamental level of computer
- [ ] how software instructions are executed by hardware
- [ ] Digital Systems

Resources:
https://www.linkedin.com/pulse/what-digital-logic-design-gulled-hassan
https://technav.ieee.org/topic/logic-design

Custom CPU Design
Build CPU

Introduction to Number Systems

#Authors : HamzaDev & IEEE802

in normal we use Decimal Number system (10 based)
Most famous other systems are: *2-based, 8-based, 16-based*

Name	Radix	Digits	
Binary	2	0,1	
Octal	8	0,1,2,3,4,5,6,7	
Decimal	10	0,1,2,3,4,5,6,7,8,9	
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F	Allows convenient handling of long binary strings, using groups of 4 bits—Base 16

Data Measurement & Sizes

Bit	Single Binary Digit (1 or 0)
Byte	8 bits
Kilobyte (KB)	1,024 Bytes
Megabyte (MB)	1,024 Kilobytes
Gigabyte (GB)	1,024 Megabytes
Terabyte (TB)	1,024 Gigabytes
Petabyte (PB)	1,024 Terabytes
Exabyte (EB)	1,024 Petabytes

General equation to convert any number system to 10-based:

$$V \times W^p$$

$$V = Value, W = Weight, P = Position$$

Example: **Base-8 number**

$$(127.4)_8$$
$$= 1\times8^3 + 2\times8^2 + 1\times8^1 + 7\times8^0 + 4\times8^{-1} = (87.5)_{10}$$

General form of base-r system

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

Coefficient: $a_j = 0$ to $r - 1$

NOTE: Arithmetic operations with numbers in base r follow the same rules as decimal numbers

Mapping Dec, Bi, Oct, Hexa

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

Represent with Base

The subscript on a number indicates its base

560_{10} implies that 560 is a decimal number

11010 implies that 11010 is a **decimal** number

11010_{10} implies that 11010 is a **decimal** number

11010_{2} implies that 11010 is a **binary** number

11010_{16} implies that 11010 is a **hexadecimal** number

11010_{8} implies that 11010 is an **octal** number

Search:

Resources:
https://youtu.be/8u_doj8CO1E?si=lSceh6iNr1V3yDdC

Decimal to-from Binary

#Authors : HamzaDev & IEEE802

Binary to Decimal

General Equation to convert B2D

$$
\begin{aligned}
Y &= \cdots b_3 b_2 b_1 b_0 . b_{-1} b_{-2} b_{-3} \cdots \\
&= \sum_i \left(b_i \times 2_{10}^i \right) \ \text{(Binary To Decimal)} \\
&= \sum_i \left(b_i \times 10_2^i \right) \ \text{(Binary representation)}
\end{aligned}
$$

Binary code	**1**	**1**	**0**	**1**	**0**	**1**
weight	25 = 32	24 = 16	23 = 8	22 = 4	21 = 2	20 = 1
Value in weight	25 * 1 = 32	24 * 1= 16	23 * 0 = 0	22 * 1 = 4	21 * 0 = 0	20 * 1 = 1

Decimal to Binary

There are 3 main ways

- **Division & Reminder (Most Significant Bit, Least Significant Bit)**

$$
\begin{array}{rcrcr|cc}
7 & \div & 2 & = & 3.5 & \text{R} \ 0.5\times 2 = & 1 \quad \text{LS} \\
3 & \div & 2 & = & 1.5 & \text{R} \ 0.5\times 2 = & 1 \quad \uparrow \\
1 & \div & 2 & = & 0.5 & \text{R} \ 0.5\times 2 = & 1 \quad \text{MS}
\end{array}
$$

$$
\text{Thus } 7_{10} = 111_2 \quad \Rightarrow \quad \text{MS} \dashrightarrow \text{LS}
$$

$$
\begin{array}{rcrcr|cc}
0.1875 & \times & 2 & = & 0.375 & 0 & \text{MS} \\
0.375 & \times & 2 & = & 0.75 & 0 & \uparrow \\
0.75 & \times & 2 & = & 1.5 & 1 & \uparrow \\
0.5 & \times & 2 & = & 1.0 & 1 & \text{LS}
\end{array}
$$

$$
\text{Thus } 0.1875_{10} = 0.0011_2 \quad \Rightarrow \quad 0.\text{MS} \dashrightarrow \text{LS}
$$

Because we multiply by 2 several time to get the decimal number from a binary code, we can reverse the steps to divide several times to get binary code of a decimal number.

Each time we divide by 2, if there is a remainder of the division then we add 1 to our binary code, if there is no remainder we add 0

Take number 72 as an example
72 / 2 = 36 => no reminders => 0
36 / 2 = 18 => no reminders => 0
18 / 2 = 9 => no reminders => 0
9 / 2 = 4 => there is reminder => 1
4 / 2 = 2 => no reminders => 0
2 / 2 = 1 => no reminders => 0
1 / 2 = 0 => there is reminder => 1

So the binary code of 72 is 1001000

- **Finding X**

take number 64 as an example

- Find x using logarithms

$\log_2(64) = \log_2(2^x)$
$\log_2(64) = x * \log_2(2)$
$\log_2(64) = x$
$\log_2(64) = 6$
$x = 6$

NOTE: if x resulted number like 6.2 just round it to the next number to be x = 7 and solve based on 2^7

- Make a table that contains 2^x to 2^0.

weight	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

- Multiply all weights by 0, to get 0 in x times.

weight	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Value in weight	$2^6 * 0 = 0$	$2^5 * 0 = 0$	$2^4 * 0 = 0$	$2^3 * 0 = 0$	$2^2 * 0 = 0$	$2^1 * 0 = 0$	$2^0 * 0 = 0$
Binary code	0	0	0	0	0	0	0

- From left to right start replacing zeros with 1 tell getting the same decimal number.

weight	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Value in weight	$2^6 * 1 = 64$	$2^5 * 0 = 0$	$2^4 * 0 = 0$	$2^3 * 0 = 0$	$2^2 * 0 = 0$	$2^1 * 0 = 0$	$2^0 * 0 = 0$
Binary code	1	0	0	0	0	0	0

NOTE: Now there is no need to completely replace the rest of zeros because we already getted 64 from the first replacing.

- When getting the same decimal number the current binary code is the wanted

Binary code of 64 is 1000000

- **Nearest Guess**

Take number 79 as an example

- See the nearest weight to your number

weight	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

In our case the nearest weight to 79 is 64

- Subtract the weight from your number, if you didn't get zero, just subtract the result from the nearest weight, and repeat these steps tell getting result 0

79 - 64 = 15
15 - 8 = 7
7 - 4 = 3
3-2 = 1
1-1 = 0

- Collect all weights you had used

Used weights: 64, 8, 4, 1

- Find all missed weights and make a table

Missed weights are: 32, 16

weights	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

- Replace all missed weights by 0 and used weights by 1

weight	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Binary code	1	0	0	1	1	1	1

So the binary code of 79 is 1001111

Code Implementation

Search, Read more, Tasks:

- [ ] Build Binary to Decimal logic gate converter
- [ ] Build Code to make conversion in C++
- [ ] How To convert a number to its 4-bit binary equivalent using logic gates?
- [ ] Build BDD

Resources:

https://www.geeksforgeeks.org/program-decimal-binary-conversion/
https://youtu.be/Jp7Y5hxMDpg?si=OPz5qsvlBLiGJBzV
binary-to-decimal
Decimal2Binary

Decimal to-from Oct

#Authors : HamzaDev & IEEE802

Oct to Decimal

General Formula to convert

$$Z = \cdots o_3 o_2 o_1 o_0 . o_{-1} o_{-2} o_{-3} \cdots$$
$$= \sum_i \left(o_i \times 10_8^i \right)$$

Example

$$\begin{aligned} 111.14_8 &= \left(1 \times 8^2\right) + \left(1 \times 8^1\right) + \left(1 \times 8^0\right) + \left(1 \times 8^{-1}\right) + \left(4 \times 8^{-2}\right) \\ &= 64 + 8 + 1 + 0.125 + 0.0625 \\ &= 73.1875_{10} \end{aligned}$$

Decimal to Oct

Division & Reminder (Most Significant Bit, Least Significant Bit)

431	÷	8	=	53.875	R	0.875×8 =	7	LS
53	÷	8	=	6.625	R	0.625×8 =	5	↑
6	÷	8	=	0.75	R	0.75×8 =	6	MS

$$\text{Thus } 431_{10} = 657_8 \quad \Rightarrow \quad \text{MS} \dashrightarrow \text{LS}$$

0.1875	×	8	=	1.5	1	MS
0.5	×	8	=	4.0	4	LS

$$\text{Thus } 0.1875_{10} = 0.14_8 \quad \Rightarrow \quad 0.\text{MS} \dashrightarrow \text{LS}$$

Code Implementation

Search, Read more, Tasks:

- [ ] Build Code to make conversion in C++
- [ ] Other ways to convert Dec to Oct

Resources:

https://youtu.be/mayNND1WZkE?si=xsCh_uJgsroPYNNJ

Decimal to-from HexaDecimal

#Authors : HamzaDev & IEEE802

HexaDecimal to Decimal

General Formula to convert

$$Z = \cdots h_3 h_2 h_1 h_0 . h_{-1} h_{-2} h_{-3} \cdots$$
$$= \sum_i \left(h_i \times 16^i \right)$$

Example

$$\begin{aligned} 2C_{16} &= \left(2_{16} \times 16^1\right) + \left(C_{16} \times 16^0\right) \\ &= \left(2 \times 16^1\right) + \left(12 \times 16^0\right) \\ &= 32 + 12 \\ &= 44_{10} \end{aligned}$$

Equivalent Table

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Octal	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17

Decimal to HexaDecimal

Division & Reminder (Most Significant Bit, Least Significant Bit)

$$\begin{array}{rcrcr|cl|l} 431 & \div & 16 & = & 26.9375 & \text{R} & 0.9375 \times 16 = & 15 = \text{F} & \text{LS} \\ 26 & \div & 16 & = & 1.625 & \text{R} & 0.625 \times 16 = & 10 = \text{A} & \uparrow \\ 1 & \div & 16 & = & 0.0625 & \text{R} & 0.0625 \times 16 = & 1 & \text{MS} \end{array}$$

$$\text{Thus } 431_{10} = 1AF_{16} \quad \Rightarrow \quad \text{MS} \dashrightarrow \text{LS}$$

$$0.1875 \times 16 = 3.0 \mid \quad 3 \quad \text{MS}$$

$$\text{Thus } 0.1875_{10} = 0.3_{16} \quad \Rightarrow \quad 0.\text{MS} \dashrightarrow \text{LS}$$

Code Implementation

Search, Read more, Tasks:

- [ ] Build Code to make conversion in C++
- [ ] Other ways to convert Dec to Hex

Resources:

https://youtu.be/ShbCYuomKzc?si=G1F3nJGvg8Af1_y-

Binary to-from Oct

#Authors : HamzaDev & IEEE802

Octal to Binary

Use Equivalent Table

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

Example

$262.6_8 = ??_2$

$$\begin{array}{lcccc} \text{Octal} & 2 & 6 & 2 & 6 \\ \text{Binary} & 010 & 110 & 010 & 110 \\ & \Downarrow & \Downarrow & \Downarrow & \Downarrow \end{array}$$

$$262.6_8 = \underbrace{010}_{2}\,\underbrace{110}_{6}\,\underbrace{010}_{2}.\underbrace{110}_{6} = 010110010.110_2 = 10110010.11_2$$

Binary to Octal

- By dividing the binary code into slots of 3 bits (3 digits of 0,1s)
- This because 2^3 = 8 & because max number which is `7` needs 3 bits `111`
- complete missed places on left by 0s
- Use Equivalent table

$10110010.11_2 = ??_8$

$$\begin{array}{lcccc} \text{Binary} & 010 & 110 & 010 & 110 \\ \text{Octal} & 2 & 6 & 2 & 6 \\ & \Downarrow & \Downarrow & \Downarrow & \Downarrow \end{array}$$

$$10110010.110_2 = \underbrace{010}_{2}\,\underbrace{110}_{6}\,\underbrace{010}_{2}.\underbrace{110}_{6} = 262.6_8$$

Resources:

https://youtu.be/7S3VPc3t2H0?si=a7oj2rISB5Az0mev

Binary to-from HexaDecimal

#Authors : HamzaDev & IEEE802

HexaDecimal to Binary

Use Equivalent Table

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

Example

$$B2.C_{16} = ??_2$$

$$\begin{array}{rccc} \text{Hexadecimal} & B & 2 & C \\ \text{Binary} & 1011 & 0010 & 1100 \\ & \Downarrow & \Downarrow & \Downarrow \end{array}$$

$$B2.C_{16} = \underbrace{1011}_{B}\underbrace{0010}_{2}.\underbrace{1100}_{C} = 10110010.1100_2$$

Binary to HexaDecimal

- By dividing the binary code into slots of 4 bits (4 digits of 0,1s)
- This because 2^4 = 16 & because max number which is `F = 15` needs 4 bits `1111`
- complete missed places on left by 0s
- Use Equivalent table

$$110101111_2 = ??_{16}$$

$$\begin{array}{rccc} \text{Binary} & 0001 & 1010 & 1111 \\ \text{Hexadecimal} & 1 & A & F \\ & \Downarrow & \Downarrow & \Downarrow \end{array}$$

$$110101111_2 = \underbrace{0001}_{1}\underbrace{1010}_{A}\underbrace{1111}_{F} = 1AF_{16}$$

Resources:

https://youtu.be/7S3VPc3t2H0?si=a7oj2rISB5Az0mev

Binary Codes

#Authors : HamzaDev & IEEE802

- Binary system is the only system that fits with **Digital Systems**
- Any Discrete element of information that is distinct among group of quantities can be represented with **Binary Code**
- Binary Codes change the symbol of the element information not it's meaning

---

BCD (Binary Coded Decimal)

This system works on 10 symbols (**0-9**), converting each digit to a specific binary value of 4 bits (**0000-1001**).

SO: Each decimal digit from the full Decimal Number is dealt individually, and taking 4 bits binary number alone.
Ex: ***12*** in decimal is = ***1 and 2*** = ***0001*** and ***0010*** = ***00010010*** = ***10010***

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (1011\ 1001)_2$$

Now Weights of normal BCD is

$$8, 4, 2, 1 = 2^3, 2^2, 2^1, 2^0$$

BCD vs Normal Decimal to Binary Conversion

BCD deals with each decimal digit of the decimal number individually, but normal conversion deals with the full decimal number as 1 digit.
Ex: take number ***185***:
Normal conversion: ***10111001***
BDC conversion: ***0001-0100-0101*** = ***101000101***

BCD to Decimal

Because each decimal digit is presented by 4 bits, we'll convert each 4 bits to single digit from right to left, and if we found less than 4 bits, we fill 0s to left.
Ex: take BCD ***101000101***:
- Will be divided into: 1-0100-0101
- Fill missing positions: 0001-0100-0101
- Convert each 4 bits to a single decimal digit: 1-8-5
- Final result: 185

Note: ***1010***, ***1011***, ***1100***, ***1101***, ***1110***, and ***1111*** are **INVALID CODE**, because they equal 10-15, and these are decimal numbers not decimal digits.

Weighted Codes

In weighted code, each bit position is assigned a weighted factor.
in normal we had weights of 2 power to position:

$$\ldots 2^3, 2^2, 2^1, 2^0$$

But now we have other weights not ***2 power based***, like:

- **2421**
- ***84-2-1***
- Normal BCD is also weighted code
- also: ***3132*** by HamzaDev

NOTE: Weighted BCD also works on **10** digit symbols (**0-9**) and 4 bits (**0000-1111**)

$0110_{BCD} = 6_{10}$

because 8*0 + 4*1 + 2*1+ 1*0 = 6

$1101_{2421} = 7_{10}$

because 2*1 + 4*1 + 2*0 + 1*1 =7

$0110_{8,4,-2,-1} = 2_{10}$

because 8*0 + 4*1 + (-2)*1 + (-1)*0 =2

to convert from weighted to Decimal, just sum the active positions
Ex: 0*110* in 8,4,-2,-1 =

$$(0 * 8) + (1 * 4) + (1 * -2) + (0 * -1) = 0 + 4 + -2 + 0 = 2$$

Gray Code (RBC = Reflected Binary Code)

other names: Unit Distance Code (UDC), Minimum Error Code (MEC), Cyclic Code.

- It is unweighted code
- Has a property that **two successive numbers differ in only one bit**
- Binary code is converted to Gray to reduce switching operation by switching the least number of bits.

Binary to Gray Conversions

- Take the MSB (Most significant Bit) as it is
- Apply XOR operation between each 2 neighbor bits
- **Ex: *1011*** in binary
  1 (MSB) ----------> 1
  0 (XOR with the top) ---------> 1
  1 (XOR with the top) ---------> 1
  1 (LSB) (XOR with the top) ---> 0
  SO, in Gray it equals to ***1110***

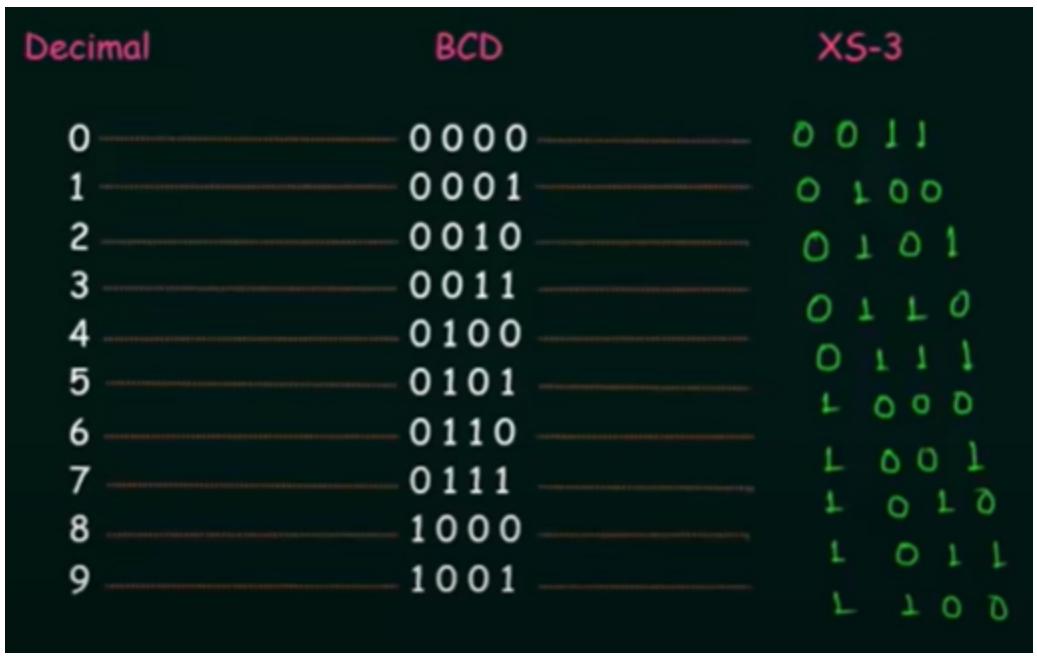
Gray to Binary Conversions

- Take the MSB (Most significant Bit) as it is
- Apply XOR operation between the result and the next bit
- **Ex: *1110*** in Gray
  1 (MSB) ----------> 1
  1 (XOR with the result) ---------> 0
  1 (XOR with the result) ---------> 1
  0 (LSB) (XOR with the result) ---> 1
  SO, in Binary it equals to ***1011***

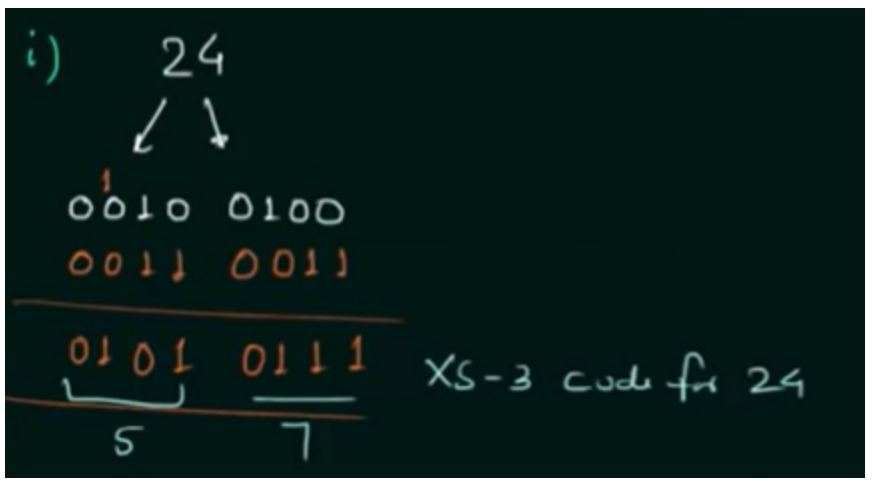
Excess-3 Code (XS-3)

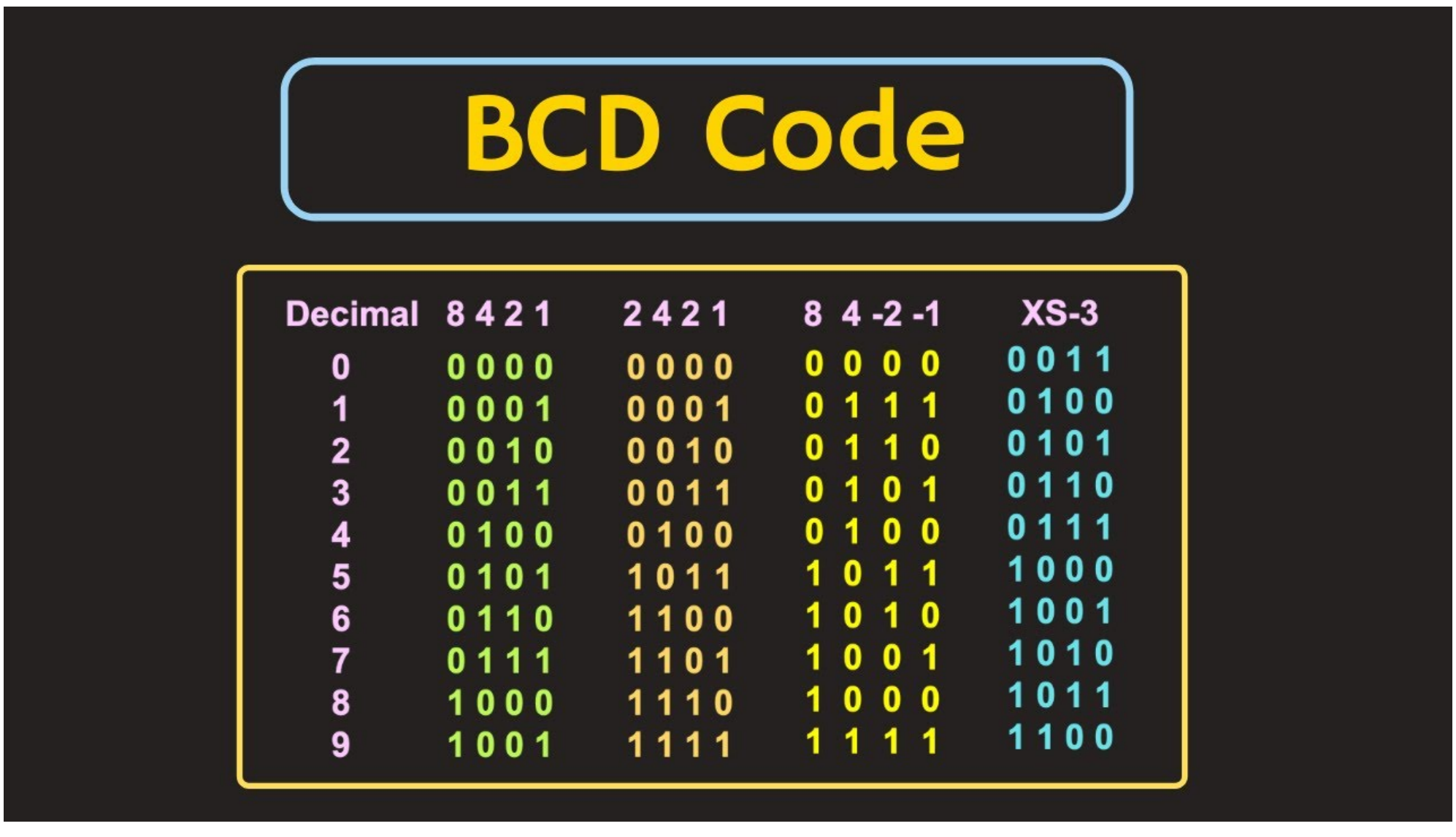
- It is unweighted code
- It is 4 bits code to convert Decimal digit symbols (**0-9**) to binary codes (**0011-1010**)
- Each decimal digit from the full Decimal Number is dealt individually
- It adds 3 to each decimal digit and then convert each decimal digit to binary code.
- Or add 0011 to the BCD code of each decimal digit to get the new binary code.

Decimal to BCD to XS-3 Conversions



Ex: 24 in decimal to XS-3





ACII

each character in the table is made of 7 bits
first 3 bits represent the column
other 4 bits represents the row

Search, Read more, Tasks:

- [ ] BCD Operations (Sum, Sub, Div, Mult)
- [ ] Gray Operations (Sum, Sub, Div, Mult)
- [ ] XS-3 Operations (Sum, Sub, Div, Mult)
- [ ] Applications of Gray code, XS-3
- [ ] Build Code to make conversion in C++
- [ ] XS-3 Self-Complementing code

Resources:
https://youtu.be/0rLiYpy2CqQ?si=sNoYW1RBxH2-Q6lD
https://youtu.be/0dPN4gh0CKI?si=63tQiKouJ3ya58gl
https://youtu.be/cF-Q5j7RUEw?si=dAt4Mbmgo_yT7xXh
https://youtu.be/-qMm9hhvp9Y?si=GMkwcAoES4mZY1LI
https://youtu.be/LHw8TVk9iOY?si=NN-8qHdznNOdFbqG

Octal to-from HexaDecimal

#Authors : HamzaDev & IEEE802

Octal to HexaDecimal

- Convert Oct to binary
- Convert Binary to Hexa

$657_8 = ??_{16}$

- **Step 1**: Convert Octal to Binary

$$657_8 = \underbrace{110}_{6}\ \underbrace{101}_{5}\ \underbrace{111}_{7} = 110101111_2$$

- **Step 2**: Convert Binary to Hexadecimal

$$110101111_2 = \underbrace{0001}_{1}\ \underbrace{1010}_{A}\ \underbrace{1111}_{F} = 1AF_{16}$$

HexaDecimal to Octal

- Convert Hexa to binary
- Convert Binary to Oct

$1AF_{16} = ??_8$

- **Step 1**: Convert Hexadecimal to Binary

$$1AF_{16} = \underbrace{0001}_{1}\ \underbrace{1010}_{A}\ \underbrace{1111}_{F} = 110101111_2$$

- **Step 2**: Convert Binary to Octal

$$110101111_2 = \underbrace{110}_{6}\ \underbrace{101}_{5}\ \underbrace{111}_{7} = 657_8$$

Search, Read more, Tasks:

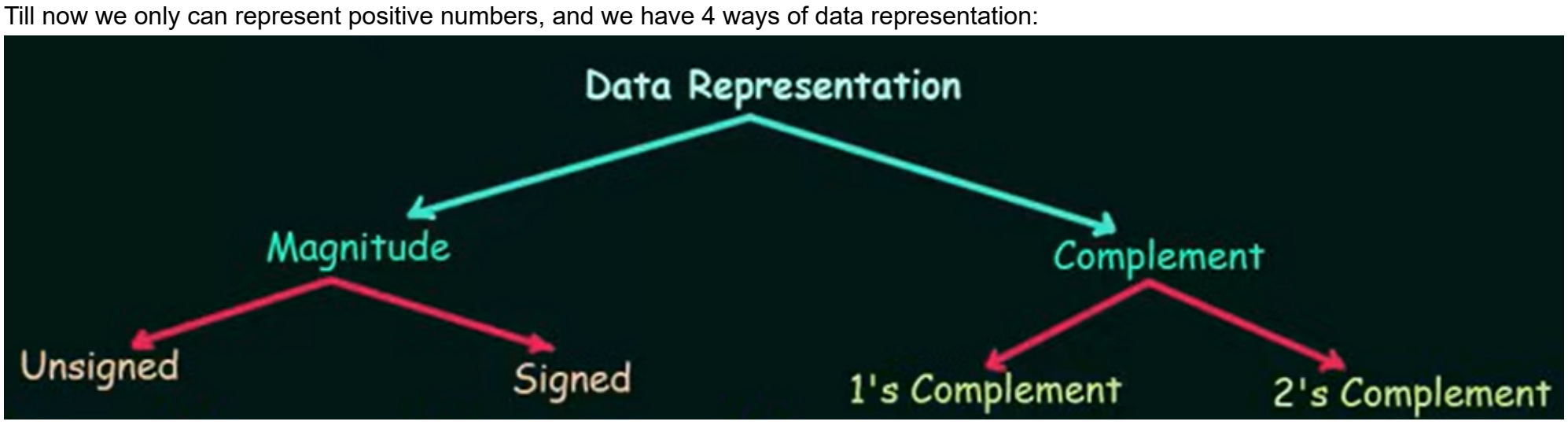
- [ ] Build Code to make conversion in C++
- [ ] Other ways to convert Oct to Hex
- [ ] Other ways to convert Hex to Oct

Resources:
https://youtu.be/ShbCYuomKzc?si=G1F3nJGvg8Af1_y-

Introduction Data Representation

Is the way of how we can represent data (numbers) into binary code

Till now we only can represent positive numbers, and we have 4 ways of data representation:



All these 4 methods, represent the positive numbers in the same way.

Search, Read more, Tasks:

- [ ] Best way of negative in computer
- [ ] Compare all of them in speed and switching number

Signed & Unsigned Magnitude

- **Unsigned:**
  - Use all bits to represent the number
  - Can't represent negatives
  - Range: `2^n - 1` while `n = number of bits`

```
+6 = 110
-6 = can't
```

- **Signed:**
  - Use last bit to represent the sing of the number
  - Can represent Negatives
  - Range: `-(2^(n-1) - 1)` to `2^(n-1) - 1` while `n = number of bits`
  - we have `+0 = 0000000` and `-0 = 1000000`

```
+6 = 0110
-6 = 1110
```

NOTE:

- Now the `1` which was add is called `MSB`
- In normal we have 7 bits, **SO:** `-6 = 1000110` and `6 = 0000110`

Search, Read more, Tasks:

- [ ] Build Code to make conversion in C++
- [ ] Operations on Un\Signed magnitude

1,2's Complement Magnitude

- ## 1's Complement:
  - Use last bit to represent the sing of the number
  - Represent the positive number first, then complement each bit
  - Range: from `-(2^(n-1) - 1)` to `2^(n-1) - 1` while `n = number of bits`

```
+6 = 0110
-6 = 1001
```

NOTE:

- In normal we have 7 bits, **SO:** `6 = 0000110` and `-6 = 1111001`
- In 1's Complement, we have `+0 = 0000000` and `-0 = 1111111`

- ## 2's Complement::
  - Use all bits to represent the number
  - Represent the positive number first, then 1's complement it, then sum 1 to it.
  - Range: from `-(2^(n-1))` to `2^(n-1) - 1` while `n = number of bits`
  - To convert from negative to positive: take 1's complement and then sum to it 1.

```
+6 = 0110
-6 = 1010
```

NOTE:

- In normal we have 7 bits, **SO:** `6 = 0000110` and `-6 = 1111010`
- In 1's Complement, we have only 0
- Last bit indicates sign, and called `MSB`

Search, Read more, Tasks:

- [ ] Build Code to make conversion in C++
- [ ] Operations on Complement

Boolean Algebra

Multiplication = AND
Sum = OR

Boolean Algebra is a set of rules used to represents logical circuits and simplify logical expression and minimizing the number of gates without changing the functionality

Why reducing number of gates?

As much we reduce the number of gates as designing & debugging becomes easier and less cost and more efficient because less operations

Definitions

(a) F1 = x'y'z + x'yz + xy' **(b) F2 = xy' + x'z**

Term
Each term requires a **gate**

3 terms in (a): **x'y'z, x'yz, xy'**
2 terms in (b): **xy', x'z**

Variable
Each variable designates an input to the gate

3 variables in both (a) and (b): **x, y, and z**

Literal
Is a single variable within a term, in complemented or uncomplemented form

8 literals in (a): **x', y', z, x', y, z, x, and y'**
4 literals in (b): **x, y', x', and z**

Boolean Postulates

Boolean postulates come ***dual***

Dual Concept

Dual: means to have 2 equations, where:

- `Or` in the first will be `AND` in the second
- `0` in the first will be `1` in the second
- and via versa
- Dual is can be used to proof the equation

Example:

$$0 + x = x$$

is dual to:

$$x * 1 = x$$

NOTE: Dual equations, are not equal to each others, but they proof each others, and have same output in duality

Example:

$$x + x' = 1$$

is dual to

$$x * x' = 0$$

Some important rules & Proof

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$

Important

Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$

Important

Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

y'*z + y*z' = y XOR z

==**yz + y'z' = y XNOR z

Some tricks

- Consensus Theorem

$$\begin{aligned} F_2 &= xy + x'z + yz \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$

- Complement of function

$$\begin{aligned} F_3 &= (x(y'z' + yz))' \\ &= x' + (y'z' + yz)' \\ &= x' + (y + z)(y' + z') \\ &= x' + yz' + y'z. \end{aligned}$$

This needs DeMorgan law, but instead of distributing, you can use dual proof and invert all to get more simplified shape directilly:

$$\begin{aligned} F_3 &= x(y'z' + yz) \\ dual\ of F_3 &= x + (y' + z')(y + z) \\ F_3' &= x' + (y + z)(y' + z'). \end{aligned}$$

How to master

Practice Proof
Practice simplification
Practice Representation (table, circuit to\from equation)

Search, Read more, Tasks:

- [ ] Redundancy Theorem

Important Proofs

Proof methods

- Drawing truth table
- Using Duality
- Simplifying
- Algebric manipulation

Theorem 1 Proof (Algebric Manipulation)

For (a): x+x=x
RHS: x+x
LHS: x

- 1: What x contains?

$$x = x * 1$$

- 2: New shape: theorem 1

$$x + x = (x + x) * 1$$

- 3: Postulate 5

$$x + x = (x + x)(x + x')$$

- 4: Distribution Law

$$x + x = x + x * x'$$

- 5: Postulate 5

$$x + x = x + 0$$

- 6: Postulate 2

$$x + x = x$$

  now the result equals the right hand side.

For (b): x*x=x
RHS: x * x
LHS: x

- 1: What x contains?

$$x = x * x$$

- 2: New shape: theorem 1

$$x + x = (x * x) + 0$$

- 3: Postulate 5

$$x + x = (x * x) + (x * x')$$

- 4: Distribution Law

$$x + x = x(x + x')$$

- 5: Postulate 5

$$x + x = x * 1$$

- 6: Postulate 2

$$x + x = x$$

Theorem 2 Proof (Algebric Manipulation)

For (a): x+1=1

- 1: Postulate 1

$$x == (x * 1)$$

$$1 == (1 * 1)$$

$$x + 1 == (x * 1) + (1 * 1)$$

- 2: Distributive

$$(x * 1) + (1 * 1) == 1 * (x + 1)$$

- 3: Theorem 2

$$1 * (x + 1) == 1 * 1 == 1$$

For (b): x*0=0

- 1: Postulate 1

$$x == (x + 0)$$

$$0 == (0 + 0)$$

$$x * 0 == (x + 0) * (0 + 0)$$

- 2: Distributive

$$(x + 0) * (0 + 0) == 0 + (x * 0)$$

- 3: Theorem 2

$$0 + (x * 0) == 0 + 0 == 0$$

Theorem 6 Proof

For (a): x+xy=x

- 1: What x contains?

$$x = x * 1$$

- 2: New shape

$$x + xy == (x * 1) + (x * y)$$

- 3: Distribution Law

$$x + xy == x(1 + y)$$

- 4: Theorem 2

$$x(1 + y) == x * 1$$

- 5: Postulate 2

$$x * 1 == x$$

For (b): x(x+y)=x

- 1: Distribution

$$x(x + y) == (x * x) + (x * y)$$

- 2: Theorem 1

$$(x * x) + (x * y) == x + (xy)$$

- 3: Now it is like **(a)** so continue from step 1

OR

- 1: Postulate 2

$$x(x + y) == (x + 0) * (x + y)$$

- 2: Distribute

$$(x + 0) * (x + y) == x + (y * 0)$$

- 3: Theorem 2

$$x + (y * 0) == x + 0$$

- 4: Postulate 2

$$x + 0 == x$$

Search, Read more, Tasks:

- [ ] Theorem 5 proof
- [ ] XOR proof

Boolean Functions - Canonical From

What is a Boolean Function

- Used to express logical relationship between binary variables
- Evaluated by determining the binary value of the expression for all possible values of the variables.
- Boolean function is expressed in 2 forms: **Canonical** & **Standard** From
- **Canonical** & **Standard** From are used to generate a logical expression of a truth table

Canonical Form

- It uses the concept of Minterms & Maxterms

MIN/MAXterms

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

MINterm:

- Equals 1 only at one combination
- number of forms 2^2

MAXterm:

- Equals 0 only at one combination
- number of forms 2^2

NOTES:

- Order of variables is important
- $$m_i = M_i'$$
- The subscript is the order and value of this combination

Express boolean function

x	y	z	Function f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- **SUM/ORing Of MINterms** (***SOM***)
  - By summing all MINterms which equal to 1

    f_1

    or summing all MINterms which equal to 0 to get the complement

    f_1'

    From the last table:

$$f_1 = \sum_{1\ of\ f_1} = \sum(1,4,7) = m_1 + m_4 + m_7$$
$$= x'y'z + xy'z' + xyz,$$
$$f_1' = \sum_{1\ of\ f_1'} = \sum_{0\ of\ f_1} = \sum(0,2,3,5,6) = m_0 + m_2 + m_3 + m_5 + m_6.$$

- **PORDUCT/ANDing Of MAXterms** (***POM***)
  - By adding all MAXterms which equal to 0

    f_1

    or adding all MAXterms which equal to 1 to get the complement

    f_1'

    From the last table:

$$f_1 = \prod_{0\ of\ f_1} = \prod(0,2,3,5,6) = M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$
$$= (x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z),$$
$$f_1' = \prod_{0\ of\ f_1'} = \prod_{1\ of\ f_1} = \prod(1,4,7) = M_1 M_4 M_7.$$

Other way to do ***POM***:

- Get minterms that do 0 f1'
- Take complement of the full function

$$f = \sum_1$$
$$= (m_i + m_j + \cdots)$$
$$f' = (m_i + m_j + \cdots)'$$
$$= m_i' \cdot m_j' \cdots$$
$$= M_i \cdot M_j \cdots$$
$$= \Pi_1$$

Conclusion: For any function f equals 1 on the set *1* and 0 on the set *0*:

$$\sum_{1\ of\ f} = \prod_{0\ of\ f} \Rightarrow f,$$

$$\sum_{0\ of\ f} = \prod_{1\ of\ f} \Rightarrow f'.$$

From Algebric to MIN\MAXterms

Method 2:

- **Another procedure** for deriving the minterms of a Boolean function:

1) Obtain the truth table of the function from the algebraic expression
2) Read minterms from the truth table

1)

Table 2.5
Truth Table for F = A + B'C

A	B	C	F		
0	0	0	0		
0	0	1	1	A'B'C	m1
0	1	0	0		
0	1	1	0		
1	0	0	1	AB'C'	m4
1	0	1	1	AB'C	m5
1	1	0	1	ABC'	m6
1	1	1	1	ABC	m7

Section 2.6

2) From the truth table:

$F = A'B'C + AB'C' + AB'C + ABC' + ABC$

$= m_1 + m_4 + m_5 + m_6 + m_7$

Boolean Functions - Standard From

What is standard form?

- It is used to combining boolean expression terms into terms ANDed\ORed together
- This makes implementation easier and minimize number of gate levels
- Minimizing gate level, cause to avoid delay and async problems
- This is done by implementing the function using `Canonical Forms` (***SOM, POM***) then simplify using **standard form** by using **Sum Of Products** (***SOP***) & **Product Of Sums** (***POS***)

More About Functions

What is a Boolean Function

- Used to express logical relationship between binary variables
- Evaluated by determining the binary value of the expression for all possible values of the variables.
- Boolean function is expressed in 2 forms: **Canonical** & **Standard** From
- **Canonical** & **Standard** From are used to generate a logical expression of a truth table

NOTE:

- Number of functions for ***n*** binary variables: ***2^n^number of rows in truth table***

Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Now each function of them has specific shape

$$
\begin{aligned}
F_0 &= 0, \\
F_1 &= m_3 &&= xy, \\
F_2 &= m_2 &&= xy', \\
F_3 &= m_2 + m_3 &&= xy' + xy \\
F_4 &= m_1 &&= x'y, \\
F_5 &= m_1 + m_3 &&= x'y + xy \\
F_6 &= m_1 + m_2 &&= x'y + xy' \\
F_7 &= M_0 &&= x + y.
\end{aligned}
$$

for the second half (in this example second 8 functions) they are complement to the first half
so F0 = F15'

Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

To understand more, how we got these equations, watch Walead youssif video: Boolean algebra: c from 12:00 to 22:00
to Link: Mathematical Logic + Set theory + Digital design

Other Logic Gates

- From a mathematical view, we don't need more gates than +*' AND OR NOT, Because Other gates are made of these 3 gates
  E.g:
  - XOR = x'y + xy' = AND, OR, NOT
  - XNOR = (x'y + xy')' = (x+y') * (x'+y) = xy + y'x' = AND, OR, NOT
  - NAND = (xy)' = x' + y' = AND, OR, NOT
- Also, AND, OR, NOT are enough to represent any function, because functions are:
  E.g:
  - Combination of **MINterms**
    F = mi+mj+mk...
    and each **MINterm** is combination of **literals** ANDed with each others
    mi = xyz
    and each **literals** is a variable or it's complement
    x or x'
  - Product of **MAXterms**
    F' = Mi*Mj*Mk...
    and each **MAXterms** is combination of **literals** ORed with each others
    Mi = x+y+z
    and each **literals** is a variable or it's complement
    x or x'

So, we can replace NOT AND with NAND, and NOT OR with NOR

For NAND

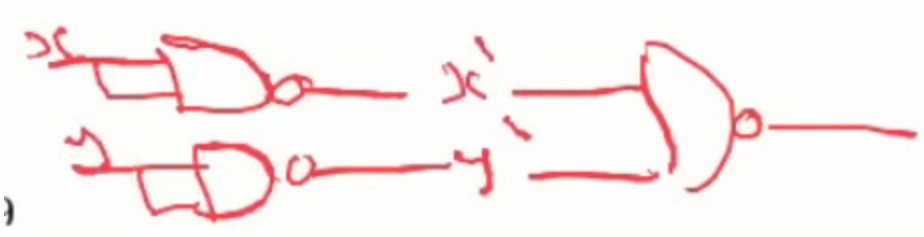
- AND can be made using 2 NANDs

x	y	AND	x NAND y	(x NAND y) NAND (x NAND y)
0	0	0	0 NAND 0 = 1	1 NAND 1 = 0
0	1	0	0 NAND 1 = 1	1 NAND 1 = 0
1	0	0	1 NAND 0 = 1	1 NAND 1 = 0
1	1	1	1 NAND 1 = 0	0 NAND 0 = 1

- NOT can be done by NANDing the variable with itself

y	y NAND y = y'
0	0 NAND 0 = 1
1	1 NAND 1 = 0

- OR can be made using 2 NANDs to complement x,y and one for ORing
  x+y = (x'y')'



x	y	OR	x NAND x = x'	y NAND y = y'	x' NAND y'
0	0	0	0 NAND 0 = 1	0 NAND 0 = 1	1 NAND 1 = 0
0	1	1	0 NAND 0 = 1	1 NAND 1 = 0	1 NAND 0 = 1
1	0	1	1 NAND 1 = 0	0 NAND 0 = 1	0 NAND 1 = 1
1	1	1	1 NAND 1 = 0	1 NAND 1 = 0	0 NAND 0 = 1

For NOR

- OR can be made using 2 NORs to complement x,y and one for ORing

x	y	OR	x NOR y	(x NOR y) NOR (x NOR y)
0	0	0	0 NOR 0 = 1	1 NOR 1 = 0
0	1	1	0 NOR 1 = 0	0 NOR 0 = 1
1	0	1	1 NOR 0 = 0	0 NOR 0 = 1
1	1	1	1 NOR 1 = 0	0 NOR 0 = 1

- NOT can be done by NORing the variable with itself

y	y NOR y = y'
0	0 NOR 0 = 1
1	1 NOR 1 = 0

- AND can be made using 2 NORs

x	y	AND	x NOR x = x'	y NOR y = y'	x' NOR y'
0	0	0	0 NOR 0 = 1	0 NOR 0 = 1	1 NOR 1 = 0
0	1	0	0 NOR 0 = 1	1 NOR 1 = 0	1 NOR 0 = 0
1	0	0	1 NOR 1 = 0	0 NOR 0 = 1	0 NOR 1 = 0
1	1	1	1 NOR 1 = 0	1 NOR 1 = 0	0 NOR 0 = 1

SO, Finally,

$\sum$ can be implemented **only** with NAND.

$\prod$ can be implemented **only** with NOR.

OK, but wait, Why using NAND, NOR, instead of OR,AND,NOT???

To avoid syncronization and delay problems in huge circuits, because each gate needs time, and if your circuit is made from nonUniform gates, this will cause these problems. BUT NOR, NAND help us uniforming gates.

NOTE: in electronics side, NOR, NAND gates are new circuits, that means they are not AND circuit and an inverter added to it.

NOTE:

Unfortunately: $\downarrow$ (NOR), $\uparrow$ (NAND) are not associative:

$$(x \downarrow y) \downarrow z = ((x+y)' + z)' = (x+y)z' = xz' + yz',$$
$$x \downarrow (y \downarrow z) = (x + (y+z)')' = x'(y+z) = x'y + x'z.$$

- Prove (by truth table) that it is commutative and associative; i.e.,

$$x \oplus y = y \oplus x,$$
$$x \oplus (y \oplus z) = (x \oplus y) \oplus z.$$

- Hence, we **DEFINE**:

$$x \oplus y \oplus z = (x \oplus y) \oplus z = x \oplus (y \oplus z)$$

NOTE:

XOR: returns 1 only when number of 1s is odd so if we have x XOR y XOR w XOR z' = 3 ones = return 1
