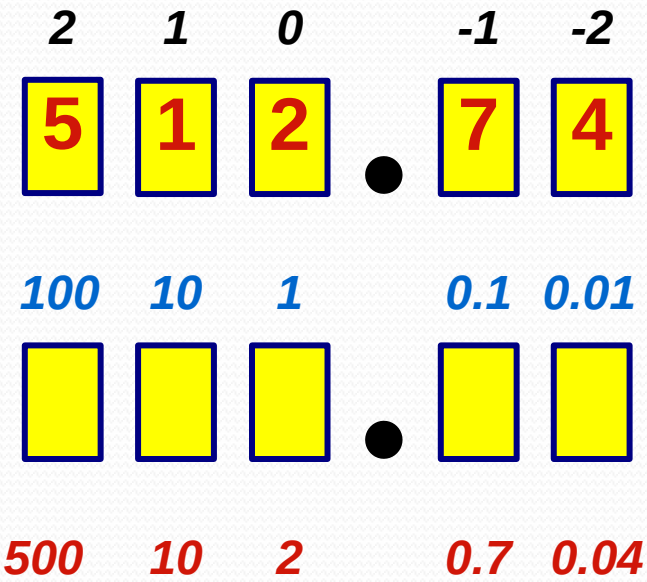
Number Systems and Digital Systems

Decimal Number System

- Base (also called radix) = 10
  - 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Digit Position
  - Integer & fraction
- Digit Weight
  - Weight = $(Base)^{Position}$
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

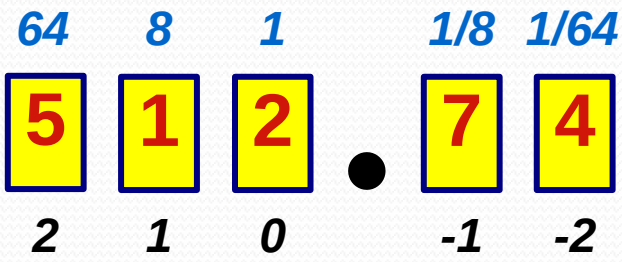


$$d_2*B^2+d_1*B^1+d_0*B^0+d_{-1}*B^{-1}+d_{-2}*B^{-2}$$

$$(512.74)_{10}$$

Octal Number System

- Base = 8
  - 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



$5_2 * 8^2 + 1 * 8^1 + 2 * 8^0 + 7 * 8^{-1} + 4 * 8^{-}$

$= (330.9375)_{10}$

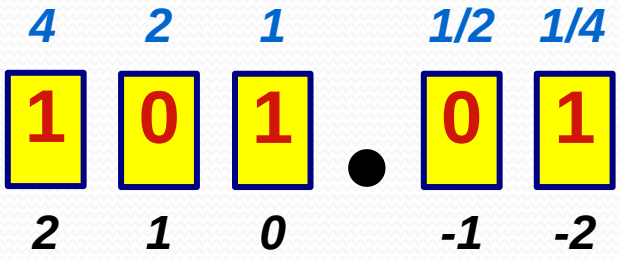
$(512.74)_8$

Binary Number System

- Base = 2
  - 2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Bit* x *Weight*"
- Formal Notation
- Groups of bits

4 bits = *Nibble*

8 bits = *Byte*



$$1_2 *2^2+0 *2^1+1 *2^0+0 *2^{-1}+1 *2^{-2}$$

$$=(5.25)_{10}$$

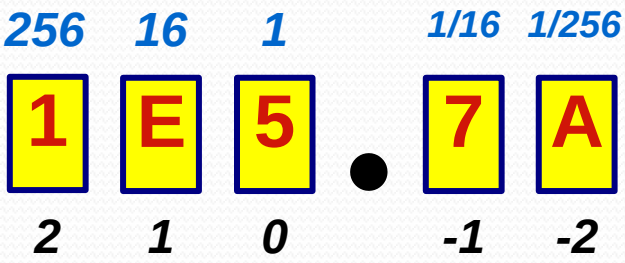
$$(101.01)_2$$

1 0 1 1

1 1 0 0 0 1 0 1

Hexadecimal Number System

- Base = 16
  - 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



$1*16^{2}+14*16^{1}+5*16^{0}+7*16^{-1}+10*16^{-2}$

$=(485.4765625)_{10}$

$(1E5.7A)_{16}$

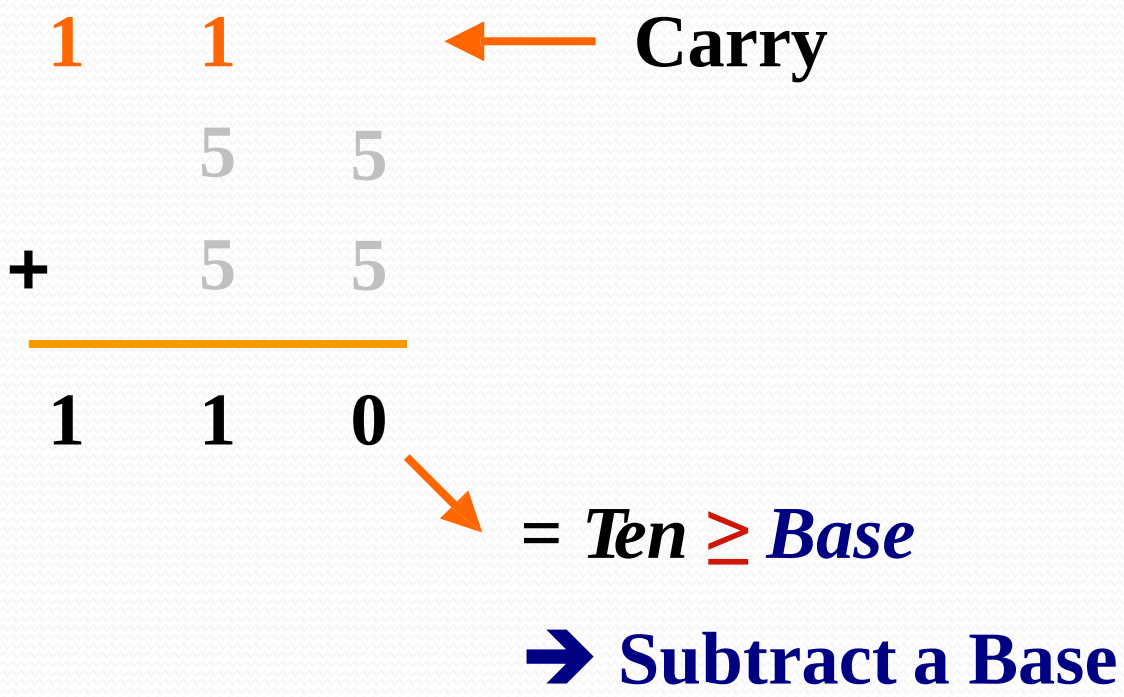
The Power of 2

n	2^n
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
4	$2^4=16$
5	$2^5=32$
6	$2^6=64$
7	$2^7=128$

n	2^n	
8	$2^8=256$	
9	$2^9=512$	
10	$2^{10}=1024$	**Kilo**
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	**Mega**
30	$2^{30}=1G$	**Giga**
40	$2^{40}=1T$	**Tera**

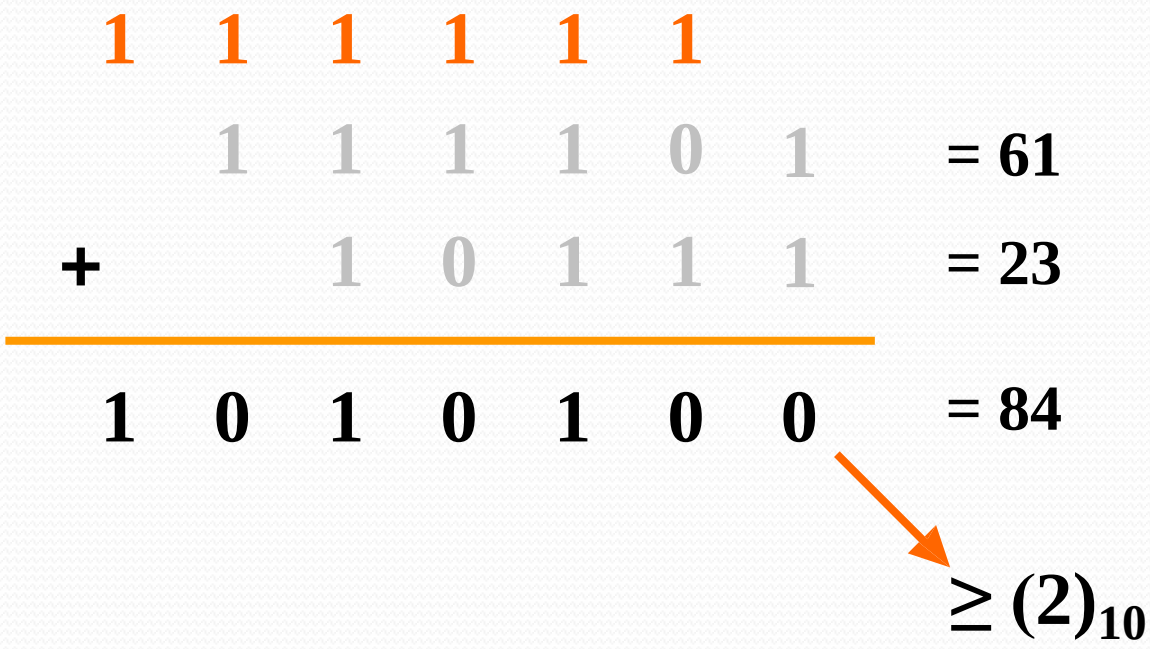
Addition

- Decimal Addition



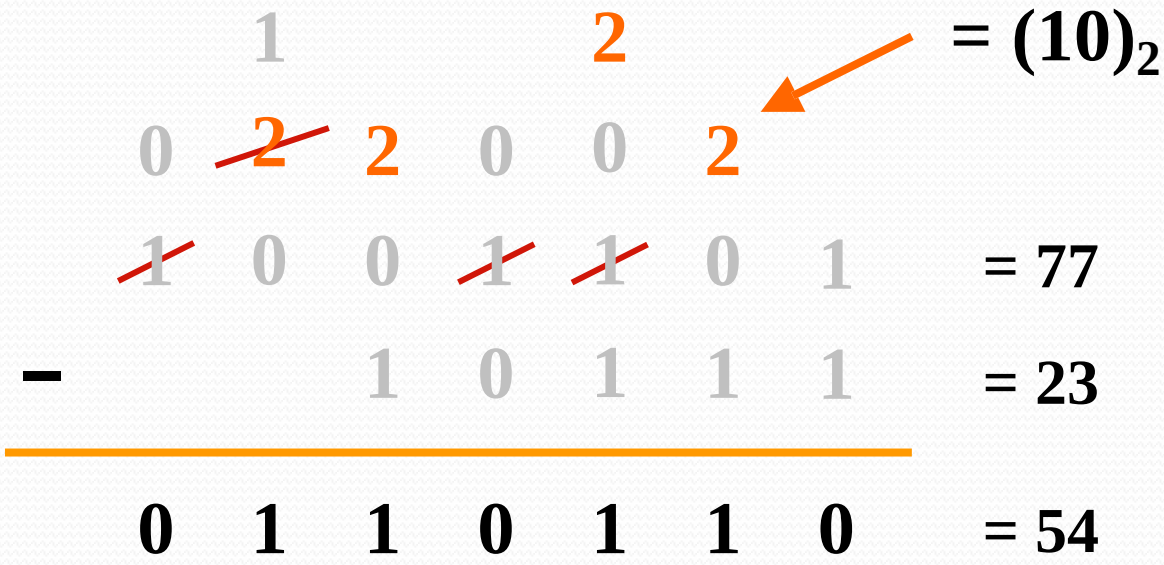
Binary Addition

- Column Addition



Binary Subtraction

- Borrow a “Base” when needed

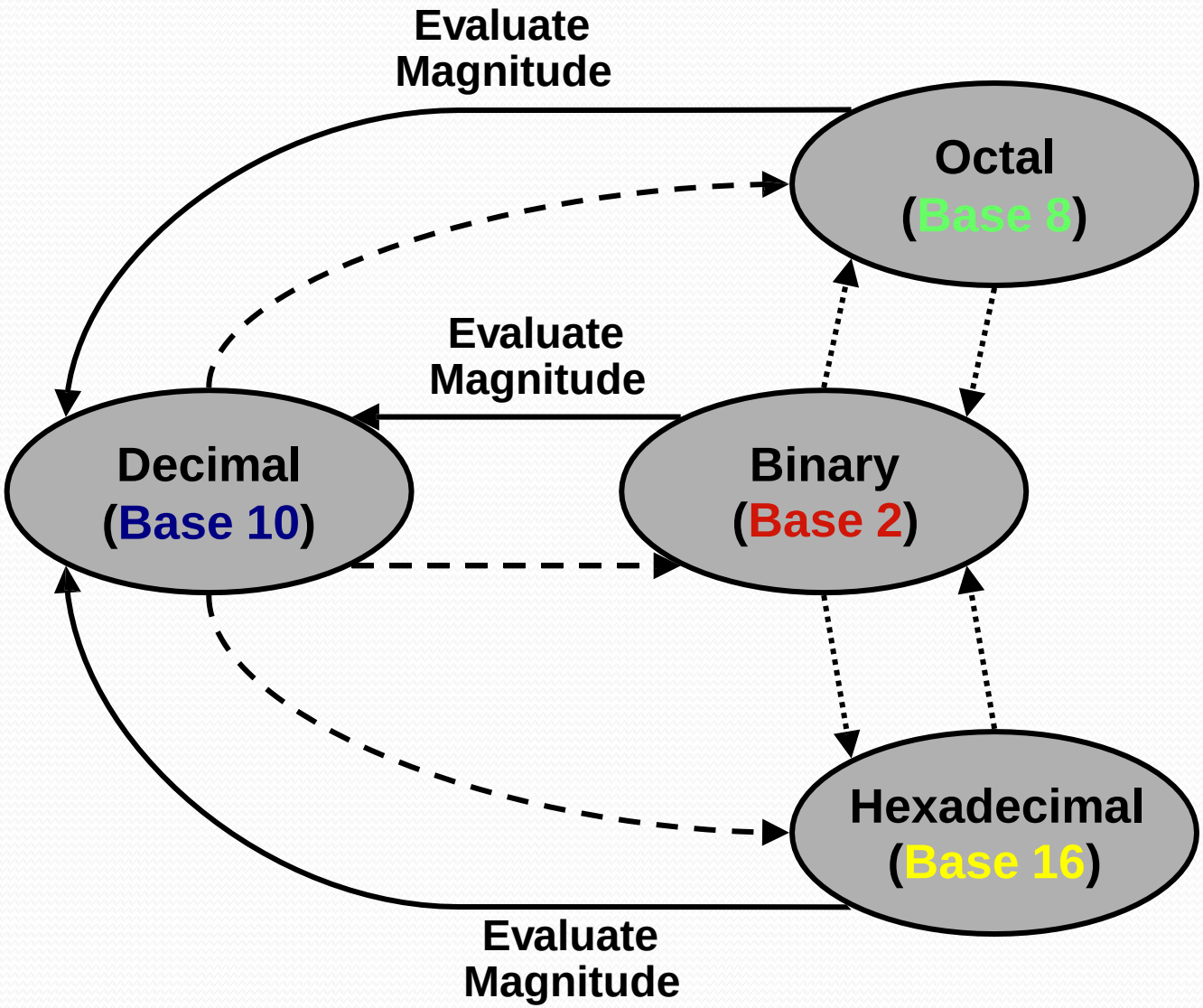


Binary Multiplication

- Bit by bit

			1	0	1	1	1
x				1	0	1	0
			0	0	0	0	0
		1	0	1	1	1	
	0	0	0	0	0		
1	0	1	1	1			
1	1	1	0	0	1	1	0

Number Base Conversions



Decimal (*Integer*) to Binary Conversion

- Divide the number by the 'Base' (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
13 / 2 =	6	1	$a_0 = 1$
6 / 2 =	3	0	$a_1 = 0$
3 / 2 =	1	1	$a_2 = 1$
1 / 2 =	0	1	$a_3 = 1$

Answer: $(13)_{10} = (a_3\, a_2\, a_1\, a_0)_2 = (1101)_2$

MSB LSB

Decimal (*Fraction*) to Binary Conversion

- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division

Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_2 = (0.101)_2$

MSB LSB

Decimal to Octal Conversion

Example: $(175)_{10}$

	Quotient	Remainder	Coefficient
175 / 8 =	21	7	$a_0 = 7$
21 / 8 =	2	5	$a_1 = 5$
2 / 8 =	0	2	$a_2 = 2$

Answer: $(175)_{10} = (a_2\, a_1\, a_0)_8 = (257)_8$

Example: $(0.3125)_{10}$

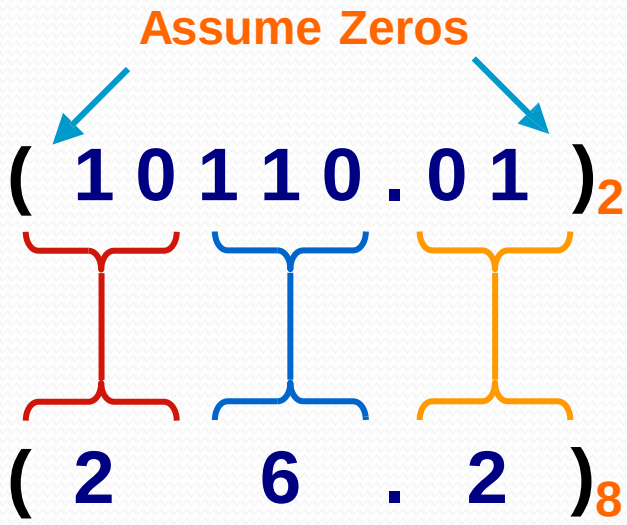
	Integer		Fraction	Coefficient
0.3125 * 8 =	2	.	5	$a_{-1} = 2$
0.5 * 8 =	4	.	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_8 = (0.24)_8$

Binary – Octal Conversion

- $8 = 2^3$
- Each group of 3 bits represents an octal digit

Example:



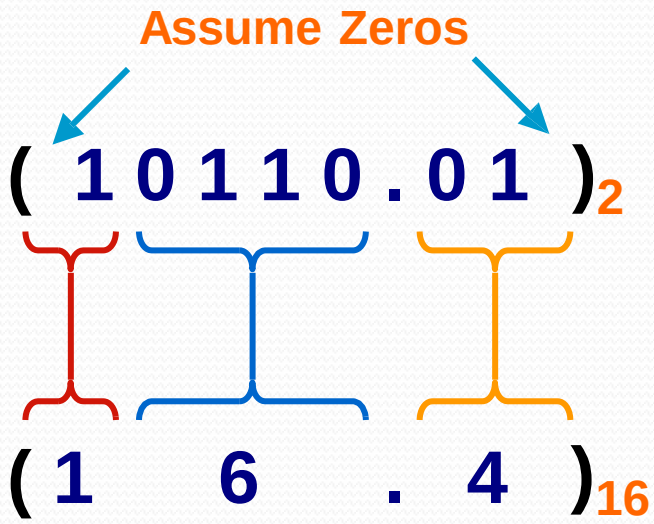
Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Works both ways (*Binary* to *Octal* & *Octal* to *Binary*)

Binary – Hexadecimal Conversion

- $16 = 2^4$
- Each group of 4 bits represents a hexadecimal digit

Example:



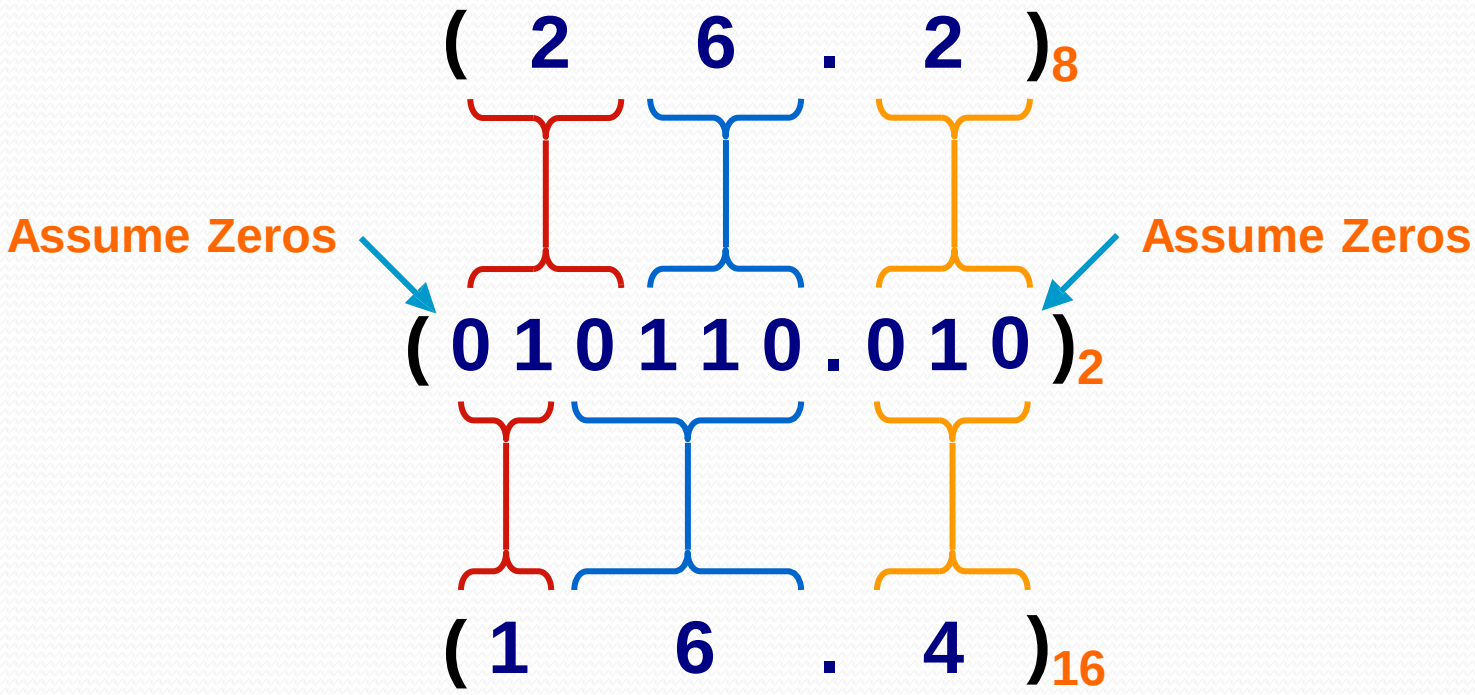
Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Works both ways (*Binary* to *Hex* & *Hex* to *Binary*)

Octal – Hexadecimal Conversion

- Convert to Binary as an intermediate step

Example:



Works both ways (*Octal* to *Hex* & *Hex* to *Octal*)

Decimal, Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hex
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	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

Complements

- There are two types of complements for each base-*r* system: the radix complement and diminished radix complement.
- **Diminished Radix Complement - (r-1)'s Complement**
  - Given a number *N* in base *r* having *n* digits, the (*r–1*)'s complement *of N* is defined as:

$$(r^n - 1) - N$$

- **Example for 6-digit <u>decimal</u> numbers**:
  - 9's complement is $(r^n - 1) - N = (10^6 - 1) - N = 999999 - N$
  - 9's complement of 546700 is 999999–546700 = 453299
- **Example for 7-digit <u>binary</u> numbers:**
  - 1's complement is $(r^n - 1) - N = (2^7 - 1) - N = 1111111 - N$
  - 1's complement of 1011000 is 1111111–1011000 = 0100111
- **Observation:**
  - Subtraction from $(r^n - 1)$ will never require a borrow
  - Diminished radix complement can be computed digit-by-digit
  - For binary: 1 – 0 = 1 and 1 – 1 = 0

Complements

- 1's Complement (*Diminished Radix* Complement)
  - All '0's become '1's
  - All '1's become '0's

Example $(10110000)_2$

⇨ $(01001111)_2$

If you add a number and its 1's complement …

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

Complements

- Radix Complement

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r - 1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r - 1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$.

- Example: Base-10

The 10's complement of 012398 is 987602
The 10's complement of 246700 is 753300

- Example: Base-2

The 2's complement of 1101100 is 0010100
The 2's complement of 0110111 is 1001001

Complements

- 2's Complement (*Radix* Complement)
  - Take 1's complement then add 1

OR

  - Toggle all bits to the left of the first '1' from the right

Example:

Number:

1's Comp.:

	1 0 1 1 0 0 0 0	**1 0 1 1 0 0 0 0**
	0 1 0 0 1 1 1 1	
+	**1**	
	0 1 0 1 0 0 0 0	**0 1 0 1 0 0 0 0**

Complements

- Subtraction with Complements
  - The subtraction of two *n*-digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geqq N$, the sum will produce and end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Complements

- Example 1.5
  - Using 10's complement, subtract 72532 – 3250.

	$M =$	72532
10's complement of	$N =$	+96750
	Sum =	169282
	Discard end carry 10^5 =	–100000
	Answer =	69282

- Example 1.6
  - Using 10's complement, subtract 3250 – 72532.

	M =	03250
10's complement of	N =	+27468
	Sum =	30718

→ There is no end carry.

→ Therefore, the answer is – (10's complement of 30718) = – 69282.

Complements

- Example 1.7
  - Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$; and (b) $Y - X$, by using 2's complement.

(a)		
	X =	1010100
	2's complement of Y =	+0111101
	Sum =	10010001
	Discard end carry 2^7 =	−10000000
	Answer. X – Y =	0010001

(b)		
	Y =	1000011
	2's complement of X =	+ 0101100
	Sum =	1101111

There is no end carry. Therefore, the answer is Y – X = – (2's complement of 1101111) = – 0010001.

Complements

- Subtraction of unsigned numbers can also be done by means of the $(r-1)$'s complement. Remember that the $(r-1)$'s complement is one less then the r's complement.
- Example 1.8
  - Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$X =$	1010100
1's complement of $Y =$	± 0111100
Sum =	10010000
End-around carry =	+ 1
Answer. $X - Y =$	0010001

(b) $Y - X = 1000011 - 1010100$

$Y =$	1000011
1's complement of $X =$	+ 0101011
Sum =	1101110

There is no end carry, Therefore, the answer is Y – X = – (1's complement of 1101110) = – 0010001.

Data Representation

The ways to represent the data are:

- Unsigned Representation
- Signed Representation
  - Signed magnitude representation
  - Signed 1’s complement representation
  - Signed 2’s complement Representation

1.6 Signed Binary Numbers

- To represent negative integers, we need a notation for negative values.
- It is customary to represent the sign with a bit placed in the leftmost position of the number since binary digits.
- The convention is to make the sign bit 0 for positive and 1 for negative.
- Example:

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

Weighted and Non weighted Code

- **Weighted Code:-**
  - In weighted code, each digit position has a weight or value. The sum of all digits multiplied by a weight gives the total amount being represented.
  - We can express any decimal number in tens, hundreds, thousands and so on.
  - Eg:- Decimal number 4327 can be written as

    $4327 = 4000+300+20+7$
  - In the power of 10, it becomes

    $4327 = 4(10^3)+3(10^2)+2(10^1)+7(100)$
  - BCD or 8421 is a type of weighted code where each digit position is being assigned a specific weight.

- **Non-weighted code:-**
  - In non-weighted code, there is no positional weight i.e. each position within the binary number is not assigned a prefixed value. No specific weights are assigned to bit position in non –weighted code.
  - The non-weighted codes are:-

    a) The Gray code b) The Excess-3 code

Binary Codes

BCD or 8421 code:-

- It is composed of four bits representing the decimal digits 0 through 9.
- The 8421 indicates the binary weights of the four bits(2^3,2^2,2^1,2^0).
- A number with k decimal digits will
- require 4k bits in BCD.
- Decimal 396 is represented in BCD with 12bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.
- A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.
- The binary combinations 1010 through 1111 are not used and have no meaning in BCD.

Table 1.4
Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

BCD or 8421 Code

- Convert following to BCD form

  a)47310 b) 31210 c) 25710 d) 11210

- Convert following BCD to their decimal equivalent

  a)10000110 b) 00110010.10010100

- Convert the following binary number numbers to their BCD equivalent.

  a)1101.012 b) 11.0112

  (Binary to decimal Decimal to BCD)

Comparison of BCD with Binary

1. BCD is less efficient than binary

eg:- $(78)_{10} = (0111\ 1000)_{BCD}$

$=(1001110)_2$

To encode the same decimal number , BCD needs more no. of bits than binary . Hence BCD is less efficient as compared to Binary

2. BCD arithmetic is more complicated than binary arithmetic.

3. Advantage of a BCD code is that the conversion from decimal to BCD or vice versa is simple.

BCD or 8421 Code

Decimal	BCD
weight	8 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Unused or Don't care conditions

1010
1011
1100
1101
1110
1111

Comparison of BCD with Binary

1. BCD is less efficient than binary

eg:- $(78)_{10} = (0111\ 1000)_{BCD}$

$=(1001110)_2$

To encode the same decimal number , BCD needs more no. of bits than binary . Hence BCD is less efficient as compared to Binary

2. BCD arithmetic is more complicated than binary arithmetic.

3. Advantage of a BCD code is that the conversion from decimal to BCD or vice versa is simple.

8 4 -2 -1 Code

Decimal	84-2-1 code
weight	8 4 -2 -1
0	0 0 0 0
1	0 1 1 1
2	0 1 1 0
3	0 1 0 1
4	0 1 0 0
5	1 0 1 1
6	1 0 1 0
7	1 0 0 1
8	1 0 0 0
9	1 1 1 1

Unused or Don't care conditions

0001
0010
0011
1100
1101
1110

2421 Code

Decimal	2421 Code
weight	2 4 2 1
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	1 0 1 1
6	1 1 0 0
7	1 1 0 1
8	1 1 1 0
9	1 1 1 1

Unused or Don't care conditions

0101
0110
0111
1000
1001
1010

Excess-3 Code

Decimal	Excess-3 code
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

Unused or Don't care conditions

0000
0001
0010
1101
1110
1111

Decimal	BCD	Excess-3	8 4 -2 -1	2 4 2 1
0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0
1	0 0 0 1	0 1 0 0	0 1 1 1	0 0 0 1
2	0 0 1 0	0 1 0 1	0 1 1 0	0 0 1 0
3	0 0 1 1	0 1 1 0	0 1 0 1	0 0 1 1
4	0 1 0 0	0 1 1 1	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 0 0	1 0 1 1	1 0 1 1
6	0 1 1 0	1 0 0 1	1 0 1 0	1 1 0 0
7	0 1 1 1	1 0 1 0	1 0 0 1	1 1 0 1
8	1 0 0 0	1 0 1 1	1 0 0 0	1 1 1 0
9	1 0 0 1	1 1 0 0	1 1 1 1	1 1 1 1

The Gray code

❖ **Introduction:**

It is non weighted code in which each number differs from previous number by a single bit.

or

Only one bit is changed as we move from one number to another successive number.

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

Dec.	Binary	Gray	Dec.	Binary	Gray
0	0 0 0 0	0 0 0 0	14	1 1 1 0	1 0 0 1
1	0 0 0 1	0 0 0 1	15	1 1 1 1	1 0 0 0
2	0 0 1 0	0 0 1 1			
3	0 0 1 1	0 0 1 0			
4	0 1 0 0	0 1 1 0			
5	0 1 0 1	0 1 1 1			
6	0 1 1 0	0 1 0 1			
7	0 1 1 1	0 1 0 0			
8	1 0 0 0	1 1 0 0			
9	1 0 0 1	1 1 0 1			
10	1 0 1 0	1 1 1 1			
11	1 0 1 1	1 1 1 0			
12	1 1 0 0	1 0 1 0			
13	1 1 0 1	1 0 1 1			

The Excess-3 code

- It is an important BCD code , is a 4 bit code and used with BCD numbers
- To convert any decimal numbers into its excess-3 form ,add 3 to each decimal digit and then convert the sum to a BCD number
- As weights are not assigned, it is a kind of non weighted codes.

- Convert the following into Excess-3 number

  a)149 b) 2546 c) 152 d) 2694

- Add the following numbers in excess-3 code

  a) 108+789 b) 275+496

BCD addition

- Add two numbers as same as binary addition
  - Case 1: If the result is less than or equals to 9 and carry is zero then it is valid BCD.
  - Case 2: If result is greater than 9 and carry is zero then add 6 in four bit combination.
  - Case 3: If result is less than or equals to 9 but carry is 1 then add 6 in four bit combination.

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	+1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

BCD Addition

- Example:
  - Consider the addition of 184 + 576 = 760 in BCD:

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6	____	0110	0110	____
BCD sum	0111	0110	0000	760

Binary Codes

- Other Decimal Codes

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

Error-Detection code

❖ **Introduction:**

Electric wires or other communication medium can transmit binary information from one location to another. Any external noise introduced into the physical communication medium may change some of the bits from 0 to 1 or vice versa. The purpose of an error-detection code is to detect such bit-reversal errors. One of the most common ways to achieve error detection is by means of a **parity bit**. A parity bit is the extra bit included to make the total number of 1's in the resulting code word either even or odd.

Even Parity : Here the total number of 1's in the message is made even

Odd Parity : Here the total number of 1's in the message is made odd

A message of 4-bits and a parity bit P are shown in the table below:

Odd parity		Even parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

A message of 3-bits and a parity bit P are shown in the table below:

3-bit Message			Odd Parity Bit	Even Parity Bit
X	Y	Z		
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Alpha-Numeric Codes

1. American Standard Code for Information Interchange (ASCII) Character Code

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

ASCII

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

ASCII Character Codes

American Standard Code for Information Interchange.

A popular code used to represent information sent as character-based data.

It uses 7-bits to represent.

ASCII uses a linear ordering of letters.

ASCII has a maximum of 128 characters.

Alpha-Numeric Codes

2. Extended Binary Coded Decimal Interchange Code(EBCDIC)

It uses 8-bits to represent.

It does not use a linear ordering of letters.

It has a maximum of 256 characters.

Each code is shown in decimal, hexadecimal, and character form.

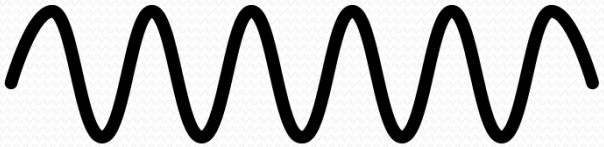
129	81	a	193	C1	A	240	F0	0
130	82	b	194	C2	B	241	F1	1
131	83	c	195	C3	C	242	F2	2
132	84	d	196	C4	D	243	F3	3
133	85	e	197	C5	E	244	F4	4
134	86	f	198	C6	F	245	F5	5
135	87	g	199	C7	G	246	F6	6
136	88	h	200	C8	H	247	F7	7
137	89	i	201	C9	I	248	F8	8
						249	F9	9
145	91	j	209	D1	J			
146	92	k	210	D2	K	64	40	blank
147	93	l	211	D3	L	76	4C	<
148	94	m	212	D4	M	77	4D	(
149	95	n	213	D5	N	78	4E	+
150	96	o	214	D6	O	79	45	|
151	97	p	215	D7	P	80	50	&
152	98	q	216	D8	Q	90	5A	!
153	99	r	217	D9	R	91	5B	$
						92	5C	*
162	A2	s	226	E2	S	93	5D	)
163	A3	t	227	E3	T	94	5E	;
164	A4	u	228	E4	U	96	60	-
165	A5	v	229	E5	V	97	61	/
166	A6	w	230	E6	W	107	6B	,
167	A7	x	231	E7	X	108	6C	%
168	A8	y	232	E8	Y	109	6D	_
169	A9	z	233	E9	Z	110	6E	>
						111	6F	?
122	7A	:	125	7D	'			
123	7B	#	126	7E	=			
124	7C	@	127	7F	"			

Digital Systems

- Digital age and information age
- Digital computers
  - General purposes
  - Many scientific, industrial and commercial applications
- Digital systems
  - Telephone switching exchanges
  - Digital camera
  - Electronic calculators,
  - Digital TV

Analog and Digital Signal

- Analog system
  - The physical quantities or signals may vary continuously over a specified range.
- Digital system
  - The physical quantities or signals can assume only discrete values.



Analog Signal

Digital Signal

Logic Gates and Boolean Algebra

What are Gates?

- The building blocks that creates digital circuits are logic gates.
- Each gate has its own logic symbol which allows complex functions to be represented by a logic diagram
- The function of each gate can be represented by a truth table or using Boolean notation

LOGIC GATES

- Types of gates
  - NOT
  - AND
  - OR
  - NAND
  - NOR
  - EX-OR
  - EX-NOR
  - BUFFER GATE

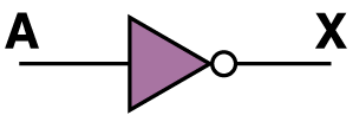
NOT Gate

- A NOT gate accepts one input value and produces one output value
- By definition, if the input value for a NOT gate is 0, the output value is 1, and if the input value is 1, the output is 0
- A NOT gate is sometimes referred to as an *inverter* because it inverts the input value

Boolean Expression

$X = A'$

Logic Diagram Symbol



Truth Table

A	X
0	1
1	0

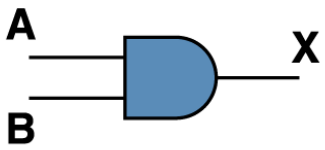
AND Gate

- An AND gate accepts two input signals
- If the two input values for an AND gate are both 1, the output is 1; otherwise, the output is 0

Boolean Expression

$$X = A \cdot B$$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

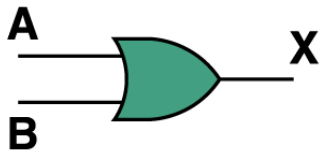
OR Gate

- If the two input values are both 0, the output value is 0; otherwise, the output is 1

Boolean Expression

$X = A + B$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

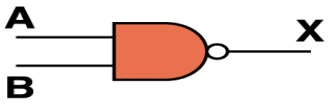
NAND and NOR Gates

- The NAND and NOR gates are essentially the opposite of the AND and OR gates, respectively.
- They are also called universal gates.

Boolean Expression

$X = (A \cdot B)'$

Logic Diagram Symbol



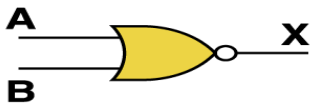
Truth Table

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Boolean Expression

$X = (A + B)'$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

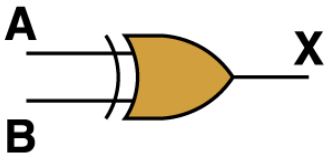
XOR Gate

- XOR, or *exclusive* OR, gate
  - An XOR gate produces 0 if its two inputs are the same, and a 1 otherwise

Boolean Expression

$$X = A \oplus B$$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

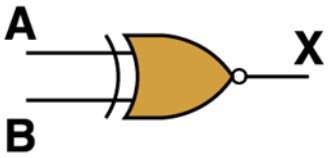
XNOR Gate

- XNOR, or *exclusive NOR*, gate
  - An XNOR gate produces 0 if its two inputs are the different, and a 1 otherwise

Boolean Expression

$$X = A \odot B$$

Logic Diagram Symbol



Truth Table

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebra

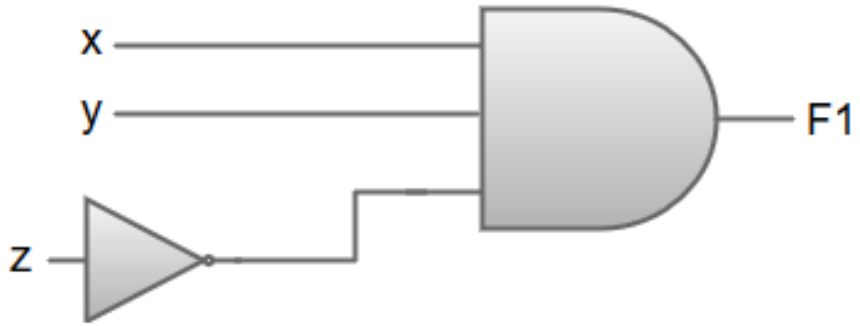
- Mathematical system, formulate logic with symbols.
- Mathematics of digital system.
- Algebraic Manipulation

Boolean Function

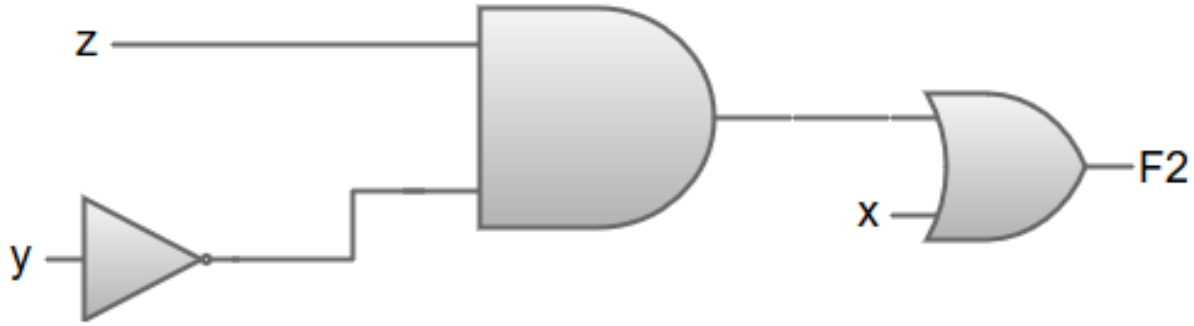
- Expression using binary operators.
- Bracket, NOT, AND, OR
- Eg; F =x.y’.z

Boolean algebra simplifications using logic gates

1. F1 = xyz'



2. F2 = x + y'z



Theorems

Boolean Expression
x * 0 = 0
x + 1 = 1
x * 1= x
x + 0 = x
x * x = x
x + x = x
x * x’ =0
x + x’ =1
(x’)’ = x

Theorems

Commutative Law

It that the interchanging of the order of operands in a Boolean equation does not change its result. For example:

OR operator → A + B = B + A
AND operator → A . B = B . A

Associative Law

It states that the AND operation are done on two or more than two variables. For example:

A .(B.C) = (A.B).C A+(B+C) = (A+B)+C

Distributive Law

It states that the multiplication of two variables and adding the result with a variable will result in the same value as multiplication of addition of the variable with individual variables. For example:

A + BC = (A + B) (A + C) and A .(B+C) = AB+ AC

Theorems

De Morgan's Theorem

(A+B)' = A'.B' (A+B+C)' = A'.B'.C'

(A.B)' = A'+B' (A.B.C)' = A'+B'+C'

Duality Principle:

To find the dual of any expression:

- change the operator
- 0 > 1, 1>0

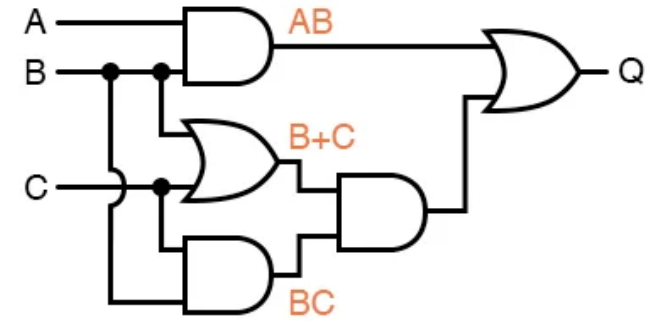
Eg: a+a'= 1 dual : a.a'=0

How to find complement?

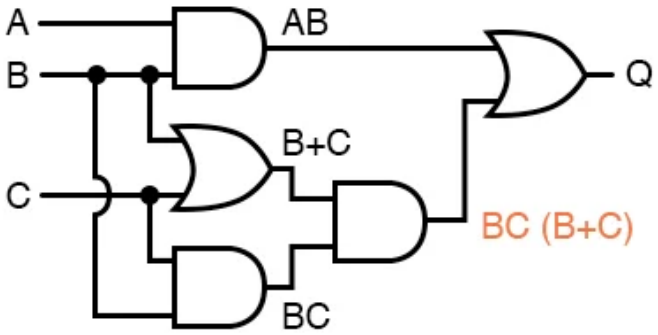
- find dual
- complement each variable.

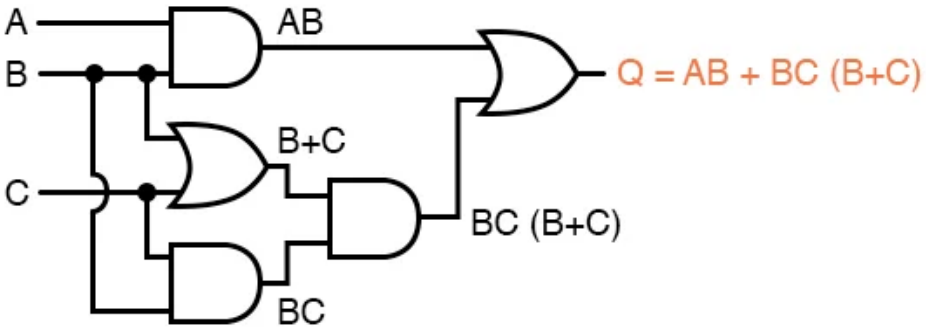
Eg: F= x+y 1.dual = x.y 2. F'= x'.y'

Gate Implementation



Q=A.B+B.C.(B+C)





Simplification of Boolean Functions

Simplify the following Boolean functions to a minimum number of literals.

- $x+x'.y = (x+x').(x+y) = 1.(x+y) = (x+y)$
- $x(x'+y) = x.x' + xy = 0+xy = xy$
- $x'y'z + x'yz + xy' = x'z(y+y') + xy' = x'z+xy'$
- $xy + x'z + yz = xy+yz + x'z + x.x' = y(x+z)+x'(x+z) = (x+z)(y+x')$
- $(x+y)(x'+z)(y+z)$

Find the complement of F = x+y’z.
Also prove F.F’= 0 and F+F’= 1.

Implement F = xy+x’y’+yz’

- using AND,OR and NOT gates
- using NOT and OR only
- using NOT and AND only

Q= A’B’+A’C+B’C’
Q= A’C+B’C’

Redundant Theorem:

1. 3 variables
2. Each variable repeated twice
3. One variable must be complemented

- Take complemented variable and remaining is redundant.

F = AB+BC’+AC = BC’+AC
F =(A+B).(A’+C).(B+C) = A+B).(A’+C)

Find the complement of F = x+y'z. Also prove F.F'= 0 and F+F'= 1.

F' = (x+y'z)' = x'.(y'.z)' = x'.(y+z') = x'y+x'z'

To prove: F.F'=0

LHS. (X+y'z). (x'y+x'z')

To prove: F+F'=1

LHS. X+y'z+x'y+x'z'

= x+x'y+y'z+x'z'

= X+y+y'z+x'z'

= x+x'z'+y+y'z

= A+ 1

=1

F+F' = [(F+F')']' = [(F'.F)]' = [0]' = 1

Canonical and Standard Forms

- Boolean functions expressed as a sum of minterms or product of maxterms are said to be in **canonical.**
  - Minterms and Maxterms
  - Sum-of-Minterms and Product-of- Maxterms
  - Product and Sum terms
  - Sum-of-Products (SOP) and Product-of-Sums (POS)

No of minterms = no of maxterms = 2^n

Definitions

- *Literal:* A variable or its complement
- *Product term:* literals connected by •
- *Sum term:* literals connected by +
- *Minterm:* a product term in which all the variables appear exactly once, either complemented or uncomplemented
- *Maxterm:* a sum term in which all the variables appear exactly once, either complemented or uncomplemented

Minterm

- Represents exactly one combination in the truth table.
- Denoted by m_j, where j is the decimal equivalent of the minterm's corresponding binary combination (b_j).
- A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.
- Example: Assume 3 variables (A,B,C), and j=3. Then, b_j = 011 and its corresponding minterm is denoted by m_j = A'BC

Maxterm

- Represents exactly one combination in the truth table.
- Denoted by M_j, where j is the decimal equivalent of the maxterm's corresponding binary combination (b_j).
- A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.
- Example: Assume 3 variables (A,B,C), and j=3. Then, b_j = 011 and its corresponding maxterm is denoted by M_j = A+B'+C'

Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truth table.
- Example:
  Assume 3 variables x,y,z (order is fixed)

Minterms: x=0; x'

x=1; x

And vice versa for maxterm

x	y	z		Minterm	Maxterm
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		$xyz = m_7$	$x'+y'+z' = M_7$

Canonical Forms

- Any Boolean function F( ) can be expressed as a *unique* **sum** of **min**terms and a unique **product** of **max**terms (under a fixed variable ordering).
- In other words, every function F() has two canonical forms:
  - Canonical Sum-Of-Products (sum of minterms)
  - Canonical Product-Of-Sums (product of maxterms)

Canonical Forms (cont.)

- Canonical Sum-Of-Products:
  The minterms included are those m_j such that F( ) = 1 in row *j* of the truth table for F( ).
- Canonical Product-Of-Sums:
  The maxterms included are those M_j such that F( ) = 0 in row *j* of the truth table for F( ).

Example

- Truth table for $f_1(a,b,c)$ at right
- The canonical sum-of-products form for f_1 is

  $f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$

  $= a'b'c + a'bc' + ab'c' + abc'$
- The canonical product-of-sums form for f_1 is

  $f_1(a,b,c) = M_0 \cdot M_3 \cdot M_5 \cdot M_7$

  $= (a+b+c)\cdot(a+b'+c')\cdot$

  $(a'+b+c')\cdot(a'+b'+c').$
- Observe that: $m_j = M_j'$

a	b	c		f_1
0	0	0		0
0	0	1		1
0	1	0		1
0	1	1		0
1	0	0		1
1	0	1		0
1	1	0		1
1	1	1		0

Shorthand: ∑ and ∏

- $f_1(a,b,c) = \sum m(1,2,4,6)$, where ∑ indicates that this is a sum-of-products form, and m(1,2,4,6) indicates that the minterms to be included are m_1, m_2, m_4, and m_6.
- $f_1(a,b,c) = \prod M(0,3,5,7)$, where ∏ indicates that this is a product-of-sums form, and M(0,3,5,7) indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.
- Since $m_j = M_j'$ for any *j*,
  $\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$

Conversion Between Canonical Forms

- Replace $\sum$ with $\prod$ (or *vice versa*) and replace those *j*'s that appeared in the original form with those that do not.
- Example:

$f_1(a,b,c) = a'b'c + a'bc' + ab'c' + abc'$

$= m_1 + m_2 + m_4 + m_6$

$= \sum(1,2,4,6)$

$= \prod(0,3,5,7)$

$= (a+b+c)\cdot(a+b'+c')\cdot(a'+b+c')\cdot(a'+b'+c')$

Conversion to SOP

- Expand *non-canonical* terms by inserting equivalent of 1 in each missing variable x:
  $(x + x') = 1$
- Remove duplicate minterms
- $f_1(a,b,c) = a'b'c + bc' + ac'$

$$= a'b'c + (a+a')bc' + a(b+b')c'$$
$$= a'b'c + abc' + a'bc' + abc' + ab'c'$$
$$= a'b'c + abc' + a'bc + ab'c'$$

Conversion to POS

- Expand noncanonical terms by adding 0 in terms of missing variables (*e.g.*, xx' = 0) and using the distributive law
- Remove duplicate maxterms
- $f_1(a,b,c)$ = (a+b+c)•(b'+c')•(a'+c')
  = (a+b+c)•(aa'+b'+c')•(a'+bb'+c')
  = (a+b+c)•(a+b'+c')•(a'+b'+c')•
  (a'+b+c')•(a'+b'+c')
  = (a+b+c)•(a+b'+c')•(a'+b'+c')•(a'+b+c')

F(A,B,C)=A+B’C

For SOP conversion;

F=A+B’C

F=A(B+B’)(C+C’)+B’C(A+A’)

F= (AB+AB’)(C+C’)+ AB’C+A’B’C

F= ABC + ABC’ + AB’C + AB’C’ + AB’C+A’B’C

F= ABC + ABC’ + AB’C + AB’C’ +A’B’C

F = ∑ m(....)

F(A,B,C)=A+B’C

For POS conversion;

F=A+B’C

F=(A+B’).(A+C) [Similar to POS]

F = (A+B’+C.C’).(A+B.B’+C)

F= (A+B’+C).(A+B’+C’).(A+B+C).(A+B’+C)

F= (A+B’+C).(A+B’+C’).(A+B+C)

F = ∏ M(…)

Karnaugh Maps

Karnaugh Maps

- Karnaugh maps (K-maps) are *graphical* representations of boolean functions.
- One ***cell*** corresponds to a row in the truth table.
- Also, one cell corresponds to a minterm or a maxterm in the boolean expression
- Multiple-cell areas of the map correspond to standard terms.

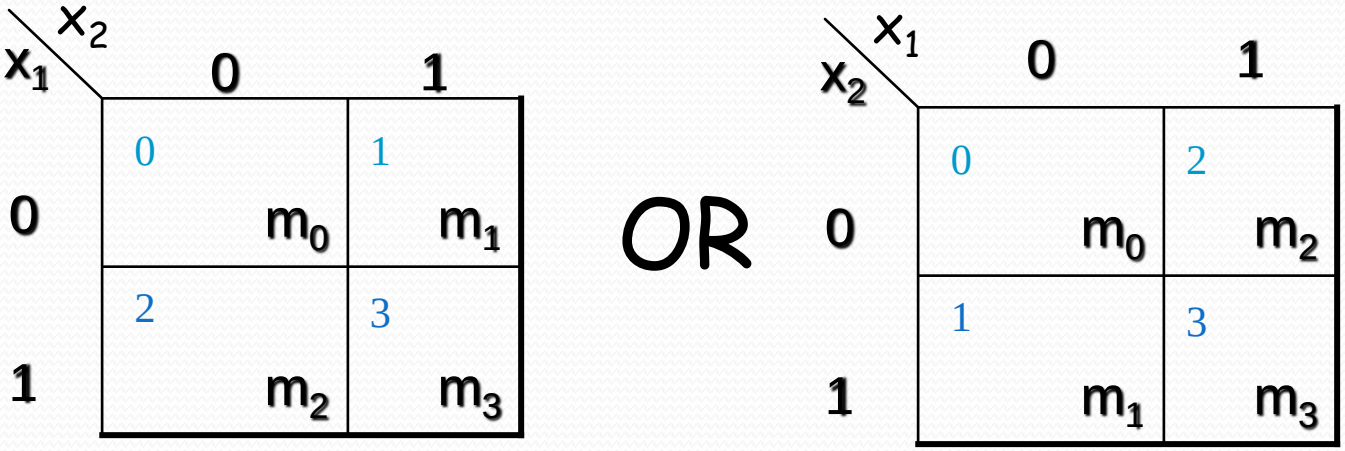
Minimization by K-map (Karnaugh map)

- Algebraic minimization of Boolean functions is rather awkward because it lacks specific rules to predict each succeeding step in the manipulative process.
- The map method provides a simple straightforward procedure for minimizing Boolean functions. This method may be regarded as a pictorial form of a truth table.
- The map method, first proposed by Veitch and modified by Karnaugh, is also known as the "Veitch diagram" or the "Karnaugh map."
  - The k-map is a diagram made up of grid of squares.
  - Each square represents one minterm.
  - The minterms are ordered according to Gray code (only one variable changes between adjacent squares).
  - Squares on edges are considered adjacent to squares on opposite edges.

Minimization by K-map (Karnaugh map)

- Karnaugh maps become clumsier to use with more than 4 variables.
- In fact, the map presents a visual diagram of all possible ways a function may be expressed in a standard form. By recognizing various patterns, the user can derive alternative algebraic expressions for the same function, from which he can select the simplest one.
- We shall assume that the simplest algebraic expression is anyone in a sum of products or product of sums that has a minimum number of literals. (This expression is not necessarily unique)

Two-Variable Map



NOTE: ordering of variables is IMPORTANT for $f(x_1,x_2)$, x_1 is the row, x_2 is the column.

Cell 0 represents $x_1'x_2'$; Cell 1 represents $x_1'x_2$; etc. If a minterm is present in the function, then a 1 is placed in the corresponding cell.

Two-variable Map Simplification

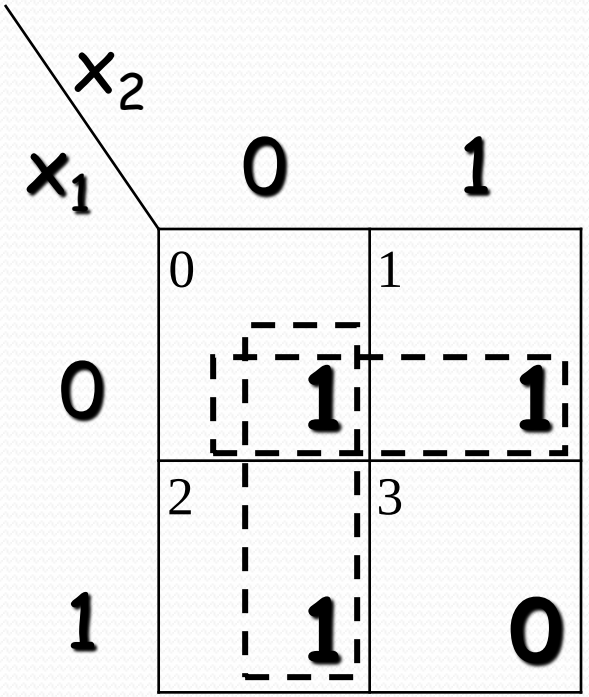
- One square(uncombined) represents a minterm of 2 literals.
- A combination of 2 adjacent squares represents a product term of single literal.
- A combination of 4 squares represents a functional value equal to 1. (F=1)

Two-Variable Map (cont.)

- Any two adjacent cells in the map differ by ONLY one variable, which appears complemented in one cell and uncomplemented in the other.
- Example:
  m_0 ($=x_1'x_2'$) is adjacent to m_1 ($=x_1'x_2$) and m_2 ($=x_1x_2'$) but NOT m_3 ($=x_1x_2$)
- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m0, m1), (m2, m3), (m0, m2) and (m1, m3)}.

2-Variable Map -- Example

- $f(x_1,x_2) = x_1'x_2' + x_1'x_2 + x_1x_2'$
  $= m_0 + m_1 + m_2$
  $= x_1' + x_2'$
- 1s placed in K-map for specified minterms m_0, m_1, m_2
- Grouping (ORing) of 1s allows simplification
- What (simpler) function is represented by each dashed rectangle?
  - $x_1' = m_0 + m_1$
  - $x_2' = m_0 + m_2$
- Note m_0 covered twice

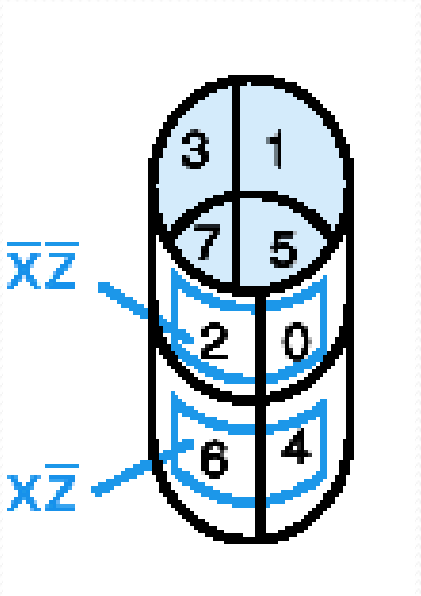


Minimization as SOP using K-map

- Enter 1s in the K-map for each product term in the function
- Group *adjacent* K-map cells containing 1s to obtain a product with fewer variables. Group size must be in power of 2 (2, 4, 8, …)
- Handle "boundary wrap" for K-maps of 3 or more variables.
- Realize that answer may not be unique

Three-Variable Map

x \ yz	00	01	11	10
0	0 m_0	1 m_1	3 m_3	2 m_2
1	4 m_4	5 m_5	7 m_7	6 m_6



-Note: variable ordering is (x,y,z); yz specifies column, x specifies row.

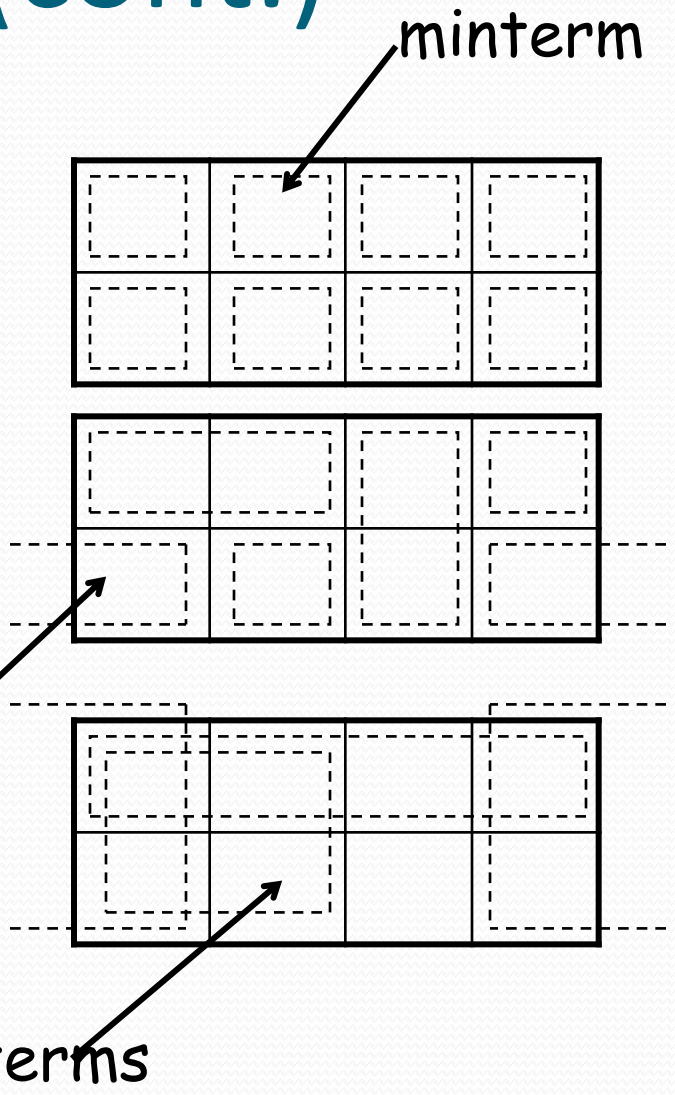
-Each cell is adjacent to ***three*** other cells (left or right or top or bottom or edge wrap)

Three-variable Map Simplification

- One square(uncombined) represents a minterm of 3 literals.
- A rectangle of 2 adjacent squares represents a product term of 2 literals.
- A rectangle of 4 squares represents a product term of 1 literal.
- A rectangle of 8 squares represents a functional value equal to 1.(F=1)

Three-Variable Map (cont.)

The types of structures that are either minterms or are generated by repeated application of the minimization theorem on a three variable map are shown at right.
Groups of 1, 2, 4, 8 are possible.

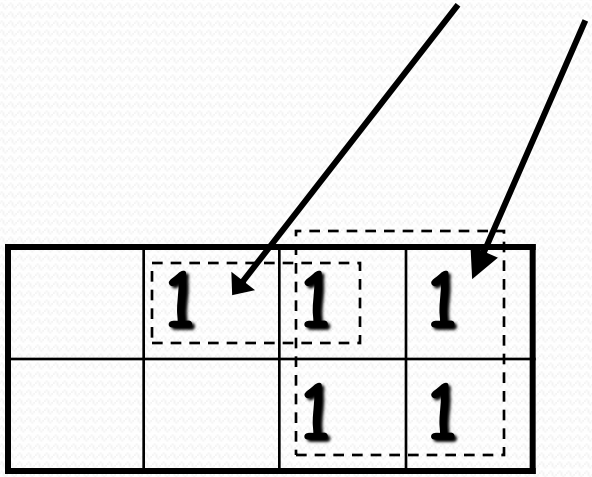


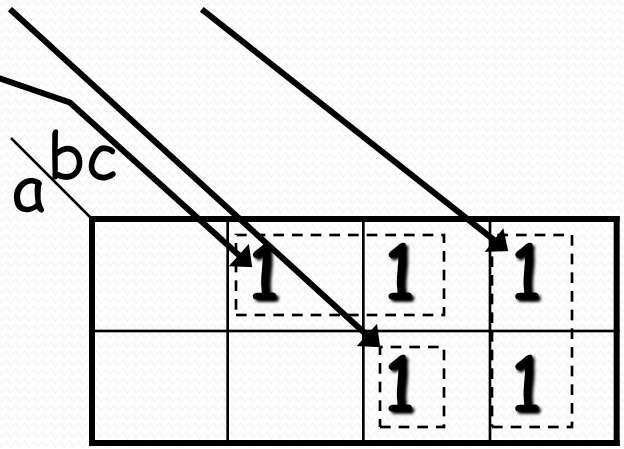
Three-Variable Map

- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m0, m1, m3, m2), (m4, m5, m7, m6), (m0, m1, m4, m5), (m1, m3, m5, m7), (m3, m2, m7, m6) and (m2, m0, m6, m4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m0, m1), (m1, m3), (m3, m2), (m2, m0), (m4, m5), (m5, m7), (m7, m6), (m6, m4), (m0, m4), (m1, m5), (m3, m7) and (m2, m6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

Simplification

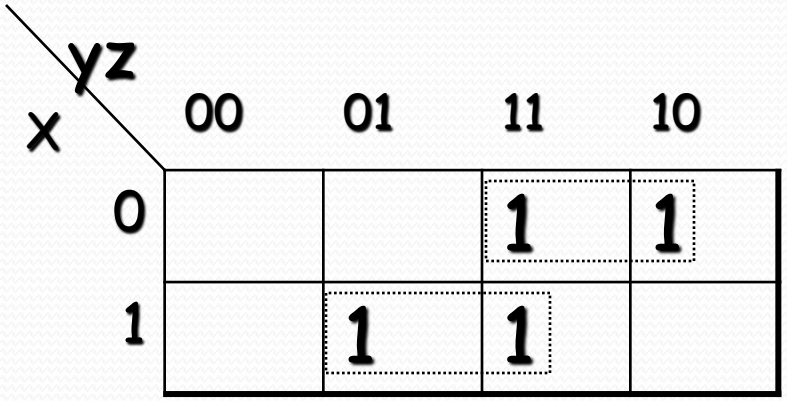
- Enter minterms of the Boolean function into the map, then group terms
- Example: f(a,b,c) = a'c + abc + bc'
- Result: f(a,b,c) = a'c+ b



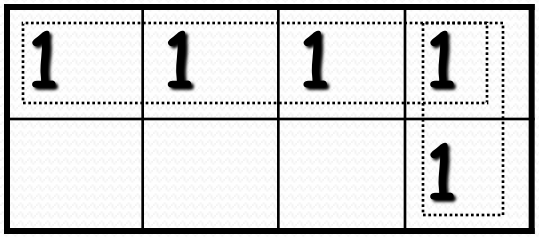


More Examples

- $f_1(x, y, z) = \sum m(2,3,5,7)$
  - **$f_1(x, y, z) = x'y + xz$**

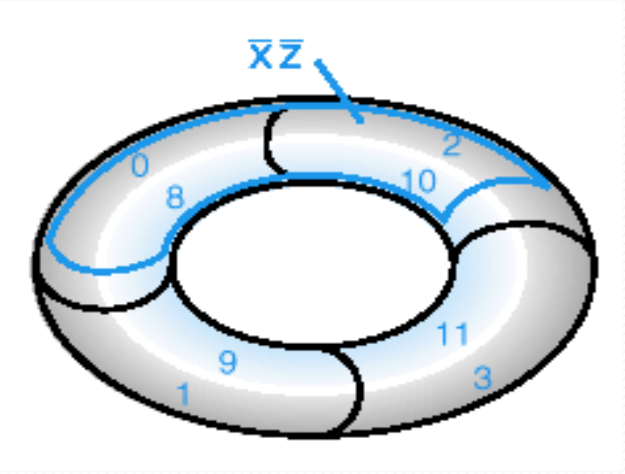


- $f_2(x, y, z) = \sum m(0,1,2,3,6)$
  - **$f_2(x, y, z) = x'+yz'$**



Four-Variable Maps

wx \ yz	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}



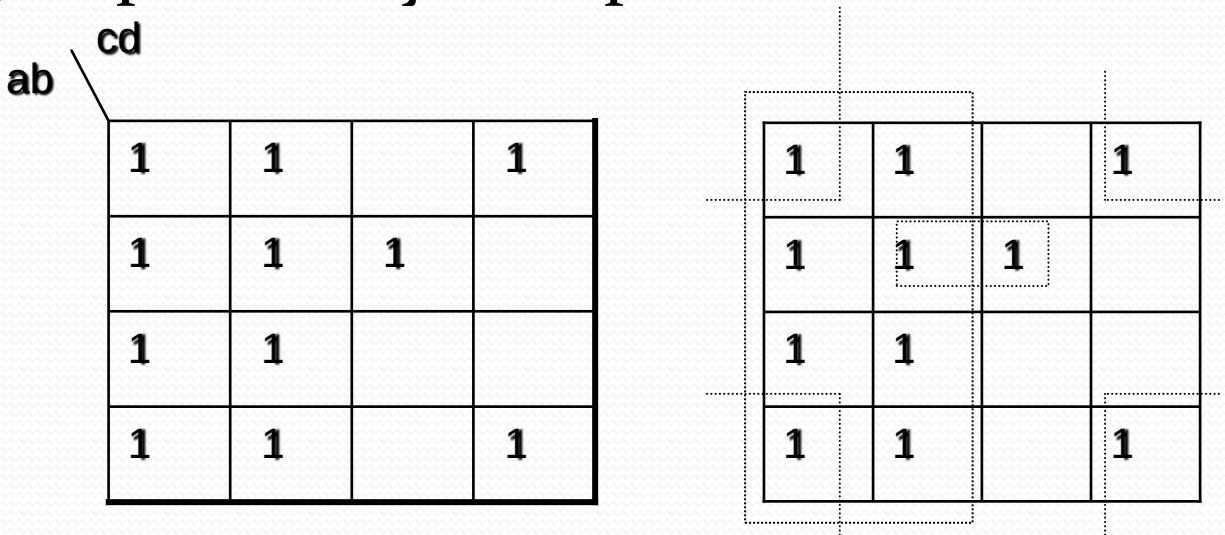
- Top cells are adjacent to bottom cells. Left-edge cells are adjacent to right-edge cells.
- Note variable ordering (WXYZ).

Four-variable Map Simplification

- One square represents a minterm of 4 literals.
- A rectangle of 2 adjacent squares represents a product term of 3 literals.
- A rectangle of 4 squares represents a product term of 2 literals.
- A rectangle of 8 squares represents a product term of 1 literal.
- A rectangle of 16 squares produces a function that is equal to logic 1.

Example

- Simplify the following Boolean function (A,B,C,D) = ∑m(0,1,2,4,5,7,8,9,10,12,13).
- First put the function g( ) into the map, and then group as many 1s as possible.



$$g(A,B,C,D) = c'+b'd'+a'bd$$