Introduction to Number Systems

UNIT I

Analog Vs Digital

- Two ways of representing numerical values
- Analog -> Numerical values as a continuous range between two expected °C extreme °C values
- Eg. Temperature of Oven can range from 0

to 100

- Digital -> Numerical range in discrete values

Number Systems

Four number system

- **Decimal (10)**
- **Binary (2)**
- **Octal (8)**
- **Hexadecimal (16)**
- **............**

Decimal numbers

$$1439 = 1 \times \underset{\underset{\text{Thousands}}{\uparrow}}{10^3} + 4 \times \underset{\underset{\text{Hundreds}}{\uparrow}}{10^2} + 3 \times \underset{\underset{\text{Tens}}{\uparrow}}{10^1} + 9 \times \underset{\underset{\text{Ones}}{\uparrow}}{10^0}$$

- Radix = 10

Binary numbers?

- Computers work only on two states
  - On
  - Off
- Basic memory elements hold only two states
  - Zero / One
- Thus a number system with two elements {0,1}
- A binary digit – bit !

Binary $\rightarrow$ Decimal

$1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

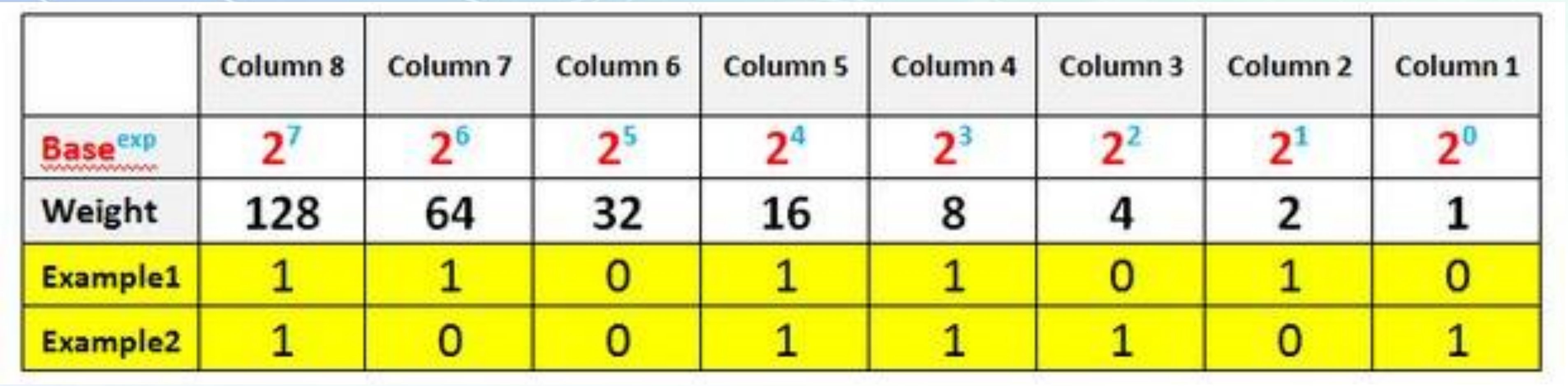
$= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$

$= 8 + 4 + 0 + 1$

$$\boxed{(1101)_2 = (13)_{10}}$$

1, 2, 4, 8, 16, 32, 64, 128, 256, 512, ….

Binary   Decimal



	Column 8	Column 7	Column 6	Column 5	Column 4	Column 3	Column 2	Column 1
Base^{exp}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Weight	128	64	32	16	8	4	2	1
Example1	1	1	0	1	1	0	1	0
Example2	1	0	0	1	1	1	0	1

Examples:

- $100101_2 = 37_{10}$

Exponents	2^5		2^4		2^3		2^2		2^1		2^0		
Place Values	32		16		8		4		2		1		
Bits	1		0		0		1		0		1		
Value	32					+	4			+	1	=	37

$10001110_2 = 142_{10}$:

Exponents	2^7		2^6		2^5		2^4		2^3		2^2		2^1		2^0		
Place Values	128		64		32		16		8		4		2		1		
Bits	1		0		0		0		1		1		1		0		
Value	128							+	8	+	4	+	2			=	142

$111101000_2 = 488_{10}$:

Exponents	2^8		2^7		2^6		2^5		2^4		2^3	2^2	2^1	2^0		
Place Values	256		128		64		32		16		8	4	2	1		
Bits	1		1		1		1		0		1	0	0	0		
Value	256	+	128	+	64	+	32			+	8				=	488

$10110101_2 = 181_{10}$:

Exponents	2^7		2^6		2^5		2^4		2^3		2^2		2^1		2^0		
Place Values	128		64		32		16		8		4		2		1		
Bits	1		0		1		1		0		1		0		1		
Value	128			+	32	+	16			+	4			+	1	=	181

Homework

- 1101_2
- 11011_2
- 101010_2
- 111100_2
- 10001101_2

Binary fraction to Decimal

Following are the steps of converting binary fractional to decimal.

- **A) Convert the integral part of binary to decimal equivalent**
  - Multiply each digit separately from left side of radix point till the first digit by 2^0, 2^1, 2^2,… respectively.
  - Add all the result coming from step 1.
  - Equivalent integral decimal number would be the result obtained in step 2.

- **B) Convert the fractional part of binary to decimal equivalent**
  - Divide each digit from right side of radix point till the end by 2^1, 2^2, 2^3, … respectively.
  - Add all the result coming from step 1.
  - Equivalent fractional decimal number would be the result obtained in step 2.
- **C) Add both integral and fractional part of decimal number.**

Binary fraction to Decimal

Let's take an example for n = 110.101

- **Step 1: Conversion of 110 to decimal** =>
  - $110_2 = (1*2^2) + (1*2^1) + (0*2^0)$
  - $=> 110_2 = 4 + 2 + 0$
  - $=> 110_2 = 6$
  - *So equivalent decimal of binary integral is 6.*
- **Step 2: Conversion of .0101 to decimal =>**
  - $0.101_2 = (1*1/2) + (0*1/2^2) + (1*1/2^3) => 0.101_2$
  - $= 1*0.5 + 0*0.25 + 1*0.125 => 0.101_2$
  - = 0.625
  - *So equivalent decimal of binary fractional is 0.625*
- **Step 3: Add result of step 1 and 2.**
  - => 6 + 0.625 = 6.625

Homework

- 11.101_2
- 110.011_2
- 101.010_2
- 1101.1100_2
- 1010.1101_2

Decimal $\rightarrow$ Binary

2	13	1	LSB
2	6	0	
2	3	1	
2	1	1	MSB
	0		

$$\boxed{(13)_{10} = (1101)_2}$$

Examples

- $(77)_{10}$
- $(120)_{10}$
- $(38)_{10}$
- $(95)_{10}$
- $(159)_{10}$

Decimal Fraction → Binary

1101.0111 = $(1\times2^{3}) + (1\times2^{2}) + (0\times2^{1}) +$
$(1\times2^{0}) + (0\times2^{-1}) + (1\times2^{-2}) +$
$(1\times2^{-3}) + (1\times2^{-4})$

= 8 + 4 + 0 + 1 + 0 + 1/4 + 1/8 + 1/16
= 8 + 4 + 0 + 1 + 0 + 0.25 + 0.125 + 0.0625 = 13.4375_{10}

Examples

- $(12.567)_{10}$
- $(28.745)_{10}$
- $(72.124)_{10}$
- $(30.897)_{10}$
- $(167.4592)_{10}$

Octal $\rightarrow$ Decimal

$$
\begin{aligned}
137 &= 1 \times 8^2 + 3 \times 8^1 + 7 \times 8^0 \\
&= 1 \times 64 + 3 \times 8 + 7 \times 1 \\
&= 64 + 24 + 7
\end{aligned}
$$

$$\boxed{(137)_8 = (95)_{10}}$$

- Digits used in Octal number system – 0 to 7

Examples

- $(267)_8$
- $(45)_8$
- $(12)_8$
- $(462)_8$
- $(125)_8$

octal fraction to decimal

Convert (2 1 . 2 1)$_8$= (?)$_{10}$

= 2 x 8^1 + 1 x 8^0 . 2 x 8^{-1} + 1 x 8^{-2}

= 2 x 8 + 1 x 1 . 2 x (1 / 8) + 1 x (1 / 64)

= 16 + 1 . (0. 2 5) + (0 . 0 1 5 6 2 5)

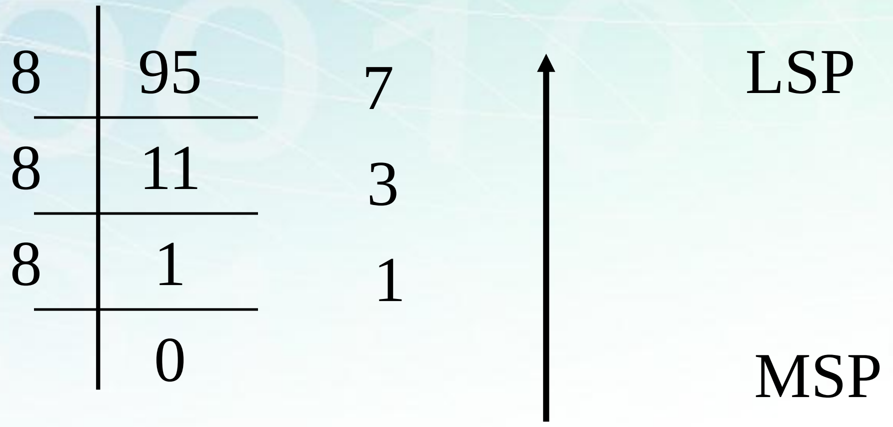
= 17 + 0. 265625

= 17 . 265625

Examples

- $(16.432)_8$
- $(213.345)_8$
- $(43.1236)_8$
- $(71.2013)_8$
- $(627.2361)_8$

Decimal $\rightarrow$ Octal

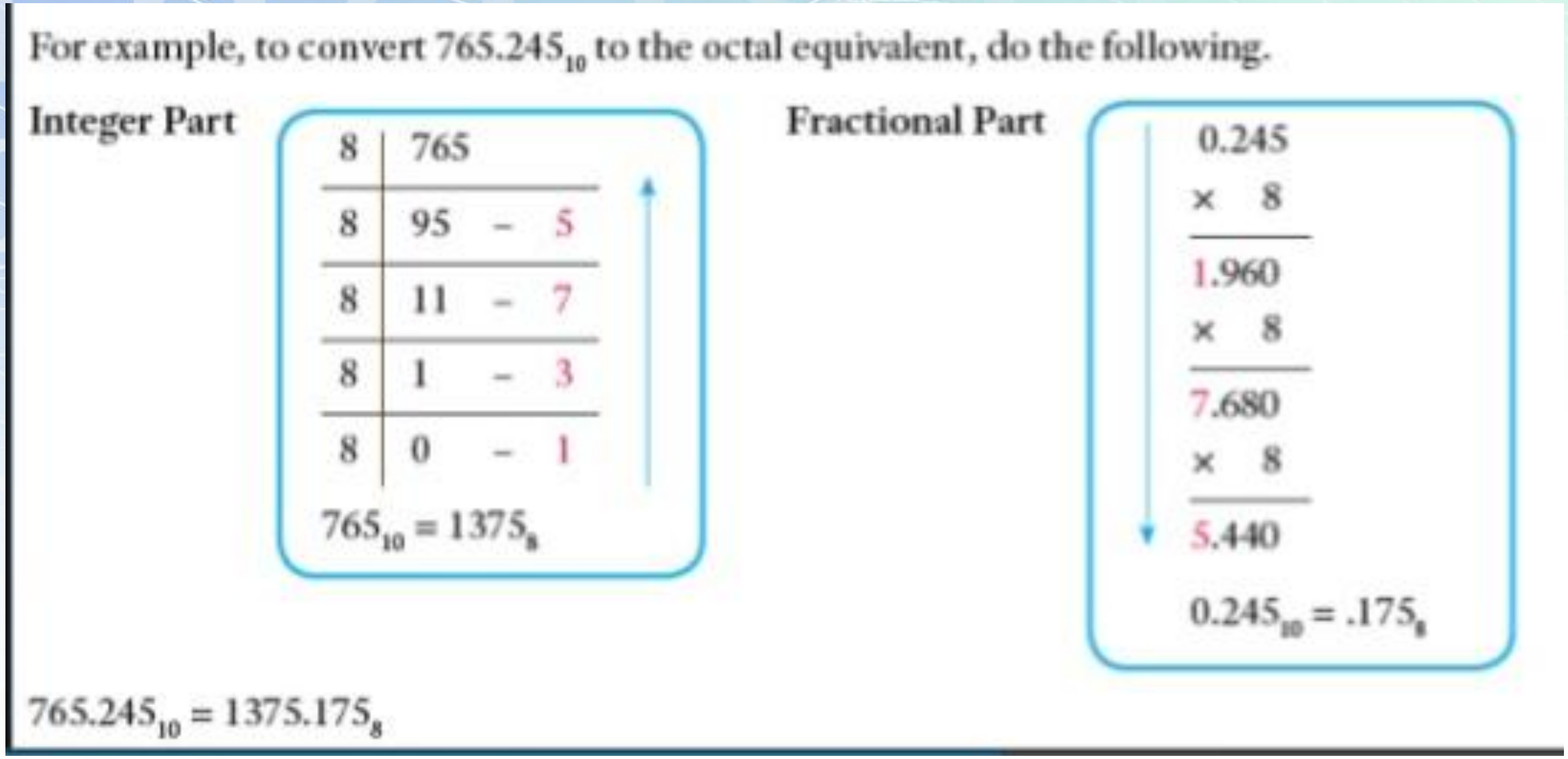


$$\boxed{(95)_{10} = (137)_8}$$

Examples

- $(149)_{10}$
- $(951)_{10}$
- $(637)_{10}$
- $(8731)_{10}$
- $(7345)_{10}$

Decimal Fraction → Octal



For example, to convert 765.245_{10} to the octal equivalent, do the following.

Integer Part

```
8 | 765
8 | 95  – 5
8 | 11  – 7
8 | 1   – 3
8 | 0   – 1
```

$765_{10} = 1375_8$

Fractional Part

```
  0.245
×    8
  1.960
×    8
  7.680
×    8
  5.440
```

$0.245_{10} = .175_8$

$765.245_{10} = 1375.175_8$

Examples

- $(25.987)_{10}$
- $(38.562)_{10}$
- $(19.281)_{10}$
- $(114.5921)_{10}$
- $(163.9817)_{10}$

Hex $\rightarrow$ Decimal

$$\begin{aligned} BAD &= 11 \times 16^2 + 10 \times 16^1 + 13 \times 16^0 \\ &= 11 \times 256 + 10 \times 16 + 13 \times 1 \\ &= 2816 + 160 + 13 \end{aligned}$$

$$\boxed{(BAD)_{16} = (2989)_{10}}$$

A = 10, B = 11, C = 12, D = 13, E = 14, F = 15

Examples

- $(A59)_{16}$
- $(901D)_{16}$
- $(ACF)_{16}$
- $(871E)_{16}$
- $(26AD)_{16}$

Decimal $\rightarrow$ Hex

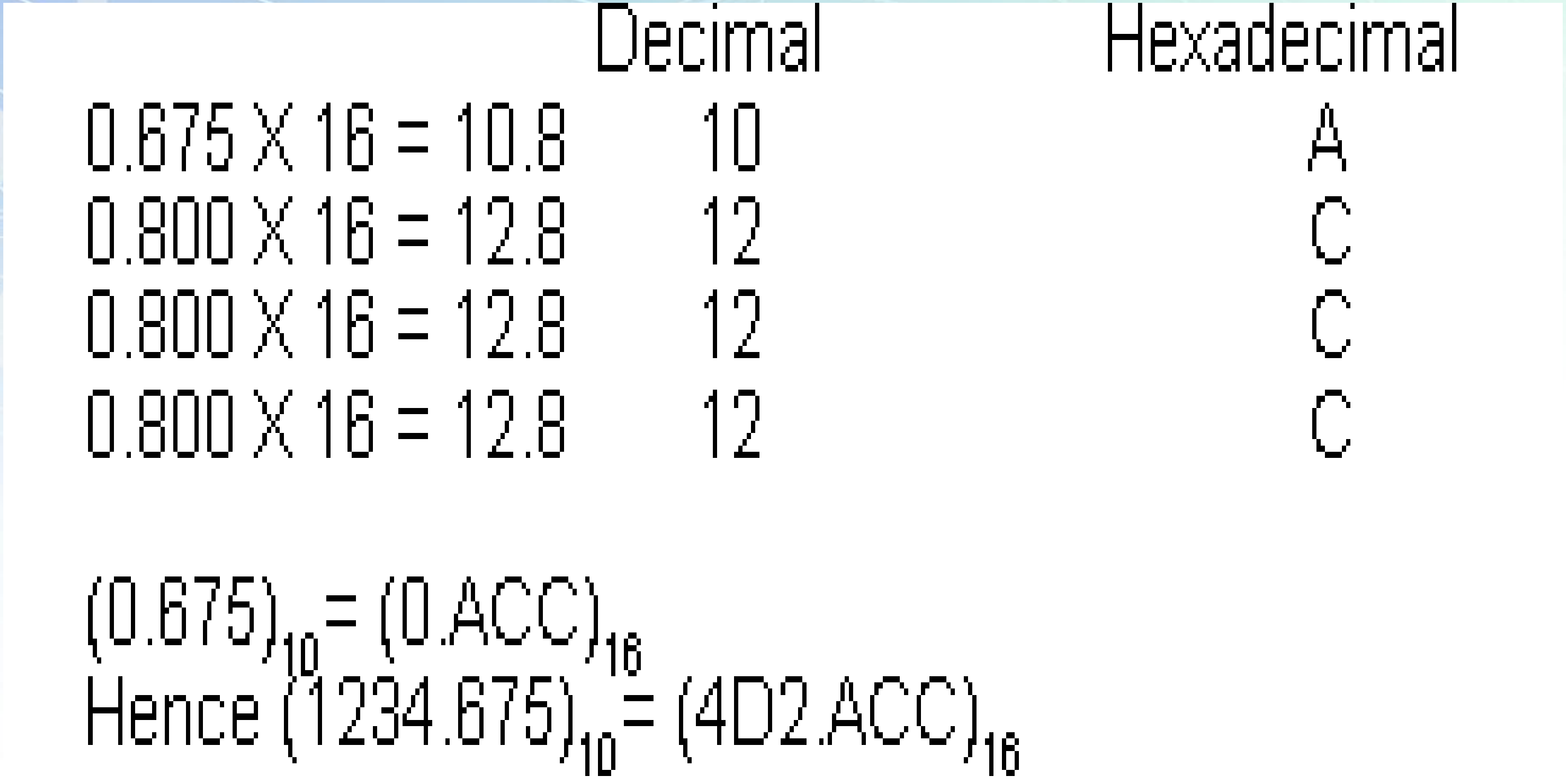
16	2989	13	LSP
16	186	10	$\uparrow$
16	11	11	
	0		MSP

$$\boxed{(2989)_{10} = (BAD)_{16}}$$

Examples

- $(149)_{10}$
- $(951)_{10}$
- $(637)_{10}$
- $(8731)_{10}$
- $(7345)_{10}$

Decimal fraction → Hex



	Decimal	Hexadecimal
0.675 X 16 = 10.8	10	A
0.800 X 16 = 12.8	12	C
0.800 X 16 = 12.8	12	C
0.800 X 16 = 12.8	12	C

$(0.675)_{10} = (0.ACC)_{16}$

Hence $(1234.675)_{10} = (4D2.ACC)_{16}$

Examples

- $(134.567)_{10}$
- $(68.912)_{10}$
- $(93.281)_{10}$
- $(169.583)_{10}$
- $(218.3189)_{10}$

Why octal or hex?

- Ease of use and conversion
- Three bits make one octal digit

  111 010 110 101

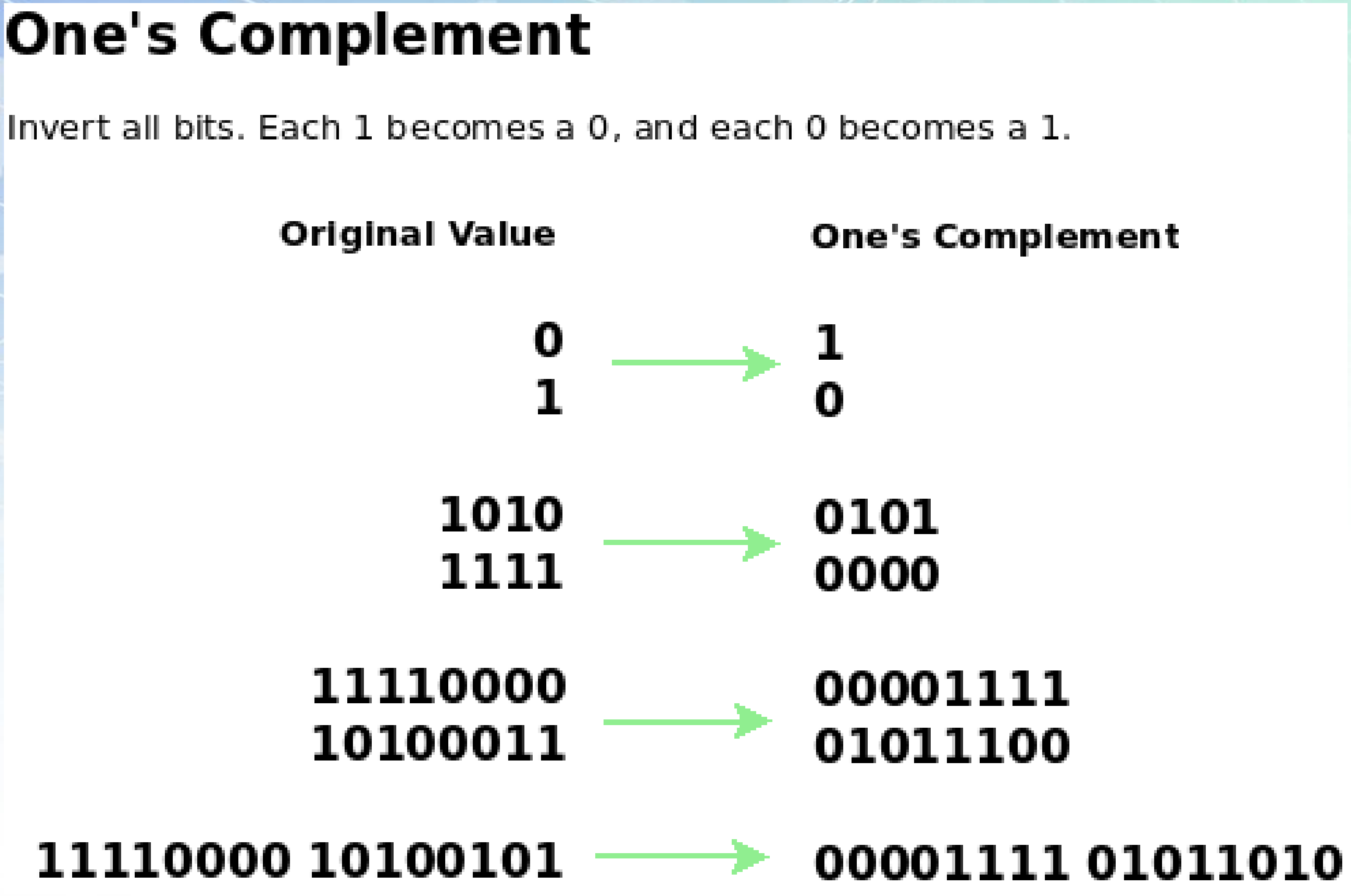
  7 2 6 5 => 7265 in octal

- Four bits make one hexadecimal digit

  1110 1011 0101

  4 bits = nibble

  E B $\bigcirc$5 $\longrightarrow$ => EB5 in hex



One's Complement

Invert all bits. Each 1 becomes a 0, and each 0 becomes a 1.

Original Value		One's Complement
0	$\rightarrow$	1
1		0
1010	$\rightarrow$	0101
1111		0000
11110000	$\rightarrow$	00001111
10100011		01011100
11110000 10100101	$\rightarrow$	00001111 01011010

Examples

- 11101111
- 10101100
- 11111011
- 100010001
- 110001100

Two's Complement

First, find the one's complement of a value, and then add 1 to it.

Original Value	11000010	10000000	10101011
One's Complement	00111101	01111111	01010100
Add 1	00111101 + 1	01111111 + 1	01010100 + 1
	00111110	10000000*	01010101

Two's Complements

* This is not an error. This is a contrived problem to show that it is possible for a two's complement to match the original value.

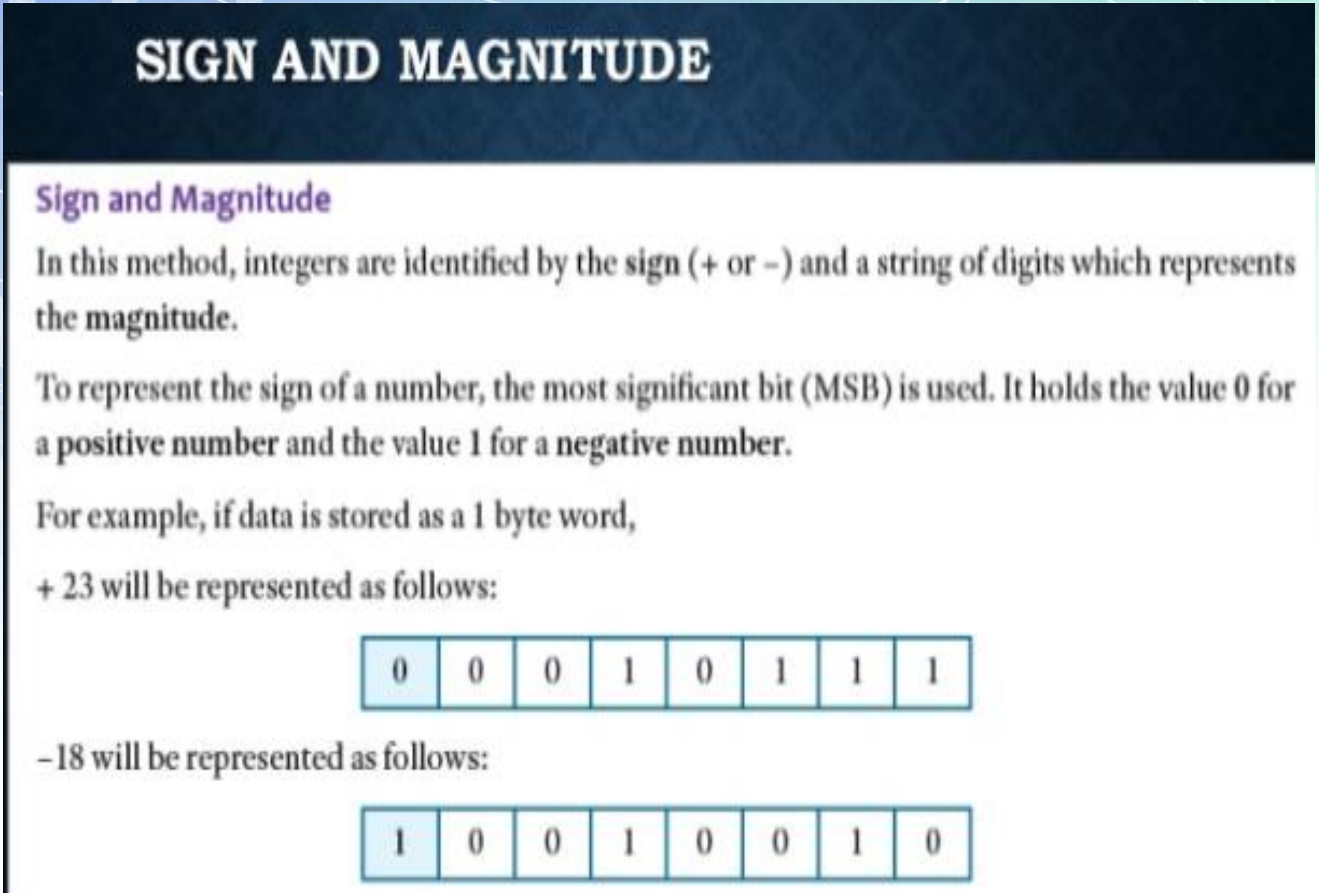
Examples

- 11101111
- 10101100
- 11111011
- 100010001
- 110001100

Negative numbers

Three representations

- Signed magnitude
- 1’s complement
- 2’s complement



SIGN AND MAGNITUDE

Sign and Magnitude

In this method, integers are identified by the **sign** (+ or –) and a string of digits which represents the **magnitude**.

To represent the sign of a number, the most significant bit (MSB) is used. It holds the value 0 for a **positive number** and the value 1 for a **negative number**.

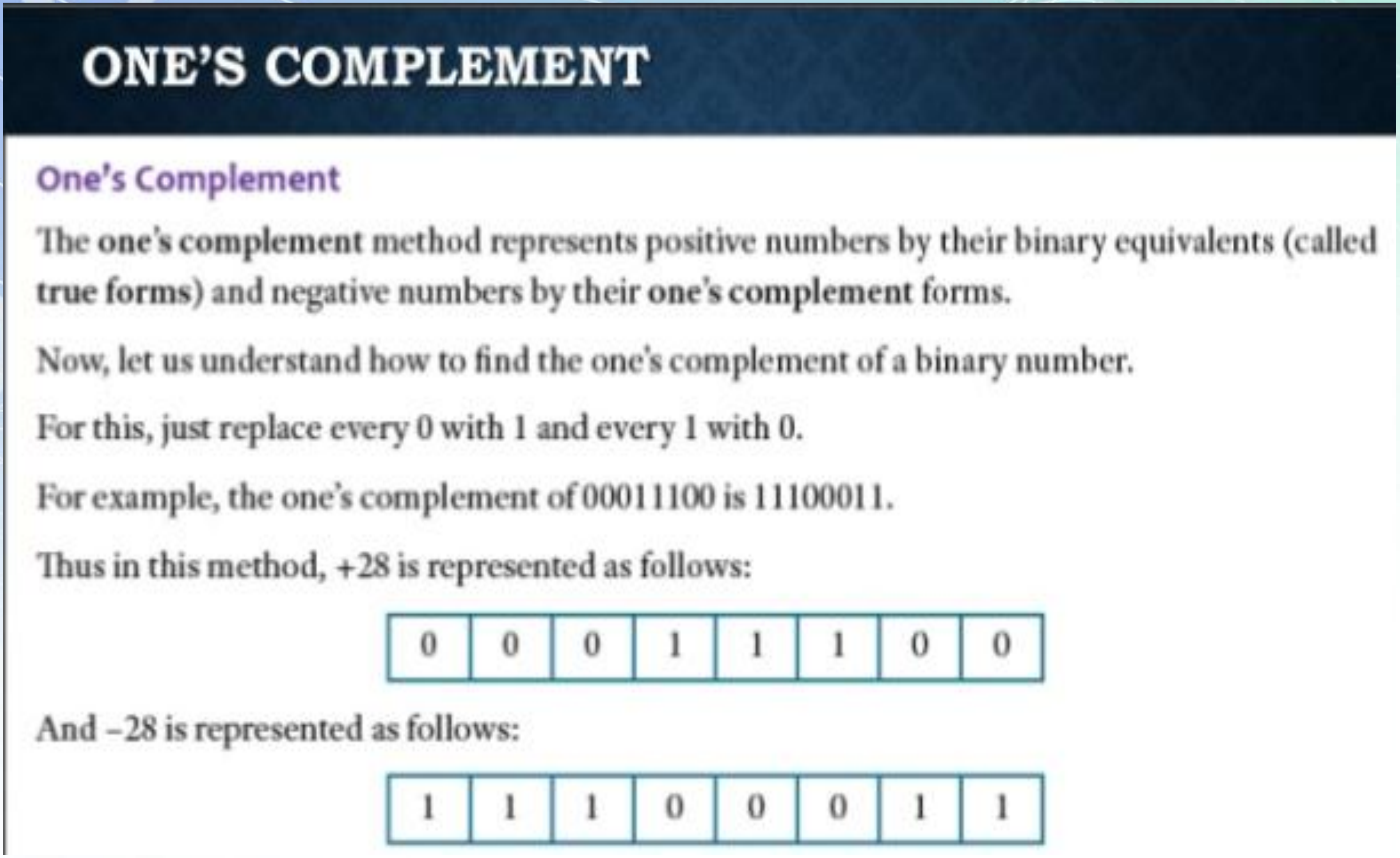
For example, if data is stored as a 1 byte word,

+ 23 will be represented as follows:

0	0	0	1	0	1	1	1

–18 will be represented as follows:

1	0	0	1	0	0	1	0



ONE'S COMPLEMENT

One's Complement

The **one's complement** method represents positive numbers by their binary equivalents (called **true forms**) and negative numbers by their **one's complement** forms.

Now, let us understand how to find the one's complement of a binary number.

For this, just replace every 0 with 1 and every 1 with 0.

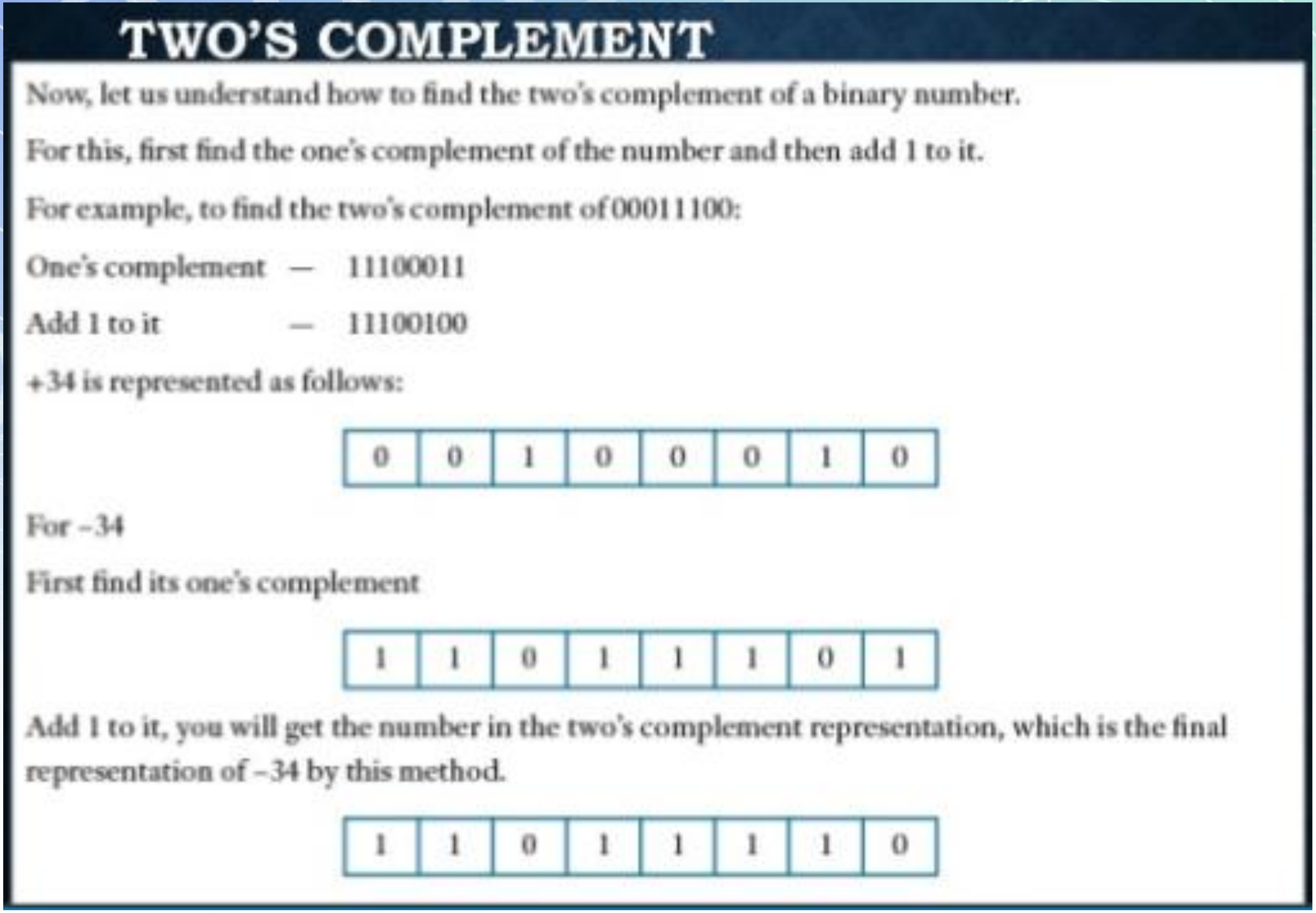
For example, the one's complement of 00011100 is 11100011.

Thus in this method, +28 is represented as follows:

0	0	0	1	1	1	0	0

And –28 is represented as follows:

1	1	1	0	0	0	1	1



TWO'S COMPLEMENT

Now, let us understand how to find the two's complement of a binary number.

For this, first find the one's complement of the number and then add 1 to it.

For example, to find the two's complement of 00011100:

One's complement	–	11100011
Add 1 to it	–	11100100

+34 is represented as follows:

0	0	1	0	0	0	1	0

For –34

First find its one's complement

1	1	0	1	1	1	0	1

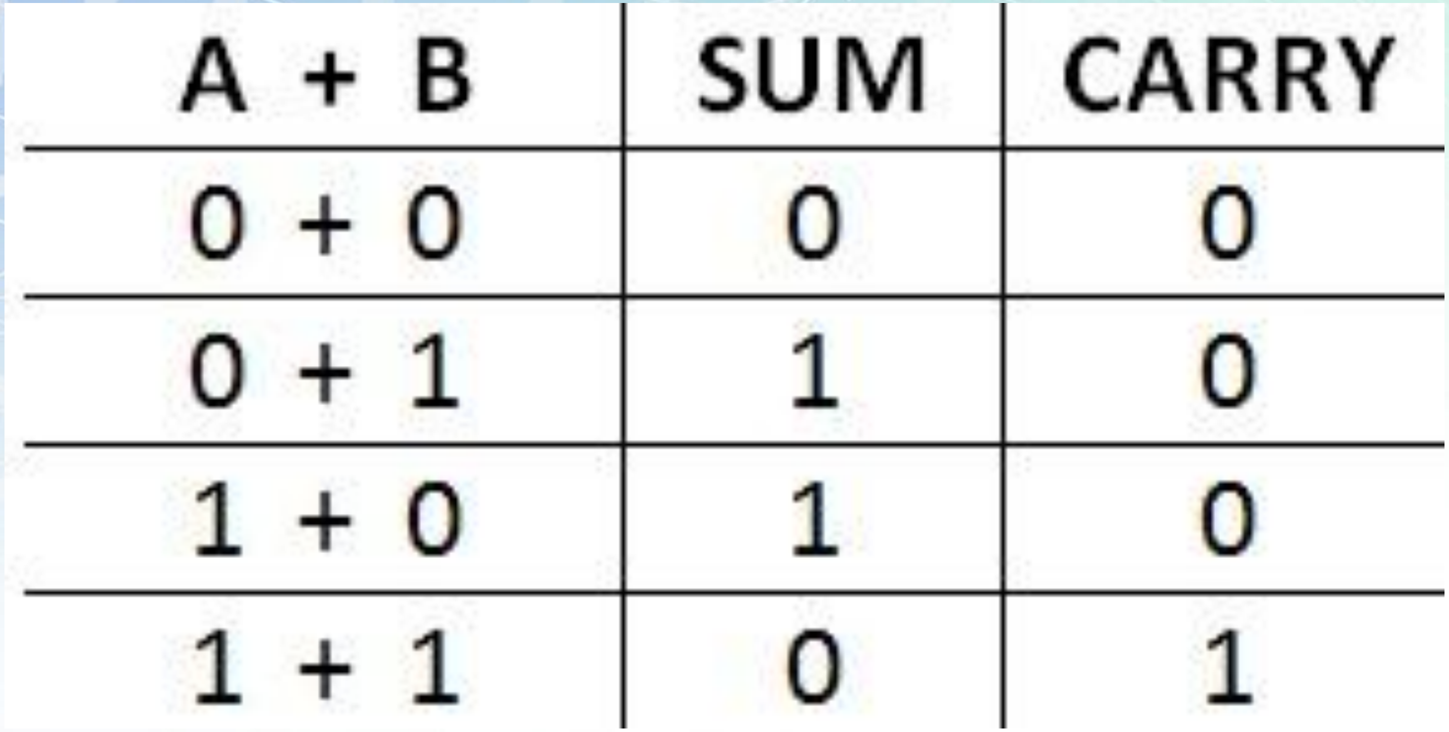
Add 1 to it, you will get the number in the two's complement representation, which is the final representation of –34 by this method.

1	1	0	1	1	1	1	0

Binary Arithmetic

- Addition / subtraction
- Unsigned
- Signed
  - Using negative numbers

Binary Addition



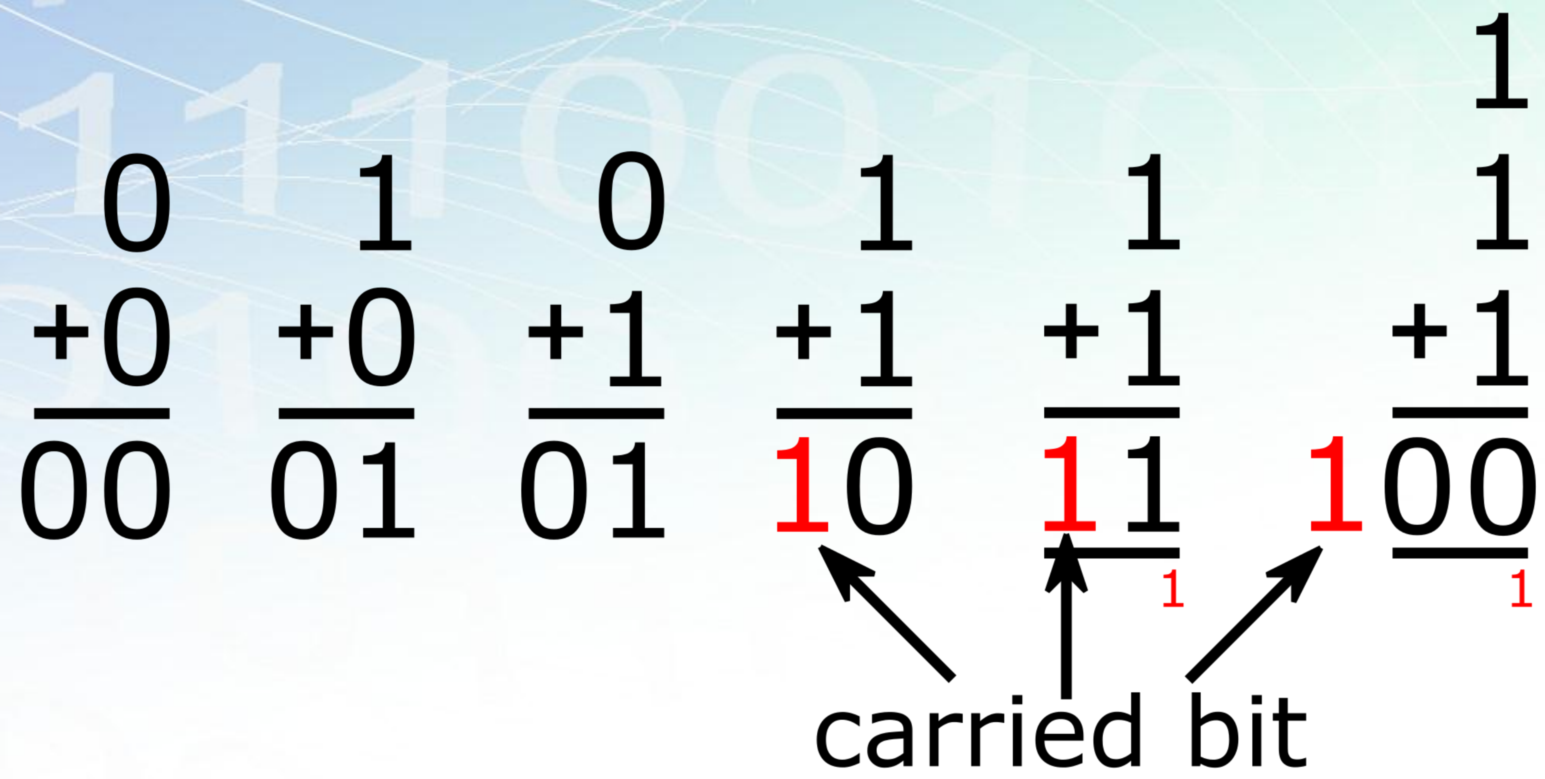
A + B	SUM	CARRY
0 + 0	0	0
0 + 1	1	0
1 + 0	1	0
1 + 1	0	1

Binary Addition

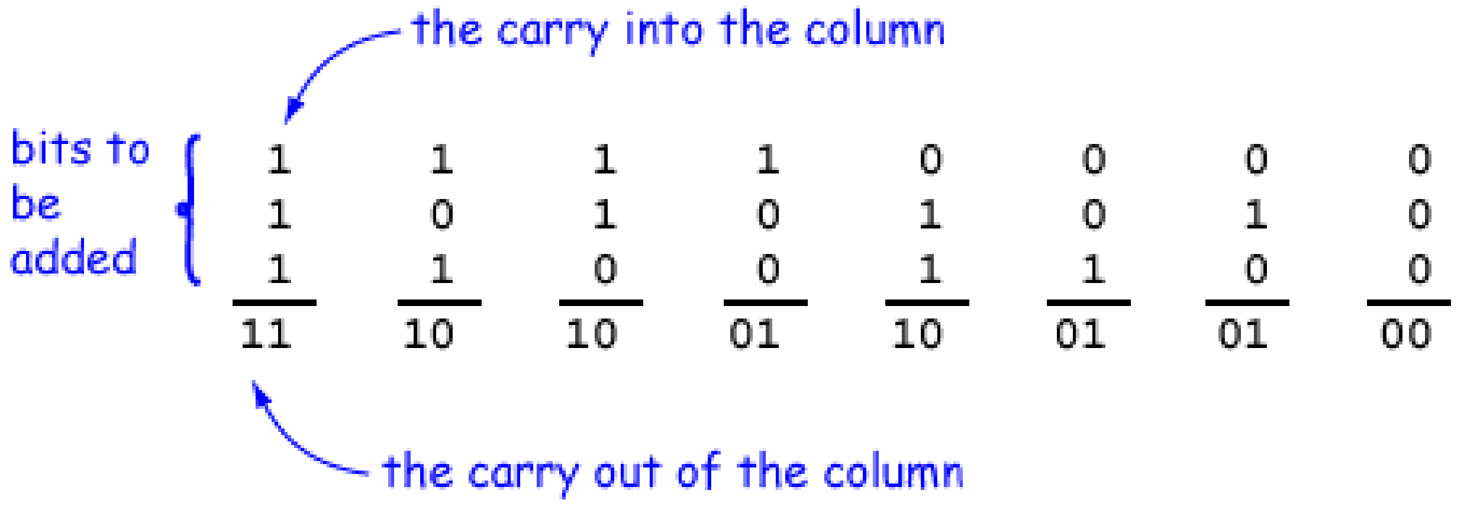
$$\begin{array}{r} 0 \\ +0 \\ \hline 00 \end{array} \qquad \begin{array}{r} 1 \\ +0 \\ \hline 01 \end{array} \qquad \begin{array}{r} 0 \\ +1 \\ \hline 01 \end{array} \qquad \begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$$

carried bit

Binary Addition



Rules for Binary Addition

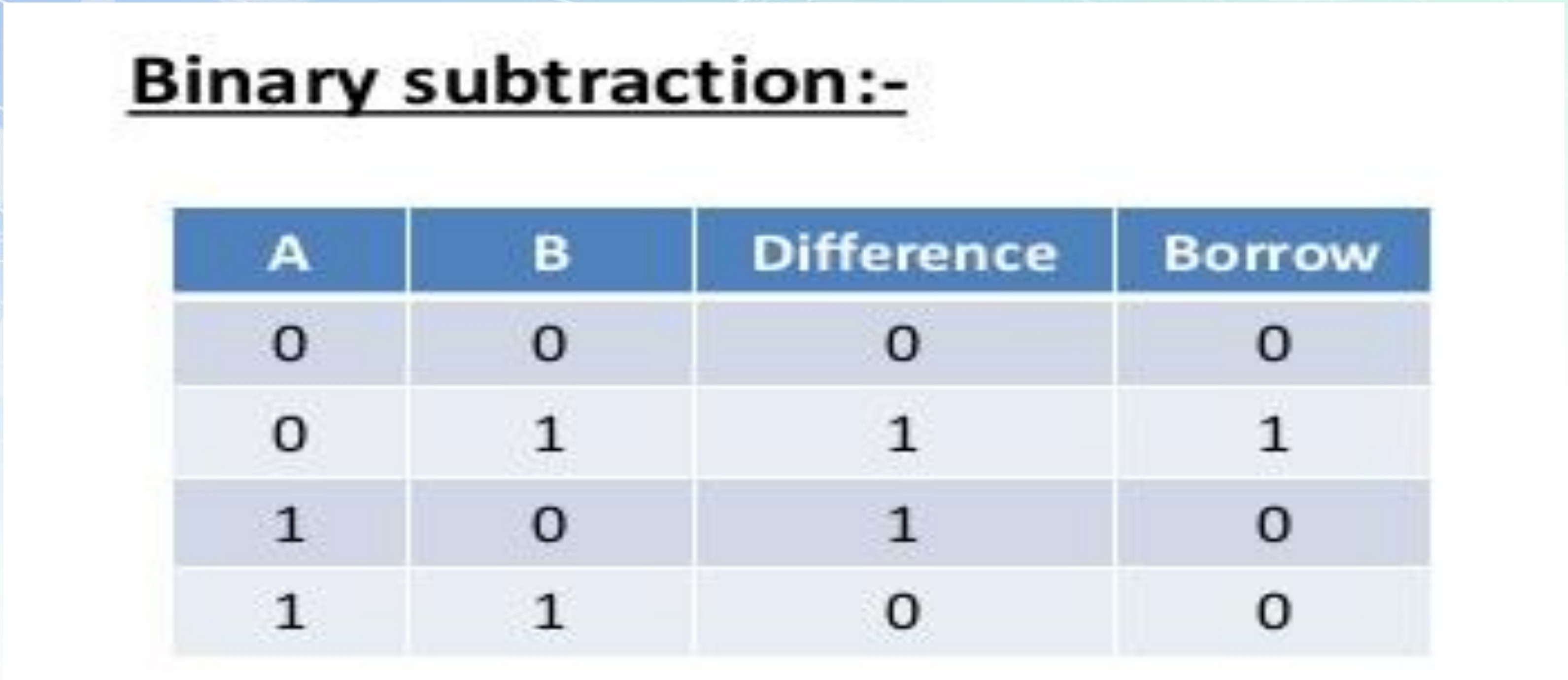


1	1	1	1	0	0	0	0
1	0	1	0	1	0	1	0
1	1	0	0	1	1	0	0
11	10	10	01	10	01	01	00

Examples

- 11101111+10001011
- 10101100+11011101
- 11111011+10101010
- 100010001+111111010
- 110001100+100001001

Binary subtraction



Binary subtraction:-

A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Multiplication

Binary, two 1-bit values

A	B	A × B
0	0	0
0	1	0
1	0	0
1	1	1

Multiplication

- Binary, two *n*-bit values
  - As with decimal values
  - E.g.,

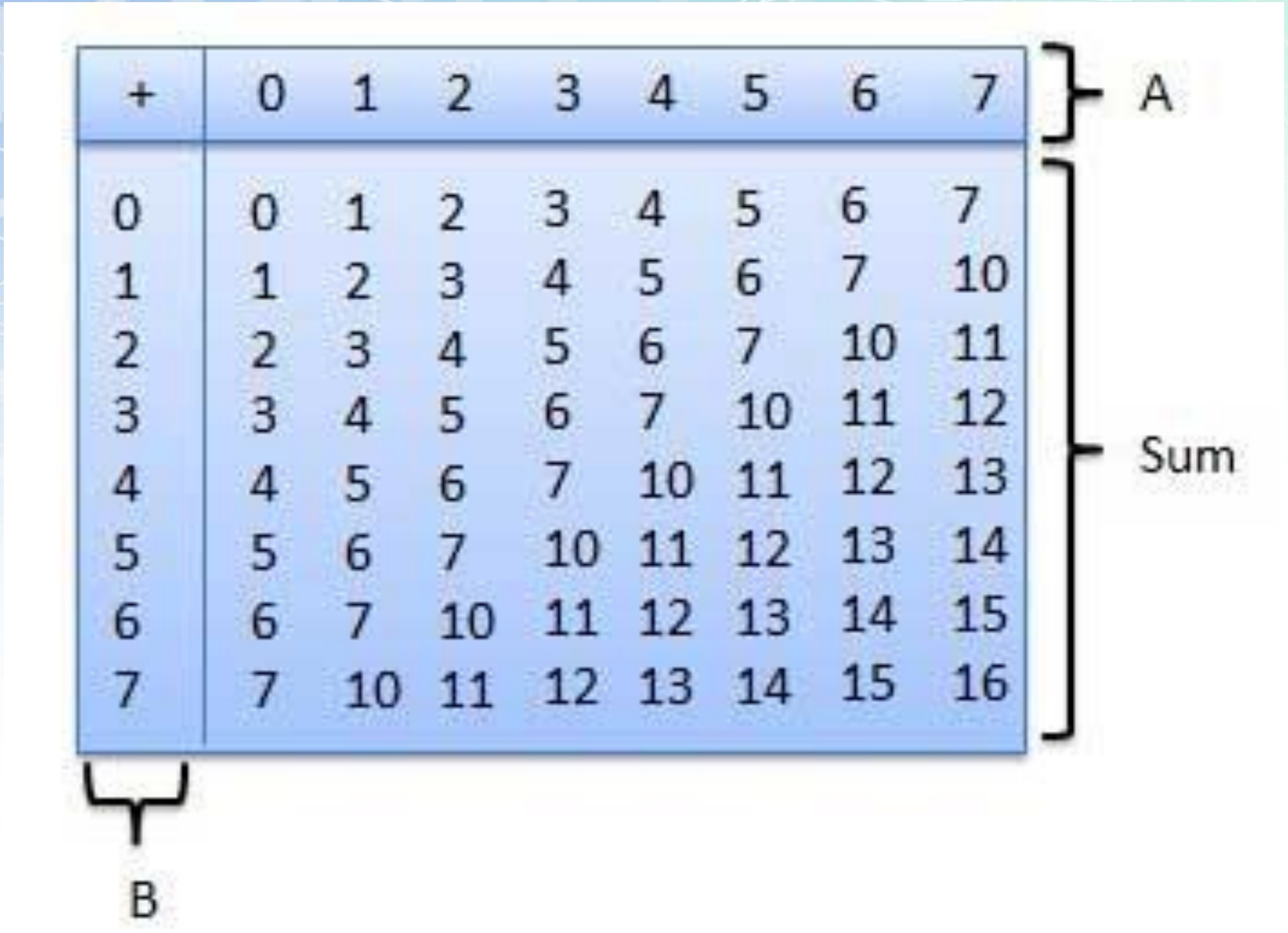
```
         1110
       x 1011
    _________
         1110
        1110
       0000
      1110
    _________
     10011010
```

Binary Division

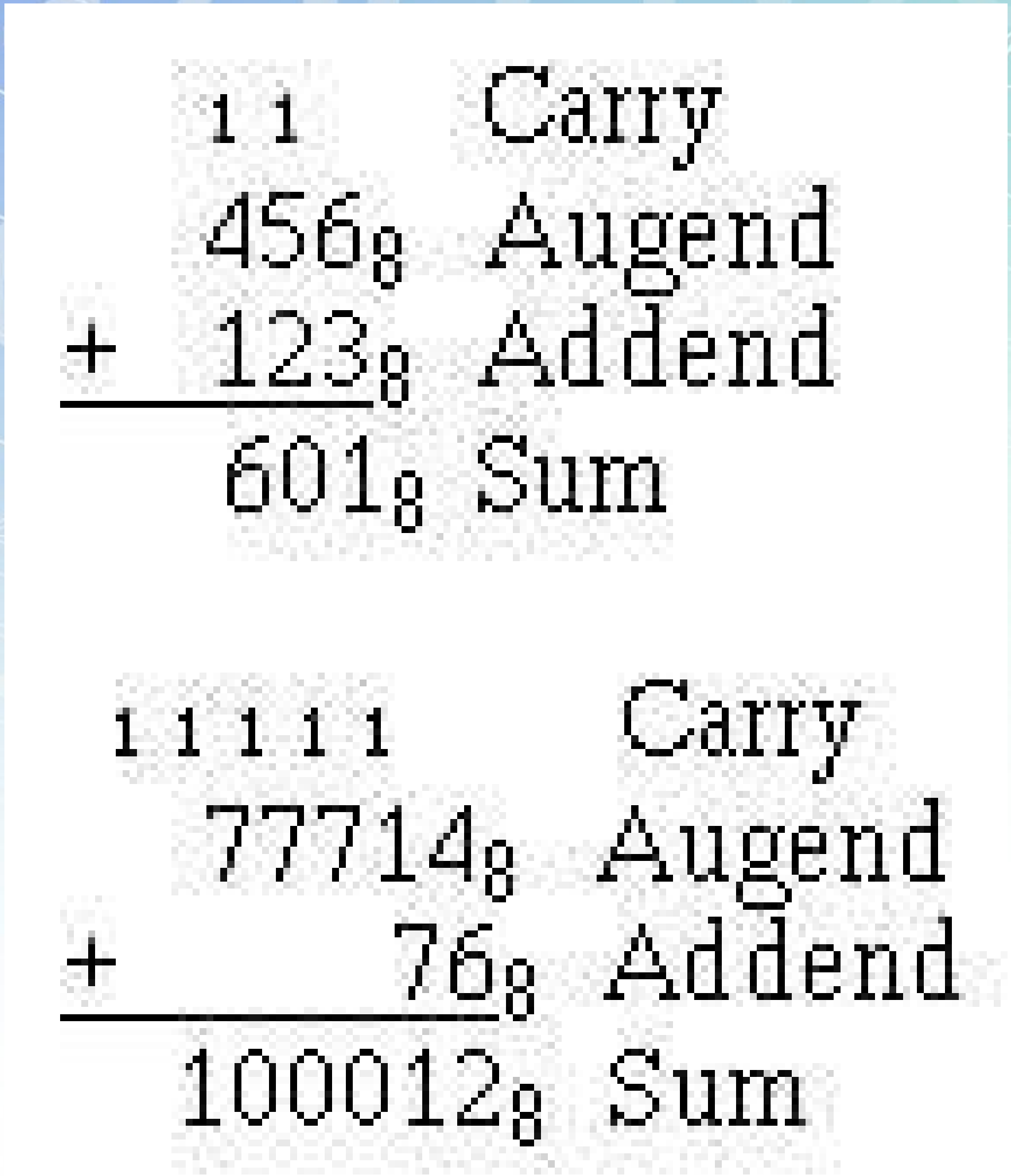
- Division in binary follows the same procedure as division in decimal.

$$\begin{array}{r} 10 \\ 11\overline{)110} \\ \underline{11} \\ 000 \end{array} \qquad \begin{array}{r} 2 \\ 3\overline{)6} \\ \underline{6} \\ 0 \end{array} \qquad \begin{array}{r} 11 \\ 10\overline{)110} \\ \underline{10} \\ 10 \\ \underline{10} \\ 00 \end{array} \qquad \begin{array}{r} 3 \\ 2\overline{)6} \\ \underline{6} \\ 0 \end{array}$$

Arithmetic in octal number system



+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16



$$\begin{array}{rrl}
 & 1\,1 & \text{Carry} \\
 & 456_8 & \text{Augend} \\
+ & 123_8 & \text{Addend} \\
\hline
 & 601_8 & \text{Sum}
\end{array}$$

$$\begin{array}{rrl}
 & 1\,1\,1\,1\,1 & \text{Carry} \\
 & 77714_8 & \text{Augend} \\
+ & 76_8 & \text{Addend} \\
\hline
 & 100012_8 & \text{Sum}
\end{array}$$

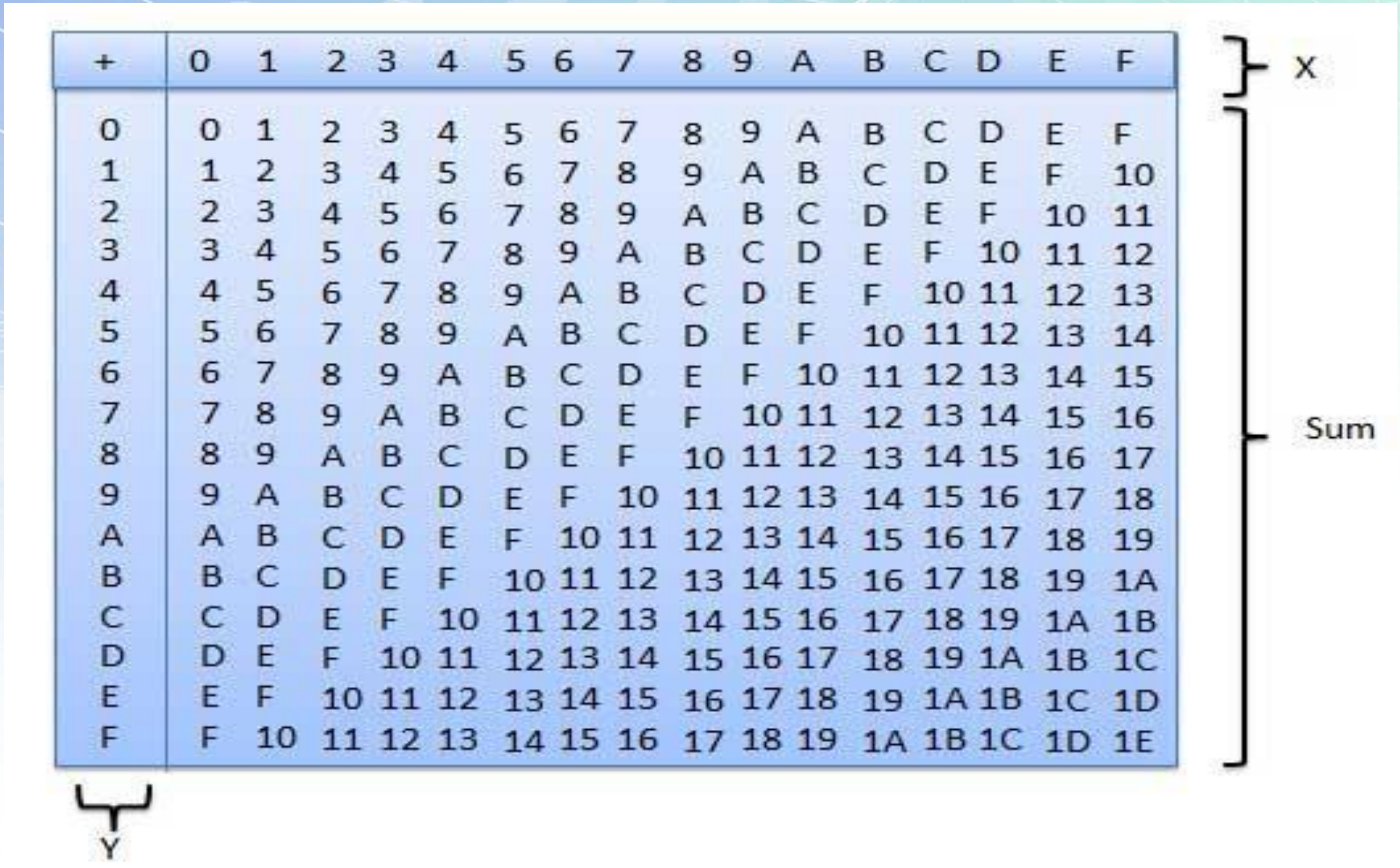
Examples

- $(162)_8+(537)_8$
- $(463)_8+(115)_8$
- $(764)_8+(231)_8$
- $(541)_8+(117)_8$
- $(751)_8+(342)_8$

Examples

- $(456)_8 - (173)_8$
- $(463)_8 - (115)_8$
- $(764)_8 - (231)_8$
- $(541)_8 - (117)_8$
- $(751)_8 - (342)_8$

Arithmetic in Hex number system



+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Example – Addition

$4A6_{16} + 1B3_{16} = 659_{16}$

$$\begin{array}{rrl} 1 & & \text{carry} \\ & 4\,A\,6 & = 1190_{10} \\ + & 1\,B\,3 & = 435_{10} \\ \hline & 6\,5\,9 & = 1625_{10} \end{array}$$

Examples

- $(ACB)_{16}+(678)_{16}$
- $(459)_{16}+(7EF)_{16}$
- $(921)_{16}+(473)_{16}$
- $(762)_{16}+(BCD)_{16}$
- $(958)_{16}+(DEF)_{16}$

$4A6_{16} - 1B3_{16} = 2F3_{16}$

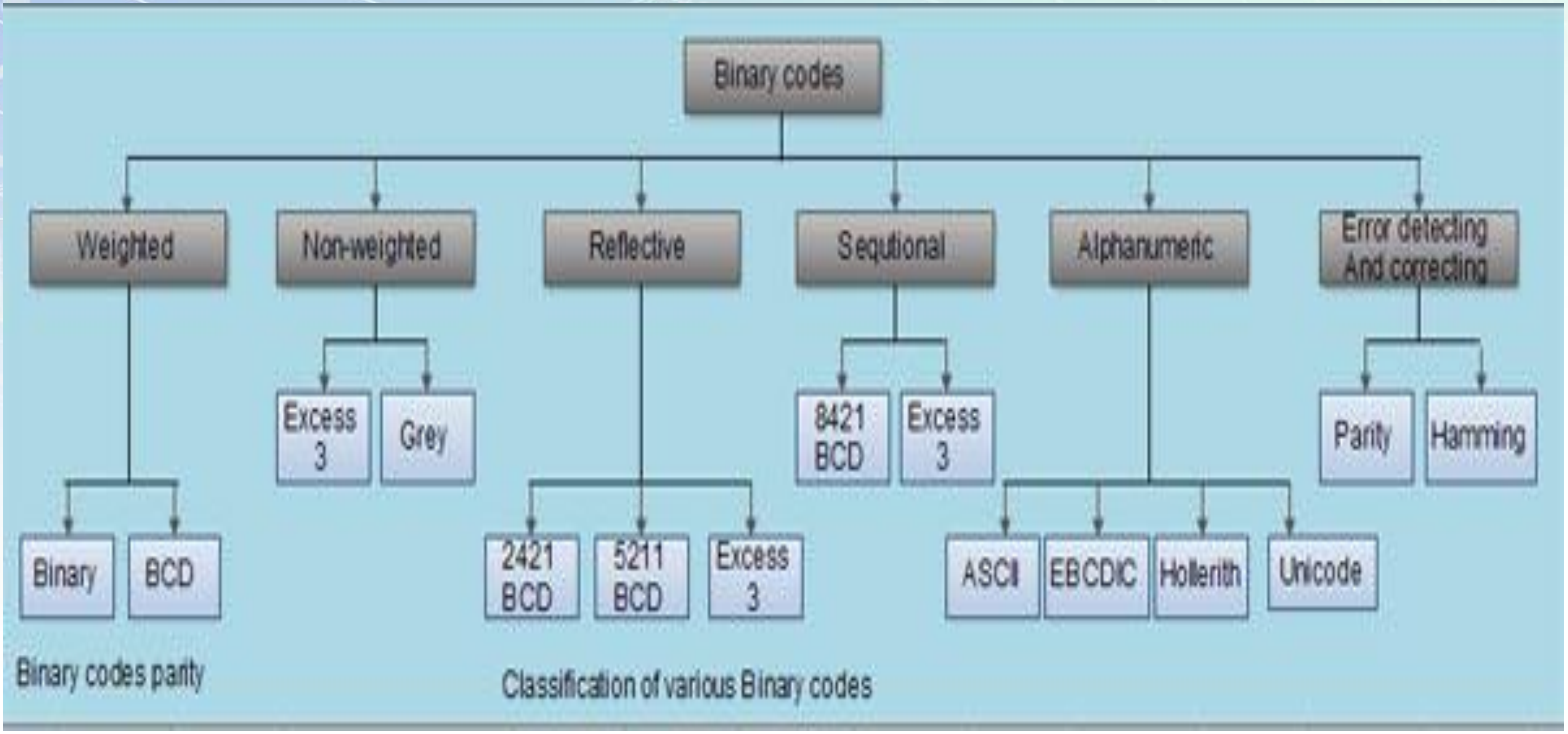
16 borrow

$$
\begin{array}{rl}
{}^{3}4\,A\,6 & = 1190_{10} \\
-\,1\,B\,3 & = 435_{10} \\
\hline
2\,F\,3 & = 755_{10}
\end{array}
$$

Examples

- $(ACB)_{16} - (678)_{16}$
- $(7EF)_{16} - (459)_{16}$
- $(921)_{16} - (473)_{16}$
- $(BCD)_{16} - (726)_{16}$
- $(DEF)_{16} - (958)_{16}$

Binary codes



Code Conversion

Binary to BCD

Step 1: Convert binary number to decimal.
Step 2: Convert decimal number to BCD.

BINARY TO BCD CONVERSION

Step1: Binary ---> Decimal

Step2: Decimal ---> BCD

Example : $(10011)_2$ ----> $(?)_{BCD}$

$$2^4\ 2^3\ 2^2\ 2^1\ 2^0$$

$$1\ 0\ 0\ 1\ 1$$

$$1 \times 2^0 = 1$$
$$1 \times 2^1 = 2$$
$$1 \times 2^4 = 16$$
$$\overline{\ \ 19\ \ }$$

$$(10011)_2 \rightarrow (19)_{10}$$

$$(19)_{10}$$

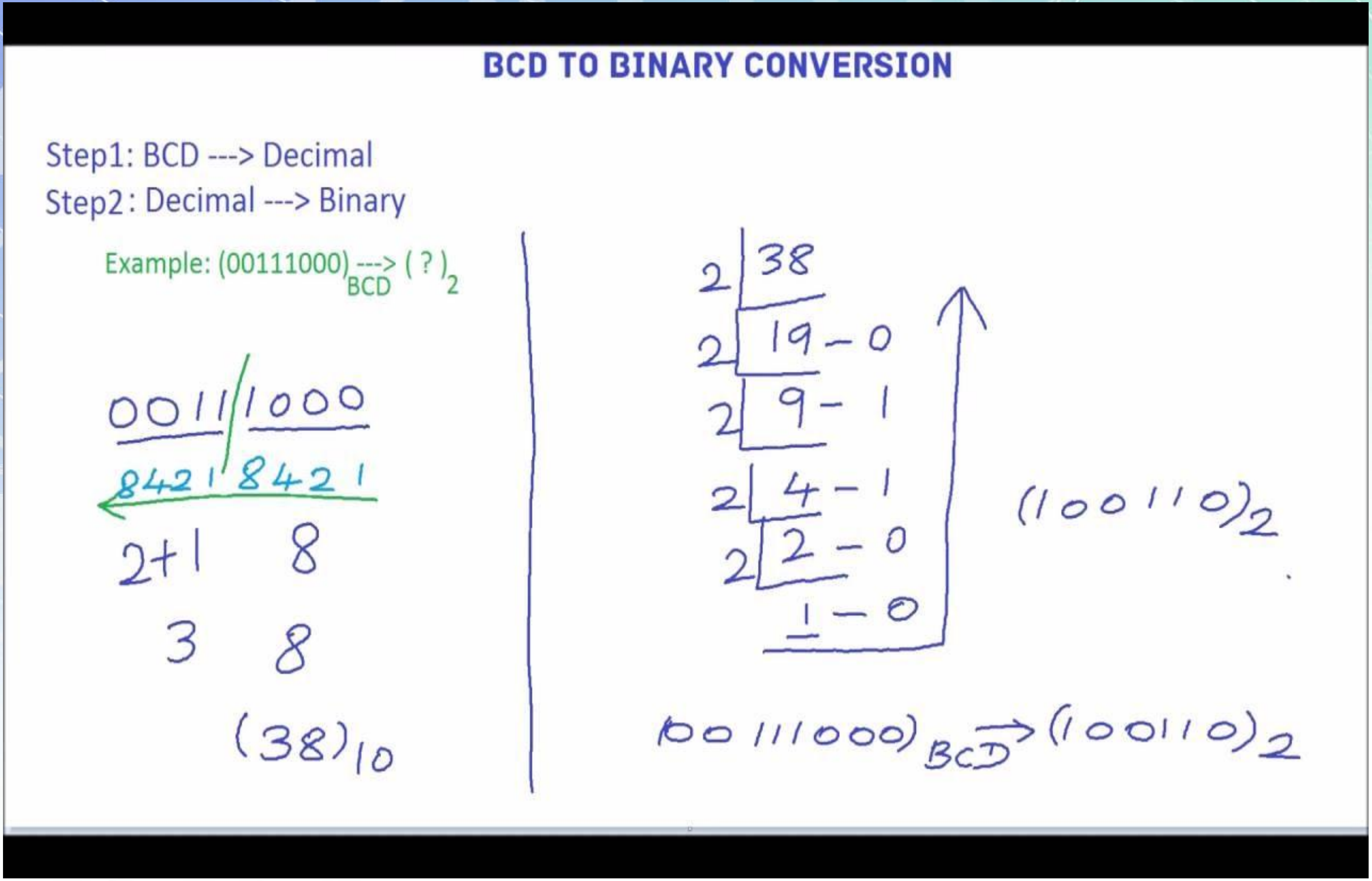
$$\begin{array}{cc} 1 & 9 \\ \downarrow & \downarrow \end{array}$$

$$(0001\ 1001)_{BCD}$$

$$(10011)_2 \rightarrow 0001100$$

Examples

- $(110101)_2$
- $(1000101)_2$
- $(1111101)_2$
- $(10000110)_2$
- $(10010010)_2$



BCD TO BINARY CONVERSION

Step1: BCD ---> Decimal

Step2: Decimal ---> Binary

Example: $(00111000)_{BCD} \dashrightarrow (\ ?\ )_2$

$\underline{0011} \mid \underline{1000}$

$8421 \mid 8421$

$2+1 \quad 8$

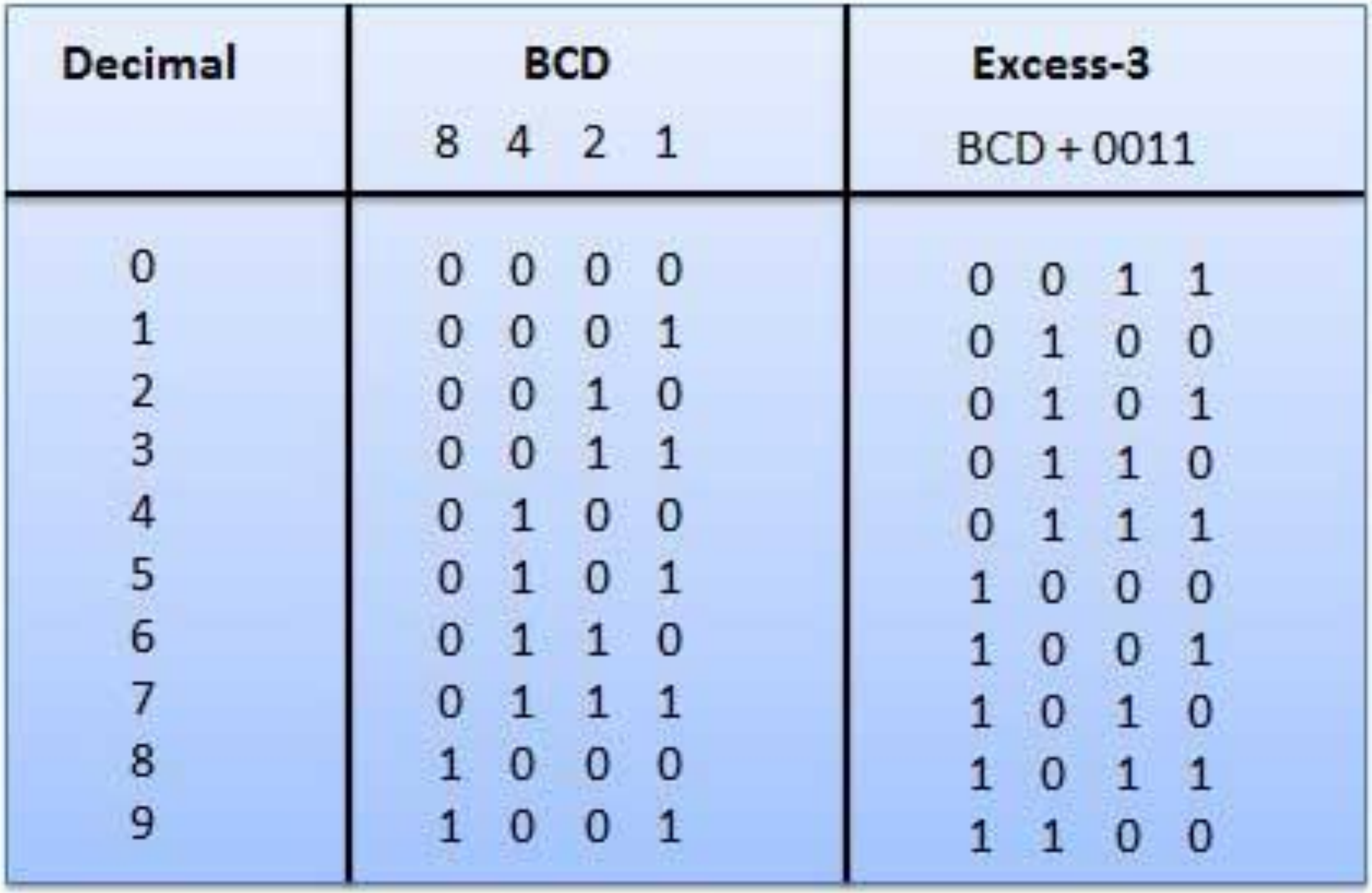
$3 \quad 8$

$(38)_{10}$

```
2 | 38
2 | 19 - 0
2 |  9 - 1
2 |  4 - 1
2 |  2 - 0
     1 - 0
```

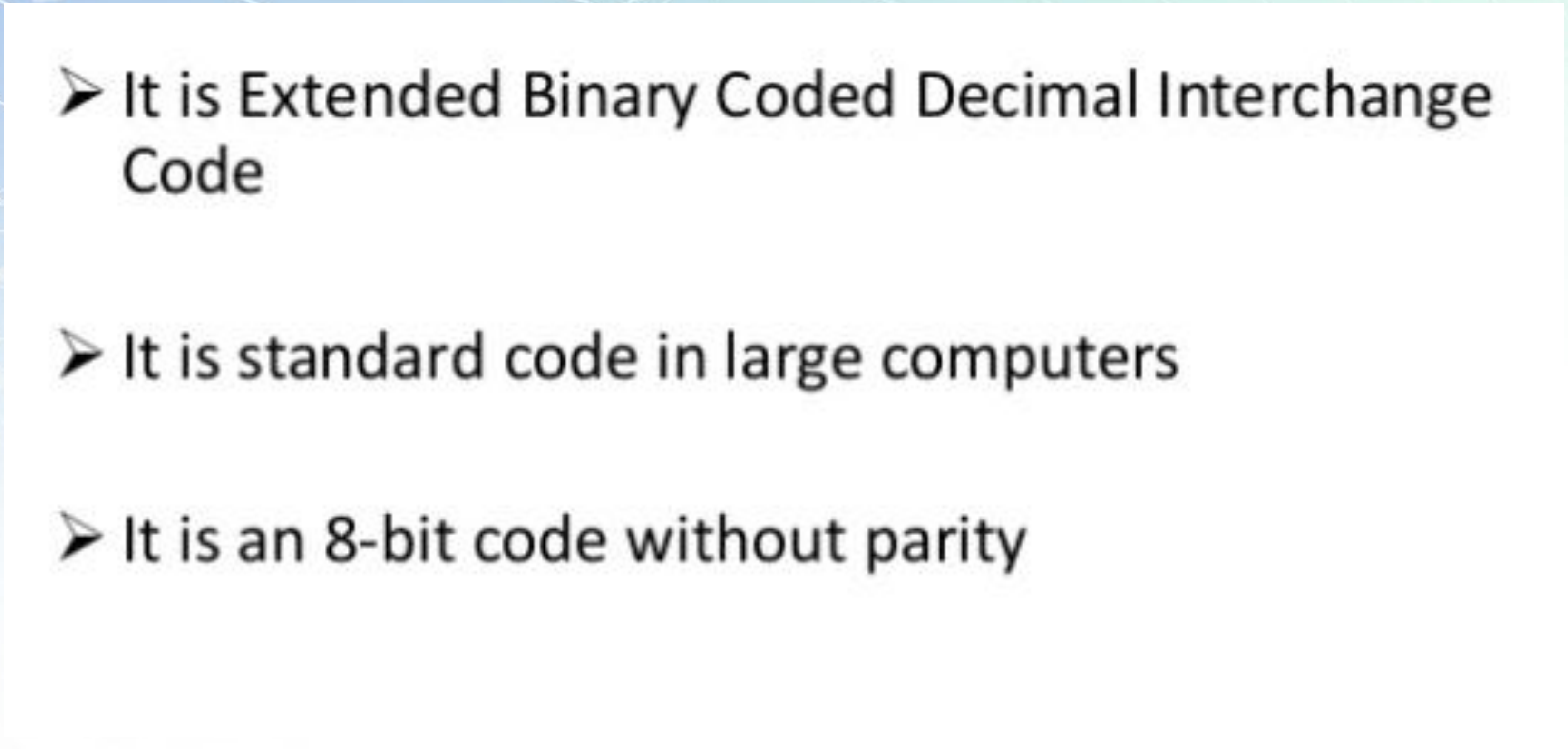
$(100110)_2$

$(00111000)_{BCD} \rightarrow (100110)_2$



Decimal	BCD				Excess-3			
	8	4	2	1	BCD + 0011			
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

EBCDIC Code



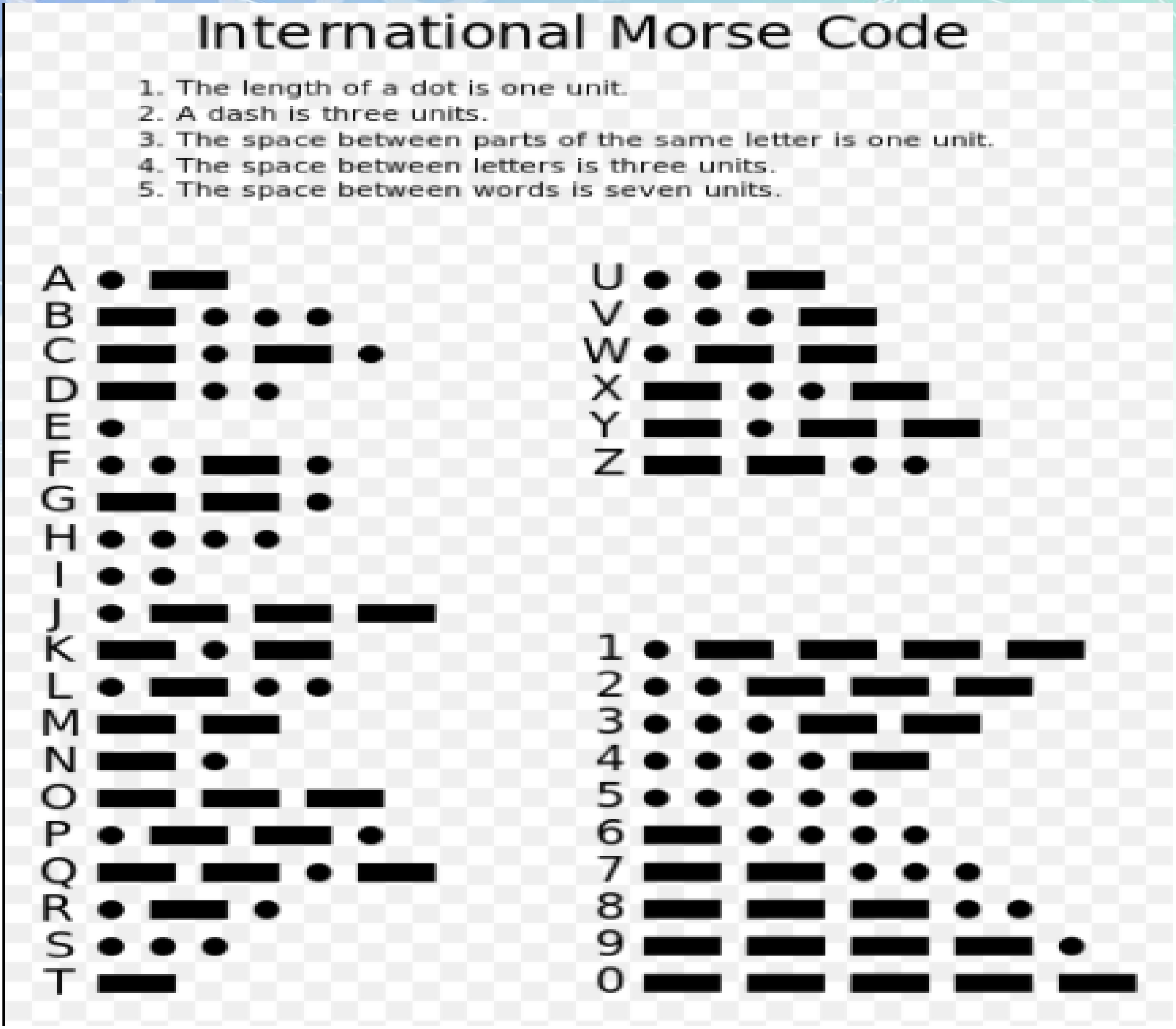
- It is Extended Binary Coded Decimal Interchange Code
- It is standard code in large computers
- It is an 8-bit code without parity

Hollerith Code

- The holes in punch cards are punched according to a code
- Each character in this code is represented by a unique combination of punched holes.

Morse code

- **Morse code** is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment.
- It is named for Samuel F. B. Morse, an inventor of the telegraph.
- Each Morse code symbol represents either a text character (letter or numeral) or a prosign and is represented by a unique sequence of dots and dashes.
- The duration of a dash is three times the duration of a dot. Each dot or dash is followed by a short silence, equal to the dot duration.
- The letters of a word are separated by a space equal to three dots (one dash), and the words are separated by a space equal to seven dots.
- The dot duration is the basic unit of time measurement in code transmission.



International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	● ▬	U	● ● ▬
B	▬ ● ● ●	V	● ● ● ▬
C	▬ ● ▬ ●	W	● ▬ ▬
D	▬ ● ●	X	▬ ● ● ▬
E	●	Y	▬ ● ▬ ▬
F	● ● ▬ ●	Z	▬ ▬ ● ●
G	▬ ▬ ●		
H	● ● ● ●		
I	● ●		
J	● ▬ ▬ ▬		
K	▬ ● ▬	1	● ▬ ▬ ▬ ▬
L	● ▬ ● ●	2	● ● ▬ ▬ ▬
M	▬ ▬	3	● ● ● ▬ ▬
N	▬ ●	4	● ● ● ● ▬
O	▬ ▬ ▬	5	● ● ● ● ●
P	● ▬ ▬ ●	6	▬ ● ● ● ●
Q	▬ ▬ ● ▬	7	▬ ▬ ● ● ●
R	● ▬ ●	8	▬ ▬ ▬ ● ●
S	● ● ●	9	▬ ▬ ▬ ▬ ●
T	▬	0	▬ ▬ ▬ ▬ ▬

Teletypewriter (TTY)

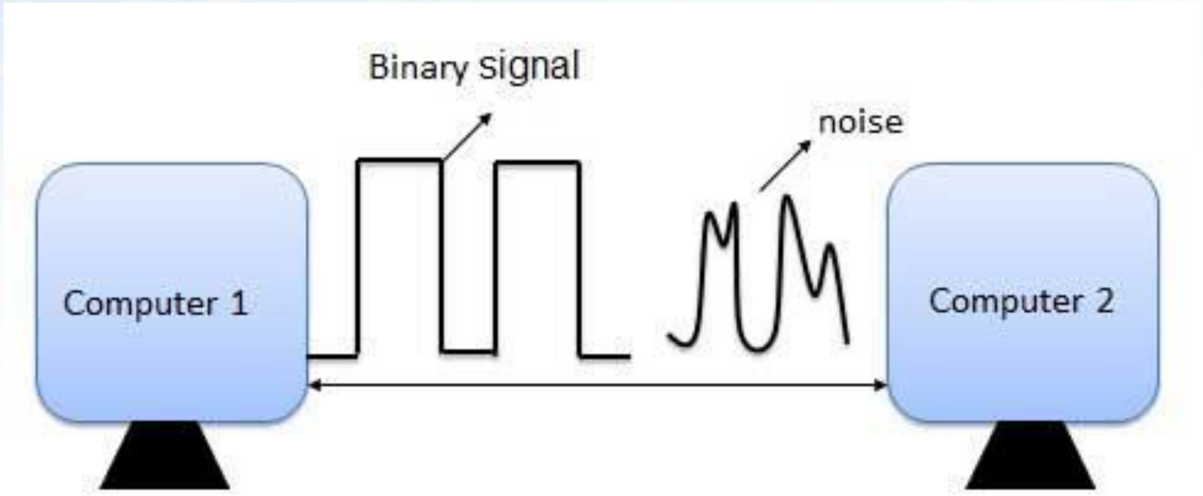
A **TTY** is a special device that lets people who are deaf, hard of hearing, or speech-impaired use the telephone to communicate, by allowing them to type messages back and forth to one another instead of talking and listening.

Error Correction & Detection

What is Error?

Error is a condition when the output information does not match with the input information.

During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from one system to other. That means a 0 bit may change to 1 or a 1 bit may change to 0.



Error-Detecting codes

Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted.

To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message.

A simple example of error-detecting code is **parity check**.

Error-Correcting codes

Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received.

This type of code is called an error-correcting code.

Error-correcting codes also deploy the same strategy as error-detecting codes but additionally, such codes also detect the exact location of the corrupt bit.

In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location.

Once the corrupt bit is located, its value is reverted (from 0 to 1 or 1 to 0) to get the original message.

Parity Checking of Error Detection

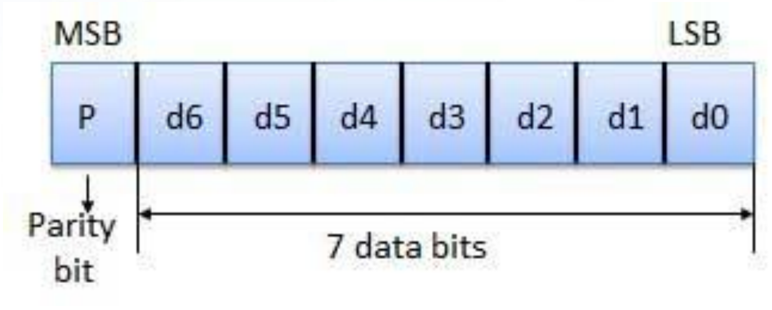
It is the simplest technique for detecting and correcting errors.

The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits.

The parity of 8-bits transmitted word can be either even parity or odd parity.

Even parity -- Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6,....).

Odd parity -- Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,....).



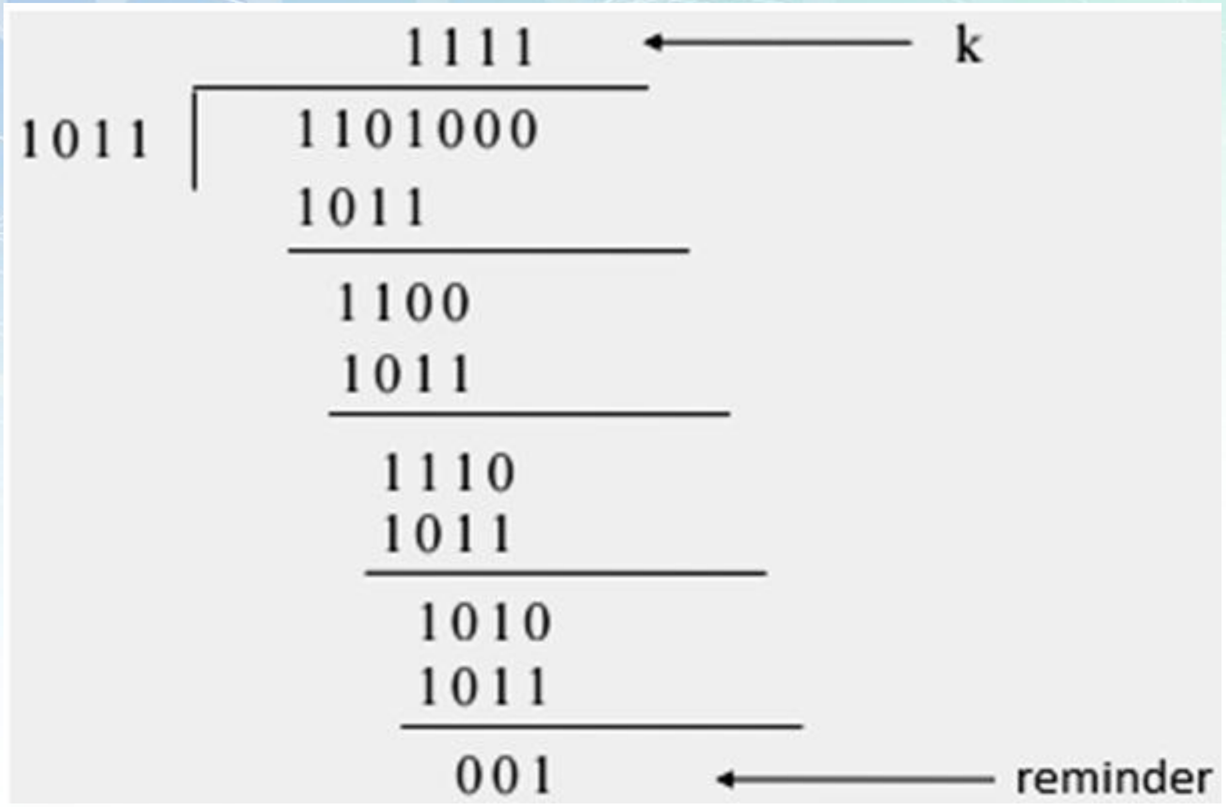
Cyclic Redundancy Check

A **cyclic redundancy check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data.

Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents.

On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption.

CRC Code Generation


1111
k
1011
1101000
1011
1100
1011
1110
1011
1010
1011
001
reminder