Computer Storage and Arithmetic

1

Computer Storage

- A computer can be logically considered to be a large collection of switches – much like a light switch

2

Computer Storage

- How many signals can we send with a light switch?
  - Two – it's ON or OFF
- We're going to consider that if the switch is ON, it's a 1
- If it's OFF, it's a 0

3

- So everything that the CPU sees comes as a stream of switch outputs that are either ON or OFF (that we are going to consider to be 1s or 0s)
- So everything stored on a computer can also be considered to be a stream of 1s and 0s
  - Data, programs, are all stored as a sequence of 1s and 0s
  - Each 1 or 0 is called a bit (8 bits in a byte)

4

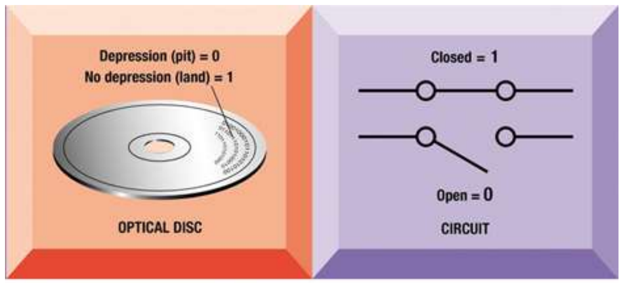
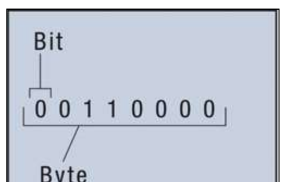


FIGURE 3-1
Ways of representing 0 and 1. Digital computers recognize only two states–off and on–usually represented by 0 and 1.

Digital Data Representation

- Bit: The smallest unit of data that a binary computer can recognize (a single 1 or 0)
- Byte = 8 bits
- Byte terminology used to express the size of documents and other files, programs, etc.
- Prefixes are often used to express larger quantities of bytes: kilobyte (KB), megabyte (MB), gigabyte (GB), etc.



Abbreviation	Approximate Size
KB	1,024 bytes
MB	1 million bytes
GB	1 billion bytes
TB	1 trillion bytes
PB	1,000 terabytes

FIGURE 2-2
Bits and bytes.

6

The Binary Numbering System

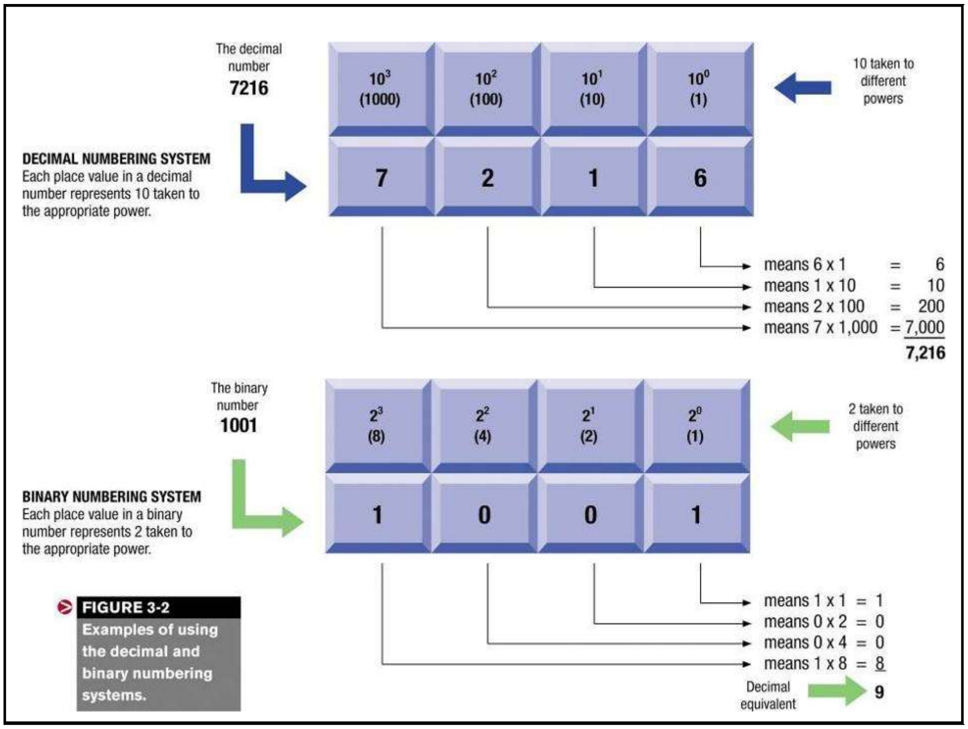
- We normally use the *decimal* numbering system, which uses 10 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9).
- Computers use the **binary numbering system,** which represents all numbers using just two symbols (0 and 1).

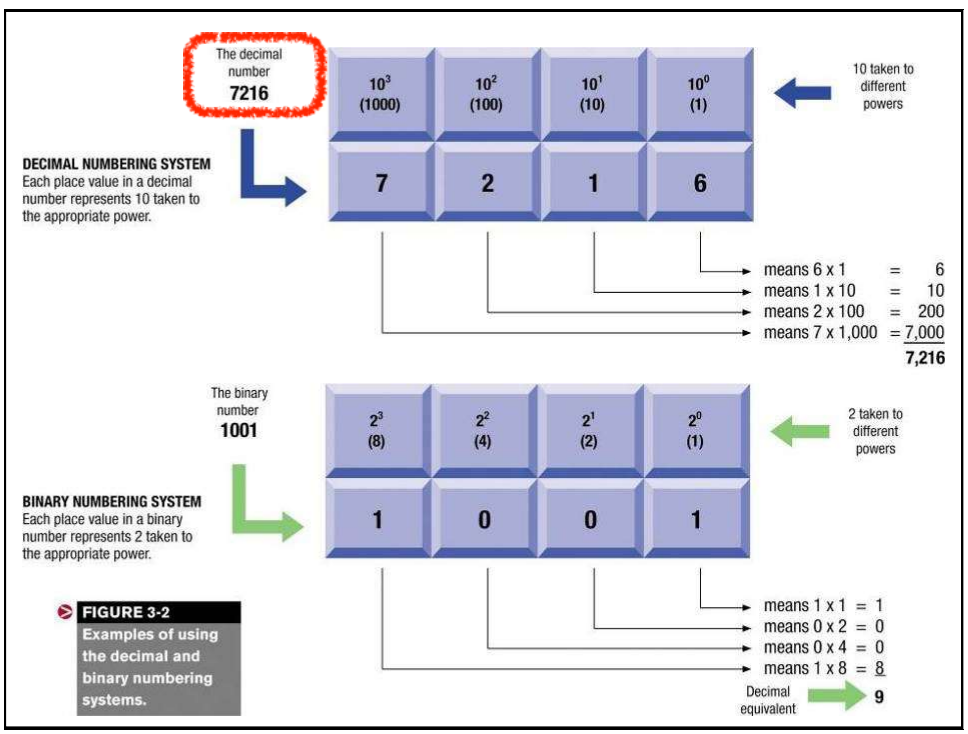
7

Understanding Decimal Numbers

- Decimal numbers are made of 10 decimal numerals: (0,1,2,3,4,5,6,7,8,9)
- But how many items does a decimal number represent?
  - $8653 = 8x10^3 + 6x10^2 + 5x10^1 + 3x10^0$
- What about fractions?
  - $97654.35 = 9x10^4 + 7x10^3 + 6x10^2 + 5x10^1 + 4x10^0 + 3x10^{-1} + 5x10^{-2}$
  - In formal notation -> $(97654.35)_{10}$

8


The decimal number
7216
10^3 (1000)
10^2 (100)
10^1 (10)
10^0 (1)
10 taken to different powers
DECIMAL NUMBERING SYSTEM
Each place value in a decimal number represents 10 taken to the appropriate power.
7
2
1
6
means 6 x 1 = 6
means 1 x 10 = 10
means 2 x 100 = 200
means 7 x 1,000 = 7,000
7,216
The binary number
1001
2^3 (8)
2^2 (4)
2^1 (2)
2^0 (1)
2 taken to different powers
BINARY NUMBERING SYSTEM
Each place value in a binary number represents 2 taken to the appropriate power.
1
0
0
1
means 1 x 1 = 1
means 0 x 2 = 0
means 0 x 4 = 0
means 1 x 8 = 8
Decimal equivalent
9
FIGURE 3-2
Examples of using the decimal and binary numbering systems.


The decimal number
7216
10^3 (1000)
10^2 (100)
10^1 (10)
10^0 (1)
10 taken to different powers
DECIMAL NUMBERING SYSTEM
Each place value in a decimal number represents 10 taken to the appropriate power.
7
2
1
6
means 6 x 1 = 6
means 1 x 10 = 10
means 2 x 100 = 200
means 7 x 1,000 = 7,000
7,216
The binary number
1001
2^3 (8)
2^2 (4)
2^1 (2)
2^0 (1)
2 taken to different powers
BINARY NUMBERING SYSTEM
Each place value in a binary number represents 2 taken to the appropriate power.
1
0
0
1
means 1 x 1 = 1
means 0 x 2 = 0
means 0 x 4 = 0
means 1 x 8 = 8
Decimal equivalent
9
FIGURE 3-2
Examples of using the decimal and binary numbering systems.

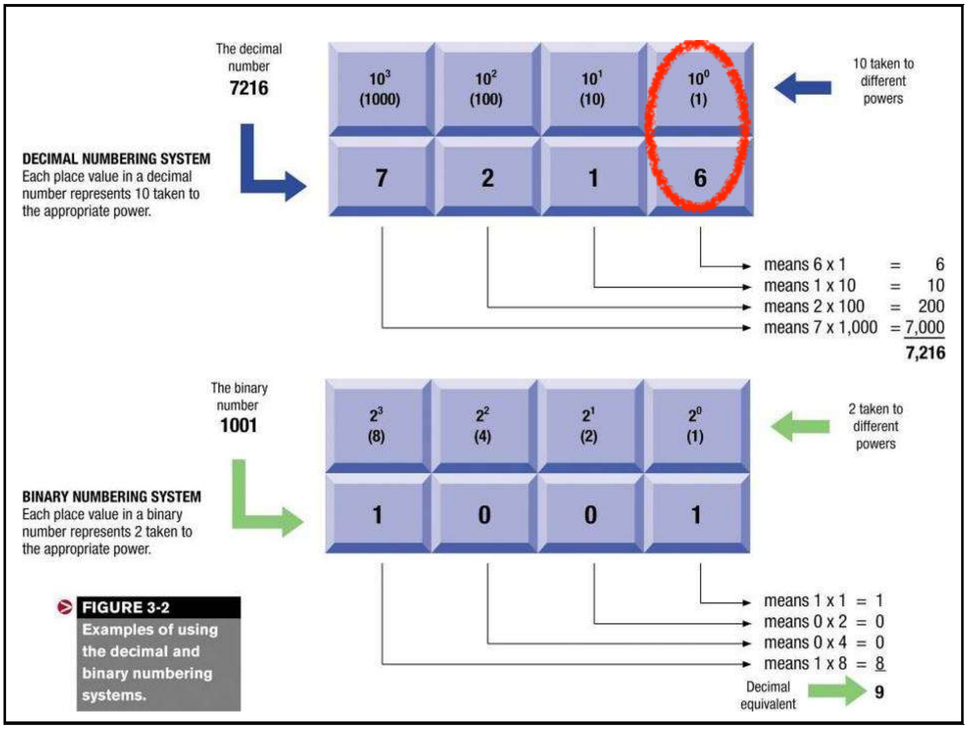


FIGURE 3-2
Examples of using the decimal and binary numbering systems.

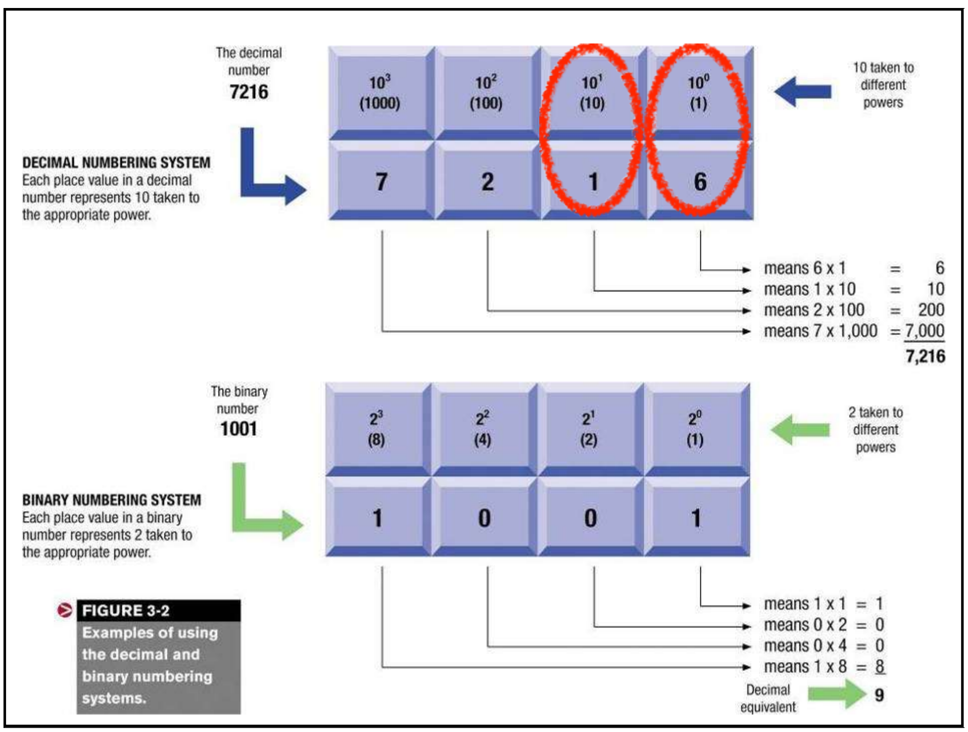
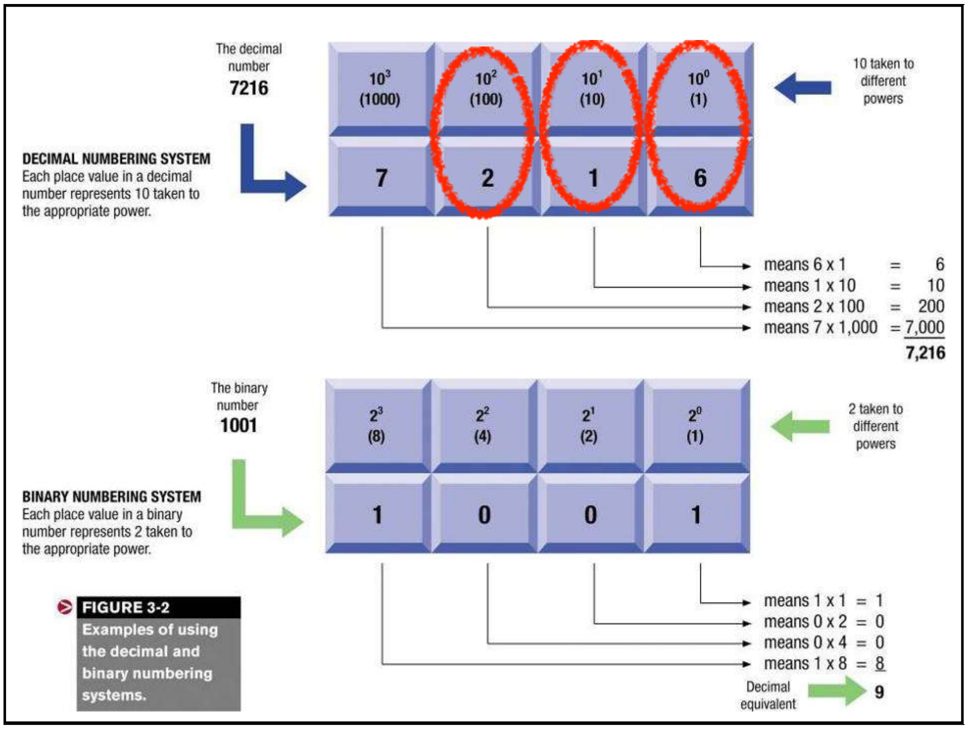
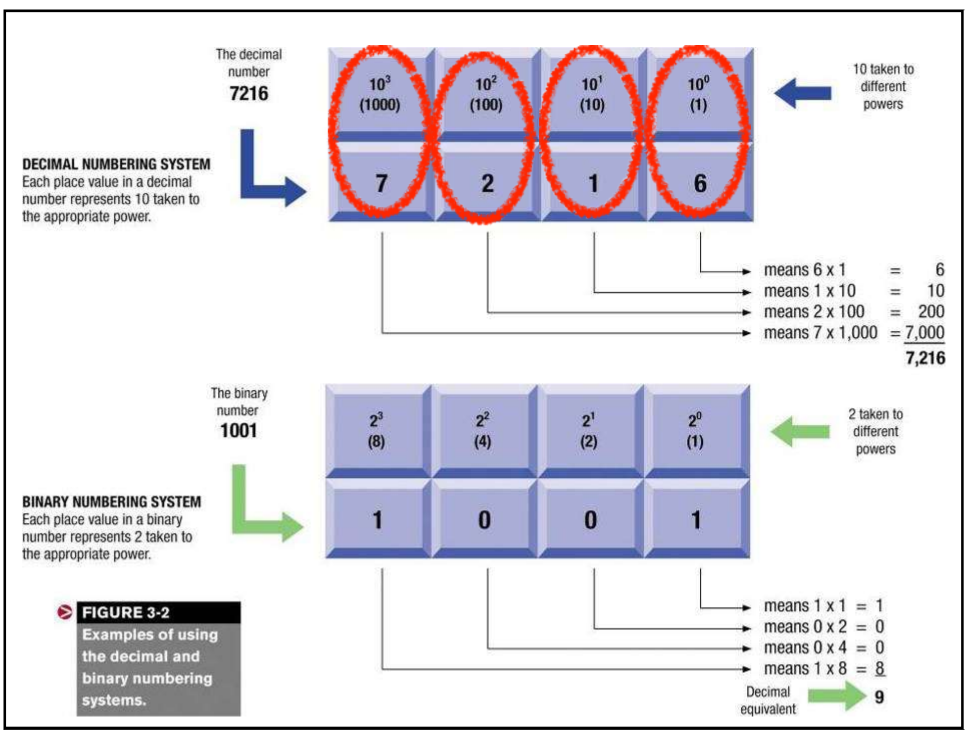


FIGURE 3-2
Examples of using the decimal and binary numbering systems.


The decimal number 7216
10^3 (1000)
10^2 (100)
10^1 (10)
10^0 (1)
7
2
1
6
10 taken to different powers
DECIMAL NUMBERING SYSTEM
Each place value in a decimal number represents 10 taken to the appropriate power.
means 6 x 1 = 6
means 1 x 10 = 10
means 2 x 100 = 200
means 7 x 1,000 = 7,000
7,216
The binary number 1001
2^3 (8)
2^2 (4)
2^1 (2)
2^0 (1)
1
0
0
1
2 taken to different powers
BINARY NUMBERING SYSTEM
Each place value in a binary number represents 2 taken to the appropriate power.
means 1 x 1 = 1
means 0 x 2 = 0
means 0 x 4 = 0
means 1 x 8 = 8
Decimal equivalent
9
FIGURE 3-2
Examples of using the decimal and binary numbering systems.


The decimal number 7216
10^3 (1000)
10^2 (100)
10^1 (10)
10^0 (1)
7
2
1
6
10 taken to different powers
DECIMAL NUMBERING SYSTEM
Each place value in a decimal number represents 10 taken to the appropriate power.
means 6 x 1 = 6
means 1 x 10 = 10
means 2 x 100 = 200
means 7 x 1,000 = 7,000
7,216
The binary number 1001
2^3 (8)
2^2 (4)
2^1 (2)
2^0 (1)
1
0
0
1
2 taken to different powers
BINARY NUMBERING SYSTEM
Each place value in a binary number represents 2 taken to the appropriate power.
means 1 x 1 = 1
means 0 x 2 = 0
means 0 x 4 = 0
means 1 x 8 = 8
Decimal equivalent
9
FIGURE 3-2
Examples of using the decimal and binary numbering systems.

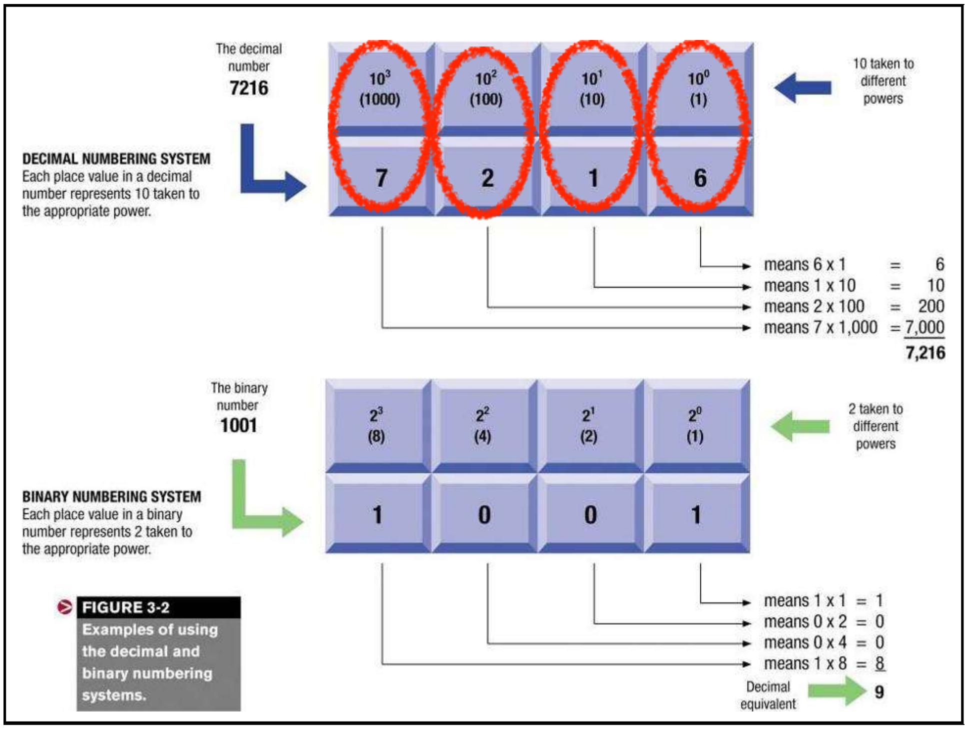


FIGURE 3-2
Examples of using the decimal and binary numbering systems.

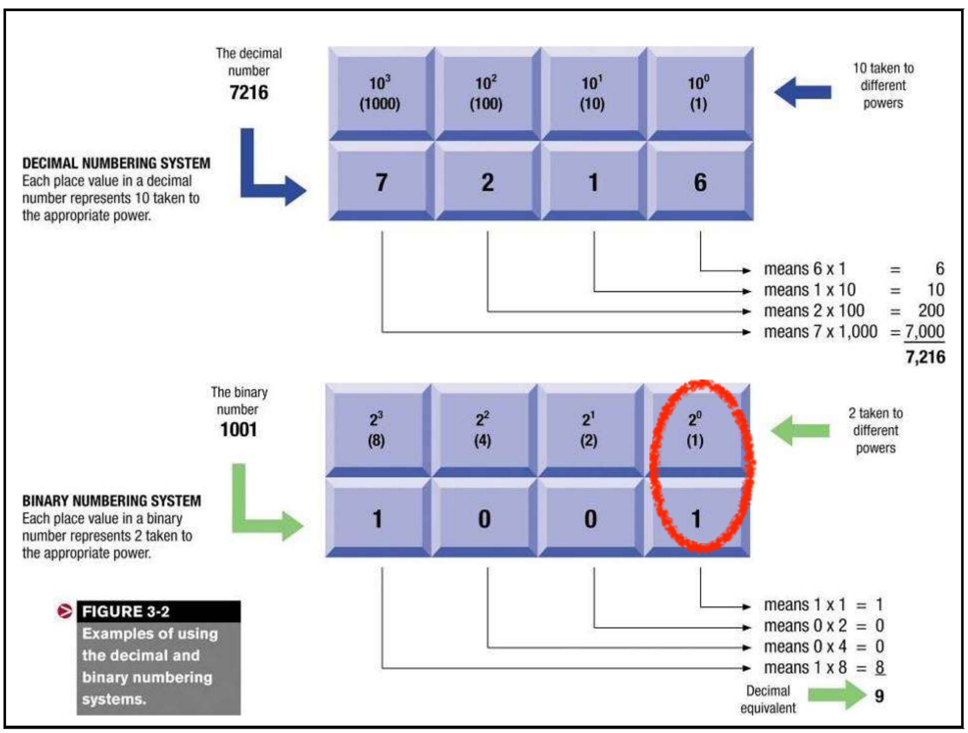


FIGURE 3-2
Examples of using the decimal and binary numbering systems.

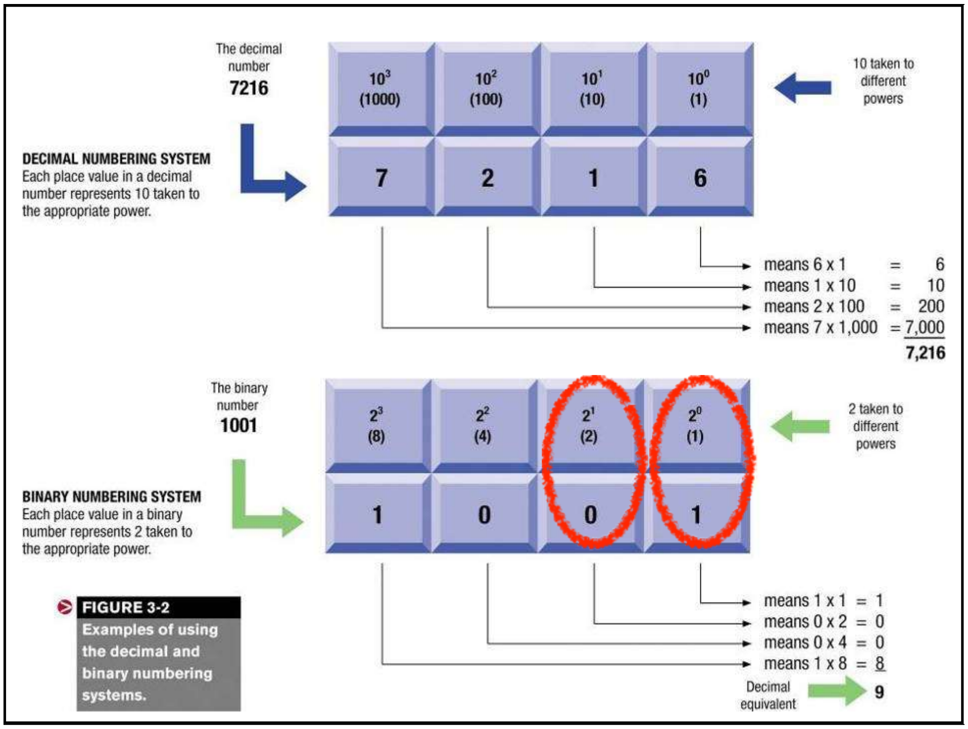


FIGURE 3-2 Examples of using the decimal and binary numbering systems.

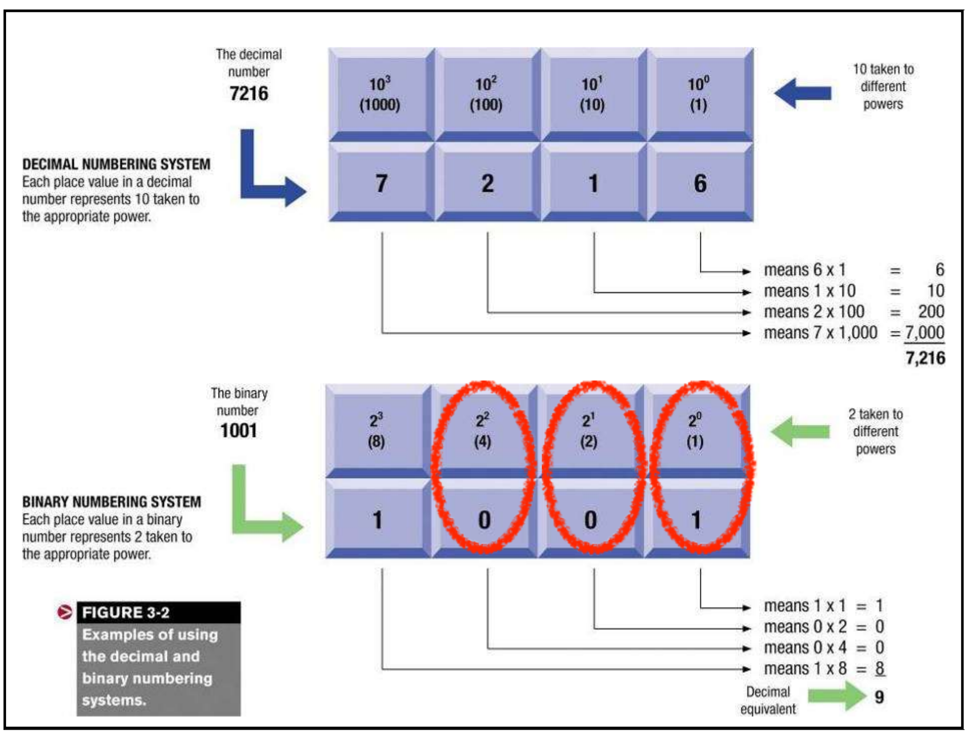


FIGURE 3-2 Examples of using the decimal and binary numbering systems.

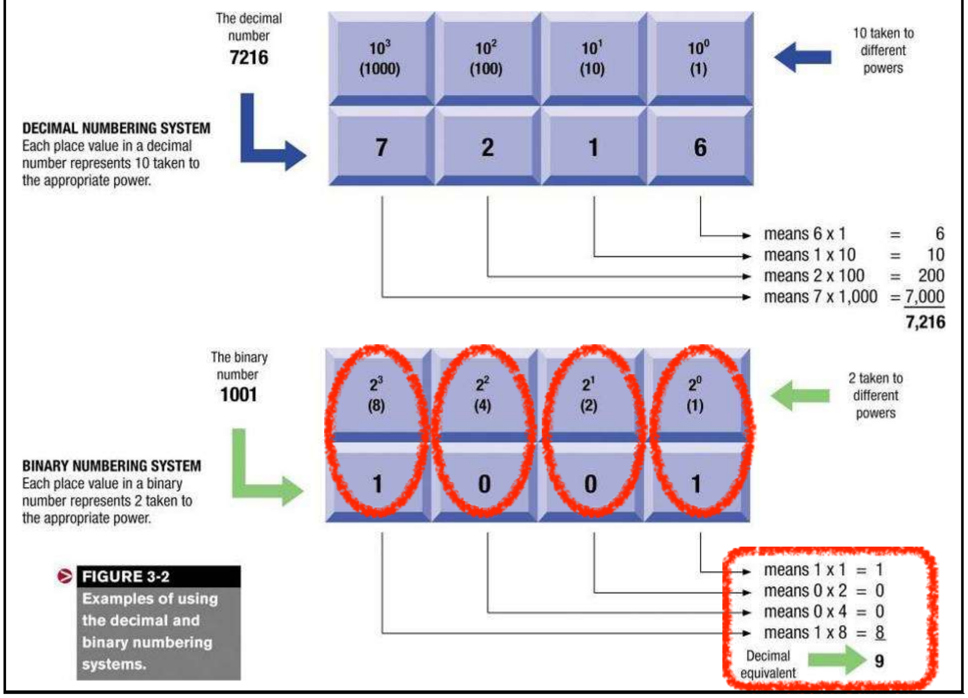
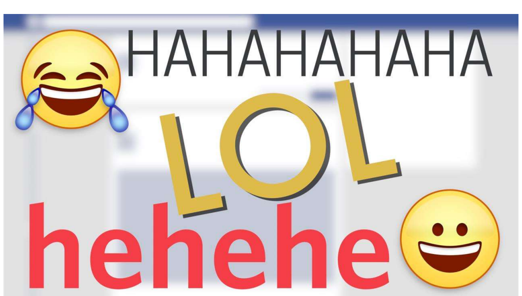


FIGURE 3-2 Examples of using the decimal and binary numbering systems.

"There are only 10 types of people in the world: those who u



Understanding Binary Numbers

- Binary numbers are made of binary digits (bits):
  - 0 and 1
- How many items does an binary number represent?
  - $(1011)_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 = (11)_{10}$
- What about fractions?
  - $(110.01)_2 = 1x2^2 + 1x2^1 + 0x2^0 + 0x2^{-1} + 1x2^{-2}$
- Groups of eight bits are called a *byte*
  - $(11001001)_2$
- Groups of four bits are called a *nibble.*
  - $(1101)_2$

21

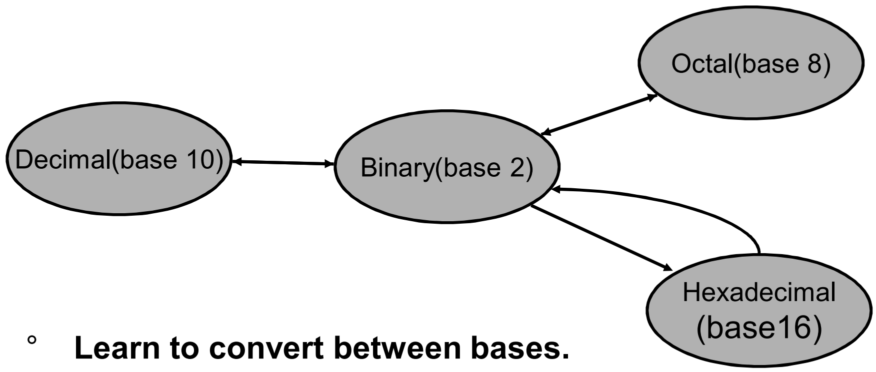
The Growth of Binary Numbers

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
50	$2^{50}=1P$	Peta

22

Conversion Between Number Bases



- **Learn to convert between bases.**

Convert an Integer *from* Decimal *to* Another Base

For each digit position:

1. **Divide decimal number by the base (e.g. 2)**
2. **The *remainder* is the lowest-order digit**
3. **Repeat first two steps until no *divisor* remains.**

Example for $(13)_{10}$:

	Integer Quotient		**Remainder**	**Coefficient**
13/2 =	6	+	½	$a_0 = 1$
6/2 =	3	+	0	$a_1 = 0$
3/2 =	1	+	½	$a_2 = 1$
1/2 =	0	+	½	$a_3 = 1$

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

Convert an Fraction *from* Decimal *to* Another Base

For each digit position:

1. **Multiply decimal number by the base (e.g. 2)**
2. **The *integer* is the highest-order digit**
3. **Repeat first two steps until fraction becomes zero.**

Example for $(0.625)_{10}$:

	Integer		Fraction	Coefficient
0.625 x 2 =	1	+	0.25	$a_{-1} = 1$
0.250 x 2 =	0	+	0.50	$a_{-2} = 0$
0.500 x 2 =	1	+	0	$a_{-3} = 1$

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

25

Understanding Octal Numbers

- Octal numbers are made of 8 digits:
  - (0,1,2,3,4,5,6,7)
- How many items does an octal number represent?
  - $(3456)_8 = 3x8^3 + 4x8^2 + 5x8^1 + 6x8^0 = (1838)_{10}$
- What about fractions?
  - $(213.5)_8 = 2x8^2 + 1x8^1 + 3x8^0 + 5x8^{-1} = (139.625)_{10}$
- Note that *each* octal digit can be represented with three bits.
  - $(111)_2 = (7)_8$

26

Convert an Integer *from* Decimal *to* Octal

For each digit position:

1. **Divide decimal number by the base (8)**
2. **The *remainder* is the lowest-order digit**
3. **Repeat first two steps until no *divisor* remains.**

Example for $(175)_{10}$:

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

Answer $(175)_{10} = (a_2\, a_1\, a_0)_2 = (257)_8$

27

Understanding Hexadecimal Numbers

- Hexadecimal numbers are made of <u>16</u> digits:
  - (0,1,2,3,4,5,6,7,8,9,A, B, C, D, E, F)
- How many items does an hex number represent?
  - $(3A9F)_{16} = 3x16^3 + 10x16^2 + 9x16^1 + 15x16^0 = (14999)_{10}$
- What about fractions?
  - $(2D3.5)_{16} = 2x16^2 + 13x16^1 + 3x16^0 + 5x16^{-1} = (723.3125)_{10}$
- Note that *each* hexadecimal digit can be represented with four bits.
  - $(1110)_2 = (E)_{16}$

28

Converting Between Base 16 and Base 2

$$3A9F_{16} = \underset{3}{\underline{0011}}\ \underset{A}{\underline{1010}}\ \underset{9}{\underline{1001}}\ \underset{F}{\underline{1111}}_2$$

- **Conversion is easy!**
  - **Determine 4-bit value for each hex digit**
- **Note that there are 2^4 = 16 different values of four bits**
- **Easier to read and write in hexadecimal.**
- **Representations are equivalent!**

29

Converting Between Base 16 and Base 8

$$3A9F_{16} = \underset{3}{\underline{0011}}\ \underset{A}{\underline{1010}}\ \underset{9}{\underline{1001}}\ \underset{F}{\underline{1111}}_2$$

$$\downarrow$$

$$35237_8 = \underset{3}{\underline{011}}\ \underset{5}{\underline{101}}\ \underset{2}{\underline{010}}\ \underset{3}{\underline{011}}\ \underset{7}{\underline{111}}_2$$

1. **Convert from Base 16 to Base 2**
2. **Regroup bits into groups of three starting from right**
3. **Ignore leading zeros**
4. **Each group of three bits forms an octal digit.**

30

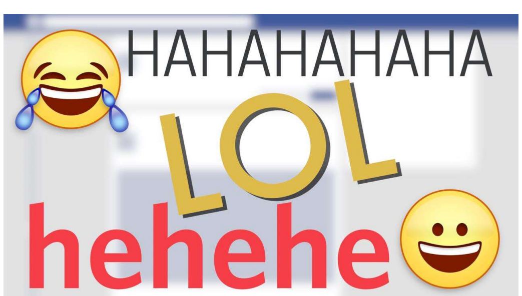
Putting It All Together

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

- **Binary, octal, and hexadecimal similar**
- **Easy to build circuits to operate on these representations**
- **Possible to convert between the three formats**

31

"Why do programmers always mix up Halloween and Ch



Binary Addition

- Very simple
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = 10
  - 1 + 1 + 1 = 11
  - 1 + 1 + 1 + 1 = 100

33

Binary Addition Example

- Let's add 111101 and 10111

```
  1  1  1  1  1  1 <------- carries
     1  1  1  1  0  1
+       1  0  1  1  1
---------------------
  1  0  1  0  1  0  0
```

34

How To Represent Signed Numbers

- **Plus and minus sign used for decimal numbers: 25 (or +25), -16, etc.**
- **For computers, desirable to represent everything as *bits.***
- **Three types of signed binary number representations: signed magnitude, 1's complement, 2's complement.**
- **In each case: left-most bit indicates sign: positive (0) or negative (1).**

Consider ***signed magnitude***:

$0\underline{0001100}_2 = 12_{10}$ — Sign bit, Magnitude

$1\underline{0001100}_2 = -12_{10}$ — Sign bit, Magnitude

35

One's Complement Representation

- **Sign magnitude doesn't work – try adding a number to its negative (e.g. 00000001 + 10000001)**
- **The one's complement of a binary number involves inverting all bits.**
  - **1's comp of 00110011 is 11001100**
  - **1's comp of 10101010 is 01010101**
- **To find negative of 1's complement number take the 1's complement.**

$0\underline{0001100}_2 = 12_{10}$ — Sign bit, Magnitude

$1\underline{1110011}_2 = -12_{10}$ — Sign bit, Magnitude

36

Two's Complement Representation

- **The two's complement of a binary number involves inverting all bits and adding 1.**
  - **2's comp of 10101010 is 01010110**
  - **First, inversion of 10101010 is 01010101**
  - **Add 1 to 01010101: 01010110.**

- **To find negative of 2's complement number take the 2's complement.**

$00001100_2 = 12_{10}$ $11110100_2 = -12_{10}$

Sign bit   Magnitude   Sign bit   Magnitude

37

2's Complement Addition

- Using 2's complement numbers, adding numbers is easy.
- Assume we have n=4 bit numbers
- Let's compute $(11)_{10} + (3)_{10}$.
  - $(11)_{10}$ = 1011_2 in 2's comp.
  - $(3)_{10}$ = 0011_2

Carry

```
         0 1 1
         1 0 1 1
+        0 0 1 1
--------------
      0  1 1 1 0
```

Ignore

38

2's Complement Addition (again)

- Now let's compute $(12)_{10} + (5)_{10}$.
- Again assume n=4 bit numbers
- Let's compute $(12)_{10} + (5)_{10}$.
  - $(12)_{10}$ = 1100_2 in 2's comp.
  - $(5)_{10}$ = 0101_2 in 2's comp.

```
Carry  ---->   1 1 1
               1 0 1 1
          +    0 1 0 1
          --------------
             1 0 0 0 0
             ^
           Ignore
```

39

2's Complement Subtraction

- Basic idea is to take the 2's complement and add
- For example, suppose we wish to subtract $+(0001)_2$ from $+(1100)_2$.
- Let's compute $(12)_{10} + (-1)_{10}$
  - $(12)_{10}$ = $+(1100)_2$ = 1100_2 in 2's comp.
  - $(-1)_{10}$ = $-(0001)_2 = 1111_2$ in 2's comp.

Step 1: Take 2's complement of 2[nd] operand
Step 2: Add binary numbers
Step 3: Ignore carry bit

```
                  1 1 0 0
            -   [ 0 0 0 1 ]
            --------------
2's comp
                  1 1 0 0
Add         +     1 1 1 1
            --------------
Final
Result          1 [ 1 0 1 1 ]
                ^
             Ignore
             Carry
```

40

Data Representation - ASCII Code

- American Standard Code for Information Interchange
- ASCII is a 7-bit code, frequently used with an 8th bit for error detection (more about that in a bit).

Character	ASCII (bin)	ASCII (hex)	Decimal	Octal
A	1000001	41	65	101
B	1000010	42	66	102
C	1000011	43	67	103
…				
Z				
a				
…				
1				
‘				

41

Coding Systems

- **Text**
  - **ASCII** and **EBCDIC**
    - Fixed-length codes that can represent any single character of data as a string of eight bits. *(A = 01000001)*
  - **Unicode**
    - A longer (32 bits per character is common) code that can be used to represent text-based data in virtually any written language.
- **Graphics data**
  - often stored as a *bitmap* which the colour to be displayed at each *pixel* stored in binary form. *(16 colours = 4 bits/pixel, 16 mill colours = 3 bytes)*
- **Audio data**
  - *waveform audio* is common; MP3 compression makes audio files somewhat smaller. *(CD quality = 2 byte samples taken at 44,100 samples/sec)*
- **Video data**
  - requires a great deal of storage space, but can be compressed. *(30 frames a sec, 256 colour image @640X480 = 307,200 bytes/frame)*

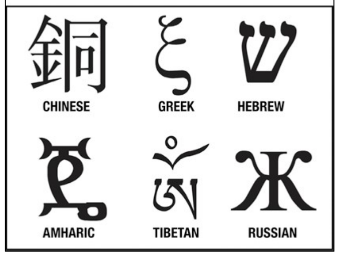
42

Coding Systems for Text-Based Data

CHARACTER	ASCII	EBCDIC
0	00110000	11110000
1	00110001	11110001
2	00110010	11110010
3	00110011	11110011
4	00110100	11110100
5	00110101	11110101
A	01000001	11000001
B	01000010	11000010
C	01000011	11000011
D	01000100	11000100
E	01000101	11000101
F	01000110	11000110
+	00101011	01001110
!	00100001	01011010
#	00100011	01111011

FIGURE 2-4
Examples from the ASCII and EBCDIC codes. These common fixed-length binary codes represent all characters as unique strings of 8 bits.

FIGURE 2-5
Unicode. Many characters, such as these, can be represented by Unicode but not by ASCII or EBCDIC.



43