

Chapter 1 Data Representation (Part 2)





Number System	Base	Coefficients
Decimal	10	0 - 9
Binary	2	0,1
Octal	8	0 - 7
Hexadecimal	16	0 - 9, A - F



- From any base-r to Decimal
- From Decimal to any base-r
- From Binary to either Octal or Hexadecimal
- From either Octal or Hexadecimal to Binary

Binary to Octal

Procedure:

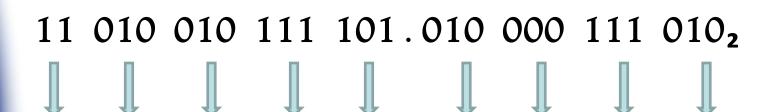
- Partition binary number into groups of 3 digits
- Convert each group to its equivalent decimal value

Example:

 $(11010010111101.010000111010)_2 =$

_____8

Binary to Octal



Binary to Octal

Thus, $(11010010111101.010000111010)_2 = (32275.2072)_8$

Binary to Hexadecimal

Procedure:

- Partition binary number into groups of 4 digits
- Convert each group to its equivalent decimal value

Example:

 $(11010010111101.010000111010)_2 =$

_____16

Binary to Hexadecimal

Thus, $(11010010111101.010000111010)_2 = (34BD.43A)_{16}$

Octal to Binary

Procedure:

 Each octal digit is converted to its 3-digit binary equivalent.

Let's try:

$$(534.123)_8 =$$

Hexadecimal to Binary

Procedure:

Each hexadecimal digit is converted to its
 4-digit binary equivalent.

Let's try:

Any Other Number System

• In general, a number expressed in **base-r** has r possible coefficients multiplied by powers of r.

$$a_n r^n + a_{n-1} r^{n-1} + a_{n-2} r^{n-2} + ... + a_0 r^0 + a_{-1} r^{-1} + a_{-2} r^{-2} + ... + a_{-m} r^{-m}$$
where
$$\mathbf{n} = \text{position of the coefficient}$$

$$\text{coefficients} = 0 \text{ to } r-1$$

Example

Base 5 number coefficients:

• (341.2)₅

Example

Base 5 number coefficients:

$$0$$
 to $r-1$ (0, 1, 2, 3, 4)

• $(341.2)_5$ = $(3 \times 5^2) + (4 \times 5^1) + (1 \times 5^0) + (2 \times 5^{-1})$

Example

Base 5 number coefficients:

$$0$$
 to $r-1$ (0, 1, 2, 3, 4)

• $(341.2)_5$ = $(3 \times 5^2) + (4 \times 5^1) + (1 \times 5^0) + (2 \times 5^{-1})$ = 75 + 20 + 1 + 0.4= $(96.4)_{10}$



Unsigned Number

- ➤ leftmost bit is the most significant bit.
- > Example.

$$01101 = 13$$

$$10110 = 22$$

Signed Number

- > Leftmost bit represents the sign.
- > Example:

$$01100 = +12$$

$$11100 = -12$$

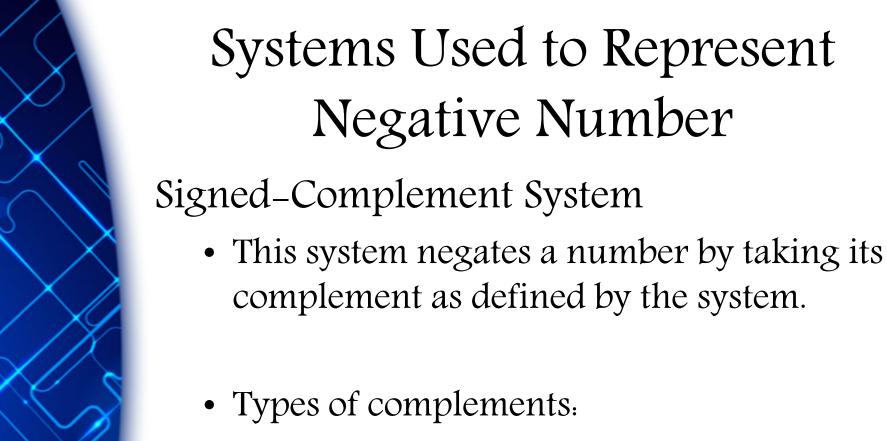
Systems Used to Represent Negative Number

Signed-Magnitude Representation

• A number consists of a magnitude and a symbol indicating whether the magnitude is positive or negative.

Examples:

$$+55 = 0110111_{2}$$
 $-55 = 1110111_{2}$
 $+126 = 011111110_{2}$ $-126 = 111111110_{2}$



- Radix-complement
- Diminished Radix-complement

Complements

 Diminished Radix Complement

General formula:

$$(r-1)$$
's C of N = $(r^n - r^{-m}) - N$

where

n = # of digits (integer)

m = # of digits (fraction)

r = base / radix

N = the given # in base-r

Complements

Diminished Radix Complement

Radix Complement

General formula.

$$(r-1)^{2}$$
s C of N = $(r^{n} - r^{-m}) - N$

General formula.

$$r's C of N = r^n - N$$

where

where

$$n = \# \text{ of bits}$$

Examples

9's C

- 012390 = 987609
- 54670.5 = 45329.4

10's C

- 012390 = 987610
- 54670.5 = 45329.5

Examples

9's C

- 012390 = 987609

1's C

- 1101100 = 0010011
- 54670.5 = 45329.4 0110111 = 1001000

10's C

2's C

- 012390 = 987610 1101100 = 0010100
 - 54670.5 = 45329.5 0110111 = 1001001

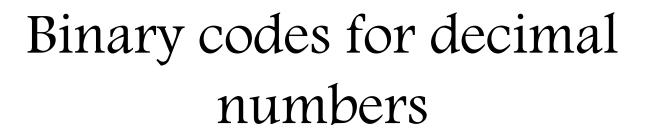
Binary Codes

Code

 a set of n-bit strings in which different bit strings represent different numbers or other things.

Binary codes are used for:

- Decimal numbers
- Character codes



At least four bits are needed to represent ten decimal digits.

Some binary codes:

- BCD (Binary-coded decimal)
- Excess-3
- Biquinary

Binary codes for decimal numbers

BCD

- straight assignment of the binary equivalent
- weights can be assigned to the binary bits according to their position

Excess-3 Code

- unweighted code
- BCD + 3

Biquinary Code

- seven-bit code with error detection properties
- each decimal digit consists of 5 0's and 2 1's

Binary codes for decimal digits

Decimal	BCD				Biquinary
Digit	8421	Excess-3	84-2-1	2421	5043210
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	0101	1000001
6	0110	1001	1010	0110	1000010
7	0111	1010	1001	0111	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000



Differences between Binary and BCD

- BCD is not a number system
- BCD requires more bits than Binary
- BCD is less efficient than Binary
- BCD is easier to use than Binary

Coding vs. Conversion

Conversion

- bits obtained are binary digits
- Example: 11 = 1011

Coding

- bits obtained are combinations of 0's and
 1's
- Example: 11 = 0001 0001



- American Standard
 Code for Information
 Interchange
 - 7-bit code
 - contains 94
 graphic
 characters and 34
 non-printing
 characters

- Extended Binary Coded Decimal Interchange Code
 - 8-bit code
 - last 4 bits range from 0000-1001

Gray Code

It is a binary number system where two successive values differ in only one digit, originally designed to prevent spurious output from electromechanical switches.

Decimal	Binary Code	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100



- No representation method is capable of representing all real numbers.
- Most real values
 must be represented
 by an
 approximation.

- Various methods can be used:
- ✓ Fixed-point number system
- ✓ Rational number system
- ✓ Floating point number system
- ✓ Logarithmic number system



- It is a method used to represent integer values.
- Disadvantages:
 - very small real numbers are not clearly distinguished
 - very large real numbers are not known accurately enough



- It is a method used to represent real numbers
- Notation:
 - Mantissa x Base^{exponent}
- Example (32-bit floating point number):

1	10000110	0010010110000000000000
Sign	Exponent (Excess-127)	Mantissa

Floating-point representation

• Example:

1	10001000	0110110000100000000000
Sign	Exponent (Excess-127)	Mantissa

Exponent: $10001000_2 = 136_{10}$ (Excess-127); 136 - 127 = 9.

De-normalize: $1.01101100001_2 \times 2^9 = 1011011000.01_2$

Convert: $1011011000.01_2 = 728.25$

Sign: (1) negative

Result: -728.25

*Note: In de-normalizing the mantissa, we add 1 (hidden bit) before the decimal point. We also truncates the trailing zeros after the rightmost significant bit.



Let's try.

0 10000001 001000000000000000000000

Exponent:

De-normalize:

Convert:

Sign:

Result: