Data Representation in the Computer

The representation and storage of different data types in the computer

Chapter 3

- Distinguish between analog and digital information
- Explain data compression and calculate compression ratios
- Explain the binary formats for negative and floating-point (rational) values
- Describe the characteristics of the ASCII and Unicode character sets

- Explain the nature of sound and its representation
- Explain how RGB values define a colour
- Discuss the nature of images and raster and vector graphics
- Explain temporal and spatial video compression

Data and Computers

- Data versus Information
- Computers are **multimedia** devices, dealing with a vast array of information categories
- Computers store, present, and help us modify
  - Numbers
  - Text
  - Audio
  - Images and graphics
  - Video

Binary and Computers

Bit (Binary digit)

Byte 8 bits

The number of bytes in a word determines the word length of the computer, but it is usually a multiple of 8

32-bit machines
64-bit machines

Analog and Digital Information

- Analog data obtained from the real world is **continuous**
- Digital data as represented in digital computers is **discrete**
- Analog data is converted to digital data by sampling it at a finite number of points. The data at these points are stored in binary format.

Electronic Signals

Important facts about electronic signals

- An analog signal continually fluctuates in voltage up and down
- A digital signal has only a high or low state, corresponding to the two binary digits
- All electronic signals (both analog and digital) degrade as they move down a line
- The voltage of the signal fluctuates due to environmental effects

Electronic Signals *(Cont'd)*

Periodically, a digital signal is **reclocked** to regain its original shape

Figure 3.2

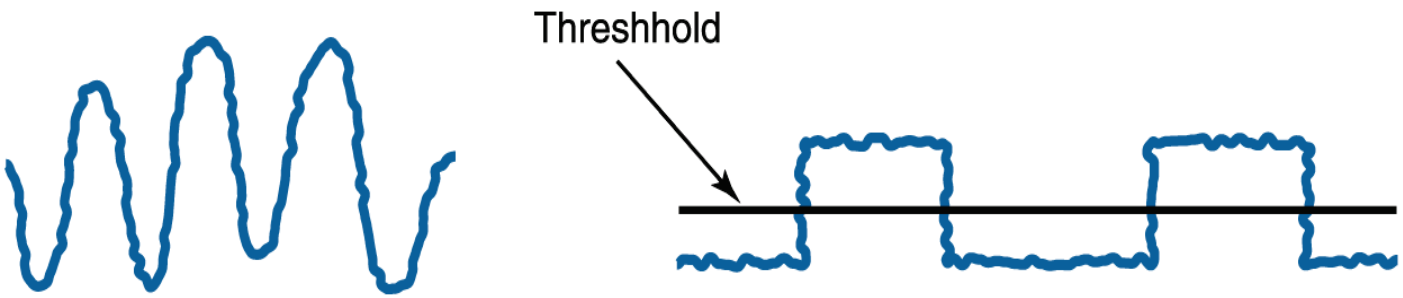


Figure 3.3
Degradation of analog and digital signals

Binary Representations

- Not only numbers, but **any** data can be represented by binary numbers

- Once we know the number of choices that any type of data requires, we can use enough binary digits to code it.

Examples

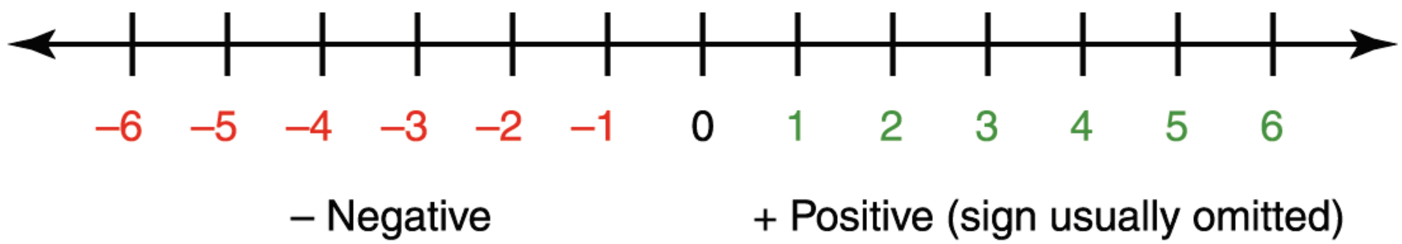
- True & false 1 bit
- Car gears 3 bits
- Numbers as many as we can afford!
- Grey scale images 8 bits
- English letters 6 bits

- How many different states can we represent with **n** bits?
- What happens every time you increase the number of bits by 1?

Representing Negative Numbers

Signed-magnitude number representation

The sign represents the ordering, and the digits represent the magnitude of the number

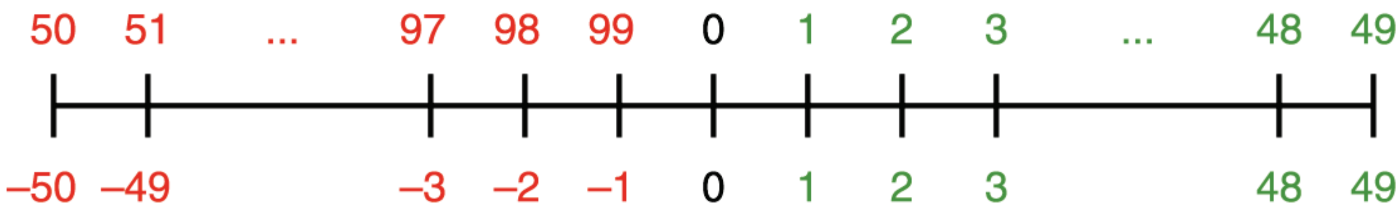


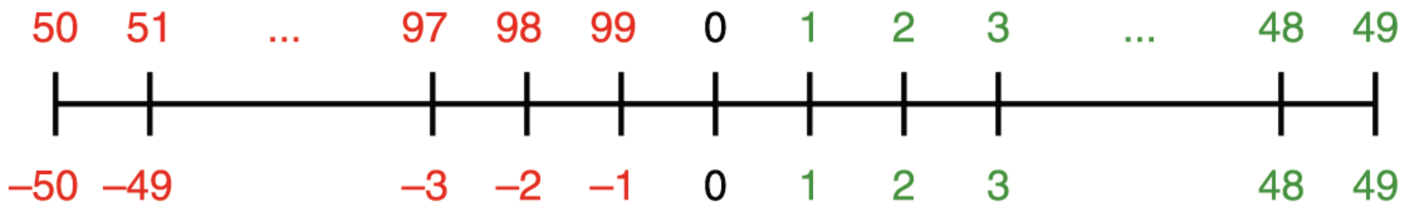
Representing Negative Numbers by Complements

Using two decimal digits:

- let 1 through 49 represent 1 through 49
- let 50 through 99 represent –50 through –1

- Easy to convert to new scheme – just subtract the number you want to make negative from 100
- Easy to convert back – just subtract a number in the negative range from 100.





To perform addition, add the numbers and discard any carry into the hundreds position

Signed-Magnitude	New Scheme
5 + – 6 – 1	5 + 94 99
– 4 + 6 2	96 + 6 2
– 2 + – 4 – 6	98 + 96 94

Now you try it

48 (signed–magnitude)
– 1
47

How does it work in the new scheme?

Subtraction

A−B=A+(−B)

Add the negative of the second to the first

Signed-Magnitude	New Scheme	Add Negative
−5	95	95
− 3	− 3	+ 97
−8		92

Try

4	*− 4*	*−4*
− 3	*+3*	*+ −3*

Representing Negative Values

Formula to compute the negative representation of a number

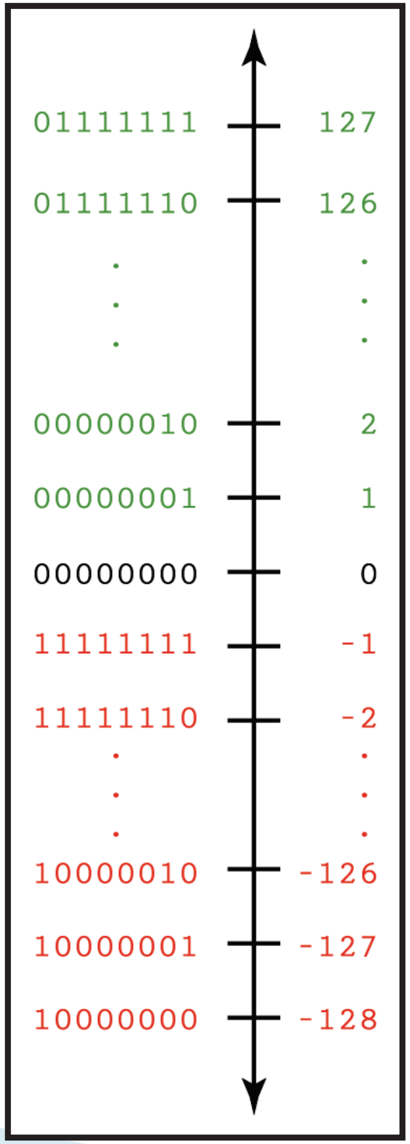
Negative$(I) = 10^k - I$, where k is the number of digits

Thus A – B is: $A + (10^k - B) = 10^k + (A-B)$

This representation is called the ten's complement

Representing Negative Values

Two's Complement
(Vertical line is easier to read)



Representing Negative Values

Addition and subtraction are the same as in 10's complement arithmetic

-127	10000001
+ 1	00000001
-126	10000010

Do you notice something interesting about the left-most bit?

Representing Negative Values

- Working out 2s complements:
- For example, for 8 bits, if we want to represent –x we would use $2^8 - x$
- That is 100000000–x
- Rather write this as 11111111 – x + 1 because the first two terms are “flip the bits of x”

<u>Rule</u>: Flip the bits of x and add 1!

Another Example

-32	-00100000	11011111+1	11100000
-64	-01000000	10111111+1	11000000
-96	-01100000		10100000

Number Overflow

What happens if the computed value won't fit?

Overflow

If each value is stored using eight bits, adding 127 to 3 overflows

$$\begin{array}{r} 01111111 \\ +\ 00000011 \\ \hline 10000010 \end{array}$$

Problems occur when mapping an unbounded world onto a bounded machine!