IS 335: Information Technology in Business
Lecture Outline
Data Representation

Objectives

- In this segment, you will learn to:
  - Describe numbering systems and their use in data representation
  - Compare different data representation methods
  - Summarize the CPU data types and explain how nonnumeric data is represented
  - Describe common data structures and their uses

Data Representation and Processing

- Capabilities required of any data/information processor–organic, mechanical, electrical, optical:
  - Recognizing external data and converting it to an internal format
  - Storing and retrieving data internally
  - Transporting data between internal storage and processing components
  - Manipulating data to produce desired results or decisions

Automated Data Processing

- Computers represent data electrically and process it with electrical switches
- Physical laws of electricity, optics, and quantum mechanics are described by mathematical formulas
- Processing operations must be based on mathematical functions

Binary Data Representation

- Binary numbers have only one of two possible values (0 or 1) per digit
- Reliably transported among computer system components
- Can be processed by two–state electrical devices (relatively easy to design and fabricate)
- Correspond directly with values in Boolean logic

Binary and decimal notations for the values 0 through 10

	Binary System (base 2)					Decimal System (base 10)			
Place	2^3	2^2	2^1	2^0		10^3	10^2	10^1	10^0
Values	8	4	2	1		1000	100	10	1
	0	0	0	0	=	0	0	0	0
	0	0	0	1	=	0	0	0	1
	0	0	1	0	=	0	0	0	2
	0	0	1	1	=	0	0	0	3
	0	1	0	0	=	0	0	0	4
	0	1	0	1	=	0	0	0	5
	0	1	1	0	=	0	0	0	6
	0	1	1	1	=	0	0	0	7
	1	0	0	0	=	0	0	0	8
	1	0	0	1	=	0	0	0	9
	1	0	1	0	=	0	0	1	0

Hexadecimal Notation

- Uses 16 as its base or radix
  - (hex = 6, and decimal = 10)
- Not enough numeric symbols available to represent 16 values; uses English letters for larger values
- Compact; advantage over binary notation
- Often used to designate memory addresses

Binary to Decimal to Hexadecimal

Binary				Decimal	Hexadecimal
8	4	2	1		
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	B
1	1	0	0	12	C
1	1	0	1	13	D
1	1	1	0	14	E
1	1	1	1	15	F

Goals of Computer Data Representation
- Compactness
- Range
- Accuracy
- Ease of manipulation
- Standardization

Goals of Computer Data Representation (continued)
- Compactness and range
  - Describes number of bits used to represent a numeric value
  - More compact data representation format; less expense to implement in computer hardware
  - Users and programmers prefer large numeric range
- Accuracy
  - Precision of representation increases with number of data bits used

Goals of Computer Data Representation (continued)
- Ease of manipulation
  - Manipulation is executing processor instructions (addition, subtraction, equality
- comparison)
  - Ease is machine efficiency
  - Processor efficiency depends on its complexity
- Standardization
  - Ensures correct and efficient data transmission
  - Flexibility to combine hardware from different vendors with minimal communication problems

CPU Data Types

- Primitive data types
  - Integer
  - Real number
  - Character
  - Boolean
  - Memory address
- Representation format for each type balances compactness, range, accuracy, ease of manipulation, and standardization

Integers

- A whole number—a value that does not have a fractional part
- Data formats can be signed or unsigned
  - Determines largest and smallest values that can be represented
  - Unsigned value is always assumed to be positive
  - Sign bit occupies bit position that would otherwise store part of a data value
  - Sign bit reduces the largest positive value that can be stored

Excess Notation

- Can be used to represent signed integers
- Divides a range of ordinary binary numbers in half; uses lower half for negative values and upper half for nonnegative values
- Always uses a fixed number of bits with the leftmost bit representing the sign (1 for nonnegative and 0 for negative values)

Bit String	Decimal Value	
1111	7	Nonnegative numbers
1110	6	
1101	5	
1100	4	
1011	3	
1010	2	
1001	1	
1000	0	
0111	-1	Negative Numbers
0110	-2	
0101	-3	
0100	-4	
0011	-5	
0010	-6	
0001	-7	
0000	-8	

Two's Complement Notation

- Nonnegative integer values are represented as ordinary binary values
- Compatible with digital electronic circuitry
  - Leftmost bit represents the sign
  - Fixed number of bit positions
  - Only two logic circuits required to perform addition on single-bit values
  - Subtraction can be performed as addition of a negative value

Range and Overflow

- Numeric range of a twos complement value is (2^{n-1}) to $(2^{n-1} – 1)$
- Overflow
  - Occurs when absolute value of a computational result contains too many bits to fit into fixed-width data format
- Avoiding overflow
  - Double-precision data formats
  - Careful programming

Range and Overflow (continued)

- Choose data format width by balancing:
  - Numeric range
  - Chance of overflow during program execution
  - Complexity, cost, and speed of processing and storage devices

Real Numbers

- Contain both whole and fractional components
- Require separation of components to be represented within computer circuitry
  - Fixed radix point (simple)
  - Floating point notation (complex)

Floating Point Notation

- Similar to scientific notation, except that 2 is the base
  - value = mantissa x $2^{exponent}$
- Many CPU-specific implementations of floating-point notation are possible
- IEEE standard 754 defines formats for floating-point data

Range, Overflow, and Underflow

- Range
  - Limited by number of bits in a floating-point string and formats of mantissa and exponent
- Overflow
  - Always occurs within the exponent
- Underflow
  - Occurs when absolute value of a negative exponent is too large to fit within allocated bits

Precision and Truncation

- Precision
  - Accuracy is reduced as the number of digits available to store mantissa is reduced
- Truncation
  - Stores numeric value in the mantissa until available bits are consumed; discards remaining bits
  - Causes an error or approximation which can magnify
  - Avoid by using integer types

Processing Complexity

- Floating point formats
  - Optimized for processing efficiency
  - Require complex processing circuitry (translates to difference in speed)
- Programmers never use real numbers when an integer will suffice (speed and accuracy)

Character Data

- Represented indirectly by defining a table that assigns numeric values to individual characters
- Characteristics of coding methods
  - All users must share same coding/decoding method
  - Coded values must be capable of being stored or transmitted
  - A coding method represents a tradeoff among compactness, range, ease of manipulation, accuracy, and standardization

Common Coding Methods

- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- ASCII (American Standard Code for Information Interchange)
  - Subset of Unicode
  - Device control
  - Software and hardware support

Partial list of ASCII and EBCDIC codes

Symbol	ASCII	EBCDIC
0	0110000	11110000
1	0110001	11110001
2	0110010	11110010
3	0110011	11110011
4	0110100	11110100
5	0110101	11110101
6	0110110	11110110
7	0110111	11110111
8	0111000	11111000
9	0111001	11111001
A	1000001	11000001
B	1000010	11000010
C	1000011	11000011
a	1100001	10000001
b	1100010	10000010
c	1100011	10000011

ASCII Limitations

- Insufficient range
  - Uses 7-bit code, providing 128 table entries (33 for device control)
  - 95 printable characters can be represented
- English-based
- Latin-1
  - Lower 128 entries ASCII-7 characters
  - Upper 128 entries multinational characters

Unicode

- Assigns nonnegative integers to represent individual printable characters (like ASCII)
- Larger coding table than ASCII
  - Uses 16-bit code providing 65,536 table entries
- Can represent written text from all modern languages
- Widely supported in modern software

Boolean Data

- Has only two data values—true and false
- Potentially most concise coding format; only a single bit is required
- To conserve memory and storage, sometimes programmers "pack" many Boolean values into a single integer

Memory Addresses

- Identifying numbers of memory bytes in primary storage
- Simple or complex numeric values depending on memory model used by CPU
  - Flat memory addresses (single integer)
  - Segmented memory addresses (multiple integers)
- •Require definition of specific coding format

Data Structures

- Related groups of primitive data elements organized for a type of common processing
- Defined and manipulated within software
- Commonly used data structures: character strings or arrays, records, and files
- Have an important role in system software development

Pointers and Addresses

- Pointer
  - Data element that contains the address of another data element
- Address
  - Location of a data element within a storage device

Arrays and Lists

- List
  - A set of related data values
- Array
  - An ordered list in which each element can be referenced by an index to its position

Arrays and Lists (continued)

- Linked list: data structures that use pointers so list elements can be scattered among
- Non-sequential storage locations
  - Singly linked lists
  - Doubly linked lists
- Easier to expand or shrink than an array

Records and Files

- Records
  - Data structures composed of other data structures or primitive data elements
  - Used as a unit of input and output to and from files or databases
- Files
  - Sequence of records on secondary storage
- Tables
  - Sequence of records stored in main memory

Methods of Organizing Files

- Sequential
  - Stores records in contiguous storage locations
- Indexed
  - An array of pointers to records
  - Efficient record insertion, deletion, and retrieval

Classes and Objects

- Classes
  - Data structures that contain traditional data elements and programs that manipulate that data
  - Programs are called methods
  - Combine related data items and extend the record to include methods that manipulate the data items
- Objects
  - One instance, or variable, of the class

Summary

- Data can be represented in many ways
- Data types are used as building blocks to create more complex data structures
- (e.g., arrays, records)
- Data representation is key to understanding hardware and software technology