Unit 1.4 Data Types, Data Structures and Algorithms

Bitwise Manipulation

Shifts
- Shifts on binary numbers are called logical shifts.
- May be a logical shift left or logical shift right.
- Move all the bits of the number a specific number of places left or right.
- Involves adding a number of zeros at the beginning or end.
- This gives a multiplication for left shifts and division for right shifts by two to the power of the number of places shifted.
- Moving one place will double or halve the number.

Masks
- Combines binary numbers with a logic gate such as AND or XOR.
- May multiply or otherwise change the involved numbers.

Karnaugh Maps

- Used to simplify Boolean expressions
- Can be used for truth tables with between two and four variables
- Values in columns and rows must be written using grey code
- Columns and rows only differ by one bit

1) Write the truth table as a Karnaugh Map
2) Highlight all the 1s
3) Only groups of 1s with edged equal to a power of 2 may be highlighted
4) Remove variables which change within the highlighting
5) Keep variables which do not change

Data Structures

Records
- A row in a file or table
- Widely used in databases
- Made up of fields

Lists
- A number of items
- Items can occur more than once
- Data can be of more than one data type

Tuples
- An ordered set of values
- Cannot be changed once initialised
- Initialised with regular rather than square brackets

Arrays
- An ordered set of elements, each of the same type.
- A 1D array is like a list.
- A 2D array is like a table.
- A 3D array is like a multi page spreadsheet.
- 2D arrays are searched first by the rows and then the columns.

Linked Lists
- Dynamic data structure.
- Stores an ordered list.
- Contents need not be in contiguous data locations.
- Items are called nodes.
- Each node contains a data field and a link or pointer field.
- The data field contains the data itself.
- The pointer field contains the address of the next item.

Graphs
- Notes connected by edges or arcs.
- Directed graphs allow edges to be traversed in one direction only.
- Undirected graphs allow edges to be traversed in both directions.
- Weighted graphs attach a cost to each arc.
- Implemented using an adjacency list or adjacency matrix.
- Adjacency matrix - easy to add nodes and to work with.
- Adjacency list - space efficient.

Trees
- Connected graphs with root and child nodes.
- A note is an item in the tree.
- An edge connects two nodes together.
- A roof is a node with no incoming nodes.
- A child is a node with incoming edges.
- A parent is a node with outgoing edges.
- A subtree is a section of a tree consisting of a parent node with child nodes.
- A leaf is a node with no child nodes.
- A binary tree is a tree where each node has two or fewer children.
- Binary trees store information in a way which is easy to search.
- They often store each node with a left and right pointer.

Positive Integers in Binary

- Each binary digit is called a bit
- Eight bits form a byte
- Four bits (half a byte) is called a nybble
- The most significant bit is furthest left
- The least significant bit is furthest right

Data Types

Integer
- A whole number
- May be positive, negative or 0
- Cannot have a fraction or decimal point
- Often used for counting objects
- e.g. 5, -1, 0, 10

Real
- Positive or negative number
- May have a decimal point
- Often used for measurements
- e.g. 5, -10, 100.556, 15.2

Character
- A single symbol
- May be a letter, number or character
- Uppercase and lowercase letters are different characters
- e.g. A, a, 5, M, ^, @

String
- A collection of characters
- Can store one or many strings
- Often used to contain text
- Leading 0s are not trimmed so useful for storing phone numbers

Boolean
- True or False only

Binary Subtraction

- Use Two's Complement.
- Use the same rules as adding a negative number.
- Use binary addition with a negative two's complement number.

Trace Tables

- A method of recording the values used within an algorithm at each stage of processing to help in troubleshooting
- Tests algorithms for logic errors which occur when the algorithm is executed.
- Simulates the steps of algorithm.
- Each stage is executed individually allowing inputs, outputs, variables, and processes to be checked for the correct value at each stage.
- A great way to spot errors

```
X = 3
Y = 1
while X > 0
    Y = Y + 1
    X = X - 1
print(Y)
```

Stage	X	Y	Output
1	3	1	
2		2	
3	2		
4		3	
5	1		
6		4	
7	0		
8			4

Normalisation

- Maximises the precision in any number of bits.
- Adjust the mantissa so that it begins with 01 for positive numbers and 10 for negative numbers.

Binary Addition

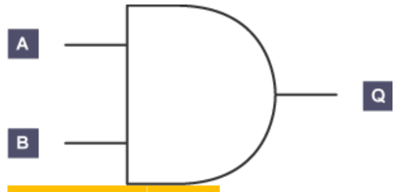
- 0 + 0 = 1
- 0 + 1 = 1
- 1 + 1 = 10
- 1 + 1 + 1 = 11

Combining and Manipulating Boolean Operations

- Boolean operators can be combined to form Boolean equations
- This follows the same way as combining standard maths operators
- The equation can be represented by a truth table
- Sometimes a long expression can share a truth table with a shorter expression
- It is better to use the shorter version.

Boolean Operators

AND - two conditions must be met for the statement to be true
Written as AND or .

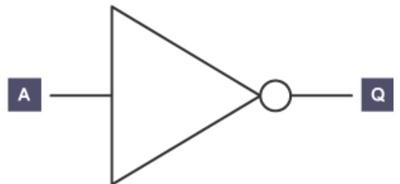


A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

OR - at least one condition must be met for the statement to be true
Written as OR or +

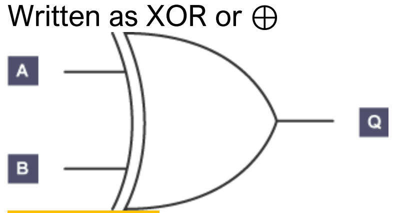
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

NOT – inverts the result, e.g. NOT(A AND B) will only be false when both A and B are true
Written as NOT or ¯



A	Q
1	0
0	1

XOR – Also know as Exclusive OR. Works the same as an OR gate, but will output 1 only if one or the other and not both inputs are 1.
Written as XOR or ⊕



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Unit 1.4 Data Types, Data Structures and Algorithms

Simplifying Boolean Algebra

De Morgan's Laws

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$

Distribution

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$$

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

$$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge (A \wedge C)$$

$$A \vee (B \vee C) \equiv (A \vee B) \vee (A \vee C)$$

Association

$$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C) \equiv A \wedge B \wedge C$$

$$(A \vee B) \vee C \equiv A \vee (B \vee C) \equiv A \vee B \vee C$$

Commutation

$$A \vee B \equiv B \vee A$$

$$A \wedge B \equiv B \wedge A$$

Double Negation

$$\neg\neg A \equiv A$$

Floating Point Numbers

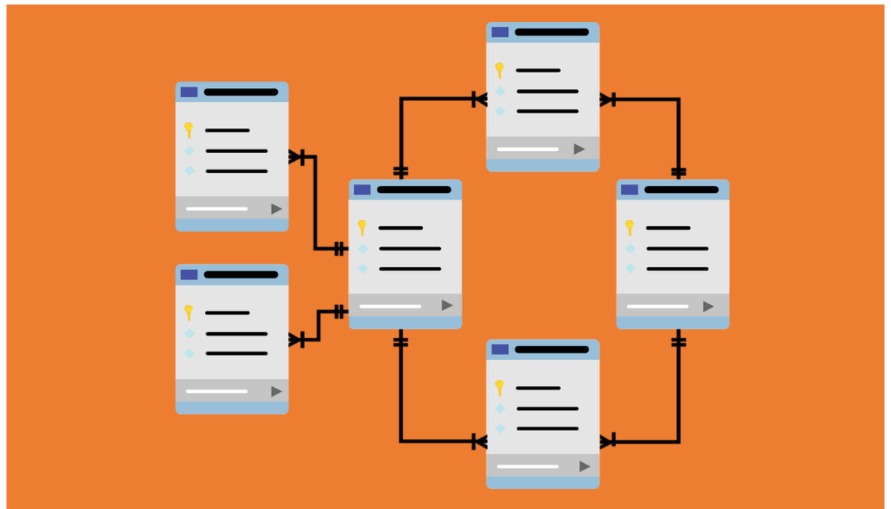
- Similar to scientific notation
- Numbers are split into Mantissa and Exponent
- The mantissa has the binary point after the most significant bit
- Then convert the exponent to decimal
- Move the binary point according to the exponent

Traversing Data Structures

Pre-order Traversal
1. Root node
2. Left subtree
3. Right subtree

In-order Traversal
1. Left subtree
2. Root node
3. Right subtree

Post-order Traversal
1. Left subtree
2. Right subtree
3. Root node



List and Queue Operations

List Operations
- isEmpty() Checks if the list is empty
- append(value) Adds a new value to the end of the list
- remove(value) Removes the value the first time it occurs
- in the list
- search(value) Searches for a value in the list.
- length() Returns the length of the list
- index(value) Returns the position of the item
- insert(position, value) Inserts a value at a given position
- pop() Returns and removes the last item in the
- list
- pop(position) Returns and removes the item at the given
- position

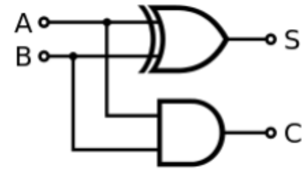
Queue Operations
- enQueue(value) Adds a new item to the end of the queue
- isEmpty() Checks if the queue if empty
- isFull() Checks if the queue is full

Logic Circuits - Adders

- Adds together the number of TRUE inputs.
- Outputs this number in binary.

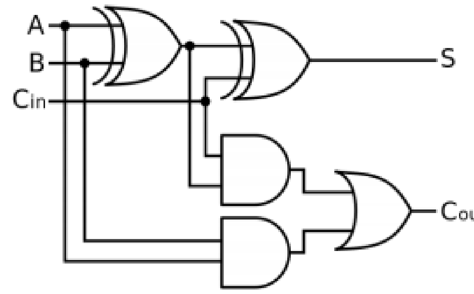
Half Adder
- Has two inputs, A and B.
- Has two outputs, SUM and CARRY.
- Has two logic gates, AND and XOR.
- When A and B are FALSE both outputs are FALSE.
- When one of A or B is true, SUM is TRUE.
- When both inputs are TRUE, CARRY is TRUE.

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Full Adder
- Like a half adder but with a third input, CARRY IN.
- Formed from two XOR gates, two AND gates and an OR gate.
- May be chained together to produce a Ripple Adder with many inputs.

A	B	C in	C out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Character Sets

- A collection of codes and their corresponding characters.

ASCII
- American standard code for information interchange
- Older character set
- Uses 7 bits representing 27 (128) characters
- Insufficient characters to represent multiple languages

Unicode
- Developed in response to ASCIIs limited characters
- Varying number of bits allows over 1 million characters
- Many characters yet to be used
- Includes different symbols and emojis

Hexadecimal

- Base 16.
- Characters 0-9 are used as usual.
- A-F are used instead of 10-15.
- Place values begin with 1 and increase in powers of 16

Converting Hexadecimal to Binary
- Convert each digit to a decimal number
- Convert these to a binary nybble
- Join the nybbles into a single binary number

Converting Hexadecimal to Decimal
- Convert to binary
- Convert the binary to decimal

Stacks and Queues

Stacks
- Last in first out
- Items can only be added or removed from the top
- Used for back or undo buttons
- Can be dynamic or static structure

Queues
- First in first out data structure
- Items are added at the beginning and removed at the end
- Used in printers and keyboards
- Linear queue with items added into the next space
- Space inefficient
- Uses pointers at the front and back
- Circular queues have a rear pointer that can loop back to the beginning to use empty space.

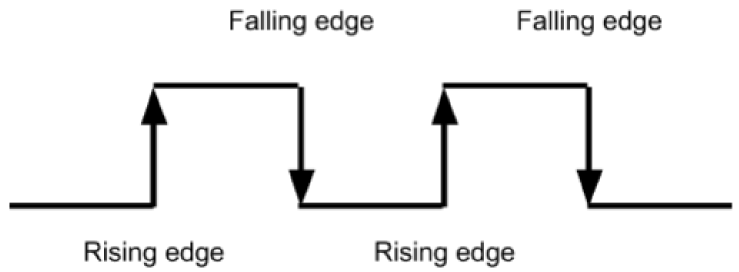
Stack and Queue Operations

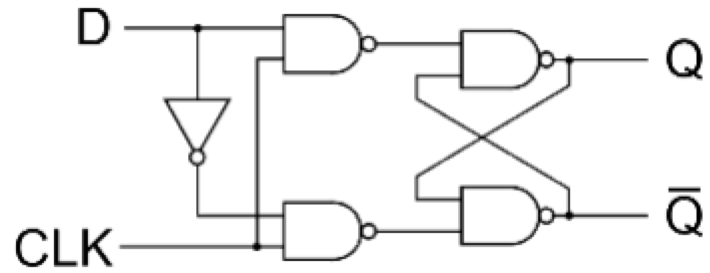
Stacks
- isEmpty() - Checks if the stack is empty
- push(value) - Adds a new value to the top of the stack
- peek() - Returns the top value of the stack
- pop() - Returns and removes the top value of the stack
- size() - Returns the size of the stack
- isFull() - Checks if the stack is full

Queues
- enQueue(value) - Adds a new item at the end of the queue
- deQueue() - Removes the item at the end of the queue
- isEmpty() - Checks if the queue if empty
- isFull() - Checks if the queue is full

Logic Circuits - D-Type Flip Flops

- Stores the value of one bit.
- Has a clock, two inputs and a control signal.
- The clock is a regular pulse from the CPU.
- The clock is used to coordinate the computer's components.
- A clock pulse has edges which either rise or fall.
- The output can only change at a rising edge.
- Used four NAND gates.
- Updates the value in Q to the value in D whenever the clock rises.
- Q is the stored value





Additional and Subtraction of Floating Point Numbers

Addition
- The exponent must be the same
- Add the mantissas
- Normalise if needed

Subtraction
- The exponents must be the same
- Covert to two's complement then add
- Use binary addition on the mantissas
- Normalise if needed