

1

Sequence and Statement

Photo-copiable digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

2

Sequence

The statement is the simplest program

It represents a single instruction and usually of code like the one shown

```
area = width * height
```

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

3

Sequence

The statement is the simplest program

It represents a single instruction and usually of code like the one shown

```
INPUT width, height
area = width * height
OUTPUT area
```

A sequence of statements (carried out in order)

This is an example of a sequence that is designed to calculate the area of a rectangle.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education


4

Variables

When necessary, labels are used to store data

A variable is a location in memory that stores a value

Think of a variable as a box



INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

5

Variables

There are three variables in this program: width, height and area. They change as the program runs.

```
INPUT width, height
area = width * height
OUTPUT area
```

Var
width
height
area
Out

The user has inputted the values 3 and 5. The width and height variables are now 3 and 5.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

6

Variables

There are three variables in this program: width, height and area. They change as the program runs.

```
INPUT width, height
area = width * height
OUTPUT area
```

Var
width
height
area
Out

The user has inputted the values 3 and 5. The width and height variables are now 3 and 5.

The value 9 is stored in the width variable; the result is stored in the height variable; the result is stored in the area variable.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

7

Variables

There are three variables in this program: width, height and area. They change as the program runs.

INPUT width, height
area = width * height
OUTPUT area

The user has inputted the values 3 and 5. The width variable is 3 and the height variable is 5.

The value 15 is stored in the width variable; the result is stored in the height variable.

The value stored in the area variable is 15.

INSPECTION COPY

COPYRIGHT
PROTECTED

8

Selection

INPUT guess
IF guess = 5 THEN
OUTPUT "Well done"
ELSE
OUTPUT "Try again"
END IF

In selection statements, the code is executed only if the condition is true.

In this example, the code is executed only if the user enters the number 5.

INSPECTION COPY

COPYRIGHT
PROTECTED

9

Selection

INPUT guess
IF guess = 5 THEN
OUTPUT "Well done"
ELSE
OUTPUT "Try again"
END IF

In selection statements, the code is executed only if the condition is true.

In this example, the code is executed only if the user enters the number 5.

This is a simple selection statement.

INSPECTION COPY

COPYRIGHT
PROTECTED

10

Selection

INPUT guess
IF guess = 5 THEN
OUTPUT "Well done"
ELSE
OUTPUT "Try again"
END IF

In selection statements, the code is executed only if the condition is true.

In this example, the code is executed only if the user enters the number 5.

This is a simple selection statement.

If the result of the condition is TRUE then the code is executed.

INSPECTION COPY

COPYRIGHT
PROTECTED

11

Selection

INPUT guess
IF guess = 5 THEN
OUTPUT "Well done"
ELSE
OUTPUT "Try again"
END IF

In selection statements, the code is executed only if the condition is true.

In this example, the code is executed only if the user enters the number 5.

This is a simple selection statement.

If the result of the condition is TRUE then the code is executed.

Otherwise this line of code will not be executed.

INSPECTION COPY

COPYRIGHT
PROTECTED

12

Selection

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

In selection statements, the code is executed only if the condition is true.

In this example, the code is executed only if the user enters the number 5.

This is a simple selection statement.

If the result of the condition is TRUE then the code is executed.

Otherwise this line of code will not be executed.

These are the different comparison operators and their meanings.

INSPECTION COPY

COPYRIGHT
PROTECTED

1

Iteration

When programming, it is often necessary to repeat instructions several times.

Iteration allows us to do this by repeating a number of times or until a condition is met.

Photo copied from digital resources may only be copied by the purchasing institution in a single site and for their own use.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

2

Iteration

When programming, it is often necessary to repeat instructions several times.

Iteration allows us to do this by repeating a number of times or until a condition is met.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

3

Types of Iteration

There are two types of iteration:

- Count-controlled**
 - Used to repeat a group of statements a set number of times.
- Condition-controlled**
 - Used to repeat a group of statements until a condition is met.

Examples:

```
FOR i = 1 TO 3  
  OUTPUT i * i  
NEXT i
```

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

4

FOR Loop

A FOR loop uses a variable as a counter; the number of times statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed:

```
FOR i = 1 TO 3  
  OUTPUT i * i  
NEXT i
```

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

5

FOR Loop

A FOR loop uses a variable as a counter; the number of times statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed:

```
FOR i = 1 TO 3  
  OUTPUT i * i  
NEXT i
```

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

6

FOR Loop

A FOR loop uses a variable as a counter; the number of times statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed:

```
FOR i = 1 TO 3  
  OUTPUT i * i  
NEXT i
```

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

7

FOR Loop

A FOR loop uses a variable as a counter; the statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed

```
FOR i = 1 to 3
  OUTPUT i*i
NEXT i
```

INSPECTION COPY

COPYRIGHT PROTECTED



8

FOR Loop

A FOR loop uses a variable as a counter; the statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed

```
FOR i = 1 to 3
  OUTPUT i*i
NEXT i
```

INSPECTION COPY

COPYRIGHT PROTECTED



9

FOR Loop

A FOR loop uses a variable as a counter; the statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed

```
FOR i = 1 to 3
  OUTPUT i*i
NEXT i
```

INSPECTION COPY

COPYRIGHT PROTECTED



10

WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
  OUTPUT i*i
  i = i + 1
WEND
```

INSPECTION COPY

COPYRIGHT PROTECTED



11

WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
  OUTPUT i*i
  i = i + 1
WEND
```

INSPECTION COPY

COPYRIGHT PROTECTED



12

WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
  OUTPUT i*i
  i = i + 1
WEND
```

INSPECTION COPY

COPYRIGHT PROTECTED



13

WHILE Loop

A WHILE loop tests a condition to determine if a statement should be executed. The statement is executed as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
  OUTPUT i*4
  i = i + 1
END WHILE
```



INSPECTION COPY

COPYRIGHT
PROTECTED

14

WHILE Loop

A WHILE loop tests a condition to determine if a statement should be executed. The statement is executed as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
  OUTPUT i*4
  i = i + 1
END WHILE
```



INSPECTION COPY

COPYRIGHT
PROTECTED

15

WHILE Loop

A WHILE loop tests a condition to determine if a statement should be executed. The statement is executed as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
  OUTPUT i*4
  i = i + 1
END WHILE
```



INSPECTION COPY

COPYRIGHT
PROTECTED

16

WHILE Loop

A WHILE loop tests a condition to determine if a statement should be executed. The statement is executed as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
  OUTPUT i*4
  i = i + 1
END WHILE
```



INSPECTION COPY

COPYRIGHT
PROTECTED

17

REPEAT UNTIL

In a REPEAT UNTIL loop, the statements are executed first, then the condition is tested. The statements are repeated as long as the condition is false.

Watch what happens to the output and the code below is executed

```
i = 1
REPEAT
  OUTPUT i*4
  i = i + 1
UNTIL i = 4
```



INSPECTION COPY

COPYRIGHT
PROTECTED

18

REPEAT UNTIL

In a REPEAT UNTIL loop, the statements are executed first, then the condition is tested. The statements are repeated as long as the condition is false.

Watch what happens to the output and the code below is executed

```
i = 1
REPEAT
  OUTPUT i*4
  i = i + 1
UNTIL i = 4
```



INSPECTION COPY

COPYRIGHT
PROTECTED

19

REPEAT UNTIL

In a REPEAT UNTIL loop, the statements are executed until the condition is false.

Watch what happens to the output and the below is executed

i = 1

REPEAT

OUTPUT i*i

i = i + 1

UNTIL i = 4

INSPECTION COPY

COPYRIGHT

PROTECTED



20

REPEAT UNTIL

In a REPEAT UNTIL loop, the statements are executed until the condition is false.

Watch what happens to the output and the below is executed

i = 1

REPEAT

OUTPUT i*i

i = i + 1

UNTIL i = 4

INSPECTION COPY

COPYRIGHT

PROTECTED



21

REPEAT UNTIL

In a REPEAT UNTIL loop, the statements are executed until the condition is false.

Watch what happens to the output and the below is executed

i = 1

REPEAT

OUTPUT i*i

i = i + 1

UNTIL i = 4

INSPECTION COPY

COPYRIGHT

PROTECTED



22

REPEAT UNTIL

In a REPEAT UNTIL loop, the statements are executed until the condition is false.

Watch what happens to the output and the below is executed

i = 1

REPEAT

OUTPUT i*i

i = i + 1

UNTIL i = 4

INSPECTION COPY

COPYRIGHT

PROTECTED



23

REPEAT UNTIL

In a REPEAT UNTIL loop, the statements are executed until the condition is false.

Watch what happens to the output and the below is executed

i = 1

REPEAT

OUTPUT i*i

i = i + 1

UNTIL i = 4

INSPECTION COPY

COPYRIGHT

PROTECTED



1

Subroutine

INSPECTION COPY

COPYRIGHT

PROTECTED



Printed and digital resources may only be copied by the purchasing institution on a single site and for their own use.

2

Subroutine

When programming, we often need to perform the same task many times, at different points in the program; this is where subroutines are handy.

A subroutine is a section of code that performs a specific task and can be called from the main routine.

Program

Subroutine 1

Subroutine 2

COPYRIGHT

PROTECTED



3

Types of Subroutine

There are two types of subroutines:

Procedure

A subroutine that does not normally return a value to the main program.

A subroutine that returns a value to the main program.

COPYRIGHT

PROTECTED



4

Parameter

When writing a subroutine you need to tell the computer what data it needs to work with.

This is usually done by adding variables to the subroutine's definition. These variables are called parameters.

```
PROCEDURE Compare(a, b)
  IF a > b THEN
    OUTPUT a
  ELSE
    OUTPUT b
  END IF
END PROCEDURE
```

COPYRIGHT

PROTECTED



5

Argument

Subroutines work with data, unless they are called without data. When data is passed to a subroutine when it is called, it is known as an argument.

```
PROCEDURE Compare(a, b)
  IF a > b THEN
    OUTPUT a
  ELSE
    OUTPUT b
  END IF
END PROCEDURE

Compare(4, 5)
```

COPYRIGHT

PROTECTED



6

Example

The procedure shown below accepts two values and outputs the largest. It is called with the values 4 and 5.

```
PROCEDURE Compare(a, b)
  IF a > b THEN
    OUTPUT a
  ELSE
    OUTPUT b
  END IF
END PROCEDURE

Compare(4, 5)
```

COPYRIGHT

PROTECTED



7

Example

The procedure shown below accepts two values and outputs the largest. It is called with the values 4 and 5.

```
PROCEDURE Compare(a, b)
  IF a > b THEN
    OUTPUT a
  ELSE
    OUTPUT b
  END IF
END PROCEDURE

Compare(4, 5)
```

COPYRIGHT

PROTECTED



8

Functions

Remember a function is a subroutine that returns a value.

The AreaCalc function is designed to calculate the area of a rectangle and return it to the calling routine where it is used.

```
FUNCTION AreaCalc(w, h)
    Area = w * h
    RETURN Area
END FUNCTION
```

```
Result = AreaCalc(5, 3)
OUTPUT Result
```

INSPECTION COPY

COPYRIGHT
PROTECTED

9

Functions

Remember a function is a subroutine that returns a value.

The AreaCalc function is designed to calculate the area of a rectangle and return it to the calling routine where it is used.

```
FUNCTION AreaCalc(w, h)
    Area = w * h
    RETURN Area
END FUNCTION
```

```
Result = AreaCalc(5, 3)
OUTPUT Result
```

INSPECTION COPY

COPYRIGHT
PROTECTED

10

Functions

Remember a function is a subroutine that returns a value.

The AreaCalc function is designed to calculate the area of a rectangle and return it to the calling routine where it is used.

```
FUNCTION AreaCalc(w, h)
    Area = w * h
    RETURN Area
END FUNCTION
```

```
Result = AreaCalc(5, 3)
OUTPUT Result
```

INSPECTION COPY

COPYRIGHT
PROTECTED

11

Functions

Remember a function is a subroutine that returns a value.

The AreaCalc function is designed to calculate the area of a rectangle and return it to the calling routine where it is used.

```
FUNCTION AreaCalc(w, h)
    Area = w * h
    RETURN Area
END FUNCTION
```

```
Result = AreaCalc(5, 3)
OUTPUT Result
```

INSPECTION COPY

COPYRIGHT
PROTECTED

12

Variable Scope

There are two types of variables: Global and Local.

Global variables are accessible throughout the entire program, including inside subroutines.

Local variables are only accessible from within the subroutine where they were defined.

When a subroutine is called, a new instance of local variables is created. When the subroutine ends, these variables are then destroyed.

INSPECTION COPY

COPYRIGHT
PROTECTED

1

Recursion

INSPECTION COPY

COPYRIGHT
PROTECTED

Photocopying this resource may only be done by the purchasing institution on a single site and for their own use.

2

Recursion



Recursion occurs when a subrou

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

3

Example

The recursive procedure below counts down
passed to it.

You will now see the result of calling the co
countdown(2).

```

1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE

```

countdown(2)
2 (FALSE)
4
5 OUTPUT
6 countdown(1)
7
8

COPYRIGHT
PROTECTED

Zig
Zag
Education

INSPECTION COPY

4

Trace Table

You need to be able to trace the execution of
a trace table.

In this example, the countdown procedure
countdown(3):

```

1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE

```

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

5

Trace Table

You need to be able to trace the execution of
a trace table.

In this example, the countdown procedure
countdown(3):

```

1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE

```

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

6

Trace Table

You need to be able to trace the execution of
a trace table.

In this example, the countdown procedure
countdown(3):

```

1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE

```

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

7

Trace Table

You need to be able to trace the execution of
a trace table.

In this example, the countdown procedure
countdown(3):

```

1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE

```

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

Trace Table

You need to be able to trace the execution of a trace table.

In this example, the countdown procedure countdown(3):

```

1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT
6     countdown(n-1)
7   END IF
8 END PROCEDURE

```

INSPECTION COPY

COPYRIGHT PROTECTED



Trace Table

You need to be able to trace the execution of a trace table.

In this example, the countdown procedure countdown(3):

```

1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT
6     countdown(n-1)
7   END IF
8 END PROCEDURE

```

INSPECTION COPY

COPYRIGHT PROTECTED



Infinite Recursion

If a subroutine calls itself, it is possible to

means the subroutine will keep running out of available memory

```

1 PROCEDURE countdown
2   IF n <= 1 THEN
3     OUTPUT
4   ELSE
5     OUTPUT
6     countdown
7   END IF
8 END PROCEDURE

```

To avoid this you need a stopping condition

INSPECTION COPY

COPYRIGHT PROTECTED



Arrays

INSPECTION COPY

COPYRIGHT PROTECTED



Please do not distribute this resource. It may only be copied by the purchasing institution on a single site and for their own use.

Arrays

A variable can only store one value; if you need to store more than one value you will need to use an array.

An array is a set of values of the same data type, identified by a single identifier.

Here is an example array designed to store names:

Index	0	1
Value	Alex	James

INSPECTION COPY

COPYRIGHT PROTECTED



Index

We access the values stored in an array using an index.

Index	0	1
Value	Alex	James

If this array was called 'names' we could access this value using: names[1]

INSPECTION COPY

COPYRIGHT PROTECTED



Two-Dimensional

Arrays can also be two-dimensional, allowing values that are basically arrays of arrays.

Here is an example two-dimensional array used in a computer game.

		Columns (Levels)		
		0	1	2
Rows (Players)	0	4		3
	1		5	3
	2	2	4	3
	3	5	3	4

COPYRIGHT PROTECTED



Pseudocode

We can create the one-dimensional names array using the following syntax:

```
names ← ["Alex", "James", "Pat"]
```

We can create the two-dimensional scores array using the following syntax:

```
scores ← [[4, 4], [3, 5, 3, 4], [2, 5, 3, 4]]
```

COPYRIGHT PROTECTED



FOR Loops and

FOR loops and arrays go well together; they allow us to cycle through each element in an array.

This example program cycles through each element in the names array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Pat"]
```

COPYRIGHT PROTECTED



FOR Loops and

FOR loops and arrays go well together; they allow us to cycle through each element in an array.

This example program cycles through each element in the names array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Pat"]
```

COPYRIGHT PROTECTED



FOR Loops and

FOR loops and arrays go well together; they allow us to cycle through each element in an array.

This example program cycles through each element in the names array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Pat"]
```

COPYRIGHT PROTECTED



FOR Loops and

FOR loops and arrays go well together; they allow us to cycle through each element in an array.

This example program cycles through each element in the names array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Pat"]
```

COPYRIGHT PROTECTED



10

FOR Loops and

FOR loops and arrays go well together; they allow you to cycle through each element in an array.

This example program cycles through each array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Pa

INSPECTION COPY

COPYRIGHT
PROTECTED

11

FOR Loops and

FOR loops and arrays go well together; they allow you to cycle through each element in an array.

This example program cycles through each array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Pa

INSPECTION COPY

COPYRIGHT
PROTECTED

12

FOR Loops and

FOR loops and arrays go well together; they allow you to cycle through each element in an array.

This example program cycles through each array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Pa

INSPECTION COPY

COPYRIGHT
PROTECTED

13

FOR Loops and

FOR loops and arrays go well together; they allow you to cycle through each element in an array.

This example program cycles through each array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Pa

INSPECTION COPY

COPYRIGHT
PROTECTED

14

FOR Loops and

FOR loops and arrays go well together; they allow you to cycle through each element in an array.

This example program cycles through each array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Pa

INSPECTION COPY

COPYRIGHT
PROTECTED

15

FOR Loops and

FOR loops and arrays go well together; they allow you to cycle through each element in an array.

This example program cycles through each array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Pa

INSPECTION COPY

COPYRIGHT
PROTECTED

FOR Loops and Arrays

FOR loops and arrays go well together; they allow you to cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Patricia"]

INSPECTION COPY

COPYRIGHT PROTECTED



FOR Loops and Arrays

FOR loops and arrays go well together; they allow you to cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Patricia"]

INSPECTION COPY

COPYRIGHT PROTECTED



Starting Index

In most programming languages the index of the first element in an array is 0.

The pseudocode featured in most exams uses the index of the first element.

Index	1	2
Value	Alex	James

INSPECTION COPY

COPYRIGHT PROTECTED



Linked Lists and Hash Tables

A Level Only

Please do not copy or reproduce this resource in any form without the permission of Zig Zag Education. This resource may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT PROTECTED



Arrays

Arrays allow us to store a series of values. Each element has an index and can be accessed using that index.

When an array is created, its size is declared. This is a section of memory; this makes it easy to manage.

Linked lists consist of **nodes** and each node contains a pointer to the next memory location, making them easy to traverse.

INSPECTION COPY

COPYRIGHT PROTECTED



Linked Lists

Each node in a linked list contains a pointer to the next node in the list.



The last node in the list is indicated by a null pointer.

INSPECTION COPY

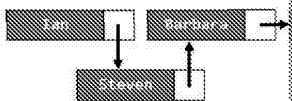
COPYRIGHT PROTECTED



4

Linked List

Each node in a linked list contains a pointer to the next node in the list.



The last node in the list is indicated.

New nodes can easily be added or removed.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig
Zag
Education

5

Disadvantages

A disadvantage of linked lists is that elements cannot be accessed directly. The entire list has to be traversed.

Hash tables offer a solution to this problem, enabling direct access to elements.

Hash tables are used when speedy insertion and deletion are required.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig
Zag
Education

6

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT PROTECTED

Zig
Zag
Education

7

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT PROTECTED

Zig
Zag
Education

8

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT PROTECTED

Zig
Zag
Education

9

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT PROTECTED

Zig
Zag
Education

10

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



COPYRIGHT

PROTECTED



11

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



COPYRIGHT

PROTECTED



12

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



COPYRIGHT

PROTECTED



13

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



COPYRIGHT

PROTECTED



14

Hash Function

This is the pseudocode for a simple hash function that takes the first letter of a key and converts it to a hash value.

The letter is converted to its ASCII value (for example, 'A' is 65) and then a constant value is subtracted from it to produce the hash value.

In this example the key is 'Carol'.

```

FUNCTION hashFunction(key)
    hash = ASCII(key[0]) - 65
    RETURN hash
  
```

COPYRIGHT

PROTECTED



15

Hash Function

This is the pseudocode for a simple hash function that takes the first letter of a key and converts it to a hash value.

The letter is converted to its ASCII value (for example, 'A' is 65) and then a constant value is subtracted from it to produce the hash value.

In this example the key is 'Carol'.

```

FUNCTION hashFunction(key)
    hash = ASCII(key[0]) - 65
    RETURN hash
  
```

COPYRIGHT

PROTECTED



16

Hash Function

This is the pseudocode for a simple hash function. It takes a letter of a key and converts it to a hash value.

The letter is converted to its ASCII value (for example, 'A' is 65) and then subtracted from it.

In this example the key is 'A'.

```
FUNCTION hashFunction(key)
    hash = ASCII(key[0]) - 65
    RETURN hash
END FUNCTION
```

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

17

Collisions

If two keys produce the same value, a collision has occurred. For example, Alex and Andy would both have the same hash value.



There are two methods of dealing with collisions: linear probing and separate chaining.

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

18

Linear Probing

The linear probing method stores the value at the next available position.



The function 'probes' the list until it finds an empty slot.

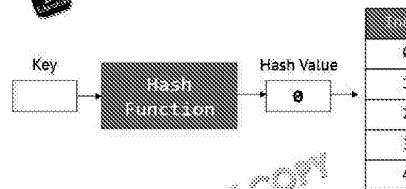
INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

19

Separate Chaining

The separate chaining method uses lists to store values that hash to the same position.



Values that hash to the same position are placed in the list at that position.

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

1

Graphs and

A Level
Only

INSPECTION COPY

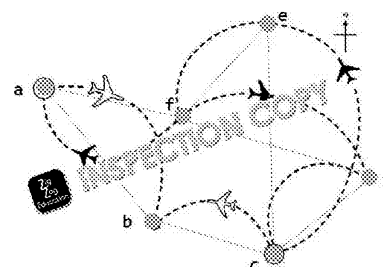
COPYRIGHT
PROTECTEDZig
Zag
Education

2

Graphs

A graph is a data structure used to represent connections between objects.

Example: airline flight paths between cities.



INSPECTION COPY

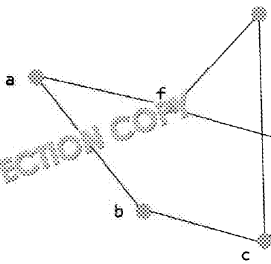
COPYRIGHT
PROTECTEDZig
Zag
Education

3

Structure

Each node in a graph is called a vertex
are called vertices

Each connection in a graph is called an edge

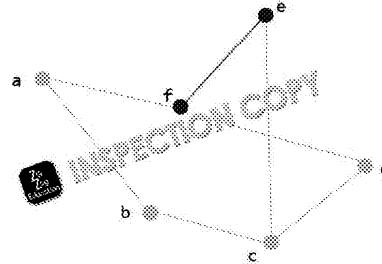


INSPECTION COPY

4

Neighbours and Neighbourhood

Neighbours are vertices connected by an edge
for example, E and F are neighbours



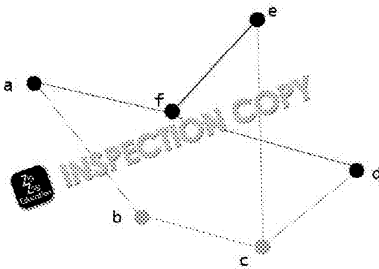
INSPECTION COPY

5

Neighbours and Neighbourhood

Neighbours are vertices connected by an edge
for example, E and F are neighbours

The degree of a vertex is the number of edges connected to it
for example, the degree of vertex a is 3



INSPECTION COPY

6

Weighted and Directed Graphs

A weighted graph is one in which the edges are labelled, with distances for example.

In a directed graph each edge has a direction associated with it.



INSPECTION COPY

7

Adjacency Matrix

An adjacency matrix is used to represent a graph
processed by a computer

If two vertices are connected, a 1 is placed in the matrix in two positions.

For example, A and D are connected so a 1 is placed in the two positions in the matrix where the vertices meet.

The remaining spaces are then filled with 0s or the null symbol (∅).

If graphs are weighted, the weight is placed in the matrix.

The adjacency matrix for an undirected graph is always symmetrical.

INSPECTION COPY

8

Adjacency List

An adjacency list is an alternative method of representing a graph

For each vertex it lists the vertices that it is connected to

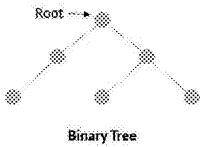
Vertex	Adjacent Vertices
A	D, E
B	A, D, C
C	B, D, E
D	A, B, C

Adjacency lists are best used when there are many vertices.

INSPECTION COPY

9 Trees

A tree is a graph that takes on the form of a graph with no cycles (only one path between



A rooted tree has a vertex that is designated

A binary tree is a rooted tree with a maximum

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Vectors

A Level
Only

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

2 Types of Vec

There are three different ways of

Data structures

Functions

Geometric points in space

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

3 Data Structure

The vector data structure is similar to an array, a vector can be accessed

The main difference is that arrays store data in a single row, whereas vectors can store data in multiple rows

[5.5, 3.0, 7.1, 6.2]

Vectors can also be represented as dictionaries

{5.5, 1:3.0, 2:7.1, 3:6.2}

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

4 Functions

Functions can also be represented

$f : S \rightarrow R$

f represents the function

S represents the values that can be used

R represents the outputs of the function

INSPECTION COPY

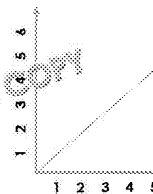
COPYRIGHT
PROTECTED
Zig
Zag
Education

5 Geometric Points

Geometric points can also represent positions

The position of the arrowhead can be represented by its coordinates, assuming the origin of the axes is (0, 0)

(6, 5)



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

6

Vector Addition

Vector addition can be used to translate

1st vector

a

a + b

Resultant from adding the two vectors

In this example, the vector that is added is vector **a** to the head of vector **b**

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

7

Scalar Multiplication

Scalar multiplication is used to

This is achieved by multiplying the scalar

The scalar is simply a real number that is

a

For example, if the scalar is **0.5** the resultant vector is

b

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

8

Convex Combination

A convex combination is a method of combining two vectors to create a third.

The new vector must be placed inside the line segment between the existing vectors.

$$t = \alpha r + \beta s$$

The existing vectors will be multiplied by the values α and β . There could be scalar values for another vector.

r

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

9

Dot Product

The dot product is a number calculated by multiplying two vectors

This formula is used to calculate

$$a \cdot b = a_x b_x + a_y b_y$$

a

a = (7, 5)

b

The dot product would be: **(7 * 8)**

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

1

Graph and Tree

A Level
Only

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

2

Tree Traversal

Traversal is the process of searching a tree structure. There are two main types of traversal:

Depth-first Traversal

Starting at the root, each node in one branch is visited before exploring the next branch.

Starting at the root, each node is visited before exploring the next branch.



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

3

Pre-Order Traversal

One type of depth-first traversal is

Steps:

1. Start at the root node.
2. Explore the left sub-tree, working top-down.
3. Explore the right sub-tree, working top-down.

1 2 4 5 3

uses include copying a tree and creating an expression tree.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

4

Post-Order Traversal

One type of depth-first traversal is

Steps:

1. Explore the left sub-tree, working bottom-up.
2. Explore the right sub-tree, working bottom-up.
3. Return to the root node.

4 2 5 1 3

Used for a binary search

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

5

In-Order Traversal

Another type of traversal is in-order

Steps:

1. Explore the left sub-tree, working bottom-up.
2. Visit the root node.
3. Explore the right sub-tree, working bottom-up.

4 2 1 5 3

uses include infix to Reverse Polish notation and converting an expression tree to an expression.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

6

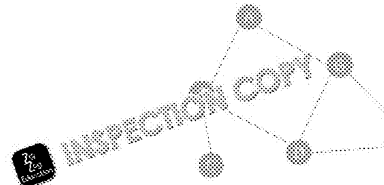
Graph Traversal

Like trees, there are two main types

Depth-first Traversal

This algorithm uses a stack.

This



INSPECTION COPY

COPYRIGHT PROTECTED

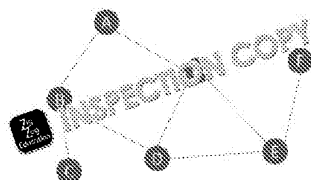
Zig Zag Education

7

Depth-First Traversal

In depth-first traversal each vertex is marked when it is visited and added to the stack.

When there are no remaining unvisited neighbours, vertices are popped off the stack until one is reached that has unvisited neighbours.



INSPECTION COPY

COPYRIGHT PROTECTED

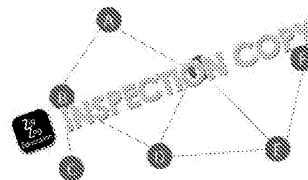
Zig Zag Education

8

Breadth-First Traversal

In breadth-first traversal each vertex is marked when it is visited and added to the queue.

When there are no remaining unvisited neighbours for the current vertex, it moves to the next vertex in the queue.



Queue

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

1

Reverse Polish

A Level Only

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

2

Infix Notation

As humans learn performing calculations we require a sequence. For example, the brackets are used to indicate the order of operations.

$$(10-5) * (5-2)$$

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

3

Infix Notation

As humans learn performing calculations we require a sequence. For example, the brackets are used to indicate the order of operations.

$$5 * 9$$

This is known as infix notation.

CPU's evaluate expressions using the stack which is compatible with infix notation.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

4

Reverse Polish Notation

Reverse Polish notation removes the need for brackets after the numbers. This makes the evaluation using a stack.

The CPU finds the first operator and applies it to the numbers in the expression.

$$(10-5) * (5-2)$$

Becomes:

$$10 5 - 5 2 *$$

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

5

Reverse Polish Notation

Reverse Polish notation removes the need for brackets after the numbers. This makes the evaluation using a stack.

The CPU finds the first operator and applies it to the numbers in the expression.

$$(10-5) * (5-2)$$

Becomes:

$$5 9 * =$$

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

6

Another Example

Here is another example of an expression being converted from Polish notation to infix notation.

$$20 6 4$$

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

7

Another Exam

Here is another example of an expression being converted from Polish notation to infix notation.

$$20 * (6 + 4)$$

INSPECTION COPY

COPYRIGHT
PROTECTED

8

Stacks

Stacks can be used to solve problems.

20 6 4

INSPECTION COPY

COPYRIGHT
PROTECTED

9

Stacks

Stacks can be used to solve problems.

20 6 4

Each time a number is encountered it is pushed onto the stack.

INSPECTION COPY

COPYRIGHT
PROTECTED

10

Stacks

Stacks can be used to solve problems.

20 6 4

Each time a number is encountered it is pushed onto the stack.

When an operator is encountered it is applied to the top two numbers on the stack.

The two numbers are then popped from the stack and the result is pushed onto it.

INSPECTION COPY

COPYRIGHT
PROTECTED

11

Stacks

Stacks can be used to solve problems.

20 6 4

Each time a number is encountered it is pushed onto the stack.

When an operator is encountered it is applied to the top two numbers on the stack.

The two numbers are then popped from the stack and the result is pushed onto it.

INSPECTION COPY

COPYRIGHT
PROTECTED

1

Searching Algorithms

A Level Only

INSPECTION COPY

COPYRIGHT
PROTECTED

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use.

2 Linear Search

Linear search is the simplest searching algorithm. It starts from the first element in the list and checks every element one by one until it finds the element it is looking for. In this example we are looking for the number 5.

3	8	1	4	7	6	9
---	---	---	---	---	---	---

```
1 FUNCTION LSearch(List, ItemToFind)
2   FOR i = 0 to length(List)
3     IF List[i] == ItemToFind THEN
4       RETURN i
5   END IF
6 NEXT i
7 RETURN -1
8 END FUNCTION
```

The problem with this method is that it is slow, especially for large lists.

INSPECTION COPY

COPYRIGHT
PROTECTED



3 Binary Search

Binary search is a more efficient way of searching lists. It works by repeatedly dividing the list in half until the element is found. In this example we are going to search for the number 5.

0	1	2	4	5	6	8
---	---	---	---	---	---	---

```
1 FUNCTION BSearch(List, ItemToFind, Min, Max)
2   MidPoint = FindMidPoint(Min, Max)
3   IF List[MidPoint] < ItemToFind THEN
4     BSearch(List, ItemToFind, MidPoint+1, Max)
5   ELSE IF List[MidPoint] > ItemToFind THEN
6     BSearch(List, ItemToFind, Min, MidPoint-1)
7   ELSE
8     RETURN MidPoint
9   END IF
9 END FUNCTION
```

COPYRIGHT
PROTECTED



INSPECTION COPY

4 Binary Search

Binary search is a more efficient way of searching lists. It works by repeatedly dividing the list in half until the element is found. In this example we are going to search for the number 8.

8

```
1 FUNCTION BSearch(List, ItemToFind, Min, Max)
2   MidPoint = FindMidPoint(Min, Max)
3   IF List[MidPoint] < ItemToFind THEN
4     BSearch(List, ItemToFind, MidPoint+1, Max)
5   ELSE IF List[MidPoint] > ItemToFind THEN
6     BSearch(List, ItemToFind, Min, MidPoint-1)
7   ELSE
8     RETURN MidPoint
9   END IF
9 END FUNCTION
```

INSPECTION COPY

COPYRIGHT
PROTECTED



5 Binary Search

Binary search is a more efficient way of searching lists. It works by repeatedly dividing the list in half until the element is found. In this example we are going to search for the number 8.

```
1 FUNCTION BSearch(List, ItemToFind, Min, Max)
2   MidPoint = FindMidPoint(Min, Max)
3   IF List[MidPoint] < ItemToFind THEN
4     BSearch(List, ItemToFind, MidPoint+1, Max)
5   ELSE IF List[MidPoint] > ItemToFind THEN
6     BSearch(List, ItemToFind, Min, MidPoint-1)
7   ELSE
8     RETURN MidPoint
9   END IF
9 END FUNCTION
```

COPYRIGHT
PROTECTED



INSPECTION COPY

6 Binary Search

Binary search is a more efficient way of searching lists. It works by repeatedly dividing the list in half until the element is found. In this example we are going to search for the number 8.

```
1 FUNCTION BSearch(List, ItemToFind, Min, Max)
2   MidPoint = FindMidPoint(Min, Max)
3   IF List[MidPoint] < ItemToFind THEN
4     Bsearch(List, ItemToFind, MidPoint+1, Max)
5   ELSE IF List[MidPoint] > ItemToFind THEN
6     Bsearch(List, ItemToFind, Min, MidPoint-1)
7   ELSE
8     RETURN MidPoint
9   END IF
9 END FUNCTION
```

INSPECTION COPY

COPYRIGHT
PROTECTED



7 Binary Search

The drawback of the binary search method is that the list must be sorted. Even if the list is already sorted, it will be slow if a new element is added to the list.

The binary search tree can be used to get around this problem. Elements can easily be added or removed from the tree.

Rules

The left subtree of a node contains values that are less than the value of the node.

The right subtree of a node contains values that are more than the value of the node.

7

COPYRIGHT
PROTECTED

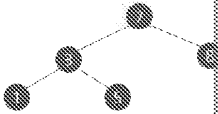


INSPECTION COPY

8

Using Binary Search

It is easy to find the highest and lowest value in a binary tree.



Lowest Value: 1

Searching for a particular value in a binary tree. At each node we simply move either right or left depending on whether its value is higher or lower than the value we are searching for. Let's search for the value 5.

INSPECTION COPY

COPYRIGHT PROTECTED



1

Sorting Algorithms

A Level Only

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT PROTECTED



2

Bubble Sort

The bubble sort algorithm works through a list of numbers and swapping them if necessary.

3 8 1 4 7 6

```

1 swap 8 and 1
2 WHILE loop
3 swap 8 and 4
4 FOR loop
5
6
7
8
9
10
11 NE
12 END
  
```

Variable	Value
swapActive	false
i	0
temp	

Bubble sort is simple to implement but inefficient.

INSPECTION COPY

COPYRIGHT PROTECTED



3

Bubble Sort

The bubble sort algorithm works through a list of numbers and swapping them if necessary.

3 1 8 4 7 6

```

1 swap 8 and 1
2 WHILE loop
3 swap 8 and 4
4 FOR loop
5
6
7
8
9
10
11 NE
12 END
  
```

Variable	Value
swapActive	true
i	1
temp	

Bubble sort is simple to implement but inefficient.

INSPECTION COPY

COPYRIGHT PROTECTED



4

Bubble Sort

The bubble sort algorithm works through a list of numbers and swapping them if necessary.

3 1 4 8 7 6

```

1 swap 8 and 1
2 WHILE loop
3 swap 8 and 4
4 FOR loop
5
6
7
8
9
10
11 NE
12 END
  
```

Variable	Value
swapActive	true
i	2
temp	

Bubble sort is simple to implement but inefficient.

INSPECTION COPY

COPYRIGHT PROTECTED



5

Bubble Sort

The bubble sort algorithm works through a list of numbers and swapping them if necessary.

3 1 4 7 8 6

```

1 swap 8 and 1
2 WHILE loop
3 swap 8 and 4
4 FOR loop
5
6
7
8
9
10
11 NE
12 END
  
```

Variable	Value
swapActive	true
i	3
temp	

Bubble sort is simple to implement but inefficient.

INSPECTION COPY

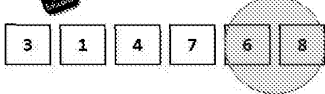
COPYRIGHT PROTECTED



6

Bubble Sort

The bubble sort algorithm works through a list and swapping them if necessary.



```

1 swap
2 WHILE
3 swap
4 FOR
5
6
7
8
9
10
11 NE
12 END

```

Variable	Value
swapActive	true
i	4
temp	

Bubble sort is simple to implement.

INSPECTION COPY

COPYRIGHT

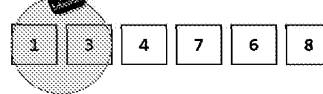
PROTECTED



7

Bubble Sort

The bubble sort algorithm works through a list and swapping them if necessary.



```

1 swap
2 WHILE
3 swap
4 FOR
5
6
7
8
9
10
11 NE
12 END

```

Variable	Value
swapActive	true
i	0
temp	

Bubble sort is simple to implement.

INSPECTION COPY

COPYRIGHT

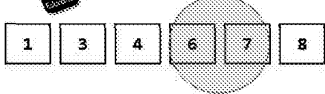
PROTECTED



8

Bubble Sort

The bubble sort algorithm works through a list and swapping them if necessary.



```

1 swap
2 WHILE
3 swap
4 FOR
5
6
7
8
9
10
11 NE
12 END

```

Variable	Value
swapActive	true
i	3
temp	

Bubble sort is simple to implement.

INSPECTION COPY

COPYRIGHT

PROTECTED



9

Merge Sort

The merge sort algorithm works by splitting elements and then gradually merging them until they are all in one sorted list.



INSPECTION COPY

INSPECTION COPY

COPYRIGHT

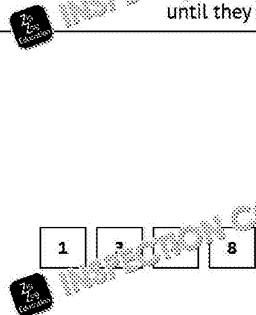
PROTECTED



10

Merge Sort

The merge sort algorithm works by splitting elements and then gradually merging them until they are all in one sorted list.



INSPECTION COPY

COPYRIGHT

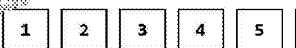
PROTECTED



11

Merge Sort

The merge sort algorithm works by splitting elements and then gradually merging them until they are all in one sorted list.



INSPECTION COPY

COPYRIGHT

PROTECTED



INSPECTION COPY

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

INSPECTION COPY

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

INSPECTION COPY

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

INSPECTION COPY

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

INSPECTION COPY

COPYRIGHT
PROTECTED

**Z^{ig}
Z^{ag}**
Education

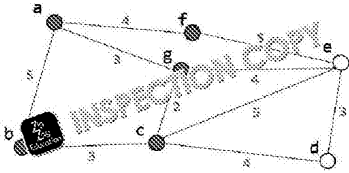
7

Worked Exam

We start by finding the distance from the starting vertex to its neighbours.

We repeat the process with the vertex furthest from the starting vertex.

The distance is only updated if a shorter route is found.



INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

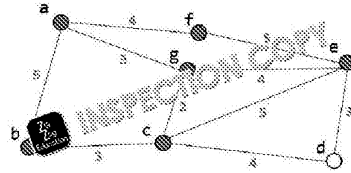
8

Worked Exam

We start by finding the distance from the starting vertex to its neighbours.

We repeat the process with the vertex furthest from the starting vertex.

The distance is only updated if a shorter route is found.



INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

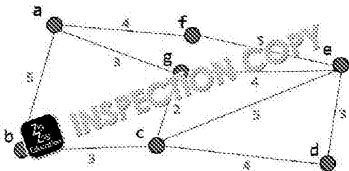
9

Worked Exam

We start by finding the distance from the starting vertex to its neighbours.

We repeat the process with the vertex furthest from the starting vertex.

The distance is only updated if a shorter route is found.



INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

10

Common Uses

There are a number of uses for the shortest path algorithm.

Finding the shortest route to drive between two addresses.

Finding the shortest route for packets to take between two devices in a network.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

1

Big O Notation

A Level Only

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

2

Big O Notation

Big O notation is used to describe how the complexity of an algorithm grows in relation to the number of items in the input.

This allows algorithms to be compared based on their complexity.

Big O looks at how long an algorithm takes to run.

An O is used as a prefix for all expressions.

n is used to refer to the number of items in the input.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

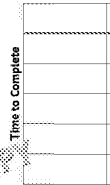
3

Constant Complexity

The time complexity remains the same no matter how many items.

$O(1)$

This graph illustrates constant complexity.



For example, finding the first item in a list takes the same amount of time regardless of the size of the list.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

4

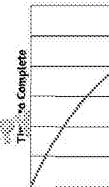
Logarithmic Complexity

A logarithm is the inverse of an exponential function.

The increase in time complexity decreases as the number of items increases.

$O(\log(n))$

This graph illustrates logarithmic complexity.



Examples of algorithms with logarithmic complexity include binary search and binary tree traversal.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

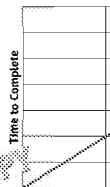
5

Linear Complexity

The time complexity is proportional to the number of items.

$O(n)$

This graph illustrates linear complexity.



Linear search is an example of linear complexity. The time to find an item in a list has to be evaluated for each item.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

6

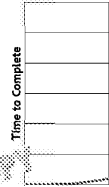
Polynomial Complexity

The rate at which time complexity increases with the number of items gets larger as the power of n increases.

$O(n^k)$

k is a constant value.

This graph illustrates polynomial complexity.



Bubble sort is an example of an algorithm with polynomial complexity.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

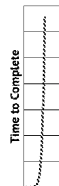
7

Exponential Complexity

The time complexity increases exponentially as the number of items gets larger.

$O(k^n)$

This graph illustrates exponential complexity.



The traveling salesman problem has exponential complexity. It involves a salesman wanting to find the shortest route to visit every city only once before returning to the starting point.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

8

Order of Complexity

This is the order of complexity from best to worst.

Complexity	Big O Notation
Constant	$O(1)$
Logarithmic	$O(\log(n))$
Linear	$O(n)$
Polynomial	$O(n^k)$
Exponential	$O(k^n)$

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

Examples

Algorithm	Big O
Linear Search	$O(n)$
Bubble Sort	$O(n^2)$
Binary Search	$O(\log n)$
Binary Tree Search	$O(\log n)$
Merge Sort	$O(n \log n)$

INSPECTION COPY

COPYRIGHT
PROTECTED

Tracing Algorithm

INSPECTION COPY

COPYRIGHT
PROTECTED

Please note that digital resources may only be copied by the purchasing institution on a single site and for their own use.

Tracing an Algorithm

Tracing is a method of testing an algorithm intended.

A trace table is used, showing the value of results of any conditions after the execution.

Line	x	OUTPUT	Condition
01	5		
02			TRUE
03		5	
04	4		
05			Return

INSPECTION COPY

COPYRIGHT
PROTECTED

Example

Line	x	OUTPUT	Condition
01	3		
02			
03			
04			
05			
06			
07			
08			
09			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
80			
81			
82			
83			
84			
85			
86			
87			
88			
89			
90			
91			
92			
93			
94			
95			
96			
97			
98			
99			
100			

INSPECTION COPY

COPYRIGHT
PROTECTED

Example

Line	x	OUTPUT	Condition
01	3		
02			
03			
04			
05			
06			
07			
08			
09			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
80			
81			
82			
83			
84			
85			
86			
87			
88			
89			
90			
91			
92			
93			
94			
95			
96			
97			
98			
99			
100			

TRUE, go to line 03

INSPECTION COPY

COPYRIGHT
PROTECTED

Example

Line	x	OUTPUT	Condition
01	3		
02			
03			
04			
05			
06			
07			
08			
09			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43			
44			
45			
46			
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
57			
58			
59			
60			
61			
62			
63			
64			
65			
66			
67			
68			
69			
70			
71			
72			
73			
74			
75			
76			
77			
78			
79			
80			
81			
82			
83			
84			
85			
86			
87			
88			
89			
90			
91			
92			
93			
94			
95			
96			
97			
98			
99			
100			

TRUE, go to line 03

INSPECTION COPY

COPYRIGHT
PROTECTED

6 Example

line	x	OUTPUT
01	1	TRUE, go to line 03
03	3	
04	2	
05		Return to line 02

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

7 Example

line	x	OUTPUT
01	1	TRUE, go to line 03
03	3	
04	2	
05		Return to line 02
02		TRUE, go to line 03

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

8 Example

line	x	OUTPUT
01	1	TRUE, go to line 03
03	3	
04	2	
05		Return to line 02
02		TRUE, go to line 03
03	2	
04	1	

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

9 Example

line	x	OUTPUT
01	1	TRUE, go to line 03
03	3	
04	2	
05		Return to line 02
02		TRUE, go to line 03
03	2	
04	1	
05		Return to line 02
02		FALSE, go to line 06

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

10 Example

line	x	OUTPUT
01	1	TRUE, go to line 03
03	3	
04	2	
05		Return to line 02
02		TRUE, go to line 03
03	2	
04	1	
05		Return to line 02
02		FALSE, go to line 06
		"End"

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Finite State M

Printed and digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

2

Finite-State Ma

An finite-state machine is a model that has states with acceptable inputs and ac

A traffic light is an example of a finite-state machine as it has a fixed number of states.



INSPECTION COPY

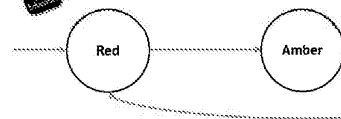
COPYRIGHT PROTECTED

Zig Zag Education

3

State Transition D

A state transition diagram is a method of representing a machine graphically



Each circle in a state transition diagram represents a state. The arrows represent transitions.

Each arrow is labelled with the input that causes the transition. In this case the input is time.

INSPECTION COPY

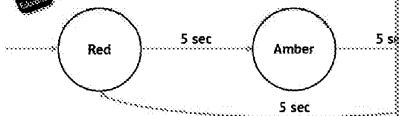
COPYRIGHT PROTECTED

Zig Zag Education

4

State Transition D

A state transition diagram is a method of representing a machine graphically



Each circle in a state transition diagram represents a state. The arrows represent transitions.

Each arrow is labelled with the input that causes the transition. In this case the input is time.

INSPECTION COPY

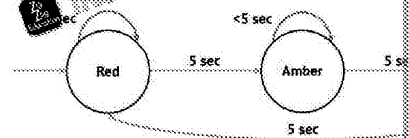
COPYRIGHT PROTECTED

Zig Zag Education

5

State Transition D

A state transition diagram is a method of representing a machine graphically



Each circle in a state transition diagram represents a state. The arrows represent transitions.

Each arrow is labelled with the input that causes the transition. In this case the input is time.

INSPECTION COPY

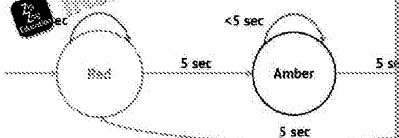
COPYRIGHT PROTECTED

Zig Zag Education

6

State Transition D

A state transition diagram is a method of representing a machine graphically



Each circle in a state transition diagram represents a state. The arrows represent transitions.

Each arrow is labelled with the input that causes the transition. In this case the input is time.

If the value of the timer is less than 5 the

INSPECTION COPY

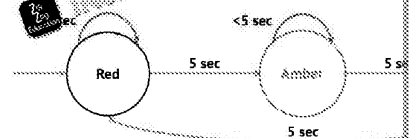
COPYRIGHT PROTECTED

Zig Zag Education

7

State Transition D

A state transition diagram is a method of representing a machine graphically



Each circle in a state transition diagram represents a state. The arrows represent transitions.

Each arrow is labelled with the input that causes the transition. In this case the input is time.

If the value of the timer is less than 5 the

INSPECTION COPY

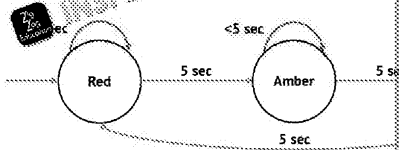
COPYRIGHT PROTECTED

Zig Zag Education

8

State Transition Diagram

A state transition diagram is a method of representing a finite state machine graphically.



Each circle in a state transition diagram represents a state. The arrows represent transitions between states.

Each arrow is labelled with the input that causes the transition. In this case the input is time.

If the value of the timer is less than 5 seconds, the state remains the same.

INSPECTION COPY

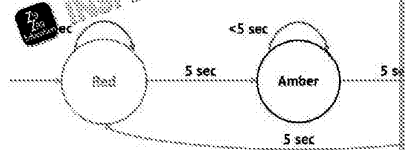
COPYRIGHT PROTECTED



9

State Transition Diagram

A state transition diagram is a method of representing a finite state machine graphically.



Each circle in a state transition diagram represents a state. The arrows represent transitions between states.

Each arrow is labelled with the input that causes the transition. In this case the input is time.

If the value of the timer is less than 5 seconds, the state remains the same.

INSPECTION COPY

COPYRIGHT PROTECTED



10

State Transition Table

State transition tables can be used to show the next state for each input on each state of the machine.

This is the state transition table for the traffic light system.

Input	Current State	Next State
<5 sec	Red	Red
5 sec	Red	Amber
<5 sec	Amber	Amber
5 sec	Amber	Red
<5 sec	Green	Green
5 sec	Green	Green

INSPECTION COPY

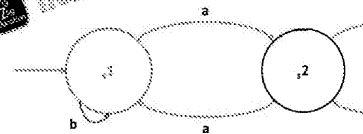
COPYRIGHT PROTECTED



11

Accepting State

Many FSMs have a final state known as an accepting state. This is indicated by a double circle.



This example FSM accepts input sequences that end in state 2. For example babbb.

The sequence babbb would be accepted as it ends in the accepting state.

INSPECTION COPY

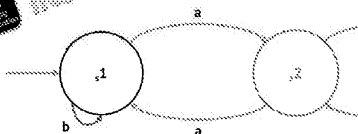
COPYRIGHT PROTECTED



12

Accepting State

Many FSMs have a final state known as an accepting state. This is indicated by a double circle.



This example FSM accepts input sequences that end in state 2. For example babbb.

The sequence babbb would be accepted as it ends in the accepting state.

INSPECTION COPY

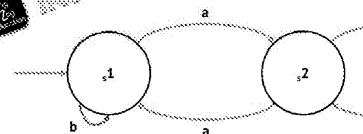
COPYRIGHT PROTECTED



13

Accepting State

Many FSMs have a final state known as an accepting state. This is indicated by a double circle.



This example FSM accepts input sequences that end in state 2. For example babbb.

The sequence babbb would be accepted as it ends in the accepting state.

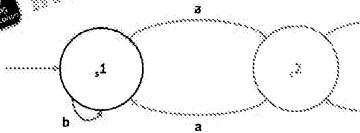
INSPECTION COPY

COPYRIGHT PROTECTED



Accepting State

Many FSMs have a final state known as an accepting state. This is indicated by a double circle.



This example FSM accepts input sequences such as babbb.

The sequence aabbbb would be accepted as it ends in the accepting state.

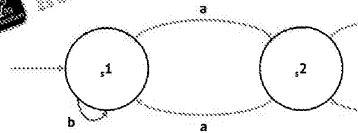
INSPECTION COPY

COPYRIGHT PROTECTED



Accepting State

Many FSMs have a final state known as an accepting state. This is indicated by a double circle.



This example FSM accepts input sequences such as babbb.

The sequence aabbbb would be accepted as it ends in the accepting state.

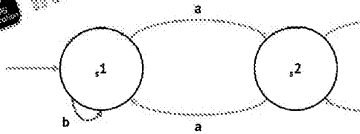
INSPECTION COPY

COPYRIGHT PROTECTED



Accepting State

Many FSMs have a final state known as an accepting state. This is indicated by a double circle.



This example FSM accepts input sequences such as babbb.

The sequence aabbbb would be accepted as it ends in the accepting state.

The sequence babb would be accepted as it ends in the accepting state.

INSPECTION COPY

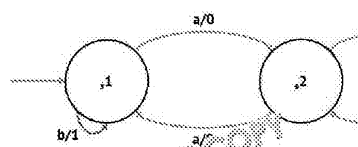
COPYRIGHT PROTECTED



Outputs

Some FSMs can produce outputs. The output is indicated by a slash followed by the output value.

Input / Output



The sequence baaaba would produce the output 100001.

The sequence ababb would produce the output 01001.

INSPECTION COPY

COPYRIGHT PROTECTED



Maths for Regular Expressions

A Level Only

INSPECTION COPY

COPYRIGHT PROTECTED



Maths for Regular Expressions

Regular expressions can also be used to define sets of values.

Different types of numeric value can be defined:

Natural Numbers

$\mathbb{N} = \{0, 1, 2, 3, \dots\}$

Integers

$\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$

Empty Set

$\{\}$ or \emptyset

INSPECTION COPY

COPYRIGHT PROTECTED



3

Set Comprehe

Sets can also be defined using set comprehension properties that a set should have.

$$A = \{ x \mid x \in \mathbb{N} \wedge x > 0 \}$$

x represents the values of

The equation that follows the \mid defines

\in is used to indicate that \mathbb{N} is a

\mathbb{N} is used to represent the natural

\wedge is used to represent

$>$ means greater than or equal to

INSPECTION COPY

COPYRIGHT PROTECTED



4

Set Comprehe

$$A = \{ x \mid x \in \mathbb{N} \wedge x > 0 \}$$

This example is used to create a set of values that are greater than

This could look like

$$A = \{ 1, 2, 3, 4, \dots \}$$

COPYRIGHT PROTECTED



INSPECTION COPY

5

Types of Se

Finite Have a fixed number of elements using natural numbers.

Infinite Have an infinite number of elements. \mathbb{N} represents all natural numbers.

Countably Infinite An infinite set of numbers that can be counted.

Key Term

Cardinality The number of elements in a set.

INSPECTION COPY

COPYRIGHT PROTECTED



6

Compact Represe

Strings can be represented in a compact way using a regular expression.

$$\{ 0^n 1^n \mid n \geq 0 \}$$

This defines a set that has strings with an equal number of 0s and 1s.

$$\{ 01, 0011, 000011, \dots \}$$

COPYRIGHT PROTECTED



INSPECTION COPY

7

Cartesian Pro

The Cartesian product is the product of two sets.

$$X = \{ 1, 2, 3 \}$$

$$Y = \{ a, b, c \}$$

The Cartesian product of X and Y is

$$\{ (1, a), (1, b), (1, c), (2, a), (2, b), (2, c), (3, a), (3, b), (3, c) \}$$

INSPECTION COPY

COPYRIGHT PROTECTED



8

Subset

This is what all the elements of one set are contained within another set.

A proper subset has fewer elements than the superset.

$$\{ 0, 1, 2 \} \subset \{ 0, 1, 2, 3, 4 \}$$

\subset means that $\{ 0, 1, 2 \}$ is a proper subset of $\{ 0, 1, 2, 3, 4 \}$.

A countable set is a set with the same number of elements as a subset of natural numbers.

COPYRIGHT PROTECTED



INSPECTION COPY

Set Operation

Union	Joining sets together to form a new set containing all the elements from both sets.
Intersection	Joining sets together to form a new set containing only the elements that are common to both sets.
Difference	Joining sets together to form a new set containing only the elements that are unique to one of the sets.

INSPECTION COPY

Regular Expression

A Level
Only

INSPECTION COPY

Please note that digital resources may only be copied by the purchasing institution on a single site and for their own use.

Regular Expression

Key Terms

Set	A set is a collection of unique elements.
Regular Expression	A form of notation that describes a set of strings.
Regular Language	A language that can be expressed by a regular expression.

INSPECTION COPY

Expression Operator

A range of operators is used to construct regular expressions.

Most of the operators refer to the previous expression.

Operator	Meaning	Example
*	0 or more times	ab*c
?	0 or 1 times	ab?c
+	1 or more times	ab+c
	Either character	(a b)c
[]	Any one of the characters	[abc]de

INSPECTION COPY

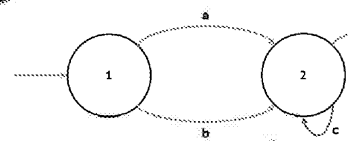
Examples

*		Accepts any string starting with 0 or more 'a's.
?	abc?	Accepts any string starting with 'abc' or not.
+	abc+	Accepts any string starting with 'abc' one or more times.
	a(b c)	Accepts any string starting with 'a' followed by either 'b' or 'c'.
[]	a[bcd]	Accepts any string starting with 'a' followed by 'b', 'c' or 'd'.

INSPECTION COPY

Finite-State Machine

All finite-state machines can also be represented by regular expressions.



The state transition diagram above represents the regular expression:

(a|b)c*de*

INSPECTION COPY

1

2

Backus-Naur Form (BNF) is an example of a formal language.



3

Notation	Meaning
< >	Encloses each element
::=	Defines the rule for a previous element
	Used to indicate OR
{ }	Encloses optional elements



4

Notation	Meaning
< >	Encloses each element
::=	Defines the rule for a previous element
	Used to indicate OR
{ }	Encloses optional elements

This example rule states that student data elements: name, address and



5

```
<character> ::= "M"
```



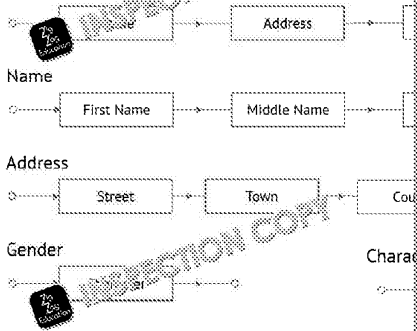
6

Used to represent a non-terminating sequence that can be reused.



Example

Student Details



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

Turing Machine

A Level
Only

Please note that digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

The Turing Machine

In 1936 Alan Turing developed a theory that can carry out any algorithm.

The Turing machine is considered the foundation of digital computers as it defines what can be computed.

It is an example of a finite-state machine.

INSPECTION COPY

COPYRIGHT
PROTECTED

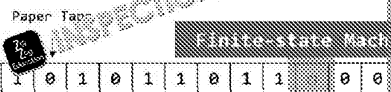
Zig
Zag
Education

Structure

This is the basic structure of a Turing Machine.

There is a read/write head that can read data from and write data to an infinite paper tape.

The paper tape is divided into square cells, each containing a character, like a binary digit. The □ symbol represents an empty cell.



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

Transition Function

Transition functions are used to control the movement of the head on the tape based on the current state and the symbol read.

They also control whether the head should move left or right.

Transition functions are represented by state transition diagrams.

The state marked with an H is the halting state, which tells the machine to stop.

1/0, R

0/1, R

The labels have the following structure: read/write, direction.

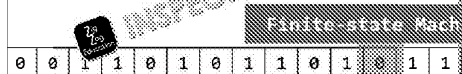
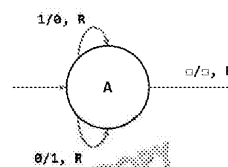
INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

Example

This example transition function reverses the tape until it reaches a blank symbol.



INSPECTION COPY

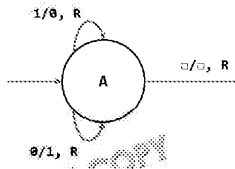
COPYRIGHT
PROTECTED

Zig
Zag
Education

6

Example

This example transition function reverse tape until it reaches a blank symbol.



Finite-state Machine

0 1 1 0 1 1 0 0 1 0 0 0 0 0

INSPECTION COPY

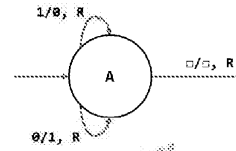
COPYRIGHT PROTECTED

Zig Zag Education

7

State Table

A Turing machine can also be described using a state table.



State	Read	Write	Move
A	1	0	R
A	0	1	R
A	□	□	R

INSPECTION COPY

COPYRIGHT PROTECTED

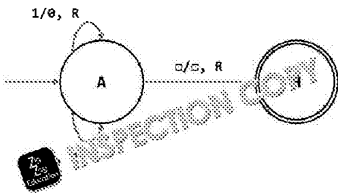
Zig Zag Education

8

Written Notation

Transition functions can also be described using written notation.

δ (Current State, Input) = (Next State, Output, Move)



INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

1

Binary and Hexadecimal

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT PROTECTED

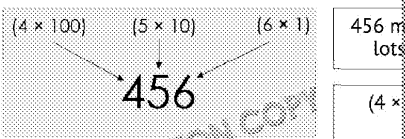
Zig Zag Education

2

Base 10

Normal numbers are written in base 10.

Each digit is worth 10 times more than the digit to its right.



456 million

(4 x 100)

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

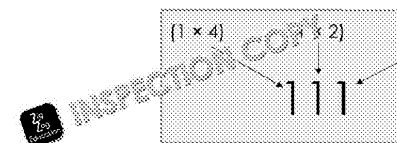
3

Binary

Computers work with a different number system.

Computers use binary which only has two digits: 0 and 1.

In binary each digit is worth twice as much as the digit to its right.



INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

Binary Exam

Converting from binary (base 2) to decimal (base 10)

16s	8s	4s	2s	1s	
1	0	0	1	1	

Write out the value of each digit at the

COPYRIGHT PROTECTED



INSPECTION COPY

Binary Exam

Converting from binary (base 2) to decimal (base 10)

16s	8s	4s	2s	1s	
1	0	0	1	1	(1x16)

Write out the value of each digit at the

Add together the value of the columns

COPYRIGHT PROTECTED



INSPECTION COPY

Binary Exam

Converting from binary (base 2) to decimal (base 10)

16s	8s	4s	2s	1s	
1	0	0	1	1	(1x16)
0	1	1	1	0	(1x8)

Write out the value of each digit at the

Add together the value of the columns

COPYRIGHT PROTECTED



INSPECTION COPY

Denoting Base

As the digits are the same, subscripts are used to state which number has been written

1111₁₀

The subscript 10 states this is a decimal number

The

COPYRIGHT PROTECTED



INSPECTION COPY

Hexadecimal

Numbers in binary can get rather large

Hexadecimal (base 16) shortens the number as each digit is worth 16 times the one to the right

$$10_{16} = 16_{10}$$

$$100_{16} = 256_{10}$$

Along with 0-9, hexadecimal also uses letters A-F

A = 10, B = 11, C = 12, D = 13, E = 14, F = 15

$$100_{16} = 160_{10}$$

$$800_{16} = 2816_{10}$$

COPYRIGHT PROTECTED



INSPECTION COPY

Hexadecimal Exam

Converting from hex to denary is also easy

16s	8s	4s	1s	
1	2	3		
A	B	C		

Write out the value of each digit at the

COPYRIGHT PROTECTED



INSPECTION COPY

10

Hexadecimal Ex

Converting from hex to denary is also

16s	1s	Working
1	2	3
A	B	C

$$(1 \times 256) + (2 \times 16) + (3 \times 1)$$

$$(10 \times 256) + (11 \times 16) + (12 \times 1)$$

Write out the value of each digit at the

Multiply the value of the column by the

INSPECTION COPY

COPYRIGHT
PROTECTED

11

Hexadecimal Ex

Converting from hex to denary is also

16s	1s	Working
1	2	3
A	B	C

$$(1 \times 256) + (2 \times 16) + (3 \times 1)$$

$$(10 \times 256) + (11 \times 16) + (12 \times 1)$$

Write out the value of each digit at the

Multiply the value of the column by the

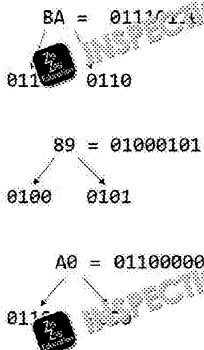
Add together the totals to get the denary

INSPECTION COPY

COPYRIGHT
PROTECTED

12

Hexadecimal to



Although computers use binary, hex is used to

Each hex digit has a value of 16

In binary, this is

To convert a two digit hex number to binary, convert each digit

Once each individual digit has been converted, join the binary digits together to

INSPECTION COPY

COPYRIGHT
PROTECTED

13

Binary to Hexad

Binary to hex is also a straightforward

Group the binary number into batches of four (starting at the right-hand

01100101

=

0110	0101
6	5

Convert each batch of four digits to a hex digit

Then just join the hex digits together

INSPECTION COPY

COPYRIGHT
PROTECTED

14

Denary to Bin

Converting from denary to binary uses subtraction

Example: 230 in decimal to binary

Start with the number line showing the powers of 2

128	64	32	16	8

INSPECTION COPY

COPYRIGHT
PROTECTED

15

Denary to Bin

Converting from denary to binary uses subtraction

Example: 230 in decimal to binary

Start with the number line showing the powers of 2

128	64	32	16	8
1				

Locate the largest number you'll need and subtract it from the denary number

INSPECTION COPY

COPYRIGHT
PROTECTED

Denary to Binary

Converting from denary to binary uses subtraction.

Example: 230 in decimal to binary

Start with the number line showing the powers of 2.

128	64	32	16	8
1				

$$230 - 128 = 102$$

Locate the largest number you'll need and subtract it from the denary you're converting.

Subtract that number from the denary you're converting, and a 1 if it is.

INSPECTION COPY

COPYRIGHT
PROTECTED



Denary to Binary

Converting from denary to binary uses subtraction.

Example: 230 in decimal to binary

Start with the number line showing the powers of 2.

128	64	32	16	8
1	1	1	0	0

$$230 - 128 = 102$$

$$102 - 64 = 38$$

$$38 - 32 = 6$$

Locate the largest number you'll need and subtract it from the denary you're converting.

Subtract that number from the denary you're converting, and a 1 if it is.

Repeat the process for each column, putting a 1 if it is.

INSPECTION COPY

COPYRIGHT
PROTECTED



Decimal to Hexadecimal

Example: 2468

$$2400 \div 16 = 154 \text{ remainder } 4$$

$$154 \div 16 = 9 \text{ remainder } 10$$

$$9 \div 16 = 0 \text{ remainder } 9$$

2468 in denary is 9A4 in hex

Remember you can always convert to binary hexadecimal if you find it easier.

Divide the denary number by 16.

Write down the remainder.

Repeat the process and write down the remainder.

Keep going until the remainder is 0.

Final remainder is the final hex number. Penultimate remainder is the next hex number. The remainder is the next hex number.

INSPECTION COPY

COPYRIGHT
PROTECTED



Binary Arithmetic

INSPECTION COPY

COPYRIGHT
PROTECTED



Please do not reproduce this resource in any form without the permission of Zig Zag Education. This resource may only be copied by the purchasing institution on a single site and for their own use.

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules.

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

INSPECTION COPY

COPYRIGHT
PROTECTED



Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules.

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \\ 0 \ 1 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline 0 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED



4

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \\ 0 \ 1 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline 1 \ 0 \end{array}$$

COPYRIGHT
PROTECTED



5

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \\ 0 \ 1 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline 1 \ 0 \end{array}$$

COPYRIGHT
PROTECTED



6

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \ 1 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline 1 \ 0 \end{array}$$

COPYRIGHT
PROTECTED



7

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \ 1 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline 1 \ 0 \end{array}$$

COPYRIGHT
PROTECTED



8

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \ 1 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline 1 \ 0 \end{array}$$

COPYRIGHT
PROTECTED



9

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \ 1 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 1 \ 0 \ 1 \\ \hline 1 \ 0 \end{array}$$

COPYRIGHT
PROTECTED



10

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \quad 1 \quad 1 \quad 0 \quad 1 \\ 0 \quad 0 \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 1 \quad 1 \quad 0 \end{array}$$

COPYRIGHT
PROTECTED



11

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \quad 1 \quad 1 \quad 0 \quad 1 \\ 0 \quad 0 \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 1 \quad 1 \quad 0 \end{array}$$

COPYRIGHT
PROTECTED



12

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \quad 1 \quad 1 \quad 0 \quad 1 \\ 0 \quad 0 \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 1 \quad 1 \quad 0 \end{array}$$

The total number of bits is bigger than 5 bits (the size of the register). This is known as an overflow.

COPYRIGHT
PROTECTED



13

Binary Multiplication

Binary multiplication is a simple process.

$$\begin{array}{r} 1 \quad 0 \quad 1 \quad 0 \\ 0 \quad 1 \quad 0 \quad 1 \times \\ \hline 1 \quad 0 \quad 1 \quad 0 \\ 1 \quad 0 \quad 1 \quad 0 \\ \hline 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \end{array}$$

Write the first number in the register.

Next add the number in the register.

COPYRIGHT
PROTECTED



14

Binary Multiplication

Here are some more examples:

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 0 \\ 0 \quad 1 \quad 1 \quad 0 \times \\ \hline 1 \quad 1 \quad 0 \quad 0 \\ 1 \quad 1 \quad 0 \quad 0 \\ \hline 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \end{array}$$

For the last example we used the following rules:

$$1 + 1 + 1 + 1 = 0 \text{ carry } 0 \text{ and carry } 1$$

This is because $1 + 1 + 1 + 1$ produces 4.

COPYRIGHT
PROTECTED



1

Two's Complement

COPYRIGHT
PROTECTED



Photocopying this document may only be done by the purchasing institution on a single site and for their own use.

2

Two's Complement

Two's complement is a method of representing binary.

The most significant bit is a negative bit.

128	64	32	16	8
1	0	0	0	1

$$-128 + 8 + 4 + 1$$

It is easy to tell whether a two's complement number is positive. If the most significant bit is a 1 then it is negative, if it is 0 then it is positive.

INSPECTION COPY

COPYRIGHT
PROTECTED

3

Subtraction

Two's complement can help with binary subtraction.

First we convert the second number to its negative equivalent in two's complement.

COPYRIGHT
PROTECTED

INSPECTION COPY

4

Subtraction

Two's complement can help with binary subtraction.

First we convert the second number to its negative equivalent in two's complement.

We do this by reversing the bits and adding 1 to the number.

COPYRIGHT
PROTECTED

INSPECTION COPY

5

Subtraction

Two's complement can help with binary subtraction.

First we convert the second number to its negative equivalent in two's complement.

We do this by reversing the bits and adding 1 to the number.

COPYRIGHT
PROTECTED

INSPECTION COPY

6

Subtraction

Two's complement can help with binary subtraction.

First we convert the second number to its negative equivalent in two's complement.

We do this by reversing the bits and adding 1 to the number.

Next we add the numbers together using the rules of binary addition.

COPYRIGHT
PROTECTED

INSPECTION COPY

7

Subtraction

Two's complement can help with binary subtraction.

First we convert the second number to its negative equivalent in two's complement.

We do this by reversing the bits and adding 1 to the number.

Next we add the numbers together using the rules of binary addition.

If there is an overflow it is discarded.

COPYRIGHT
PROTECTED

INSPECTION COPY

8

Another Exam



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

9

Another Exam

Reverse the bits of the second number.



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

10

Another Exam

Reverse the bits of the second number.



Add 1 to the second number.



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

11

Another Exam

Reverse the bits of the second number.



Add 1 to the second number.

Add the numbers together using the rules of binary addition.



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

12

Another Exam

Reverse the bits of the second number.



Add 1 to the second number.

Add the numbers together using the rules of binary addition.

Discard any overflow.



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

INSPECTION COPY

Fraction

INSPECTION COPY

Photocopying this document may only be
permitted by the purchasing institution on a
single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

Fixed Point

Fixed point is the simplest method of representing a number using binary.

In a fixed-point binary number the values point are halved.

8	4	2	1	1/2
1	0	1	0	1

$$8 + 2 = 10$$

10.5

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

Another Exam

$$1/2 = 0.5$$

$$1/4 = 0.25$$

$$1/8 = 0.125$$

$$1/16 = 0.0625$$

8	4	2	1	1/2
1	0	1	0	1

$$8 + 2 = 10 \quad 0.5 +$$

11.625

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

Floating Point

Floating point is an alternative method of representing a number with a fractional part.

As the name suggests, the binary point can be in a fixed position.

A floating-point number is divided into the mantissa and the exponent.

8	4	2	1	1/2	1/4	1/8	1/16
0	1	1	0	1	0	0	0

Mantissa

This is the actual number

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

Floating Point

There are many different formats of floating point, we are going to be working with the two's complement format.

8	4	2	1	1/2	1/4	1/8	1/16
0	1	1	0	1	0	0	0

Mantissa

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

Floating Point

There are many different formats of floating point, we are going to be working with the two's complement format.

8	4	2	1	1/2	1/4	1/8	1/16
0	1	1	0	1	0	0	0

Mantissa

The value of the exponent is 3, so we need three places to the right.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

Floating Point

There are many different formats of floating point, we are going to be working with the two's complement format.

8	4	2	1	1/2	1/4	1/8	1/16
0	1	1	0	1	0	0	0

Mantissa

The value of the exponent is 3, so we need three places to the right.

$$4 + 2 + 0.5 = 6.5$$

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

Another Exam

Both the mantissa and the exponent are in binary. If either starts with a 1 it needs to be converted by flipping the bits and adding 1.

1	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---

Mantissa

INSPECTION COPY

COPYRIGHT PROTECTED



Another Exam

Both the mantissa and the exponent are in binary. If either starts with a 1 it needs to be converted by flipping the bits and adding 1.

0	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Mantissa

INSPECTION COPY

COPYRIGHT PROTECTED



Another Exam

Both the mantissa and the exponent are in binary. If either starts with a 1 it needs to be converted by flipping the bits and adding 1.

16	8	4	2	1	1/2	1/4	1/8
0	1	0	1	0	0	1	1

Mantissa

$$8 \times 2 + 0.25 + 0.125$$

INSPECTION COPY

COPYRIGHT PROTECTED



Another Exam

Both the mantissa and the exponent are in binary. If either starts with a 1 it needs to be converted by flipping the bits and adding 1.

16	8	4	2	1	1/2	1/4	1/8
0	1	0	1	0	0	1	1

Mantissa

$$8 \times 2 + 0.25 + 0.125$$

We know the result is a negative number so we add a negative sign.

INSPECTION COPY

COPYRIGHT PROTECTED



Negative Exponent

If the exponent is negative we shift the binary point to the right.

0	1	0	0	0
---	---	---	---	---

Mantissa

INSPECTION COPY

COPYRIGHT PROTECTED



Negative Exponent

If the exponent is negative we shift the binary point to the right.

First we have to convert the exponent to a positive number by adding 1 to the number of places the binary point is shifted.

0	1	0	0	0
---	---	---	---	---

Mantissa

INSPECTION COPY

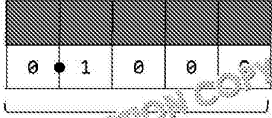
COPYRIGHT PROTECTED



Negative Expo

If the exponent is negative we shift the bin
than to the right.

First we have to convert the exponent to a
the number of places the binary po



Antissa

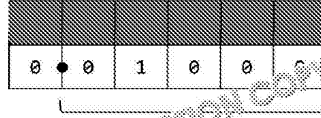
COPYRIGHT
PROTECTED



Negative Expo

If the exponent is negative we shift the bin to the left.
If the exponent is positive we shift the bin to the right.

First we have to convert the exponent to a
the number of places the binary po



Santissa

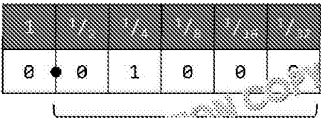
COPYRIGHT
NOTED



Negative Expo

If the exponent is negative we shift the binary point to the right.

First we have to convert the exponent to a
the number of places the binary po



Antitissa

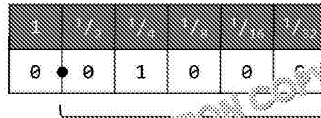
COPYRIGHT
PROTECTED



Negative Expo

If the exponent is negative we shift the bin
than to the right.

First we have to convert the exponent to a
the number of places the binary po



Santissa

COPYRIGHT
PROTECTED

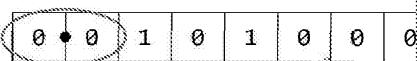


0.25

Normalisati

When representing numbers in binary we want the smallest number of bits possible; to do this we

To normalise a number you look at the standard deviation and whether there are any repeated values.



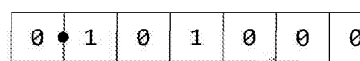
COPYRIGHT
PROTECTED



Normalisati

When representing numbers in binary we want the smallest number of bits possible; to do this we

To normalise a number you look at the standard deviation and whether there are any repeated values.



The repeated value should be removed and

COPYRIGHT
PROTECTED



Normalisation

When representing numbers in binary we want the smallest number of bits possible; to do this we

To normalise a number you look at the start of the number to see whether there are any repeated

0 • 1 0 1 0 0 0

The repeated value needs to be removed and

the binary point has moved one place so the value is updated by subtracting 1

INSPECTION COPY

COPYRIGHT
PROTECTED



Comparison

It is faster to do calculations using fixed-point numbers than floating-point numbers

Floating-point numbers can represent a large range of fractional parts when compared to fixed-point numbers

This means that floating-point numbers are used where you need to represent a wide range of values

On the other hand, fixed-point numbers are used where the range of processing is more important

INSPECTION COPY

COPYRIGHT
PROTECTED



Rounding Error

Some numbers cannot be represented using a finite number of bits. In this case the number is rounded to the nearest representable number

The rounding error is the difference between the original value and the rounded value. There are two different methods of rounding: truncation and rounding to the nearest

Absolute Error

The difference between the actual value and the rounded number

The difference between the actual value and the rounded number

INSPECTION COPY

COPYRIGHT
PROTECTED



Example

If we wanted to represent the decimal value 5.6 in binary we would end up with

101.10011
(5.59375)

The absolute error is

$5.6 - 5.59375 = 0.00625$

The relative error is

$0.00625 / 5.6 = 0.001116$

INSPECTION COPY

COPYRIGHT
PROTECTED



Error Checking

When transmitting or storing data, it is important to ensure that the data is not corrupted

There are four main techniques for error checking:

Parity Bit

Majority Vote

Checksum

INSPECTION COPY

COPYRIGHT
PROTECTED



Error Checking

When transmitting or storing data, it is important to ensure that the data is not corrupted

There are four main techniques for error checking:

Parity Bit

Majority Vote

Checksum

INSPECTION COPY

COPYRIGHT
PROTECTED



Parity Bit

An extra bit is added as the most significant bit to the total number of 1s either in the data or in the parity bit.

Odd Parity

The number of bits in the bit pattern is odd.

Bit Pattern
101101
(5 × 1)

Even Parity

The number of bits in the bit pattern is even.

Bit Pattern
101101
(5 × 1)

Errors can be detected by checking the parity of the data received; if the parity is no longer the same, an error has occurred.

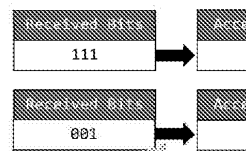
INSPECTION COPY

COPYRIGHT PROTECTED



Majority Vote

Majority vote is a method of error correction where the data is transmitted three times.



The bit that occurs the most is kept as the value.

INSPECTION COPY

COPYRIGHT PROTECTED



Checksum

Data is transmitted in blocks called packets.

The checksum is a hash of the data in the packet, which is sent along with the packet.

When the packet is received, the checksum is calculated. If the checksums match then the data has been received correctly.

If however the two checksums don't match, an error has occurred.

INSPECTION COPY

COPYRIGHT PROTECTED



Check Digit

It is common for human error to occur due to long numbers, for example customer numbers.

To prevent this, a check digit is often added to the end of the number that is calculated using the rest of the digits.

For example, all books have an ISBN number, which is a 10-digit number with a check digit at the end.

Another example is credit card numbers where a check digit is used to ensure the number has been entered correctly.

INSPECTION COPY

COPYRIGHT PROTECTED



Example

MOD11 is a variation method for calculating a check digit. The MOD11 method includes the following steps:

Each digit is given a multiplication factor value. It starts at 2 and increases by 1 for each digit.

	4	3	2	5	6	7
Factor	8	7	6	5	4	3
Result	32	56	18	30	4	21

Each digit is multiplied by its factor and the results are added together.

The result is divided by 11. Finally the remainder is used to give the check digit.

$$167/11 = 15 \text{ remainder } 2$$

$$11 - 2 = 9$$

$$48361739$$

INSPECTION COPY

COPYRIGHT PROTECTED



Example

To verify the number, the same process is repeated, but the check digit is included and the multiplier starts at 1.

	1	2	3	4	5	6	7	8
Factor	8	7	6	5	4	3	2	1
Result	32	56	18	30	4	21	14	8

Once again the result is divided by 11.

$$167/11 = 15 \text{ remainder } 2$$

If the remainder is 0 then the number is correct.

INSPECTION COPY

COPYRIGHT PROTECTED



Comparison

With the parity method, if two bits change the same an error will not be detected.

Parity bits, checksums and check digits may be used but they can't correct the error.

Majority voting is a method of error correction that can correct an error as well as detect it.

However, majority voting is inefficient as the entire data needs to be sent for every bit.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Bitmapped Images

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Please note that digital resources may only be copied by the purchasing institution on a single site and for their own use.

Bitmapped Images

Images can be represented using a grid of elements called pixels.

1 pixel

Each pixel is mapped to a specific location, hence the term 'bitmapped'.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Resolution

The resolution of an image is the width of the image in pixels and the height of the image in pixels.

5 pixels

5 pixels

For example:

The resolution of the image is:

width in pixels

height in pixels

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Colour Depth

Colour depth refers to the number of bits used to represent each pixel in a bitmapped image.

Black-and-white images have a colour depth of 1 bit allocated to each pixel. 0 represents a black pixel and 1 represents a white pixel.

The image below has a colour depth of 3 bits.

Colour	Bit Pattern
White	000
Grey	001
Green	100
Blue	111

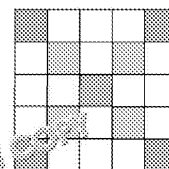
INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Storage Requirements

The storage requirements of a bitmapped image can be calculated using this formula:

resolution in pixels × colour depth



Storage requirements: $25 \times 3 = 75$ bits

We normally divide by 8 to get the answer in bytes.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

6

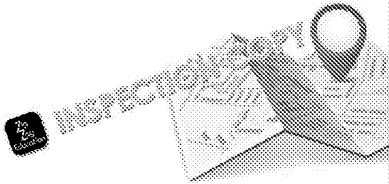
Metadata

Most bitmaps and image files contain additional information.

Width

Height

Many images also contain geolocation data which the image was taken from.



INSPECTION COPY

COPYRIGHT
PROTECTED



1

Vector Graphics

A Level
Only

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED



2

Vector Graphics

Vector graphics use a combination of lines and shapes to represent images.

The properties of each geometric shape are stored.

This information is used by the computer to draw the image.



INSPECTION COPY

COPYRIGHT
PROTECTED



3

Properties

Typical properties of object

Object type

Square

Co-ordinates

(1,1) (3,1) (3,3) (1,3)

Fill colour

Blue

Edge colour

Green

Edge width

2px

Edge style

Dashed

INSPECTION COPY

COPYRIGHT
PROTECTED



4

Advantages

Geometric images stored as vectors require less storage space compared with bitmaps.

Geometric images stored as vectors can be scaled for secondary storage.

Vector graphics can be resized without losing quality.



INSPECTION COPY

COPYRIGHT
PROTECTED



5

Disadvantages

Only suitable for images made up of simple geometric shapes.

Unsuitable to represent photographs or images with great detail within the same space.

Complex images can take a long time to load.



INSPECTION COPY

COPYRIGHT
PROTECTED



1

Digital Sound

INSPECTION COPY

Photo copied digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

2

Analogue vs Digital

Analogue data can have any value, can be continuous.

Digital data can only have fixed values; it can only be 1s and 0s.

1234.5678 100

Analogue Digital

For a computer to work with data, the data has to be digitised, i.e. it has to be digitised

INSPECTION COPY

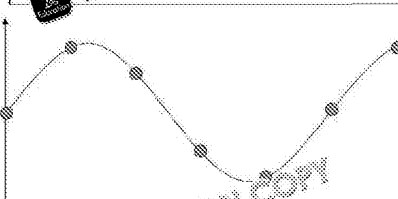
COPYRIGHT PROTECTED

Zig Zag Education

3

Sampling Rate

The first stage in digitising a sound wave is to look at the wave at regular intervals and take a sample.



The sampling rate, measured in hertz (Hz), is the number of samples taken per second. The greater the rate, the more accurate the digital representation.

INSPECTION COPY

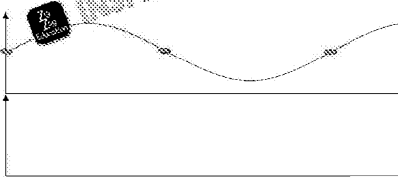
COPYRIGHT PROTECTED

Zig Zag Education

4

Nyquist

Too low a sampling rate and the computer will not be able to reconstruct the original wave.



INSPECTION COPY

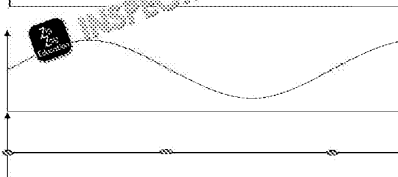
COPYRIGHT PROTECTED

Zig Zag Education

5

Nyquist

Too low a sampling rate and the computer will not be able to reconstruct the original wave.



INSPECTION COPY

COPYRIGHT PROTECTED


Zig Zag Education

6

Nyquist

Too low a sampling rate and the computer will not be able to reconstruct the original wave.

Nyquist's Theorem: To rebuild a wave, the sampling rate must be at least twice the highest frequency of the wave.



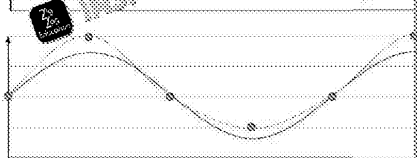
INSPECTION COPY

COPYRIGHT PROTECTED

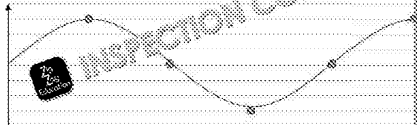
Zig Zag Education

7 Bit Depth

Another way to increase the quality of the sound is by increasing the bit depth.



This wave only has four levels (2 bits) but the remaining levels are lost.



This wave has eight levels (3 bits) and fits the original wave more closely.

INSPECTION COPY

COPYRIGHT PROTECTED



8 File Size

To calculate the file size of a sound file you can use the following formula:

Sampling rate (in Hz)
×
bit depth (in bits)
×
channels
×
duration (in seconds)



$2,120,000 \text{ bits} / 8 = 2,640$

If the sound is a mono sound then there will be one channel. If it is a stereo sound then there will be two channels.

INSPECTION COPY

COPYRIGHT PROTECTED



1 Data Compression

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT PROTECTED



2 Compression

Compression is the process of reducing the size of a file.

It is particularly beneficial to compress multimedia files such as images, videos and audio as they typically use a lot of storage space.

Multimedia files are often compressed so they can be sent over the Internet.

Other file types, such as text, can also be compressed.

INSPECTION COPY

COPYRIGHT PROTECTED



3 Types of Compression

There are two types of compression:

Lossless

The data is compressed without permanently removing any data.

The original data can be recovered.

INSPECTION COPY

COPYRIGHT PROTECTED



4 Lossy Compression

An example of lossy compression is the MP3 audio file format.

It removes parts of the sound that are out of the range of normal human hearing.

This means that people shouldn't be able to notice any difference in the sound.

INSPECTION COPY

COPYRIGHT PROTECTED



5

Comparison

	Advantages	Disadvantages
Lossless	All data is retained.	Can be slow.
Lossy	Usually results in smaller file sizes when compared to lossless compression.	Permanently loses some data.

INSPECTION COPY

COPYRIGHT
PROTECTED

6

Run-Length Encoding

Run-length encoding (RLE) is one of the simplest forms of lossless compression.

It works by identifying sequences of the same character. For example, the sequence AAAAA could represent AAAAAA.

11000111001
2(1), 3(0), 2(1), 2(0), 1

INSPECTION COPY

COPYRIGHT
PROTECTED

7

Dictionary-Based

Dictionary-based methods work by identifying repeated sequences of data.

Each sequence is represented by a unique code.

1011 0011 0011
0011 1111 1011

There are only three different sequences in the above data, so they can be represented using a single code.

1011 = 00 0011 = 01 0011 = 01

The compressed data would then be:

00 01 01 11 01 1

INSPECTION COPY

COPYRIGHT
PROTECTED

1

Encryption

INSPECTION COPY

COPYRIGHT
PROTECTED

Please do not distribute this resource. It may only be copied by the purchasing institution on a single site and for their own use.

2

Encryption

Encryption is the process of changing a message into a form that is unreadable without a special key.

Key Terms:

Cipher	A method of encryption.
Plaintext	The original message before encryption.
Ciphertext	The message after it has been encrypted.

INSPECTION COPY

COPYRIGHT
PROTECTED

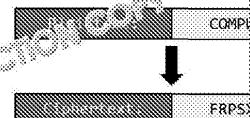
3

Caesar Cipher

Used by Julius Caesar, this is one of the earliest forms of encryption.

It works by offsetting the letters of the alphabet by a certain number. For example, with a shift of 3, A becomes D, B becomes E, and so on.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S



INSPECTION COPY

COPYRIGHT
PROTECTED

Each page of a one-time pad features a sequence of random numbers. The plaintext is encrypted by mixing each digit

[illegible]

Once the message is received, it is decrypted

There are only two copies of
once they have been used they are de

INSPECTION COPY



Suppose we want to encrypt the letter **E** using this OTP.

We start by converting the first letters of bob and the OTP to a numerical value (in this alphabet).

Plaintext	
H	01000 (8)
OTP	
P	(4)

Plaintext	OTP	CT
0	0	
0	0	
0	1	
0	0	
0	0	



Suppose we want to encrypt the letter *d* using this OTP.

We start by converting the first letters of both words and the OTP to a numerical value (in this case, using the alphabet).

Plaintext	
H	01000 (8)

OTP	
D	10000 (4)

Plaintext	OTP	Ciphertext
0	0	
0	0	
0	1	
0	0	
0	0	

INSPECTION COPY



We can decrease the message by
reducing the process.

Ciphertext	
L	01100 (12)

OTP	
D	00100 (4)



We can determine the message by reversing the process.

The diagram illustrates the XOR operation for encryption and decryption. It shows three main components: Plaintext, Key, and Ciphertext.

Encryption:

- Plaintext:** L (01)
- Key:** D (00)
- Ciphertext:** 01100 (12)

Decryption:

- Ciphertext:** 01100
- Key:** 00
- Plaintext:** H (01000)

The XOR operation is shown as 01 XOR 00 = 01, and 01 XOR 00 = 01, resulting in 0101. The final result is 01000.

INSPECTION COPY



The Caesar cipher is a very weak form of encryption. It is based on the fact that the alphabet is a fixed set of 26 letters. If you know the key, you can work out the key all messages c

Vernam, however, is the only encryption method that is secure if it is used properly.

If every letter in the OTP is truly random then it makes it impossible to break code.



1

Logic Gate

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT PROTECTED

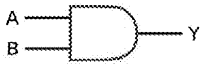
Zig Zag Education

2

Logic Gate


Logic gates are the basic components of a logic circuit. Each logic gate performs a different function.

These are the three basic logic gates.



AND Gate

Output is 1 only if both inputs are 1



OR Gate

Outputs a 1 if one or both of the inputs are 1

INSPECTION COPY

COPYRIGHT PROTECTED

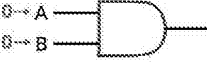
Zig Zag Education

3

Truth Table

All the possible outcomes of a logic diagram are shown in a truth table.

This is the truth table for the AND Gate.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

INSPECTION COPY

COPYRIGHT PROTECTED

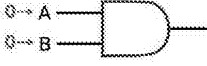
Zig Zag Education

4

Truth Table

All the possible outcomes of a logic diagram are shown in a truth table.

This is the truth table for the OR Gate.



Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

INSPECTION COPY

COPYRIGHT PROTECTED

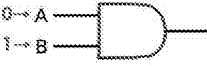
Zig Zag Education

5

Truth Table

All the possible outcomes of a logic diagram are shown in a truth table.

This is the truth table for the AND Gate.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

INSPECTION COPY

COPYRIGHT PROTECTED

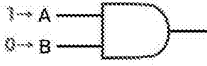
Zig Zag Education

6

Truth Table

All the possible outcomes of a logic diagram are shown in a truth table.

This is the truth table for the OR Gate.



Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

INSPECTION COPY

COPYRIGHT PROTECTED

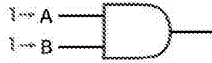
Zig Zag Education

7

Truth Table

All the possible outcomes of a logic diagram are shown in a truth table.

This is the truth table for the



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

COPYRIGHT
PROTECTED

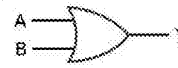
Zig
Zag
Education

INSPECTION COPY

8

Truth Table

Logic Gate



Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

COPYRIGHT
PROTECTED

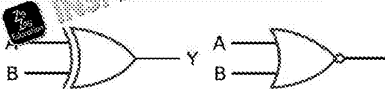
Zig
Zag
Education

INSPECTION COPY

9

Other Gate

There are some other logic gates you



XOR Gate

NOR Gate

Only outputs a 1 if one of the inputs is 1 (not both).

This is equivalent to an OR gate followed by a NOT gate.

COPYRIGHT
PROTECTED

Zig
Zag
Education

INSPECTION COPY

10

Truth Table

Logic Gate



Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

COPYRIGHT
PROTECTED

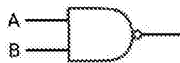
Zig
Zag
Education

INSPECTION COPY

11

Truth Table

NAND Gate



Input A	Input B	Output Y
0	0	1
0	1	1
1	0	1
1	1	0

COPYRIGHT
PROTECTED

Zig
Zag
Education

INSPECTION COPY

12

Boolean Expression

Logic diagrams can also be represented using

Logic Gate		
AND		
OR		
NOT		

COPYRIGHT
PROTECTED

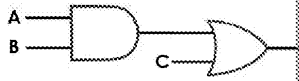
Zig
Zag
Education

INSPECTION COPY

13

Combining G

Logic gates can be combined to build



Boolean Expression: $Y =$

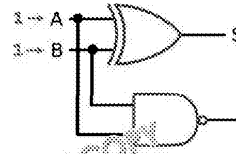
INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

14

Half Adder

The half adder circuit is used to add two numbers, producing a sum and a carry.



$1 + 1 = 0$ carry

INSPECTION COPY

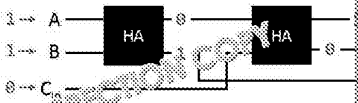
COPYRIGHT
PROTECTEDZig
Zag
Education

15

Full Adder

Two half adders can be joined together along with an OR gate to form a full adder.

A full adder enables the carry from the last addition to be passed forward.



INSPECTION COPY

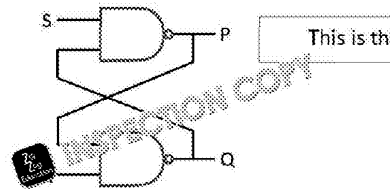
COPYRIGHT
PROTECTEDZig
Zag
Education

16

Flip-Flops

Flip-flops are logic circuits made up of logic gates and memory.

They stay in the same state even after the power is stopped.



This is the

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

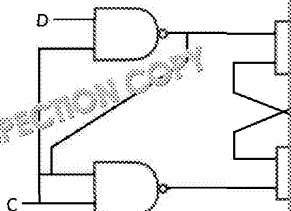
17

D-Type Flip-F

Two flip-flop circuits can be joined together to form a D-type flip-flop.

The D-type flip-flop is used to delay a signal.

It is delayed by one pulse of the clock signal.



INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

1

Boolean Algebra

INSPECTION COPY

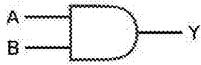
COPYRIGHT
PROTECTEDZig
Zag
Education

Photocopying this resource may only be
permitted by the purchasing institution on a
single site and for their own use.

2

General Ident

When writing out Boolean expressions it is possible to write expressions without changing the order of the inputs.



For



For



it is the OR function it is also possible to write the expression as $(A+B) + C = A + (B+C)$

INSPECTION COPY

COPYRIGHT
PROTECTED

3

Simple AND Ide

Looking at the truth table for an AND gate it is identified.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1



INSPECTION COPY

COPYRIGHT
PROTECTED

4

Simple AND Ide

Looking at the truth table for an AND gate it is identified.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

If both inputs are the same then the output is 1.



INSPECTION COPY

COPYRIGHT
PROTECTED

5

Simple AND Ide

Looking at the truth table for an AND gate it is identified.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

If both inputs are the same then the output is 1.

If you have two inputs then the output is 1.



INSPECTION COPY

COPYRIGHT
PROTECTED

6

Simple AND Ide

Looking at the truth table for an AND gate it is identified.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

If both inputs are the same then the output is 1.

If you have two inputs then the output is 1.

If one of the inputs is 1 then the output is 1.



INSPECTION COPY

COPYRIGHT
PROTECTED

7

Simple AND Ide

Looking at the truth table for an AND gate it is identified.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

If both inputs are the same then the output is 1.

If you have two inputs then the output is 1.

If one of the inputs is 1 then the output is 1.

If one input is 1 then the output is 1.



INSPECTION COPY

COPYRIGHT
PROTECTED

8

Simple OR Identifier

Looking at the truth table for an OR gate, identified.

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

9

Simple OR Identifier

Looking at the truth table for an OR gate, identified.

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

If both inputs are the

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

10

Simple OR Identifier

Looking at the truth table for an OR gate, identified.

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

If both inputs are the

If you have two

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

11

Simple OR Identifier

Looking at the truth table for an OR gate, identified.

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

If both inputs are the

If you have two

If one input determines

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

12

Simple OR Identifier

Looking at the truth table for an OR gate, identified.

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

If both inputs are the

If you have two

If one input determines

If one input is

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

13

Multiplying Expressions

Sometimes you will need to multiply out brackets in the expression.

Multiplying brackets in Boolean algebra is the same as in GCSE Maths.

$$(A+B) \cdot (C+D) = A \cdot C + A \cdot D + B \cdot C + B \cdot D$$

First AND First

First AND Second

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

More Advanced Sim

Take the expression $A + B$
it can be rewritten as $A.1 + A.B$.

This can be rewritten as $A.(1 + B)$
(as A appears in both parts of the expression)

The bracketed part simplifies to 1 (as anything OR'd with 1 gives 1)
giving $A.1$ which simplifies to A .

Similarly, $A + (A.B)$ can be expanded
(by multiplying each part in the bracketed part by A)

The first part simplifies to A (anything AND'd with itself gives itself)
leaving $A + A.B$ which, as above, simplifies to A .

INSPECTION COPY

COPYRIGHT PROTECTED



De Morgan's Laws

When designing a circuit it is usually cheaper to use a single gate, so it's best to be able to redesign systems using only AND or only OR gates.

We use De Morgan's Laws to change between AND and OR functions.

To change OR to an AND function

Change the operator to an AND

Invert each part of the expression

or invert the whole expression

De Morgan's Laws: $\overline{A+B} = \overline{A}.\overline{B}$ and $\overline{A.B} = \overline{A} + \overline{B}$

INSPECTION COPY

COPYRIGHT PROTECTED



Checking for Simp

It's always a good idea to try to get your expression as simple as possible. Make sure there are no unnecessary terms.

A	Input B	(B.A)	B + (B.A)
0	0	0	0
0	1	0	0
1	0	1	1
1	1	1	1

The table shows that the expression $B + (B.A)$ is equivalent to B .

The result is B .

As can be seen from the table, the expression $B + (B.A)$ is equivalent to B .

INSPECTION COPY

COPYRIGHT PROTECTED



Assembly Language

INSPECTION COPY

COPYRIGHT PROTECTED



Machine Code

Each type of CPU is designed to carry out a set of instructions. Each instruction is represented by a unique number. This is called machine code.

Machine Code
1110 0001 1010 0000 0011 0000 0000 1001
1110 0010 0100 0011 0001 0000 0000 1010
1110 0000 1000 0000 0000 0000 0000 0001

It is hard for humans to read and write machine code. A language was developed to make it easier.

A machine language features a set of mnemonics. Each mnemonic represents one machine code instruction.

INSPECTION COPY

COPYRIGHT PROTECTED



Mnemonic

These are the main mnemonics that are commonly used.

Mnemonic	Description
LDR	Load from memory location into a register
ADD	Add data in a register
SUB	Subtract data in a register
STR	Store values from register into a memory location
B	Branch to a marked position in the program
CMP	Compare values in a register
AND	Bitwise logical AND operation
ORR	Bitwise logical OR operation
EOR	Bitwise logical XOR operation
MVN	Bitwise logical NOT operation
LSL	Logically shift the value left, using the carry flag
LSR	Logically shift the value right, using the carry flag
HALT	Stops the program

INSPECTION COPY

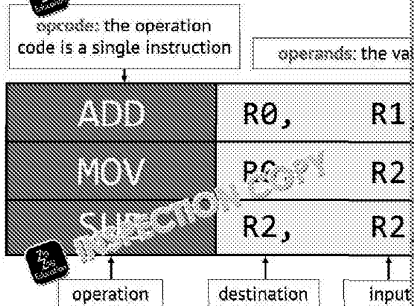
COPYRIGHT PROTECTED



4

Instruction Format

This is the standard format for writing instructions in assembly language.



INSPECTION COPY

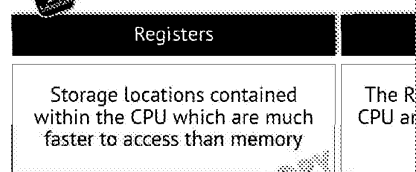
COPYRIGHT
PROTECTED

Zig
Zag
Education

5

Storage

Two different types of storage are used with instructions.



INSPECTION COPY

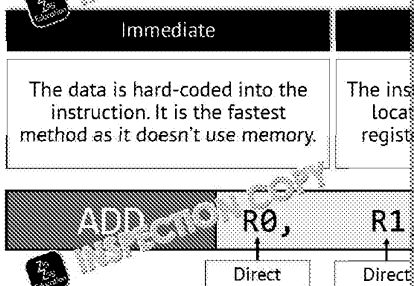
COPYRIGHT
PROTECTED

Zig
Zag
Education

6

Addressing

There are different methods to access data and these are called memory addressing.



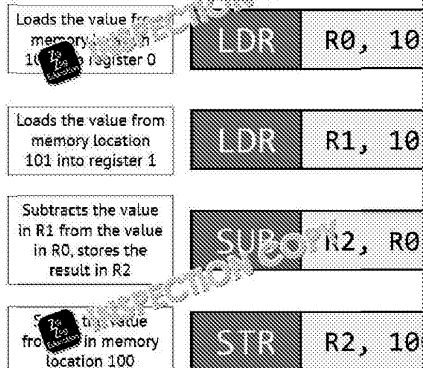
INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

7

Example



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

8

Logical Shift

Multiplication and division are very CPU intensive. Logical shift is an alternative method for multiplication and division by powers of 2.

0	0	1	0	1	1	0
0	1	0	1	1	0	1
1	0	1	1	0	1	0

As you can see, shifting the number one place to the left is equivalent to multiplying by 2 and shifting one place to the right is equivalent to dividing by 2.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

9

Branching

Branching is a way to jump to different parts of the program.

Instruction	Operands	Description
MOV	R0, #50	The value 50 is moved into register 0
MOV	R1, #0	The value 0 is moved into register 1
CMP	R0, R1	The values stored in registers 0 and 1 are compared
BGT	numIsGT	If the value stored in register 0 is greater than the value stored in register 1, the program branches to the label numIsGT
STR	R1, 100	Stores the value in register 1 into memory location 100
HALT		Stops the program
numIsGT:		A label used to define a point in the program
STR	R0, 100	Stores the value stored in register 0 into memory location 100
HALT		Stops the program

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

1

INSPECTION COPY

Relational Data

INSPECTION COPY

A Level Only

Photocopiable/digital resources may only be copied by the purchasing institution on a single site and for their own use.

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

2 Database

Databases are used to store data in a structured way. This is the basic structure of a database.

Primary Key
A field that uniquely identifies each record. Ensures all records are unique.

Table
Data is stored in tables.

student_id	First Name
001	Lois
002	Mel
003	Eloise
004	Nicola

Sometimes two fields are used together to uniquely identify a record. This is known as a composite key.

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

3

Relational Database

The most commonly used database is a relational database.

It allows data to be organised in a way that is easy to process.

In a relational database, data is separated into tables. Each one stores data relating to a single entity.

An entity is something in the real world that can be identified in a database; for example, a product.

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

4

Foreign Key

A foreign key is a primary key from another table that is used to link the two tables together.

Student_id	Firstname	Lastname	Teacher_id
001	Alex	Bennett	002
002	James	Hadwen	001
003	Eloise	Robert	002
004	Patrick	Mc Brown	003

In this example the **teacher_id** field is used to link the students to their teachers.

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

5

Entity Relationship

We define the relationship between tables using entity relationship diagrams.

These are the most common types of relationship:

Relationship Type	Symbol
One-to-one	
One-to-many	
Many-to-many	

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

Examples

6

Each student can only have one form group
have multiple students

Form

Each teacher has one form and each form has multiple students

Teacher

Each teacher has multiple students and each student has multiple teachers.

Teacher

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

1

Structured Query Language (SQL)

A Level Only

Photocopiable/digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

2

Structured Query Language (SQL)

Structured query language (SQL) is the language used to communicate with databases. Here is an example of a SQL statement:

```
SELECT FirstName FROM Student WHERE Gender = 'M'
```

Fields to show The table

This statement selects the **FirstName** field from the **Student** table and only shows the male students. Consider the following table:

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	1
3	Sarah	Smith	60

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

3

Results

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	1
3	Sarah	Smith	60

SELECT FirstName FROM Student WHERE Gender = 'M'

Fields to show The table

The following are the results of the query:

FirstName
John
Ben

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

4

Comparison Operators

The following are the comparison operators used in SQL:

Comparison operator
Equal to (=)
Less than (<)
Greater than (>)
Less than or equal to (<=)
Greater than or equal to (>=)
Not equal to (≠)

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

5

Combining Conditions

StudentID	FirstName	LastName	Age
1	John	Curtis	55
2	Ben	Jackson	75
3	Sarah	Smith	80

We can use the **AND** operator to combine conditions in a SQL statement:

```
SELECT FirstName FROM Student WHERE Gender = 'M' AND Age > 50
```

These are the results of the query:

FirstName
Ben

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

6

Inserting a Record

A new record can be added to a table using the **INSERT** statement:

```
INSERT INTO Student (FirstName, LastName, Age) VALUES ('Eloise', 'Roberts', '9')
```

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	1
3	Sarah	Smith	60
4	Eloise	Roberts	9

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

7

Deleting a Record

Records can be deleted from a table using the DELETE statement.

```
DELETE FROM Student WHERE StudentID = 4
```

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	11
4	Eloise	Roberts	9

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

8

Updating a Record

Records can be updated using the UPDATE statement.

```
UPDATE Student
SET Address = "45 Cleveley Road"
WHERE StudentID = 4
```

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	11
4	Eloise	Roberts	9

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

9

Multiple Tables

Data can be retrieved from multiple tables using the JOIN statement.

```
SELECT FirstName, LastName, TeacherID
FROM Student, Teacher
WHERE Student.TeacherID = Teacher.ID
```

StudentID	FirstName	LastName	TeacherID
001	Alex	Bennett	002
002	James	Hadwen	001
003	Eloise	Roberts	002
004	Patrick	Johnson	003

The JOIN keyword is used to join the tables.

Dot notation is used to specify which table the field belongs to.

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

10

Sorting Results

The results of a query can be sorted using the ORDER BY clause.

```
SELECT FirstName FROM Student
ORDER BY LastName
```

This will order the results by the students' last names.

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

11

Creating a Table

A new table can be created using the CREATE TABLE statement.

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY NOT NULL,
  FirstName VARCHAR(20),
  LastName VARCHAR(20),
  Gender VARCHAR(1)
)
```

NOT NULL means the field can't be empty.

CHAR is a string with a variable length. The value in the brackets is the maximum length.

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

1

Database Normalisation

A Level
Only

INSPECTION COPY

COPYRIGHT
PROTECTEDZig
Zag
Education

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use.

2 Normalisation

Normalisation is the process of structuring data to ensure that it is stored efficiently.

Key Terms

Data Redundancy This is when fields are repeated in different records.

Atomic Level Data stored at an atomic level is not further decomposed; for example, the fields **Title**, **First Name**, and **Last Name**.

Normal Form A database that has been normalised to a certain level.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

3 First Normal Form

There are different levels of normalisation requirements.

Requirements

There is no repeated data and only one value per field.

The data is atomic.

Each record is unique.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

4 1NF Example

Here is an example of repeated data. There are two email addresses (the same) for John.

StudentID	Firstname	Lastname	Email
1	John	Curtis	jcurtis@hotmail.com
2	Ben	Jackson	bjackson@me.com

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

5 Second Normal Form

Requirements

Meets the requirements for 1NF.

All non-key attributes should depend on all key attributes. This can be achieved by creating separate tables.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

6 2NF Example

This **Student** table contains data about each student. The **First Name** and **Last Name** fields don't depend on the **StudentID**.

StudentID	Firstname	Lastname
001	Alex	Bennett
002	James	Hadwen
003	Eloise	Roberts
004	Patrick	McGowan

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

7 2NF Example

This **Student** table contains data about each student. The **First Name** and **Last Name** fields don't depend on the **StudentID**.

They should therefore be in a separate table.

StudentID	Firstname	Lastname
001	Alex	Bennett
002	James	Hadwen
003	Eloise	Roberts
004	Patrick	McGowan

TeacherID	First Name	Last Name
001	Mr	Ford
002	Mr	Smith
003	Mrs	Patel

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Third Normal Form

Requirements

Meets the requirements for

Non-key attributes should not depend on

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

3NF Exam

The *FormP* field in the **Teacher** table d

StudentID	FirstName	LastName	Age
001	Alex	Bennett	
002	James	Hadwen	
003	Eloise	Roberts	
004	Patrick	Dua-Parm	

TeacherID	TeacherName	FormP	FormS
001		Ford	75FD
2	Mr	Smith	7ASM
	Mrs	Patel	7APT

COPYRIGHT
PROTECTED

Zig
Zag
Education

3NF Exam

The *FormP* field in the **Teacher** table d

To solve this we can create a separate

StudentID	FirstName	LastName	Age
001	Alex	Bennett	
002	James	Hadwen	
003	Eloise	Roberts	
004	Patrick	Dua-Parm	

TeacherID	TeacherName	FormP	FormS
001		Ford	75FD
2	Mr	Smith	7ASM
	Mrs	Patel	7APT

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education