1.1 Data Representation Notes

Candidates should be able to:

- Show understanding of binary magnitudes and difference between binary prefixes and decimal prefixes.
- Show understanding of the basis of different number systems.
- Perform binary addition and subtraction Using positive & negative binary integers
- Describe practical applications where BCD and Hexadecimal are used
- Show understanding of and be able to represent character data in its internal binary form, depending on the **character set** used
- Understand difference between and use:
  - kibi and kilo
  - mebi and mega
  - gibi and giga
  - tebi and tera
- Use binary, denary, hexadecimal number bases and BCD and one's and two's complement representation for binary numbers
- Convert an integer value from one number base /representation to another.
- Show understanding of how overflow can occur
- Familiar with ASCII, extended ASCII and Unicode.

Number System

"Number system is a way of representing **amounts** or **quantity** of something"

Types of Number System

Decimal Number System

Number system that we use in our day-to-day life is decimal **(denary)** number system. It has **base** 10 as it uses 10 digits from 0 to 9.

In decimal number system, successive positions to the left of decimal point represent units, tens, hundreds, thousands and so on. Each position represents a specific power of the base (10).

For example,

$= (8x10^3)+ (2x10^2)+ (7x10^1)+ (4xl0^0)$

$= (8x1000)+ (2x100)+ (7x10)+ (4xl)$

$= 8000 + 200 + 70 + 4$

$= 8274$

Binary Number System

Binary number system builds numbers from elements called **bits**. Each bit can be represented by any two mutually exclusive states **1** and **0**. Base of binary number system is 2.In **binary system** we have a one's column, a two's columns, a four's column, an eight's column, and so on, as illustrated below.

32	16	8	4	2	1
2^5	2^4	2^3	2^2	2^1	2^0

1.1 Data Representation Notes

Denary to Binary Conversion:

To convert denary number to binary number, repeatedly divide quotient by the base 2, until the quotient is one, making note of the remainders at each step. Then, write remainders in reverse, starting at the bottom and appending to the right each time.

Example: Find Binary equivalent of decimal number 203.

Quotient		Remainder	
$203_{10} \div 2 =$	101	1	
$101_{10} \div 2 =$	50	1	↑
$50_{10} \div 2 =$	25	0	↑
$25_{10} \div 2 =$	12	1	↑
$12_{10} \div 2 =$	6	0	↑
$6_{10} \div 2 =$	3	0	↑
$3_{10} \div 2 =$	1	1	↑

Reading remainder from bottom to Top and appending to right each time gives $(11001011)_2$. This is binary equivalent of $(203)_{10}$.

Binary to Denary System:

Convert following binary number, 0011000111100110, into a denary number.

32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	1	1	1	1	0	0	1	1	0

To convert this number to denary, each time a 1 appears in a column, column value is added to total: 8192 + 4096 + 256 + 128 + 64 + 32 + 4 + 2 = 12 774

Hexadecimal Number System

Hexadecimal Number System consists of **16 digits** from 0 to 9 and A to F. Alphabets A to F represent decimal numbers from 10 to 15 i.e (A = 10, F = 15.)

Base of this number system is 16. Each **digit position** in hexadecimal system represents a power of **16**.

65 536	4 096	256	16	1
(16^4)	(16^3)	(16^2)	(16^1)	(16^0)

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Nibble: Term given to a group of **four bits**.

1.1 Data Representation Notes

Conversion of Hexadecimal to Binary Number

Convert hex number (4 5 A) in to binary number

Using Table, find the 4-bit code for each digit:

4=0 1 0 0 , **5**= 0 1 0 1 , **A**= 1 0 1 0

Put the groups together to form the binary number:

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

Conversion from binary to Hexadecimal System

To convert from binary to hexadecimal, start from right and moving left, split binary number into groups of 4 bits.

If last group has less than 4 bits, then simply fill in with 0s from left. Take each group of 4 bits and convert it into equivalent hexadecimal digit.

Convert the binary number 10110101 to a hexadecimal number.

Step #1 First split this up into groups of 4 bits: 1011 0101

Step #2 Then, using above Table find the equivalent hexadecimal digits

1011= B and 0101 = 5

Hence hexadecimal equivalent of 10110101 is B5

Conversion Hexadecimal to Denary Number

To convert hexadecimal number to denary, take each hexadecimal digit and multiply it by its value. Add the totals together to obtain the denary value.

Example: Convert hex 45A into denary system.

First multiply each digit by its value:

256	16	1	
(4 × 256 = 1024)	(5 × 16 = 80)	(10 × 1 = 10)	(Note: A = 10)

Then add the total together . 1024 + 80 +10 =

Denary number = **1 1 1 4**

Note: A binary number is odd if its right most digit is one.

Conversion from Denary to Hex

To convert from denary to hexadecimal involve repetitive division by 16. The remainders are then read from **BOTTOM** to **TOP** to give the hexadecimal value.

1.1 Data Representation Notes

16	2004	
16	125	remainder: 4
16	7	remainder: 13
	0	remainder: 7

read the remainder from bottom to top to get the hexadecimal number:
7 D 4

Everything you see on a computer, images, sounds, games, text, videos, etc. Whatever it is, it will be stored as a string of **ones** and **zeroes.**

Internal Coding of Integer

Coding used in a computer system is based on bits being grouped together with **eight bits** representing a **byte**. **Right-hand bit** is referred to as the **least significant** bit and the **left-hand bit** as the **most significant** or top bit.

Bits in a byte are numbered **right to left** starting at bit **0** and ending at bit **7**.

Coding Unsigned Integer:

Unsigned integer can be stored as binary number. Only decision to be made is how many **bytes** should be used. If choice is to use two bytes (16 bits) then range of values that can be represented is 0 to 2^{16} - 1 which is **0** to **65535**.

Coding Signed Integer:

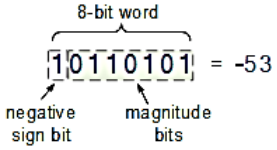
To represent signed integer we use following method.

Sign and Magnitude

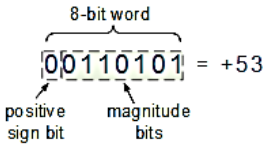
In this method, bit at the far left of the bit pattern (sign bit) - indicates whether number is positive or negative. Rest of bits in pattern store **magnitude** of the number.

Negative numbers are represented using sign and magnitude or two's complement.

Negative Signed Binary Numbers



Positive Signed Binary Numbers



Smallest possible number using sign and magnitude method representation is -127 (or 11111111) and the largest possible number is +127 (or 01111111).

Disadvantage: Sign and magnitude approach is ineffective due to requirement to compare sign and magnitude before performing addition or subtraction, leading to

slow execution of these operations.

Two's complement Representation

Two's complement method is a binary representation technique used in computing for the representation of **signed integers.**

Advantage of Two Complement Number:

- One advantage of two's complement method is that addition and subtraction of signed numbers become simplified.
- Arithmetic operations can be performed using same hardware for both positive and negative numbers, making it an efficient representation in computer architecture.

One's complement:

It is defined as the binary number obtained if each binary digit is individually subtracted from 1 which, in practice, means that each 0 is switched to 1 and each 1 switched to 0.

Example: Find 1's complement of binary number 10001001.
Invert each bit of binary number, so 1's complement of given number will be 01110110.

- **Uses of 1's Complement Binary Numbers:**

1's complement is mainly use in signed Binary number representation and arithmetic operations for Binary numbers, e.g., additions, subtractions, etc

- **Two's complement:**

Two's complement is defined as binary number obtained if 1 is added to one's complement number.

Alternative Method of two complement:

Start from least significant bit and move left ignoring any zeros up to first 1 which is also ignored. Any remaining bits are then changed from 0 to 1 or from 1 to 0.
Example,
Expressing number 10100100 in two's complement form, leaves right hand 100 unchanged then the remaining 10100 changes to 01011 so the result is 01011100.

- **Representing Positive denary as equivalent two's complement form:**
  - Convert the denary value to a binary value.
  - Add a 0 in front of this binary value.
- **Representing Negative denary as equivalent two's complement form:**

i. Find positive binary value for negative number.
ii. Add a 0 to front of the number, to indicate that it is positive.
iii. Then find two complement.

Represent - 4 in two's complement form
1. 4 = 100
2. Adding 0 to the front becomes 0100
3. Find two complement of 0100 i.e 1100 (-8 + 4 = -4)

- **Converting two's complement binary representing Positive value into denary:**
  Leading zero is ignored and then convert the remaining binary into denary number.

- **Converting two's complement binary number representing a negative number into a denary value**

Example
Converting negative number expressed in two's complement form to denary number. Consider the two's complement binary number 10110001.
Solution: Sum the individual position values but treat the most significant bit as a negative value From the original binary number 10110001 this produces the following: -128 + 0 + 32 + 16 + 0 + 0 + 0 + 1 = **- 79**.

Differences between sign magnitude representation and two's complement form.

Signed denary number to be represented	Sign and magnitude representation	Two's complement representation
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
−0	1000	Not represented
−1	1001	1111
−2	1010	1110
−3	1011	1101
−4	1100	1100
−5	1101	1011
−6	1110	1010
−7	1111	1001
−8	Not represented	1000

Points to note about two's Complement

- There is only **one** representation of zero.
- Starting from **lowest negative value**, each successive higher value is obtained by **adding 1** to the binary code. In particular, when all digits are 1 the next step is to **roll over** to an **all-zero** code..
  Example: Adding 1 to negative value of (-7) you will get value of 6 and so on.
- Just adding a leading zero to an unsigned binary value converts it to two's complement representation of corresponding positive number.
- You use a two's complement conversion to change sign of a number from positive to negative or from negative to positive. We say that the two's

complement values are **self-complementary**.

- We can add any number of leading **zeros** to representation of **Positive** value without changing value.
- We can add any number of leading **ones** to representation of **Negative** value without changing value.

Lowest and highest integer values that is possible to represent in two's complement using a **8 bit** is **-128** and **+127** respectively.

Binary Arithmetic

Binary Addition:

For binary addition, starting at the **least significant** position applies.

Rules for the addition of binary digits are:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 1 = 0 with a carry of 1
- 1 + 1 + 0 = 0 with a carry of 1
- 1 + 1 + 1 = 1 with a carry of 1

		1	0	1	1
+		1	1	1	0
	1	1	0	0	1

Example: Add 0 0 1 0 0 1 0 1 (37 in denary) and 0 0 1 1 1 0 1 0 (58 in denary)

0	0	1	0	0	1	0	1
			+				
0	0	1	1	1	0	1	0
			=				
0	1	0	1	1	1	1	1

This gives us 0 1 0 1 1 1 1 1, which is 95 in denary; the correct answer.

Binary Subtraction:

Starting at the least significant position applies. Rules for subtraction of binary digits are:

- 0– 0 = 0
- 0 – 1 = 1 after a borrow
- 1– 0 = 1
- 1– 1 = 0

Example:

Subtraction of binary equivalent of denary 11 from binary equivalent of denary 14 can be done as.

	1	1	1	0
–	1	0	1	1
	0	0	1	1

Steps followed from right to left are:

- 1 is larger than 0 so 1 is borrowed giving subtraction of 1 from 10 leaving 1
- Because of borrow the 1 is reduced to 0 so that 1 is to be subtracted from 0. This requires a further borrow giving subtraction of 1 from 10 leaving 1
- Because of the borrow the 1 is reduced to 0 leaving subtraction of 0 from 0
- 1– 1 gives 0 Answer is the binary value for denary 3.

Binary Subtraction Alternative Method:

To carry out subtraction in binary, we convert the number being subtracted into its negative equivalent using two's complement, and then add the two numbers.

Example: Carry out the subtraction 95 – 68 in binary.

Step 1: Convert the two numbers into binary:

95 = 0 1 0 1 1 1 1 1

68 = 0 1 0 0 0 1 0 0 2

Step 2: Find the two's complement of 68:

invert the digits:	1	0	1	1	1	0	1	1
add 1:								1
which gives:	1	0	1	1	1	1	0	0 = −68

Step 3: Add 95 and −68:

	0	1	0	1	1	1	1	1
					+			
	1	0	1	1	1	1	0	0
					=			
1	0	0	0	1	1	0	1	1

Additional ninth bit is simply ignored leaving binary number 0 0 0 1 1 0 1 1 (denary equivalent of 27, which is correct result of subtraction).

Over Flow

A CPU with a capacity of 8 bits has a capacity of up to 11111111 in binary. If one more bit was added there would be an **overflow error.**

Example: 8-bit Overflow

8-bit overflow occurs in the binary sum 11111111 + 1 (denary: 255 + 1).

```
  1 1 1 1 1 1 1 1
+ 0 0 0 0 0 0 0 1
-----------------
1 0 0 0 0 0 0 0 0
```

Total is number bigger than 8 digits, and when this happens CPU drops overflow digit because the computer cannot store it anywhere, and computer thinks 255 + 1 = 0.

Most CPUs use much bigger word size than 8 bits. Many PC have 64-bit CPU. 64-bit CPU can handle number larger than 18 quintillion (18,446,744,073,709,551,615 to be precise).

Binary Coded Decimal

BCD is an **encoding** for **decimal** numbers in which each digit is represented by its own **binary** sequence. In BCD we can use the **binary number** from 0000-1001 only, which are the decimal equivalent from 0-9 respectively. Example

0 0 0 0 = 0	0 1 0 1 = 5
0 0 0 1 = 1	0 1 1 0 = 6
0 0 1 0 = 2	0 1 1 1 = 7
0 0 1 1 = 3	1 0 0 0 = 8
0 1 0 0 = 4	1 0 0 1 = 9

Suppose if a number have **single** decimal digit then it's equivalent BCD will be the respective **four binary** digits of that **decimal number**.

Note: If a denary number with more than one digit is to be converted to BCD, there are **two** options for **BCD**;

Method # 1:

We store one BCD code in **one byte** leaving **four bits unused**.

Example: BCD representations of the denary digits **8503**:

00001000	00000101	00000000	00000011

Method # 2:

In this method, we **packed BCD** where two **4-bit codes** are stored in one byte.

Example:

BCD representations of the denary digits 8503:

10000101	00000011

BCD Application and Justification: 9618/12/O/N/23

- Application that performs financial or **banking calculations** because it is difficult to represent decimal values exactly in normal binary and financial transactions use only two decimal places and must be accurate, no accumulating errors.
- **Electronic displays**, e.g. calculators, digital clocks because visual displays only need to show individual digits and because conversion between denary and BCD is easier
- Storage of **date and time** in BIOS of a PC because conversion with denary is easier.

BCD Addition

To perform addition in BCD, you can first add-up in binary format, and then perform conversion to BCD afterwards. This conversion involves adding 6 to each group of four digits that has a value of greater-than 9. **For example**:

9 + 5 = 14 in **denary**.

[1001] + [0101] = [1110] in **binary**.

However, in BCD, we cannot have value greater-than 9 (1001) per-nibble. To correct this, add 6 (0110) to that answer:

[0000 1110] + [0000 0110] = [0001 0100]

which gives us two nibbles, [0001] and [0100] which correspond to "1" and "4" respectively. This gives us the 14 in BCD which is the correct result.

Correction factor.

To counteract this in BCD arithmetic, processor recognize that an impossible value has been produced and apply a method to remedy this. Remedy is to add **0110** whenever problem is detected.

For 0 to 9 decimal numbers both binary and BCD is equal but when decimal number is more than one digit, BCD differs from binary.

Internal coding of text

ASCII Code:

ASCII stands for **American Standard Code for Information Interchange.** They are used to **encode** alphanumeric data in computers and electronic devices to represent input and output in a more scientific manner.

ASCII represent **2^7 = 128** codes as it is a **seven bit** code.

Extended ASCII-8:

Latest development in field of ASCII code is development of **8 bit** code which is known as ASCII-8. As it is an 8 bit code it can represent **2^8 = 256** characters.

Character Set: A defined list of characters recognized by the computer hardware and software. Each character is represented by a number and take one byte. ASCII character set uses the numbers 0 through 127 to represent all English characters as well as special control characters.

- The ASCII code for “A” and “a” in denary number System: A = 65 • a = 97

Unicode and ASCII both are standards for encoding texts.

Boolean value is stored in a computer in the form 0 or 1 **OR** a byte of all 0 or a byte of all 1. Computer don’t accept Yes or No.

- Sixth bit changes from 1 to 0 when comparing **lower** and **uppercase** characters. This makes **conversion** between the two an **easy** operation.
- Character sets (such as a to z, 0 to 9, and so on) are grouped together in sequence, which speeds up **usability**.

Unicode

Unicode allows characters in a code form to represent all languages of world, thus supporting many operating systems, search engines and internet browsers used globally.

Unicode overlap with standard ASCII code, since the first 128 (English) characters are the same. Unicode can support 65,536 different characters in total as it take **two bytes** to store each character.

Code Point:

Unicode has its own special terminology and symbolism. For example, a character code is referred to as a 'code point'.

Code point is identified by **U+** followed by a **4-digit** hexadecimal number.

Example: U+0041 which is **code point** corresponding to alphabetic character **A**. The **0041** are hexadecimal characters representing **two** bytes.

Note: Code points **U+0000** to **U+00FF** define characters which are a duplicate of those in standard ASCII.

Important Points about Unicode:

- Unicode create a **universal standard** that covered all languages and all writing systems.
- Produce a more efficient coding system than ASCII
- Adopt uniform encoding where each character is encoded as 16-bit or 32-bit code.
- Create unambiguous encoding where each 16-bit or 32-bit value always represents same character
- Reserve part of code for private use to enable a user to assign codes for their own characters and symbols.

Similarity between ASCII and Unicode are; 9618/O/N/22/P11

- Both represent each character using a unique code
- Unicode will contain all characters that ASCII contains // ASCII is a subset of Unicode.

1.1 Data Representation Notes

Difference between ASCII and Unicode are; 9618/O/N/22/P11

- Unicode can go up to 32 bits per character whereas ASCII is 7 or 8 bits
- Unicode can represent a wider range of characters than ASCII
- different languages are represented using Unicode, ASCII is only for one language

A **problem** arises when the computer retrieves a piece of data from its memory. Imagine that the data is 01000001. Is this the number 65, or is it A?

They are both stored in the same way, so how can it tell the difference?

The answer is that characters and numbers are stored in different parts of the memory, so it knows which one it is by knowing where about it was stored.

Decimal Prefix

A **prefix** to define the magnitude of a value. Examples are kilo(K), mega (M) , giga (G) , tera (T) and peta (P) representing factors of 10^3 , 10^6 , 10^9 , 10^{12} and 10^{15} respectively.

Binary Prefix

A **prefix** to define the magnitude of a value. Examples are kibi (Ki), mebi (Mi) , gibi (Gi) , tebi (Ti) and pebi (Pi) representing factors of 2^{10} , 2^{20} , 2^{30} , 2^{40} and 2^{50} respectively.

System of Units (SI)			Binary Numeral			
Factor	Name	Symbol	Factor	Name	Symbol	# of Bytes
10^3	kilobyte	KB	2^{10}	kibibyte	KiB	1,024
10^6	megabyte	MB	2^{20}	mebibyte	MiB	1,048,576
10^9	gigabyte	GB	2^{30}	gibibyte	GiB	1,073,741,824
10^{12}	terabyte	TB	2^{40}	tebibyte	TiB	1,099,511,627,776
10^{15}	petabyte	PB	2^{50}	pebibyte	PiB	1,125,899,906,842,624
10^{18}	exabyte	EB	2^{60}	exbibyte	EiB	1,152,921,504,606,846,976
10^{21}	zettabyte	ZB	2^{70}	zebibyte	ZiB	1,180,591,620,717,411,303,424
10^{24}	yottabyte	YB	2^{80}	yobibyte	YiB	1,208,925,819,614,629,174,706,176

ESQ: State one difference between a kibibyte and a megabyte 9618/12/O/N/23

- kibibyte is binary prefix and megabyte is denary prefix
- kibibyte = 1,024 bytes // 2^{10} bytes while megabyte = 1000 kilobytes //1 000 000 bytes // 10^3 kilobytes // 10^6 bytes

ESQ: Difference and similarity between Kilobyte and Kibibyte:

Difference: Kilobyte is a decimal unit of measurement equal to 1000 bytes. kibibyte is a binary unit of measurement equal to 1024 bytes.

Similarity: Kilobyte (KB) and kibibyte (KiB) are both units of digital information storage.

1.1 Data Representation Notes

Measurement of the size of Computer Memories

- **Convert 34 560 bytes into kibibytes**

$$344560\text{B} = \frac{34560}{1024}\text{KiB} = 33.75\,\text{KiB}$$

- **Convert 34 56000 bytes into mebibytes:**

$$3456000\text{B} = \frac{\left(\frac{3456000}{1024}\right)}{1024}\text{MiB} = 3.296\,\text{MiB}$$

Example: How many mp3 files of size 2.4 MiB could be stored on a 4 GiB USB?

Solution:

$$\frac{(4 \times 1024)\text{MiB}}{2.4\text{MiB}} = 1076$$

****

Exam Style Question

Q#1 Find how -28 would be expressed in two's complement notation?

Solution: First we write out 28 in binary form. **0 0 0 1 1 1 0 0**

Then we invert the digits. 0 becomes 1, 1 becomes 0. **1 1 1 0 0 0 1 1**

Then we add 1. **1 1 1 0 0 1 0 0**

That is how one would write -28 in 8 bit binary.

****