Grade 12 A/L ICT

Lesson 3: Data Representation

- 3.1 Number Systems and Signed Numbers: Binary, Octal, Hex and 2’s Complement
- 3.2 Character Representation: BCD, EBCDIC, ASCII and Unicode
- 3.3 Binary Arithmetic and Bitwise Logic: Add, Subtract, AND, OR, XOR, NOT

© 2026 schoolict.net (Ruwan Mangala Suraweera)
All rights reserved.

Grade 12 A/L ICT · Lesson 3

3.1 Number Systems and Signed Numbers: Binary, Octal, Hex and 2's Complement

Syllabus 3.1. A computer stores **everything as 0s and 1s**. In this part you learn **why only two states** are used, the four **number systems** (decimal, binary, octal, hexadecimal), how to **convert** between them step by step, and how a computer stores **negative numbers** using **sign-magnitude, 1's complement and 2's complement**, where the **most significant bit (MSB)** is the sign bit.

1. Why computers use two states

Inside a computer, every circuit is like a **switch**: it is either **ON or OFF**. A wire has **high voltage or low voltage**, a disk spot is **magnetised one way or the other**, and a CD spot is a **pit or a land**. We write these two states as **1 and 0**. One 0 or 1 is called a **bit** (binary digit).

- Two states are **easy to make and detect** – the circuit only has to tell high from low.
- They are **reliable**: a small amount of electrical noise does not change a 1 into a 0.
- They match **Boolean logic** (true/false), so circuits can do arithmetic and decisions.
- Groups of bits: **4 bits = nibble**, **8 bits = byte**. With n bits you get **2^n different patterns**.

🧱 **Worked example: How many different patterns can be made with 8 bits? With 16 bits?**
8 bits: 2^8 = **256** patterns (0 to 255). 16 bits: 2^{16} = **65,536** patterns.

2. The four number systems

Every number system has a **base** (radix) – the number of different digits it uses. The value of a digit depends on its **place**: each place is worth **base × the place on its right**.

System	Base	Digits	Place values	Example
Decimal	10	0–9	… 1000, 100, 10, 1	156_{10}
Binary	2	0, 1	… 8, 4, 2, 1	10011100_2
Octal	8	0–7	… 512, 64, 8, 1	234_8
Hexadecimal	16	0–9, A–F (A=10 … F=15)	… 4096, 256, 16, 1	$9C_{16}$

💡 Write the base as a small number after the value, for example **1010_2**, so nobody confuses 10 in binary (two) with 10 in decimal (ten).

Decimal	Binary (4 bits)	Octal	Hex

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

> 💡 Learn this 0–15 table by heart. It is the key to every fast conversion between binary, octal and hex.



How to convert between number systems

3. Decimal to binary, octal and hex – divide and keep the remainders

Method: divide the decimal number by the new base again and again. Write down each **remainder**. Stop when the quotient is 0. Read the remainders **from the bottom to the top**.

> 🧮 **Worked example: Convert 156_{10} to binary.**
> Divide by 2 (table below). Reading the remainders from bottom to top: 156_{10} = **10011100_2**.
> Check: 128 + 16 + 8 + 4 = 156.

Division	Quotient	Remainder

156 ÷ 2	78	0
78 ÷ 2	39	0
39 ÷ 2	19	1
19 ÷ 2	9	1
9 ÷ 2	4	1
4 ÷ 2	2	0
2 ÷ 2	1	0
1 ÷ 2	0	1

🧮 **Worked example: Convert 156_{10} to octal.**
Divide by 8: 156 ÷ 8 = 19 r 4, 19 ÷ 8 = 2 r 3, 2 ÷ 8 = 0 r 2. Bottom to top: **234_8**. Check: 2×64 + 3×8 + 4 = 156.

Division	Quotient	Remainder
156 ÷ 8	19	4
19 ÷ 8	2	3
2 ÷ 8	0	2

🧮 **Worked example: Convert 2026_{10} (this year) to hexadecimal.**
Divide by 16. A remainder of 10 to 15 is written as a letter (14 = E). Bottom to top: 2026_{10} = **$7EA_{16}$**. Check: 7×256 + 14×16 + 10 = 2026.

Division	Quotient	Remainder
2026 ÷ 16	126	10 = A
126 ÷ 16	7	14 = E
7 ÷ 16	0	7

4. Binary, octal and hex to decimal – multiply by place values

Method: write the place value under each digit (1, base, base^2, …from the right). Multiply each digit by its place value and **add** the answers.

🧮 **Worked example: Convert 101101_2 to decimal.**
32 + 0 + 8 + 4 + 0 + 1 = **45_{10}**. (Shortcut: just add the place values under the 1s.)

Digit	1	0	1	1	0	1
Place value	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

Digit × place value	1 × 32 = 32	0 × 16 = 0	1 × 8 = 8	1 × 4 = 4	0 × 2 = 0	1 × 1 = 1

🧮 **Worked example: Convert 347_8 to decimal.**
3×64 + 4×8 + 7×1 = 192 + 32 + 7 = **231_{10}**.

Digit	**3**	**4**	**7**
Place value	$8^2 = 64$	$8^1 = 8$	$8^0 = 1$
Digit × place value	3 × 64 = 192	4 × 8 = 32	7 × 1 = 7

🧮 **Worked example: Convert $2AF_{16}$ to decimal.**
A = 10 and F = 15. 2×256 + 10×16 + 15×1 = 512 + 160 + 15 = **687_{10}**.

Digit	**2**	**A**	**F**
Place value	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
Digit × place value	2 × 256 = 512	10 × 16 = 160	15 × 1 = 15

5. Binary ↔ octal and hex – the grouping shortcut

Because 8 = 2^3 and 16 = 2^4, **one octal digit = 3 bits** and **one hex digit = 4 bits**. Group the bits **from the right**, add 0s on the left if needed, and change each group to one digit. To go back, change each digit to 3 or 4 bits.

🧮 **Worked example: Convert 11010110_2 to octal and to hexadecimal.**
Groups of 3 (pad to 011 010 110): octal **326_8**. Groups of 4 (1101 0110): 1101 = D, 0110 = 6, so hex **$D6_{16}$**. Both equal 214 in decimal.

Group	**011**	**010**	**110**
Value	3	2	6

Group	**1101**	**0110**
Value	D	6

🧮 **Worked example: Convert 725_8 and $3C9_{16}$ to binary.**
Octal: each digit → 3 bits: 7 = 111, 2 = 010, 5 = 101, so **111010101_2**. Hex: each digit → 4 bits: 3 = 0011, C = 1100, 9 = 1001, so **$0011\ 1100\ 1001_2$** (the leading 00 can be dropped).

Digit	**7**	**2**	**5**
Bits	111	010	101

Digit	**3**	**C**	**9**

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

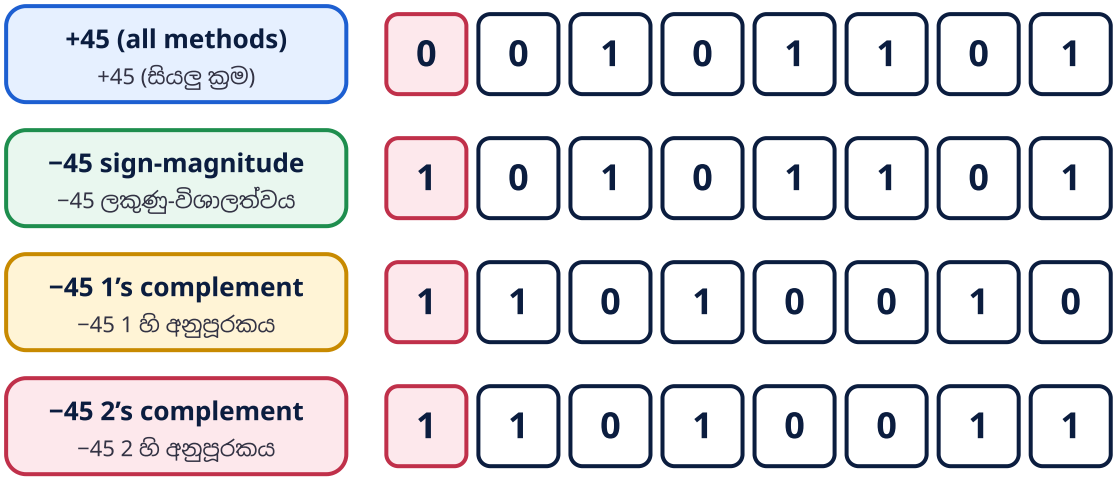
Bits	0011	1100	1001

🧮 **Worked example: Convert 157_8 to hexadecimal, and $2F_{16}$ to octal.**
Go **through binary**. 157_8 → 001 101 111 → regroup in 4s: 0110 1111 → **$6F_{16}$**. $2F_{16}$ → 0010 1111 → regroup in 3s: 000 101 111 → **57_8**.

💡 Never convert octal to hex digit by digit. Always go octal → binary → hex (or hex → binary → octal).

6. Unsigned and signed integers

An **unsigned** integer uses all its bits for the size, so it can only be 0 or positive. 8 bits give 0 to 255. A **signed** integer must also show **+ or −**. The computer uses the **leftmost bit, the most significant bit (MSB)**, as the **sign bit**: **0 = positive, 1 = negative**.



Three ways to store −45 in 8 bits

Sign-magnitude
MSB = sign, the other 7 bits = the size. −45 = **1010 1101**.

1's complement
Write +45, then **flip every bit**. −45 = **1101 0010**.

2's complement
Flip every bit, then **add 1**. −45 = **1101 0011**.

🧮 **Worked example: Represent +45 and −45 in 8-bit sign-magnitude, 1's complement and 2's complement.**
+45 = 0010 1101 in all three (positive numbers look the same). −45: sign-magnitude **1010 1101**, 1's complement **1101 0010**, 2's complement **1101 0011**.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

Step	Bits
1. Write +45 in 8 bits	0010 1101
2. Flip every bit (1's complement)	1101 0010
3. Add 1 (2's complement)	**1101 0011**

💡 **2's complement shortcut:** copy the bits from the right up to and including the **first 1**, then flip all the bits to its left. +45 = 00101101 → 11010011.

🧱 **Worked example: What decimal number is the 8-bit 2's complement value 1110 1100?** MSB is 1, so it is negative. Flip: 0001 0011, add 1: 0001 0100 = 20. So the answer is **−20**. (Or: the MSB is worth −128, so −128 + 64 + 32 + 8 + 4 = −20.)

8-bit method	Smallest	Largest	Zeros	Formula (n bits)
Unsigned	0	255	1	0 to $2^n - 1$
Sign-magnitude	−127	+127	two (+0, −0)	$-(2^{n-1} - 1)$ to $2^{n-1} - 1$
1's complement	−127	+127	two (+0, −0)	$-(2^{n-1} - 1)$ to $2^{n-1} - 1$
2's complement	−128	+127	one	-2^{n-1} to $2^{n-1} - 1$

Why 2's complement wins: it has **only one zero**, it gives one extra negative number (−128), and the **same adder circuit** can add and subtract (you will see this in 3.3). That is why almost every processor uses it.

NEW 2026 **Hex in everyday tech:** web colours are hex – **#1D5FD1** means red 1D (29), green 5F (95), blue D1 (209). **IPv6** addresses are 128 bits written as 8 groups of 4 hex digits, e.g. 2001:0db8:0000:0000:0000:0000:0000:0001.

🚀 Beyond the syllabus (2026 update) *Not required for the exam – read for interest.* **Beyond the syllabus – floating point (IEEE 754).** Fractions such as 3.75 are not stored as integers. Computers use the **IEEE 754** standard: a sign bit, an exponent and a mantissa (like scientific notation in binary). A 32-bit float has 1 sign bit, 8 exponent bits and 23 mantissa bits. 3.75 = $11.11_2 = 1.111 \times 2^1$. AI chips in 2025–2026 also use tiny formats such as **FP8 and FP4** to run large models faster.

Key Words

Bit – The smallest unit of data: a single 0 or 1.

Byte – A group of 8 bits.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

Nibble – A group of 4 bits, equal to one hex digit.

Base (radix) – The number of different digits a number system uses.

Binary – The base 2 number system using only 0 and 1.

Octal – The base 8 number system using digits 0 to 7.

Hexadecimal – The base 16 number system using 0 to 9 and A to F.

Place value – The value of a position, a power of the base.

MSB – The leftmost bit; the sign bit in a signed number.

Unsigned integer – A whole number stored with no sign, so it is zero or positive.

Sign-magnitude – A sign bit followed by the size of the number in binary.

1's complement – A negative number made by flipping every bit of the positive number.

2's complement – A negative number made by flipping every bit and adding 1.

Sample Q&A

1. Why do computers use the binary system?

> **Answer:** Electronic circuits have two stable states (on/off, high/low voltage). Two states are easy to make, easy to detect, reliable against noise, and match Boolean logic.

2. Convert 156_{10} to binary, octal and hexadecimal.

> **Answer:** Binary 10011100_2, octal 234_8, hexadecimal $9C_{16}$.

3. Why is hexadecimal used by programmers?

> **Answer:** One hex digit stands for exactly 4 bits, so long binary values become short and easy to read, and converting back to binary is quick. Examples: memory addresses, colour codes, MAC and IPv6 addresses.

4. What is the MSB and what does it show in a signed number?

> **Answer:** The MSB is the most significant (leftmost) bit. In a signed number it is the sign bit: 0 means positive and 1 means negative.

5. Write −20 in 8-bit 2's complement.

> **Answer:** +20 = 0001 0100; flip = 1110 1011; add 1 = 1110 1100.

6. Give two disadvantages of sign-magnitude.

> **Answer:** It has two zeros (00000000 and 10000000), and normal binary addition gives wrong answers when the signs are different, so extra circuits are needed.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

MCQ

1. Why do computers use binary?

A. It uses ten digits
B. Hex cannot be stored
C. Circuits have two reliable states
D. Binary numbers are shorter than decimal
E. Humans read binary easily

2. What is 156_{10} in binary?

A. 10011010
B. 11001100
C. 10101100
D. 10011101
E. 10011100

3. What is 101101_2 in decimal?

A. 45
B. 46
C. 37
D. 44
E. 54

4. What is $2AF_{16}$ in decimal?

A. 527
B. 671
C. 703
D. 687
E. 2715

5. What is 11010110_2 in hexadecimal?

A. B6
B. C6
C. D6
D. D3
E. 6D

6. What is 11010110_2 in octal?

A. 332
B. 326
C. 623
D. 316
E. 362

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

7. What is $3C9_{16}$ in binary?

A. 1111001001
B. 1111001100
C. 1011001001
D. 1111000110
E. 1100111001

8. What is 157_8 in hexadecimal?

A. 9D
B. F6
C. 6F
D. 6E
E. 157

9. How many bits does one hexadecimal digit stand for?

A. 16
B. 3
C. 8
D. 2
E. 4

10. What is −45 in 8-bit 2’s complement?

A. 00101101
B. 11010011
C. 10101101
D. 11010010
E. 11010100

11. What is −45 in 8-bit sign-magnitude?

A. 11010011
B. 00101101
C. 10101100
D. 11010010
E. 10101101

12. The 8-bit 2’s complement value 11101100 equals:

A. 20
B. 236
C. −108
D. −20
E. −19

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

13. What is the range of an 8-bit 2’s complement integer?

A. −127 to +127
B. −255 to +255
C. 0 to 255
D. −128 to +127
E. −128 to +128

14. Which method has only one representation of zero?

A. 2’s complement
B. Sign-magnitude
C. None of them
D. 1’s complement
E. Both sign-magnitude and 1’s complement

15. What is the largest unsigned number in 8 bits?

A. 255
B. 127
C. 511
D. 128
E. 256

16. In a signed binary number the MSB with value 1 means:

A. There is an overflow
B. The number is positive
C. The number is zero
D. The number is odd
E. The number is negative

17. Which term means: “The smallest unit of data: a single 0 or 1”?

A. Octal
B. Bit
C. Nibble
D. Place value
E. Sign-magnitude

18. Which term means: “A group of 8 bits”?

A. MSB
B. 1’s complement
C. Byte
D. Hexadecimal
E. Base (radix)

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

19. Which term means: "A group of 4 bits, equal to one hex digit"?

A. Place value
B. Nibble
C. Binary
D. 2's complement
E. Unsigned integer

20. Which term means: "The number of different digits a number system uses"?

A. Octal
B. Bit
C. Sign-magnitude
D. Base (radix)
E. MSB

Structured Questions

1. (a) Convert 2026_{10} to binary showing the division. (b) Convert your answer to octal and hexadecimal by grouping. *(3 marks)*

> (a) repeated ÷2, remainders bottom to top = 11111101010_2; (b) groups of 3 = 3752_8; groups of 4 = $7EA_{16}$.

2. Convert to decimal showing place values: (a) 101101_2 (b) 347_8 (c) $2AF_{16}$. *(3 marks)*

> (a) 32+8+4+1 = 45; (b) 3×64+4×8+7 = 231; (c) 2×256+10×16+15 = 687.

3. Represent −45 in 8 bits using (a) sign-magnitude (b) 1's complement (c) 2's complement. *(3 marks)*

> (a) 10101101; (b) 11010010; (c) 11010011.

4. (a) Give the range of 8-bit unsigned and 8-bit 2's complement integers. (b) Give two reasons why 2's complement is preferred. *(4 marks)*

> (a) unsigned 0 to 255; 2's complement −128 to +127; (b) only one zero; the same adder does addition and subtraction.

5. (a) Find the decimal value of the 8-bit 2's complement number 11101100. (b) Convert $2F_{16}$ to octal. *(4 marks)*

> (a) MSB 1 so negative; flip and add 1 gives 20; value −20; (b) 0010 1111 → 000 101 111 = 57_8.

Summary

- Computers use two states (0 and 1) because they are easy, reliable and match logic.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

- Bases: decimal 10, binary 2, octal 8, hexadecimal 16 (A–F = 10–15).
- Decimal → base: divide and read remainders bottom to top. Base → decimal: multiply by place values and add.
- Binary ↔ octal: groups of 3 bits; binary ↔ hex: groups of 4 bits, from the right.
- Signed numbers use the MSB as the sign bit: sign-magnitude, 1's complement (flip), 2's complement (flip + 1).
- 8-bit 2's complement range is −128 to +127 with only one zero, so CPUs use it.

MCQ answer key – 3.1

1. **C** – On/off states are easy to make and detect.
2. **E** – 156 = 128 + 16 + 8 + 4.
3. **A** – 32 + 8 + 4 + 1 = 45.
4. **D** – 2×256 + 10×16 + 15 = 687.
5. **C** – 1101 = D, 0110 = 6.
6. **B** – 011 010 110 = 3 2 6.
7. **A** – 3 = 0011, C = 1100, 9 = 1001.
8. **C** – 001 101 111 → 0110 1111 = 6F.
9. **E** – 16 = 2^4.
10. **B** – Flip 00101101 → 11010010, add 1 → 11010011.
11. **E** – Sign 1 then 45 in 7 bits.
12. **D** – −128 + 64 + 32 + 8 + 4 = −20.
13. **D** – -2^7 to $2^7 - 1$.
14. **A** – SM and 1's complement have +0 and −0.
15. **A** – $2^8 - 1 = 255$.
16. **E** – MSB is the sign bit.
17. **B** – Bit – The smallest unit of data: a single 0 or 1.
18. **C** – Byte – A group of 8 bits.
19. **B** – Nibble – A group of 4 bits, equal to one hex digit.
20. **D** – Base (radix) – The number of different digits a number system uses.

© 2026 schoolict.net (Ruwan Mangala Suraweera). All rights reserved.

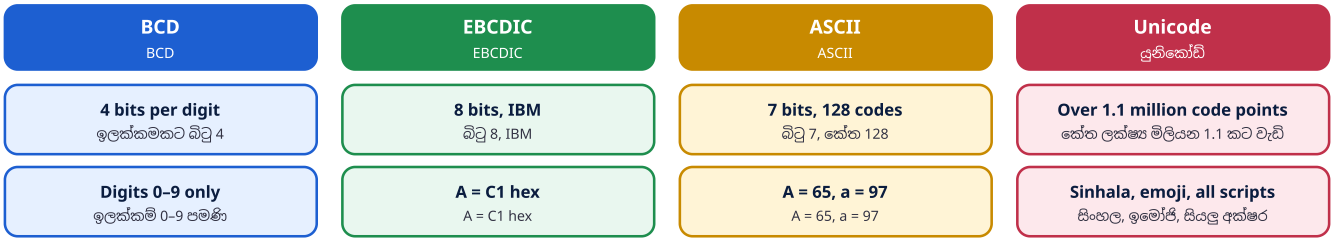
Grade 12 A/L ICT · Lesson 3

3.2 Character Representation: BCD, EBCDIC, ASCII and Unicode

Syllabus 3.2. When you press **A** on the keyboard, the computer does not store a picture of the letter. It stores a **number code**. In this part you learn the four main coding schemes – **BCD, EBCDIC, ASCII and Unicode** – how to **convert symbols to codes and back**, and the **advantages and disadvantages** of each.

1. What is a character code?

A **character** is any letter, digit, symbol or space, such as A, 7, ?, අ or 😀. A **character set (coding scheme)** is an agreed table that gives every character a **unique number**. Because both computers use the same table, text typed on one machine shows correctly on another.



The four coding schemes in this lesson

2. BCD – Binary Coded Decimal

In **BCD** each **decimal digit** is changed to its own **4-bit** code (0 = 0000 … 9 = 1001). The codes 1010 to 1111 are **not used**. BCD is used where numbers must be **shown as digits** quickly and exactly: **calculators, digital clocks, meters and banking**.

> 🧮 **Worked example: Write 2026 in BCD.**
> Take each digit on its own: 2 → 0010, 0 → 0000, 2 → 0010, 6 → 0110. So 2026 = **0010 0000 0010 0110** (16 bits).

Digit	2	0	2	6
BCD	0010	0000	0010	0110

> 🧮 **Worked example: Compare 59 in BCD and in pure binary.**
> BCD: 5 → 0101, 9 → 1001, so **0101 1001** (8 bits). Pure binary: 59 = 32 + 16 + 8 + 2 + 1 = **111011** (6 bits). BCD needs **more bits**, but each digit is easy to read.

> 🧮 **Worked example: Which decimal number is the BCD code 0111 0000 0100?**
> Split into groups of 4: 0111 = 7, 0000 = 0, 0100 = 4. The number is **704**. (Do not convert the whole thing as one binary number!)

3. EBCDIC

EBCDIC (Extended Binary Coded Decimal Interchange Code) is an **8-bit** code made by **IBM** in the 1960s for its **mainframe** computers. It has 256 codes. The letters are **not in one continuous run**: A–I are C1–C9, J–R are D1–D9 and S–Z are E2–E9 (hex). Digits 0–9 are F0–F9. It is still used in **banks and big IBM mainframes** today.

> 🧮 **Worked example: Write the word ICT in EBCDIC (hex and binary).**
> I = C9, C = C3, T = E3, so **C9 C3 E3** (hex) = 11001001 11000011 11100011.

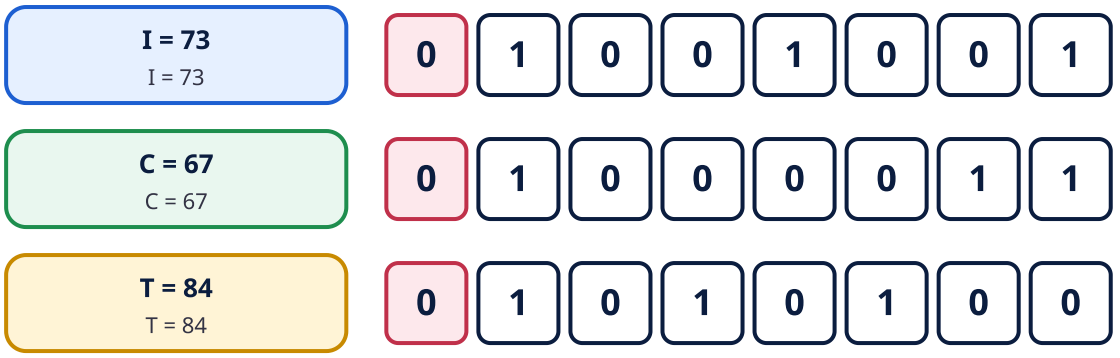
4. ASCII

ASCII (American Standard Code for Information Interchange, 1963) is a **7-bit** code with **128 codes (0–127)**. It is stored in one byte. **Extended ASCII** uses all 8 bits for 256 codes (extra symbols such as é and £). ASCII is the base of almost all modern codes.

Codes	Characters	Remember
0–31, 127	Control codes (Enter = 13, Tab = 9)	Not printed
32	Space	32 = 0100000
48–57	Digits 0–9	“0” = 48
65–90	Capital A–Z	“A” = 65
97–122	Small a–z	“a” = 97

> 🧮 **Worked example: Write the word ICT in ASCII (decimal, hex and 8-bit binary). How many bytes does it need?**
> I = 73, C = 67, T = 84 → hex 49 43 54 → binary **01001001 01000011 01010100**. 3 characters × 1 byte = **3 bytes**.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved



Each character takes one byte: 3 characters = 3 bytes

The word ICT stored in ASCII

🧮 **Worked example: Decode the ASCII codes 01000011 01000001 01010100.**
67 = C, 65 = A, 84 = T. The word is **CAT**.

🧮 **Worked example: The ASCII code of "E" is 69. Find the codes of "e" and "H" without a table.**
Small letters are **32 more** than capitals: "e" = 69 + 32 = **101**. Letters are in order: H is 3 after E, so "H" = 69 + 3 = **72**.

💡 The **character** "7" (code 55) is not the **number** 7 (binary 111). Phone numbers and NIC numbers are stored as text, so each digit uses one byte.

5. Unicode

ASCII has no space for Sinhala, Tamil, Chinese or emoji. **Unicode** gives every character in every language its own **code point**, written **U+** and a hex number. There is room for **1,114,112 code points** (U+0000 to U+10FFFF). The first 128 code points are the **same as ASCII**. Sinhala letters are in the block **U+0D80 to U+0DFF**.

UTF-8
1 to 4 bytes per character. ASCII letters use 1 byte. Used by almost all websites.

UTF-16
2 or 4 bytes per character. Used inside Windows, Java and JavaScript.

UTF-32
Always 4 bytes per character. Simple but uses the most space.

🧮 **Worked example: Find the Unicode code point and UTF-8 bytes of the Sinhala letter "අ".**
"අ" is **U+0D85** (decimal 3461). In UTF-8 it takes **3 bytes: E0 B6 85**. In UTF-16 it takes 2 bytes.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved



From a Sinhala letter to stored bits

🧱 **Worked example: How many bytes does the word “ලංකා” need in UTF-8 and in UTF-16? Compare with “ICT” in ASCII.**
“ලංකා” is 4 code points (ල, ං, ක, ා). UTF-8: 4 × 3 = **12 bytes**. UTF-16: 4 × 2 = **8 bytes**. “ICT” in ASCII is only 3 bytes. Unicode supports more languages but can need more space.

🧱 **Worked example: The emoji 😀 is code point U+1F600. How many bytes does it use in UTF-8?**
Code points above U+FFFF need **4 bytes** in UTF-8: **F0 9F 98 80**.

6. Advantages and disadvantages

Code	Bits	Advantages	Disadvantages
BCD	4 per digit	Easy to convert to and from decimal; exact for money; easy to show on displays	Digits only; wastes bits (1010–1111 unused); slower arithmetic
EBCDIC	8	256 codes; works with old IBM mainframe systems	Letters not continuous; not compatible with ASCII; English only
ASCII	7 (8 extended)	Simple; small (1 byte); letters in order; used everywhere	Only 128 (or 256) characters; no Sinhala, Tamil or emoji
Unicode	8 to 32 (UTF-8/16/32)	All languages and emoji; one world standard; UTF-8 matches ASCII	Needs more storage for non-English text; more complex

NEW 2026 **Unicode 18.0** was released on **16 September 2026** and has **172,808 characters** (Unicode 17.0 in 2025 had 159,801). Every new version adds scripts, symbols and a few new emoji, which reach phones in the following months.

🚀 Beyond the syllabus (2026 update) *Not required for the exam – read for interest.* **Beyond the syllabus – UTF-8 rules the web.** In 2026 about **98% of all websites** use UTF-8. Many emoji are really **several code points joined together**: a family emoji or a skin-tone emoji is built with a **Zero Width Joiner (U+200D)**, so one emoji on screen can take 10 or more bytes. Sinhala uses the same joiner: ශ්‍රී needs ශ + ් + ZWJ + ර + ී.

Key Words

Character – Any letter, digit, symbol or space that can be stored.

Character set – An agreed table that gives each character a unique number.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

BCD – A code that stores each decimal digit as its own 4 bits.	**EBCDIC** – An 8-bit IBM mainframe code with letters in separate runs.
ASCII – A 7-bit code with 128 characters, A = 65.	**Extended ASCII** – An 8-bit version of ASCII with 256 codes.
Unicode – A world standard that gives every character of every language a code point.	**Code point** – The number of a character in Unicode, written like U+0041.
UTF-8 – A Unicode encoding that uses 1 to 4 bytes and matches ASCII.	**UTF-16** – A Unicode encoding that uses 2 or 4 bytes per character.
Control code – A code that is not printed but controls the device, such as Enter.	**Encoding** – The rule for turning characters into bits.

Sample Q&A

1. Why do we need character codes?

> **Answer:** A computer can only store numbers in binary. A character code gives each character a number, and because everyone uses the same code, text can be shared between computers.

2. Write 2026 in BCD and say one place where BCD is used.

> **Answer:** 0010 0000 0010 0110. BCD is used in calculators and digital clocks.

3. Write “ICT” in 8-bit ASCII binary.

> **Answer:** 01001001 01000011 01010100

4. Give two differences between ASCII and Unicode.

> **Answer:** ASCII uses 7 bits and has 128 characters for English; Unicode has over a million code points for all languages. ASCII uses 1 byte; Unicode (UTF-8) uses 1 to 4 bytes.

5. Why was Unicode important for Sinhala users?

> **Answer:** Before Unicode, Sinhala text used special fonts that mapped letters onto ASCII codes, so text broke on other computers. Unicode gives Sinhala its own code points (U+0D80–U+0DFF), so Sinhala works on every phone, website and search engine.

6. Give one advantage and one disadvantage of EBCDIC.

> **Answer:** Advantage: works with existing IBM mainframe systems (256 codes). Disadvantage: letters are not continuous and it is not compatible with ASCII.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

MCQ

1. How many characters can standard 7-bit ASCII represent?

A. 128
B. 127
C. 256
D. 1,114,112
E. 64

2. What is the ASCII code of “A”?

A. 193
B. 65
C. 41
D. 64
E. 97

3. The ASCII code of “a” is 97. What is the code of “d”?

A. 100
B. 99
C. 104
D. 68
E. 101

4. What is 59 in BCD?

A. 0110 1001
B. 1001 0101
C. 0101 1001
D. 0101 1010
E. 0011 1011

5. Which of these is NOT a valid BCD digit?

A. 1001
B. 1100
C. 0000
D. 0111
E. 1000

6. The BCD code 0011 1000 stands for:

A. 56
B. 83
C. 11
D. 38
E. 3.8

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

7. Which code was made by IBM for mainframes?

A. EBCDIC
B. UTF-16
C. BCD
D. UTF-8
E. ASCII

8. In EBCDIC the letter “A” is (hex):

A. 41
B. F1
C. 65
D. A1
E. C1

9. Which Unicode block holds the Sinhala script?

A. U+0D80 – U+0DFF
B. U+0900 – U+097F
C. U+0000 – U+007F
D. U+0B80 – U+0BFF
E. U+1F600 – U+1F64F

10. How many bytes does the letter “අ” (U+0D85) use in UTF-8?

A. 4
B. 3
C. 1
D. 2
E. 8

11. Which encoding is used by almost all websites in 2026?

A. EBCDIC
B. UTF-32
C. BCD
D. UTF-8
E. 7-bit ASCII only

12. The ASCII codes 01001001 01010100 spell:

A. it
B. TI
C. HT
D. IU
E. IT

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

13. What is a main disadvantage of ASCII?

A. It is only used by IBM
B. It uses 32 bits per letter
C. Letters are not in order
D. It cannot store digits
E. It cannot represent Sinhala or Tamil

14. How many bytes does UTF-32 use for every character?

A. 1
B. 4
C. 1 to 4
D. 3
E. 2

15. The first 128 Unicode code points are the same as:

A. Hexadecimal
B. EBCDIC
C. Extended EBCDIC
D. ASCII
E. BCD

16. Which term means: “Any letter, digit, symbol or space that can be stored”?

A. BCD
B. Code point
C. Character
D. Control code
E. Extended ASCII

17. Which term means: “An agreed table that gives each character a unique number”?

A. EBCDIC
B. Unicode
C. UTF-8
D. Character set
E. Encoding

18. Which term means: “A code that stores each decimal digit as its own 4 bits”?

A. UTF-16
B. ASCII
C. BCD
D. Character
E. Code point

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

19. Which term means: “An 8-bit IBM mainframe code with letters in separate runs”?

A. Character set
B. Extended ASCII
C. EBCDIC
D. UTF-8
E. Control code

20. Which term means: “A 7-bit code with 128 characters, A = 65”?

A. BCD
B. Encoding
C. Unicode
D. UTF-16
E. ASCII

Structured Questions

1. (a) Write 59 in BCD. (b) Write it in pure binary. (c) Which needs more bits and why? *(3 marks)*

> (a) 0101 1001; (b) 111011; (c) BCD needs 8 bits against 6, because each digit uses 4 bits and codes 1010–1111 are wasted.

2. (a) Write “ICT” in ASCII decimal and binary. (b) How many bits are needed to store it in 8-bit ASCII? *(3 marks)*

> (a) 73, 67, 84; 01001001 01000011 01010100; (b) 3 × 8 = 24 bits.

3. Decode the ASCII binary 01000011 01000001 01010100 and write the same word in EBCDIC hex. *(4 marks)*

> 67 = C; 65 = A; 84 = T, word CAT; EBCDIC C3 C1 E3.

4. (a) What is a Unicode code point? (b) Give the code point and UTF-8 bytes of “ඇ”. (c) Give two advantages of Unicode. *(4 marks)*

> (a) the unique number of a character, written U+hex; (b) U+0D85, E0 B6 85; (c) supports all languages and emoji; one world standard compatible with ASCII.

5. Compare BCD, EBCDIC, ASCII and Unicode by number of bits and one use of each. *(4 marks)*

> BCD 4 bits per digit, calculators; EBCDIC 8 bits, IBM mainframes; ASCII 7 bits, English text and keyboards; Unicode 8 to 32 bits, websites and phones in all languages.

Summary

- A character code gives every character a unique number so text can be stored and shared.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

- BCD: each decimal digit in 4 bits (0000–1001); used in calculators and clocks.
- EBCDIC: 8-bit IBM mainframe code; A = C1 hex; letters not continuous.
- ASCII: 7 bits, 128 codes; A = 65, a = 97, 0 = 48; small = capital + 32.
- Unicode: a code point for every character in every language; UTF-8 uses 1–4 bytes.
- Sinhala is U+0D80–U+0DFF and takes 3 bytes per letter in UTF-8.

MCQ answer key – 3.2

1. **A** – $2^7 = 128$.
2. **B** – A = 65 (hex 41).
3. **A** – 97 + 3 = 100.
4. **C** – 5 = 0101, 9 = 1001.
5. **B** – Only 0000–1001 are used.
6. **D** – 0011 = 3, 1000 = 8.
7. **A** – EBCDIC is IBM's 8-bit code.
8. **E** – A = C1 in EBCDIC.
9. **A** – Sinhala is U+0D80–U+0DFF.
10. **B** – U+0800–U+FFFF need 3 bytes.
11. **D** – About 98% of sites use UTF-8.
12. **E** – 73 = I, 84 = T.
13. **E** – Only 128 characters, English only.
14. **B** – 32 bits = 4 bytes, always.
15. **D** – Unicode keeps ASCII for compatibility.
16. **C** – Character – Any letter, digit, symbol or space that can be stored.
17. **D** – Character set – An agreed table that gives each character a unique number.
18. **C** – BCD – A code that stores each decimal digit as its own 4 bits.
19. **C** – EBCDIC – An 8-bit IBM mainframe code with letters in separate runs.
20. **E** – ASCII – A 7-bit code with 128 characters, A = 65.

© 2026 schoolict.net (Ruwan Mangala Suraweera). All rights reserved.

Grade 12 A/L ICT · Lesson 3

3.3 Binary Arithmetic and Bitwise Logic: Add, Subtract, AND, OR, XOR, NOT

Syllabus 3.3. The ALU inside the CPU does all its work with bits. In this part you learn **binary addition** (with carries, even when adding several numbers), **binary subtraction** (with borrowing, and with 2's complement), what **overflow** is, and the **bitwise logical operations NOT, AND, OR and XOR**.

1. The rules of binary addition

Binary addition works **exactly like decimal addition**, column by column from the **right**. The only difference: in binary, **2 is written as 10**, so you write 0 and **carry 1** to the next column.

Bits added එකතු කරන බිටු	Sum bit එකතු බිටුව	Carry රැගෙන යාම
0 + 0 0 + 0	**0** 0	**0** 0
0 + 1 0 + 1	**1** 1	**0** 0
1 + 1 1 + 1	**0** 0	**1** 1
1 + 1 + 1 1 + 1 + 1	**1** 1	**1** 1

The four rules of binary addition

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 1 = 10 → write **0**, carry **1**
- 1 + 1 + 1 = 11 → write **1**, carry **1**

🧱 **Worked example: Add $1011_2 + 0110_2$.**
Right column: 1 + 0 = 1. Next: 1 + 1 = 10, write 0 carry 1. Next: 0 + 1 + 1 = 10, write 0 carry 1. Next: 1 + 0 + 1 = 10, write 0 carry 1. Last carry goes down. Answer **10001_2**. Check: 11 + 6 = 17 ✔

	2^4	2^3	2^2	2^1	2^0

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

Carry	1	1	1		
	0	1	0	1	1
	0	0	1	1	0
Sum	**1**	**0**	**0**	**0**	**1**

🧮 **Worked example: Add three numbers: 1011 + 1101 + 0111.**
In every column three or four 1s may meet. Count the 1s (plus the carry): 3 = 11, so write **1** and carry **1**. Doing this for each column gives **11111_2**. Check: 11 + 13 + 7 = 31 ✔

	2^4	2^3	2^2	2^1	2^0
Carry	1	1	1	1	
	0	1	0	1	1
	0	1	1	0	1
	0	0	1	1	1
Sum	**1**	**1**	**1**	**1**	**1**

💡 **Many numbers?** If a column adds up to 4 (100 in binary), write 0 and carry 1 **two columns** to the left. Or add two numbers first, then add the third to the answer – this is safer in an exam.

2. Binary subtraction with borrowing

- 0 − 0 = 0, 1 − 0 = 1, 1 − 1 = 0
- 0 − 1: **borrow 1** from the next column on the left. The borrowed 1 is worth **2** here, so 2 − 1 = **1**.
- The column that lent the 1 is now **one less**.

🧮 **Worked example: Subtract $11010_2 - 01011_2$.**
Right column: 0 − 1, borrow → 2 − 1 = 1. Next: 1 − 1 − 1 (paying back the borrow) is below 0, so borrow again: 2 + 1 − 1 − 1 = 1. Continue the same way; the table shows where each borrow was paid. Answer **01111_2**. Check: 26 − 11 = 15 ✔

	2^4	2^3	2^2	2^1	2^0
Borrow paid	−1	−1	−1	−1	
	1	1	0	1	0
−	0	1	0	1	1
Difference	**0**	**1**	**1**	**1**	**1**

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

🧮 **Worked example: Subtract $10000_2 - 00001_2$ (a long chain of borrows).**
The right column needs a borrow, but its neighbours are 0, so the borrow travels to the leftmost 1. That 1 becomes 0 and every 0 on the way becomes 1. Answer **01111_2** (16 − 1 = 15).

	2^4	2^3	2^2	2^1	2^0
Borrow paid	−1	−1	−1	−1	
	1	0	0	0	0
–	0	0	0	0	1
Difference	**0**	**1**	**1**	**1**	**1**

3. Subtraction with 2's complement

A CPU does not need a separate subtract circuit. To work out **A − B** it adds: **A + (2's complement of B)**. Any carry out of the left end (the 9th bit for 8-bit numbers) is **thrown away**.

🧮 **Worked example: Work out 45 − 20 using 8-bit 2's complement.**
+45 = 0010 1101. −20 = 1110 1100 (flip 00010100 and add 1). Add them: 100011001. Throw away the 9th bit → **0001 1001** = 25 ✔

	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Carry	1	1	1		1	1			
	0	0	0	1	0	1	1	0	1
	0	1	1	1	0	1	1	0	0
Sum	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**

4. Overflow

An 8-bit 2's complement number can only hold −128 to +127. If the true answer is **outside this range**, the result is wrong. This is called **overflow**. A sign check finds it: **adding two positives gives a negative**, or **adding two negatives gives a positive**.

🧮 **Worked example: Add 100 + 50 in 8-bit 2's complement. What goes wrong?**
0110 0100 + 0011 0010 = **1001 0110**. The MSB is 1, so the computer reads **−106**, not 150. Two positives gave a negative: **overflow**, because 150 > 127.

5. Bitwise logical operations

A **bitwise** operation works on **each pair of bits on its own**, column by column. There is **no carry**.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

A	B	A AND B	A OR B	A XOR B	NOT A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

AND
1 only if **both** bits are 1.

OR
1 if **at least one** bit is 1.

XOR
1 if the bits are **different**.

NOT
Flips every bit (one input).

🧱 **Worked example: A = 11001010 and B = 10101100. Find A AND B, A OR B, A XOR B and NOT A.**
Work column by column using the truth table: AND = **10001000**, OR = **11101110**, XOR = **01100110**, NOT A = **00110101**.

	b7	b6	b5	b4	b3	b2	b1	b0
A	1	1	0	0	1	0	1	0
B	1	0	1	0	1	1	0	0
A AND B	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
A OR B	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**
A XOR B	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**
NOT A	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved



Bitwise operations on A and B

6. Using bitwise operations (masks)

A **mask** is a bit pattern used to pick out or change certain bits. **AND** with a mask **checks or clears** bits, **OR sets** bits to 1, and **XOR toggles** (flips) bits.

🧱 **Worked example: Is bit 3 of A = 11001010 set? Use a mask.**
11001010 AND 00001000 = **00001000**. The answer is not 0, so bit 3 **is 1**.

🧱 **Worked example: Change the ASCII letter "A" (01000001) to "a" using one bitwise operation.**
Small letters are 32 more, and 32 = 00100000. 01000001 OR 00100000 = **01100001** = 97 = "a".

NEW 2026 **Bitwise AND finds a network.** A computer with IP address 192.168.1.130 and subnet mask 255.255.255.0 does 192.168.1.130 AND 255.255.255.0 to get its network address **192.168.1.0**. Your home router does this for every packet.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

🚀 **Beyond the syllabus (2026 update)** *Not required for the exam – read for interest.* **Beyond the syllabus – 2's complement in every CPU.** Every modern processor (Intel, AMD, Apple M-series, ARM chips in phones) stores signed integers in 2's complement. The C++20 and C23 language standards now **require** it. The ALU sets a **carry flag** for unsigned overflow and an **overflow flag** for signed overflow. XOR is also the heart of encryption such as AES: data XOR key hides the data, and XOR with the same key again brings it back.

Key Words

Carry – The 1 moved to the next column when a column adds up to 2 or more.

Borrow – Taking 1 from the next column on the left when 0 − 1 is needed.

Overflow – When the true result is too big or too small for the number of bits.

Sum bit – The bit written in a column after adding.

Bitwise operation – An operation done on each pair of bits separately, with no carry.

AND – Gives 1 only when both bits are 1.

OR – Gives 1 when at least one bit is 1.

XOR – Gives 1 when the two bits are different.

NOT – Flips every bit: 0 becomes 1 and 1 becomes 0.

Mask – A bit pattern used with AND, OR or XOR to check or change chosen bits.

ALU – The part of the CPU that does arithmetic and logic.

Truth table – A table that shows the output for every input combination.

Sample Q&A

1. State the four rules of binary addition.

> **Answer:** 0 + 0 = 0; 0 + 1 = 1; 1 + 1 = 0 carry 1; 1 + 1 + 1 = 1 carry 1.

2. Add 1011 + 1101 + 0111.

> **Answer:** 11111 (11 + 13 + 7 = 31).

3. Subtract 11010 − 01011.

> **Answer:** 01111 (26 − 11 = 15).

4. How does a CPU subtract using only an adder?

> **Answer:** It finds the 2's complement of the number being subtracted and adds it. The carry out of the left end is ignored. Example: 45 − 20 = 00101101 + 11101100 = 00011001 = 25.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

5. What is overflow? Give an example.

> **Answer:** Overflow happens when the answer does not fit in the bits. In 8-bit 2's complement, 100 + 50 = 150 is more than 127, so the result 10010110 reads as −106.

6. Find 11001010 XOR 10101100 and say what XOR shows.

> **Answer:** 01100110. XOR shows 1 wherever the two bits are different.

MCQ

1. What is 1011 + 0110?

A. 1101
B. 10001
C. 10101
D. 11001
E. 10011

2. 1 + 1 + 1 in binary gives:

A. Sum 0, carry 1
B. Sum 0, carry 0
C. Sum 1, carry 1
D. Sum 1, carry 0
E. Sum 3, no carry

3. What is 1011 + 1101 + 0111?

A. 11011
B. 111111
C. 11101
D. 11111
E. 10111

4. What is 11010 − 01011?

A. 01110
B. 01111
C. 01101
D. 10001
E. 10111

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

5. What is 10000 − 00001?

A. 11111
B. 01110
C. 10001
D. 00001
E. 01111

6. In binary subtraction, 0 − 1 needs:

A. A carry to the next column
B. Nothing, the answer is 0
C. An overflow flag
D. A NOT operation
E. A borrow from the next column

7. To find A − B a CPU usually calculates:

A. A + 2's complement of B
B. A XOR B
C. A AND NOT B
D. B − A then NOT
E. A + B

8. In 8-bit 2's complement, 00101101 + 11101100 (after dropping the 9th bit) is:

A. 10011001
B. 11100011
C. 00011001
D. 00011000
E. 100011001

9. Adding 100 + 50 in 8-bit 2's complement gives a negative answer. This is:

A. Overflow
B. A mask
C. A borrow
D. Underflow of a float
E. Correct, 150 is negative

10. 11001010 AND 10101100 = ?

A. 10001010
B. 10001000
C. 01100110
D. 11101110
E. 01110111

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

11. 11001010 OR 10101100 = ?

A. 11101110
B. 01100110
C. 10001000
D. 11101111
E. 11111110

12. 11001010 XOR 10101100 = ?

A. 10011001
B. 01100111
C. 10001000
D. 11101110
E. 01100110

13. NOT 11001010 = ?

A. 11001010
B. 11001011
C. 01010011
D. 00110101
E. 00110100

14. Which operation gives 1 only when both bits are 1?

A. ADD
B. OR
C. NOT
D. AND
E. XOR

15. Which operation with mask 00100000 changes “A” into “a”?

A. NOT
B. 2’s complement
C. OR
D. AND
E. Subtraction

16. IP 192.168.1.130 AND mask 255.255.255.0 gives:

A. 192.168.1.0
B. 192.168.1.255
C. 0.0.0.130
D. 255.255.255.130
E. 192.168.1.130

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

17. Which term means: “The 1 moved to the next column when a column adds up to 2 or more”?

A. ALU
B. Overflow
C. Carry
D. XOR
E. AND

18. Which term means: “Taking 1 from the next column on the left when 0 − 1 is needed”?

A. OR
B. NOT
C. Sum bit
D. Borrow
E. Truth table

19. Which term means: “When the true result is too big or too small for the number of bits”?

A. Mask
B. Overflow
C. Bitwise operation
D. Carry
E. XOR

20. Which term means: “The bit written in a column after adding”?

A. Borrow
B. AND
C. NOT
D. ALU
E. Sum bit

Structured Questions

1. (a) Add 1011 + 0110 showing the carries. (b) Check your answer in decimal. *(3 marks)*

> (a) carries in three columns; sum 10001; (b) 11 + 6 = 17.

2. Add the three binary numbers 1011, 1101, 0111 and give the answer in binary and decimal. *(3 marks)*

> Each column: count the 1s plus carry; sum 11111; decimal 31.

3. (a) Subtract 11010 − 01011 using borrowing. (b) Do 45 − 20 in 8-bit 2’s complement. *(3 marks)*

> (a) 01111; (b) 00101101 + 11101100 = 100011001, drop the 9th bit; 00011001 = 25.

© 2026 schoolict.net (Ruwan Mangala Suraweera) · All rights reserved

4. (a) What is overflow? (b) Show that 100 + 50 overflows in 8-bit 2's complement. *(4 marks)*

> (a) result outside the range the bits can hold; (b) 01100100 + 00110010 = 10010110; MSB 1 so reads −106; two positives gave a negative.

5. A = 11001010, B = 10101100. Find (a) A AND B (b) A OR B (c) A XOR B (d) NOT A (e) one use of AND with a mask. *(5 marks)*

> (a) 10001000; (b) 11101110; (c) 01100110; (d) 00110101; (e) checking a bit or finding a network address with a subnet mask.

Summary

- Binary addition: 1 + 1 = 0 carry 1, 1 + 1 + 1 = 1 carry 1; work from the right.
- For many numbers, count the 1s in each column plus the carry.
- Subtraction: 0 − 1 borrows 1 (worth 2) from the left column.
- A CPU subtracts by adding the 2's complement and dropping the extra carry.
- Overflow: the true answer is outside the range of the bits (8-bit: −128 to +127).
- Bitwise: AND both 1, OR any 1, XOR different, NOT flips; masks check, set or toggle bits.

MCQ answer key – 3.3

1. **B** – 11 + 6 = 17 = 10001.
2. **C** – 3 = 11.
3. **D** – 11 + 13 + 7 = 31.
4. **B** – 26 − 11 = 15.
5. **E** – 16 − 1 = 15.
6. **E** – Borrow 1 (worth 2).
7. **A** – Subtraction becomes addition.
8. **C** – 45 + (−20) = 25 = 00011001.
9. **A** – 150 > 127.
10. **B** – 1 only where both are 1.
11. **A** – 1 wherever at least one is 1.
12. **E** – 1 where bits differ.
13. **D** – Flip every bit.
14. **D** – AND truth table.
15. **C** – OR sets bit 5 to 1 (+32).
16. **A** – 130 AND 0 = 0.
17. **C** – Carry – The 1 moved to the next column when a column adds up to 2 or more.
18. **D** – Borrow – Taking 1 from the next column on the left when 0 − 1 is needed.
19. **B** – Overflow – When the true result is too big or too small for the number of bits.
20. **E** – Sum bit – The bit written in a column after adding.

© 2026 schoolict.net (Ruwan Mangala Suraweera). All rights reserved.