1.1 Number systems

Name: _____ Class: _____ Date: _____ **Total: 34 marks**

KEY WORDS denary • hexadecimal • binary addition • overflow • two's complement

Objective

Build the skills to answer exam questions on **number systems** – denary/binary/**hexadecimal** conversion, **binary addition**, **overflow**, shifts, and **two's complement**.

You must be able to:

- convert between denary, binary and hexadecimal
- add binary numbers and identify overflow
- carry out a logical shift; use two's complement for negatives

1 Key knowledge and worked examples

Conversions and two's complement

place value	128	64	32	16	8	4	2	1
bits	**1**	0	0	**1**	0	**1**	**1**	0

$150 = 128 + 16 + 4 + 2 \Rightarrow$ `10010110`

Place values 128…1; the 1s sit under the values that sum to the number

	sign bit (negative)							
place value	-128	64	32	16	8	4	2	1
bits	**1**	**1**	0	**1**	**1**	0	0	0

$-128 + 64 + 16 + 8 = -40$

Two's complement: the MSB is worth −128, so 11011000 = −40

Denary 100 → binary: $64 + 32 + 4 \rightarrow$ `01100100` → hex `64`.

■ **The general methods, step by step**

Denary to binary – divide by 2 and read the remainders bottom-to-top.

step	division	remainder
1	$150 \div 2 = 75$	0
2	$75 \div 2 = 37$	1
3	$37 \div 2 = 18$	1
4	$18 \div 2 = 9$	0
5	$9 \div 2 = 4$	1
6	$4 \div 2 = 2$	0
7	$2 \div 2 = 1$	0
8	$1 \div 2 = 0$	1

Reading bottom-to-top: `10010110`. Check with place values: $128 + 16 + 4 + 2 = 150$.

Binary to hexadecimal – split into nibbles of four bits from the right, then write each as one hex digit: `1100 1010` → `C A` → `CA`. Going back, expand each hex digit into four bits.

Binary addition. Add column by column from the right, carrying 1 when a column makes 2 (`1 + 1 = 10`). If a carry leaves the eighth column, the answer needs a ninth bit that the register cannot hold: that is **overflow**, and the stored answer is wrong.

Logical shift. A left shift of n places moves every bit n to the left, fills the right with zeros and multiplies the value by 2^n; a right shift divides by 2^n. Bits shifted out are lost, so a left shift can also lose the value.

Two's complement. Write the negative number by starting from the positive binary, inverting every bit, then adding 1. To read one, the leftmost bit is worth -128: `11011000` $= -128 + 64 + 16 + 8 = -40$.

Why hexadecimal is used: it is shorter than binary and easier for a person to read and copy without mistakes, and each hex digit maps to exactly four bits. It appears in MAC addresses, IP version 6 addresses, HTML colour codes, memory dumps and error codes.

2 Practice

2.1 Convert the denary number 45 to 8-bit binary. [2]

2.2 Convert the binary number `11001010` to hexadecimal. [2]

2.3 Convert the denary number 200 to 8-bit binary using the division method, showing your working. [3]

2.4 Convert the hexadecimal number `3F` to denary. [2]

2.5 Give two reasons why hexadecimal is used to represent binary numbers, and name two places where it is used. [4]

3 Exam-style questions

3.1 An 8-bit register can hold the denary values: [1]

- ☐ **A** 0 to 127
- **B** 0 to 255
- **C** 0 to 256
- **D** 1 to 255

3.2 Two 8-bit binary numbers are added: `01101100` + `00111010`.

(a) Work out the binary result. [2]

(b) Convert your answer to denary, and state whether overflow has occurred. [2]

3.3 A register holds `00110100`. Perform a logical left shift of 2 places and state the effect on the denary value. [2]

3.4 A microprocessor uses 8-bit registers.

(a) Convert the denary number 172 to 8-bit binary. [2]

(b) Convert your answer to (a) to hexadecimal. [2]

(c) The register holds `10110101`. This binary number is added to `01011100`. Complete the addition and give the 8-bit answer. [3]

(d) Explain whether overflow has occurred in part (c), and state what overflow means. [2]

(e) The register `00011010` has a logical left shift of 3 places applied. Give the result, and calculate the effect on the denary value. [3]

(f) The same register is used to hold two's complement numbers. Give the denary value of `11101100`. [2]

Solutions

Each answer shows the steps, then the **result**.

2.1

- denary → 8-bit binary: place values $128, 64, 32, 16, 8, 4, 2, 1$
- $45 = 32 + 8 + 4 + 1$
- `00101101`

2.2

- binary → hex: split into nibbles from the right
- `1100` = 12 = C and `1010` = 10 = A
- **`CA`**

2.3

- denary → binary: divide by 2, record remainders, read **bottom to top**

```
200 ÷ 2 = 100  rem 0
100 ÷ 2 =  50  rem 0
 50 ÷ 2 =  25  rem 0
 25 ÷ 2 =  12  rem 1
 12 ÷ 2 =   6  rem 0
  6 ÷ 2 =   3  rem 0
  3 ÷ 2 =   1  rem 1
  1 ÷ 2 =   0  rem 1   ← left-most bit
```

- remainders bottom → top: **`11001000`**
- check: $128 + 64 + 8 = 200$

2.4

- hex → denary: 3F means $3 \times 16 + \text{F}$ and F = 15
- $48 + 15 =$ **63**

2.5

- any **two** reasons: shorter than binary / easier to read and copy / each hex digit is exactly four bits
- any **two** uses: MAC addresses / IPv6 addresses / HTML colour codes / memory dumps / error codes

3.1 B

- an 8-bit register stores $2^8 = 256$ different values, **0 to 255**
- A is 7-bit unsigned; C goes one past the last code; D misses 0

3.2

(a)

- add column by column from the right

```
  0110 1100
+ 0011 1010
------------
  1010 0110
```

- **10100110**

(b)

- 1-bits at $128 + 32 + 4 + 2 =$ **166**
- $166 \leq 255$ (no ninth bit), so **no overflow**

3.3

- 00110100 shifted left 2 places, zeros fill the right
- **11010000**; a left shift of 2 multiplies by $2^2 = 4$

3.4

(a)

- $172 = 128 + 32 + 8 + 4$
- **10101100**

(b)

- split 1010 1100
- 1010 = 10 = A and 1100 = 12 = C, so **AC**

(c)

- add column by column from the right

```
  1011 0101
+ 0101 1100
------------
1 0001 0001
```

- stored 8-bit result **00010001**, with a carry out of bit 7

(d)

- **yes, overflow**
- the true sum needs 9 bits, but the 8-bit register cannot store the ninth, so the stored answer is wrong

(e)

- 00011010 shifted left 3 places → **11010000**
- denary was $16 + 8 + 2 = 26$ and becomes $128 + 64 + 16 = 208$
- a left shift of 3 multiplies by $2^3 = 8$, and $26 \times 8 = 208$

(f)

- two's complement: the MSB is worth -128
- $-128 + 64 + 32 + 8 + 4 = \mathbf{-20}$