**C1 | The use and interpretation of number systems of computer data representation methods:**

**Units of digital data:**

* All parts of a computer system use one of two values to hold or communicate data or code.
* These two values are thought of as ‘1 or 0’, sometimes noticed by ‘True or False’.
* The unit of digital data is called a bit (from **b**inary dig**it**)
* The basic unit of digital data is the symbol ‘**V**’ – This is made up of 8 bits. This gives 256 different possible combinations.

**Ranking order of Capacity:**

1. Bit
2. Nibble (4 bits)
3. Byte (8 bits)
4. Kilobyte (1024 bytes)
5. Megabytes (1024 Kilobytes)
6. Gigabyte (1024 Megabytes)
7. Terabyte (1024 Gigabyte)
8. Petabyte (1024 Terabyte)

**Binary & Hexadecimal:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** | **Denary Result:** |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 255 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 128 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 103 |

|  |  |  |
| --- | --- | --- |
| **Denary:** | **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 |

To convert a byte to hexadecimal, divide it into two 4-bit numbers and then convert each into a hexadecimal digit. The byte of 0100001 can be expressed as 41 in hexadecimal. (0001 = 1 / 0100 = 4)

**Binary Coded Decimal (BCD):**

* This is an easy way to represent decimal values.
* It is a 4-bit code used to represent the decimal numerals 0 to 9.
* The 4 bit numbers above 9 are not used in this system

|  |  |
| --- | --- |
| **Denary:** | **Binary:** |
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

**Example:** Convert 25 to BCD

* Convert each decimal numeral to BCD
* 2 = 0010
* 5 = 0101
* Therefore 25 = 00100101 BCD

**Example:** Convert 0110010111 to decimal

* Start at the BCD and group BCD bits into blocks of four.
* Convert each block into a BCD number.
* 0110010111
* 0111 = 7
* 1001 = 9
* 01 = 1
* Therefore 0110010111 BCD = 197

**C1 | The use of binary arithmetic (Including BCD) to perform calculations: addition, subtraction, multiplication & division:**

**Binary Addition Rules:**

* Rule 1: 0 + 0 = 0
* Rule 2: 0 + 1 = 1 or 1 + 0 = 1
* Rule 3: 1 + 1 = 0, then carry 1
* Rule 4: 1 + 1 + 1 = 1, then carry 1

**Example:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **1** | **1** | **1** | **0** | **1** | **1** |
| + | **0** | **1** | **1** | **0** | **1** | **0** | **0** | **0** |
| **Equals:** | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

Sometimes there may be an extra carry left over. This is called an overflow. It means that the two 8-bit numbers added together need more than 8 bits. They need 9. You may be asked to explain what an overflow is in the exam.

**Binary Subtraction:**

You will require to use the Binary Addition rules for subtracting because of the ‘**Two’s Complement Method**’.

**Remember these two key features:**

* Invert the original (0s become 1s and 1s become 0s)
* Then add 1

**For example:**

* 35 – 12. In Binary, we basically do 35 + -12. The answer is 23.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |  |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | = 12 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | Invert |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Add 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 2s complement |
|  |  |  |  |  | 1 | 1 |  | Carry 1 |

* Now add the 2s complement to 35

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |  |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 35 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 2s complement |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 23 |
| 1 | 1 |  |  |  |  |  |  | Carry |

* Note that there is a carry bit at the end. This is ignored. This is normal. When the calculation is made, a 1 should overflow from the high end (left) of the answer.
* If there is no overflow, the answer is negative, shown as two’s complement, so needs to be converted back to binary.

**Binary Multiplication Rules:**

* Rule 1: 0 x 0 = 0
* Rule 2: 1 x 0 = 0
* Rule 3: 1 x 1 = 1

**Example:** Multiply the Binary numbers 101 & 101.

1. **Put the numbers in a table:**

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | 0 | 1 |
| Binary Number 2: | 1 | 0 | 1 |
|  |  |  |  |

1. **Multiply the top row by the bottom right number, in this case 1.**

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | 0 | **1** |
| Binary Number 2: | 1 | 0 | **1** |
|  |  |  | ***1*** |

* 1 x 1 = 1

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | **0** | 1 |
| Binary Number 2: | 1 | 0 | **1** |
|  |  | ***0*** | 1 |

* 0 x 1 = 0

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | **1** | 0 | 1 |
| Binary Number 2: | 1 | 0 | **1** |
|  | ***1*** | 0 | 1 |

* 1 x 1 = 1

1. **Add a 0 on the next row:**

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | 0 | 1 |
| Binary Number 2: | 1 | 0 | 1 |
|  | *1* | *0* | *1* |
|  |  |  | **0** |

1. **Multiply the top row by second number on row 2. In this case, 0. The answers go to the left the 0 you added in Step 3.**

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | 0 | **1** |
| Binary Number 2: | 1 | **0** | 1 |
|  | 1 | 0 | 1 |
|  |  | ***0*** | 0 |

* 1 x 0 = 0

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | **0** | 1 |
| Binary Number 2: | 1 | **0** | 1 |
|  | 1 | 0 | 1 |
|  | ***0*** | 0 | 0 |

* 0 x 0 = 0

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | **1** | 0 | 1 |
| Binary Number 2: | 1 | 0 | 1 |
|  | 1 | 0 | 1 |
| ***0*** | 0 | 0 | 0 |

* 1 x 0 = 0

1. **Add two 0s on the next row**

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | 0 | 1 |
| Binary Number 2: | 1 | 0 | 1 |
|  | 1 | 0 | 1 |
| *0* | *0* | *0* | *0* |
|  |  | **0** | **0** |

* **NOTE:** Each time you move across one bit on the bottom row, you add another 0. So for the first bit (the rightmost), you add no 0s. For the second, you add one 0. For the third, you add two 0s. For the fourth, you add three 0s, etc.

1. **Multiply the top row by the third number on row 2. In this case 1. The answers go to the left of the 0s you added in Step 5.**

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | 0 | **1** |
| Binary Number 2: | 1 | 0 | 1 |
|  | **1** | 0 | 1 |
| 0 | 0 | 0 | 0 |
|  | ***1*** | 0 | 0 |

* 1 x 1 = 1

|  |  |  |  |
| --- | --- | --- | --- |
| Binary Number 1: | 1 | **0** | 1 |
| Binary Number 2: | 1 | 0 | 1 |
|  | **1** | 0 | 1 |
| 0 | 0 | 0 | 0 |
| ***0*** | 1 | 0 | 0 |

* 0 x 1 = 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Binary Number 1: | 1 | 0 | 1 |
|  | Binary Number 2: | 1 | 0 | 1 |
|  |  | **1** | 0 | 1 |
|  | 0 | 0 | 0 | 0 |
| ***1*** | 0 | 1 | 0 | 0 |

* 1 x 1 = 1

1. **Now you have multiplied all of the top row by all of the bottom row, add together the resulting numbers.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 1 | 0 | 1 |  |
|  |  | 0 | 0 | 0 | 0 |  |
|  | 1 | 0 | 1 | 0 | 0 |  |
| Answer: | **1** | **1** | **0** | **0** | **1** |  |
|  |  | *1* |  |  |  | Carry 1 |

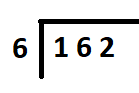
* When adding binary numbers, remember to work from right to left one column at a tune, check which rule it meets, and don’t forget to include carries.

**Binary Division:**

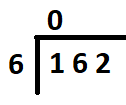
For Binary Division, we use the ‘ Bus Stop Method ‘

**Example:** *Denary* | 162 / 6

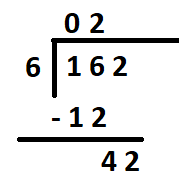
We write the dividend (162) under the bus stop and the divisor (6) before it.



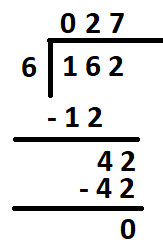
Starting at the left, the divisor cannot be divided into the first dividend digit (1), so we write a 0 above the bus stop for that column and move right.



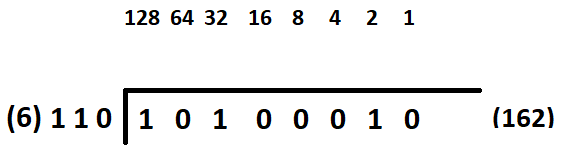
The divisor can be divided into 16 twice, so we write a 2 above the bus stop for that column and then carry out the division, which leaves a remainder of 4, so we move to the right to bring down the next digit from the dividend, giving 42.



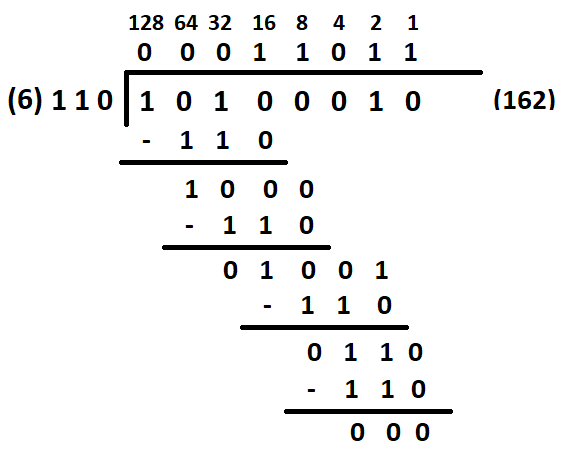
The divisor can be divided into 42 seven times, so we write a 7 above the bus stop for that column and then, as we have reached the lowest digit of the dividend, the calculation is complete.



**Example:** *Binary* | 162 / 6



Starting at the left, if the divisor cannot be divided into the dividend, write a 0 above the bus stop for that column then move right. If the divisor can be divided into the dividend, write a 1 above the bus stop for that column and then carry out the division and move right.



**C1 | The use of binary to represent negative and floating point numbers:**

**Negative:** ‘Two Complements’ are used to represent binary negative numbers. It’s a method used in Binary Arithmetic. It enabled the use of subtraction to reuse other circuits already in the CPU for adding and inversion using NOT gates.

**Floating Point Binary Numbers:** These are used to represent very large and very small numbers. This is similar to standard form. It is used with denary numbers, where the number is shortened to a mantissa with a single digit followed by a set number of decimal places and an exponent to remember how many places the decimal point moved.

**Example:**

Using standard form, the number:

27,563,498,208 would be written as 2.76 x 1010

2.76 representing the number to two decimal places. (mantissa)

x 1010 showing that the decimal point moved 10 times to between 2 and 7 in 2.76. (exponent is 10)

Accuracy is lost with standard form because to convert 2.76 x 1010 back we would get 27,600,000,000 as there is no way of recovering the detail lost in rounding the mantissa to two decimal places.

A floating point binary number has the mantissa normalised to 0.1 (1.0 for two’s complement negative numbers) with a set number of digits, and the exponent shows how many places the binary point has moved to normalise the mantissa.

Mantissa and exponent – these are used to represent very large or very small numbers in a small space using floating point or scientific notation. The mantissa is the number part and the exponent shows how many places the point has moved.