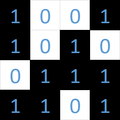
**Unit 19 Computer System Architecture**

****

**Data Representation Worksheet**

**Student ID**

|  |
| --- |
| **1598230452** |

**Student Name**

|  |
| --- |
| **Chris Bollard** |

Contents

[Part 1 – Number Systems 2](#_Toc448754459)

[Part 2 – Representing Text 5](#_Toc448754460)

[Part 3 – Fixed and Floating Point 7](#_Toc448754461)

[Part 4 – Number System Conversion 11](#_Toc448754462)

[Part 5 – Boolean Logic 16](#_Toc448754463)

# Part 1 – Number Systems

****

**Complete the table below to show how DECIMAL can be used to store the value 212**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number Base** | **10** |  | **System Values** | **0,1,2,3,4,5,6,7,8,9** |

|  |  |
| --- | --- |
| **Maximum Value Using 8 Digits** | **99,999,999** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Place Values** | **10,000,000** | **1,000,000** | **100,000** | **10,000** | **1,000** | **100** | **10** | **1** |
|
|
|
| **Frequency Value** |  |  |  |  |  | **2** | **1** | **2** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Addition Equivalent** | **2** | x | **1** | = | | **2** |
| **1** | x | **10** | = | | **10** |
| **2** | x | **100** | = | | **200** |
|  | x |  | = | |  |
|  | x |  | = | |  |
| **Total** | | | | | **212** | |

**Complete the table below to show how BINARY can be used to store the value 212**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number Base** | **2** |  | **System Values** | **0,1** |

|  |  |
| --- | --- |
| **Maximum Value Using 8 Digits** | **11,111,111** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Place Values** | **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
|
|
|
| **Frequency Value** | **1** | **1** | **0** | **1** | **0** | **1** | **0** | **0** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Addition Equivalent** | **1** | x | **4** | = | | **4** |
| **1** | x | **16** | = | | **16** |
| **1** | x | **64** | = | | **64** |
| **1** | x | **128** | = | | **128** |
|  | x |  | = | |  |
| **Total** | | | | | **212** | |

**Complete the table below to show how HEXADECIMAL can be used to store the value 212**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number Base** | **16** |  | **System Values** | **0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F** |

|  |  |
| --- | --- |
| **Maximum Value Using 8 Digits** | **FFFFFFFF** |

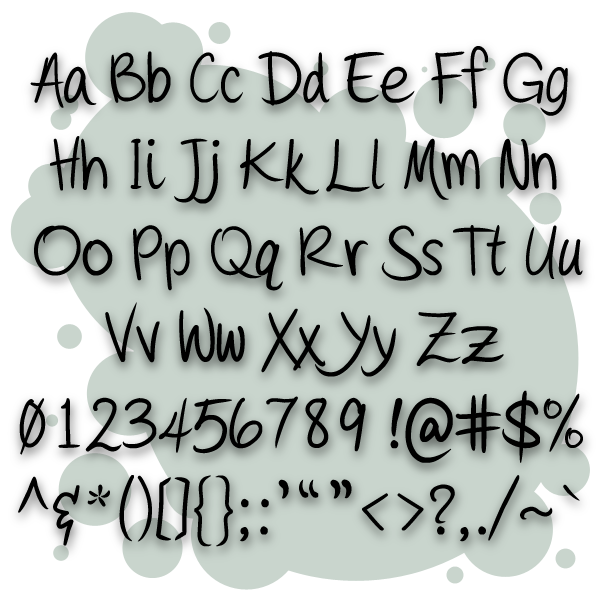
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Place Values** |  |  |  |  |  | **256** | **16** | **1** |
|
|
|
| **Frequency Value** |  |  |  |  |  |  | **D** | **4** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Addition Equivalent** | **4** | x | **1** | = | | **4** |
| **D(13)** | x | **16** | = | | **208** |
|  | x |  | = | |  |
|  | x |  | = | |  |
|  | x |  | = | |  |
| **Total** | | | | | **212** | |

**Show below how the number 1024 can be represented as Binary Coded Decimal**

|  |
| --- |
| 1 0 2 4  0001 0000 0010 0100  EQUALS  0001000000100100  Binary coded decimal is a way of representing a number by using four bits. The reason why four bits is used is because it allows the numbers 0-9 to be represented which are all the numbers in the decimal number system. As in the above example larger numbers are represented in one long string but still read in groups of four bits. |

# Part 2 – Representing Text



**Explain ASCII and how the characters “Hello World” can be represented**

|  |
| --- |
| **ASCII (American Standard Code for Information Interchange) is the most common format for text files. The standard was introduced to provide unity as to how characters, numbers and symbols are represented in binary.**  **To be able to convert ASCII into a different representation, a table must be used to lookup the corresponding representation.**  http://www.asciitable.com/index/asciifull.gif  **By using the table above, “Hello World” can be represented in decimal as:**  **72 101 108 108 111 32 87 111 114 108 100** |

# Part 3 – Fixed and Floating Point



**Explain Fixed Point Numbers**

|  |
| --- |
| **Fixed point numbers are one way in which a computer can store fractional numbers. The way that this is accomplished is by the processor placing an invisible decimal point in the middle of a memory register. For example, the decimal point in an 8 bit number would between bit 4 and 5.**  **The problem with using a fixed point number is that there is a large limitation on the size of the number that can be stored. For example, the largest number that could be stored in 8 bit would be '15.9375'. Additionally, the limited size will also affect accuracy in terms of the fractional. For example, if you wished to store the number 0.3 in 8 bit then the closest that it could be is 0.25.**  **It is important to note that computers will store these numbers in the form of binary. For example, 7.5 would be stored as 0111.1000** |

**Explain Floating Point Numbers**

|  |
| --- |
| **Like fixed point numbers this refers to how a computer can store fractional numbers. The difference between floating and fixed point is that floating point method has the ability to move the decimal point.**  **There are three parts to storing a floating point number: the base, exponent & mantissa.**  **The base number relates to the amount of numbers within a number system. An example of this is the decimal number system used by humans. The base of this system is ten because there are 10 numbers 0-9.**  **The exponent number comes when normalising a number. It is the amount of places the decimal point has moved to get to the right of the first non-zero number. It is a positive number if the decimal point moves left and negative if it moves right.**  **Finally, the mantissa (significant) are the numbers that come after the decimal point.**  **The first step is to normalize the number by moving the decimal point to the right of the first non-zero number.**  **The next step is to record the amount of times the decimal point has moved, positive number if it moved left and negative if it moved right.**  **The third step is to complete the formula by adding the base. The base is always going to be 2 because there are only 2 numbers in binary, the language that computers understand. An example of this would be:**  **1101.01011 = 1.101010111x23** |

**Demonstrate how the number 3.3 can be represented in 16 bit Floating Point Binary**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **The first step is to convert 3.3 into binary.**  **To convert an integer the number must be continually divided by 2 until 0.5 is reached. Throughout this process you must write a one must be if there is a fraction and zero if there is not. When writing a one or zero it's important to remember that you must start with the least significant bit, working from right to left.**  **3/2= 1.5=1**  **1/2=0.5=1**  **=11**  **To convert a fractional number you must continually multiple by 2 until 1.0 has been reached. If an integer is apparent then a one must be written and if not a zero. This time the process works from left to right starting with the most significant bit.**  **0.3x2=0.6=0**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **Although I have not reached 1.0 I have stopped because 16 bit floating point numbers can only store 10 bits for the fractional part.**  **Now I must put these two parts together as shown below.**  **11.0100110011**  **Now I need to move the decimal point to the right of the first non-zero number remembering that if it moves left then it is a positive number.**  **1.10100110011. The point moved 1 place to the left.**  **After this I must offset the exponent to avoid negative numbers. To do this I must do 2(exponent bits-1)-1. In 16 bit the exponent is 5 bits so**  **2(5-1)=24=16-1=15**  **Now I have to add this number to the amount of places the decimal point moved which was 1 so**  **15+1=16**  **I then have to change this number to binary as it's the only way the computer can store it so**  **16 8 4 2 1**  **1 0 0 0 0**  **Before I store the exponent I must add the sign of a number which can be done by placing a 0 because it is a positive number. Now I can put all the pieces together as shown below:**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Sign** | **Exponent (5 bits)** | | | | | **Mantissa** | | | | | | | | | | | **0** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | |

**How would storing 3.3 in 32 and 64 bit Floating Point Binary be different?**

|  |
| --- |
| **The binary figure would be different because there are a different amount of bits available for both the exponent and the mantissa.**  **Also because 0.3 is not divisible by one it will produce an infinitely recurring number. This means that the more bits available the more accurate the number is. So in this case the 64 bit number will be more accurate than the 32 bit number.**  **3.3 in 32 bit floating point binary:**  **3=11**  **------------**  **0.3x2=0.6=0**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **=11.01001100110011001100110**  **=**3.2999999523162841796875  3.3 in 64 bit floating point binary  3=11  ---------  **0.3x2=0.6=0**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **0.8x2=1.6=1**  **0.6x2=1.2=1**  **0.2x2=0.4=0**  **0.4x2=0.8=0**  **=11.0100110011001100110011001100110011001100110011001100**  **=**3.29999999999999982236431605997495353221893310546875 |

# Part 4 – Number System Conversion

[](http://www.freshideawebsites.com/wp-content/uploads/2013/04/conversion-large.jpg)

**Complete the following conversions and show your workings**

|  |
| --- |
| **12310 to Binary**  **123/2=61.5=1**  **61/2=30.5=1**  **30/2=15=0**  **15/2=7.5=1**  **7/2=3.5=1**  **3/2=1.5=1**  **1.5/2=0.5=1**  **=1111011**  **This method compromises of dividing the number by 2 each time until the number 0.5 is reached. When a fractional appears a one must written and when it is an integer a zero must be written. When writing the numbers you must work from right to left, starting with the least significant bit.** |
| **BA1116 to Binary**  B  1011  A  1010  1  0001  1  0001  =1011101000010001  To convert hexadecimal to binary you must first convert any letters to numbers. In this case B=11 and A=10. Because the highest number which is 16 fits into four bits this is how many digits long it must be. To get the number in binary you have to use the numbers 8, 4, 2 and 1 using one number only once. When a number is used a 1 must be written underneath it and if not a 0 is written. |
| **10111110102 to Decimal**  **512 256 128 64 32 16 8 4 2 1**  **1 0 1 1 1 1 1 0 1 0**  **512+128+64+32+16+8+2=762**  **=762**  **The method I used was to add the place holder values above the binary number. Then I took all the numbers that had a one underneath them and added them together.** |

**Complete the following conversions and show your workings**

|  |
| --- |
| **276610 to Hexadecimal**  **2766/2=1383=0**  **1383/2=691.5=1**  **691/2=345.5=1**  **345/2=172.5=1**  **172/2=86=0**  **86/2=43=0**  **43/2=21.5=1**  **21/2=10.5=1**  **10/2=5=0**  **5/2=2.5=1**  **2/2=1=0**  **1/2=0.5=1**  **=1010 1100 1110**  **8421 8421 8421**  **A C E**  **This method compromises of dividing the number by 2 each time until the number 0.5 is reached. When a fractional appears a one must written and when it is an integer a zero must be written. When writing the numbers you must work from right to left, starting with the least significant bit.**  **After that I split the numbers into groups of four and then added the place holder values below these numbers. Then I take all the numbers in the group of four with a one above them and add them together. I also needed to remember that if the number was 10 or higher then it would be represented by a letter. 10=A and 12=C** |
| **BAD16 to Decimal**  **B A D**  **11 10 13**  **1011 1010 1101**  **2048 1024 512 256 128 64 32 16 8 4 2 1**  **1 0 1 1 1 0 1 0 1 1 0 1**  **2048+512+256+128+32+8+4+1=2989**  **=2989**  **The first step is to convert the letters into their number equivalent in hexadecimal. I must then convert these numbers to binary using the placeholders 8, 4, 2 and 1, writing a 1 whenever I use a number. I now need to put these sets of four numbers together into one large number. Then I put the place holders over them and add them together when there is a 1 below them.** |
| **10011101010110100011110010101012 to Hexadecimal**  **0100 1110 1010 1101 0001 1110 0101 0101**  **4 E A D 1 E 5 5**  **=4EAD1E55**  **Firstly, I split the large string into sets of four starting from the right. Then by using placeholders of 8, 4, 2 and 1 I was able to figure out their hexadecimal value. I also needed to remember that if the value was 10 or greater then it would be a letter.** |

**Complete the following conversions and show your workings**

|  |
| --- |
| **7.15310 to Floating Point**  **Move the decimal point to the right of the first non-zero number and record how many times the decimal point has moved**  **= 0**  **The number is in decimal so the base is**  **=10**  **=7.153 x 100** |
| **1.1559 x 10-3 to Fixed Point**  **Move the decimal point to the left 3 times because it is to the power of negative 3**  **= 0.001155910** |
| **100111.0101012 to Floating Point**  **Move the decimal point to the right of the first non-zero number**  **=1.00111010101**  **Record the change in places of the decimal point**  **=5**  **Base is 2 because it is binary so**  **= 1.00111010101 x 25** |

**Complete the following tables of conversions**

|  |  |  |
| --- | --- | --- |
| **Binary** | **Decimal** | **Hexadecimal** |
| **10011100**2 | 156**10** | 9C**16** |
| **1011111011**2 | **76310** | 2FB**16** |
| **10101000**2 | 168**10** | **A816** |
| **11010011**2 | 211**10** | D3**16** |
| **1000000110100**2 | **103410** | 40A**16** |
| **111000100**2 | 452**10** | **1C416** |
| **10101010**2 | 170**10** | AA**16** |
| **1010100110010**2 | **542610** | 1532**16** |
| **101010111100**2 | 2748**10** | **ABC16** |

|  |  |
| --- | --- |
| **Floating Point** | **Fixed Point** |
| **4.675 x 102** | 467.5**10** |
| **3.245765 x 103** | **3245.76510** |
| **12.234567 x 104** | 122345.67**10** |
| **1.1011 x 21** | **11.0112** |
| **1.11011 x 2-2** | 0.0111011**2** |
| **1.01101 x 2-4** | **0.0001011012** |

# Part 5 – Boolean Logic



**Identify the following Logic gates and complete the Truth Tables**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D:\Users\simonm\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\ZTG4A4RL\Logic-gate-and-us[1].png   |  |  |  | | --- | --- | --- | | **Gate** | AND | | | **A** | **B** | **Y** | | 0 | 0 | 0 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 1 | | D:\Users\simonm\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\ZTG4A4RL\Logic-gate-or-us[1].png   |  |  |  | | --- | --- | --- | | **Gate** | OR | | | **A** | **B** | **Y** | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 1 | |
| D:\Users\simonm\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\ZTG4A4RL\Logic-gate-inv-us[1].png   |  |  |  | | --- | --- | --- | | **Gate** | NOT | | | **A** | **B** | **Y** | | 0 |  | 1 | | 1 |  | 0 | |  |  |  | |  |  |  | | D:\Users\simonm\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\T00WSYZ2\Logic-gate-nand-us[1].png   |  |  |  | | --- | --- | --- | | **Gate** | NAND | | | **A** | **B** | **Y** | | 0 | 0 | 1 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 0 | |
| D:\Users\simonm\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\T00WSYZ2\xor_gate[1].jpg   |  |  |  | | --- | --- | --- | | **Gate** | XOR | | | **A** | **B** | **Y** | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 0 | | D:\Users\simonm\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\JTJZU8Q6\Logic-gate-nor-us[1].png   |  |  |  | | --- | --- | --- | | **Gate** | NOR | | | **A** | **B** | **Y** | | 0 | 0 | 1 | | 0 | 1 | 0 | | 1 | 0 | 0 | | 1 | 1 | 0 | |

**Draw the Logic Circuit for the following Boolean Expression**

**And show the change in inputs/outputs for the following values**

**A = 1 B = 0 C = 1 D = 1**

**And find the value of Y**

|  |
| --- |
| A AND B= 0  C OR NOT D= 1  (A AND B) OR (C OR NOT D)= 1  A EXCLUSIVE OR C= 0  B AND D= 0  NOT ( A EXCLUSIVE OR C) AND (B AND D)=1  (NOT((A AND B) OR (C OR NOT D)) OR (NOT(A EXCLUSIVE OR C) AND (B AND D)))= 0 |