**M1 – Explain, using examples, how floating point numbers can be represented in binary**

**Introduction**

In this report, I will explain, using examples, how floating point is represented in binary. I will give clear examples of how this is used. I will also say the benefits of how it is represented in floating point and standard form. Lastly, I will explain how it is used in 32 bit and 64 bit processors.

**Fixed-point numbers**

Fixed-point numbers are numbers that represents usually base-2 or base-10. This is different from integer numbers because an integer is just a number. Fixed-point numbers are decimal numbers. We are used to using standard form (128 64 32 16 8 4 2 1), but these place values are ½, ¼, and others. These place values are used for it to convert fixed point to denary or binary numbers. For example, 1.23 can be represented as 1230 in a fixed-point data type. It is very different to integer number. They are more examples that are different, but I only named one of them.

**Examples**

**68.25**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** | **:** | **½** | **¼** | **1/8** | **1/16** |
| **0** | **1** | **0** | **0** | **0** | **1** | **0** | **0** | **:** | **0** | **1** | **0** | **0** |

This is an example of how fixed-point is used. As you can see, the fixed-point numbers are ½ and the rest. This is calculated normally by adding the numbers to get 68, 64+4. However, they want the code to be with ‘.25’ they have added with it. ¼ as a decimal is known as 0.25. If you add this to the 68 that is already complete, this makes 68.25. All you do is put one in from of ¼ and the rest would be zero. You only put one in if you want to add it together.

**Table**

|  |  |  |
| --- | --- | --- |
| **Binary Fraction** | **Fraction** | **Decimal Fraction** |
| **0.1** | **1/2** | **0.5** |
| **0.01** | **1/4** | **0.25** |
| **0.001** | **1/8** | **0.125** |
| **0.0001** | **1/16** | **0.0625** |

This table explains how each of them work. As you can see, for each column, there is a pattern going on. They are more added on and the more the pattern is common. For the binary fraction, each of the numbers are added on. As you can see, ½ is 0.1 and ¼ is 0.01. It added a one to it. For fraction, it doubles each time. After 1/8, it would be 1/16. For decimal fraction, the numbers times itself to get the other one. 0.5\*0.5=0.25. Another example could be 0.25\*0.25 = 0.0625. However, it does not work for all of them. If you want to find out the decimal, all you need to do is divide the 1 with 16 to find out the answer.

**Floating-point numbers**

Floating-point is a method that represents an estimation used for real numbers. The real number can be anything – such as minus or a positive number and it can consist of the base used. It uses a method such as 3\*10. This is just an example of it. This is used to scale numbers. The benefits of floating point and standard form is that it can represent much wider range of values. If anything is necessary and it is in need of converting, you can use this to convert it. For example, standard form is used in any of the conversions we have made e.g. floating-point to denary.

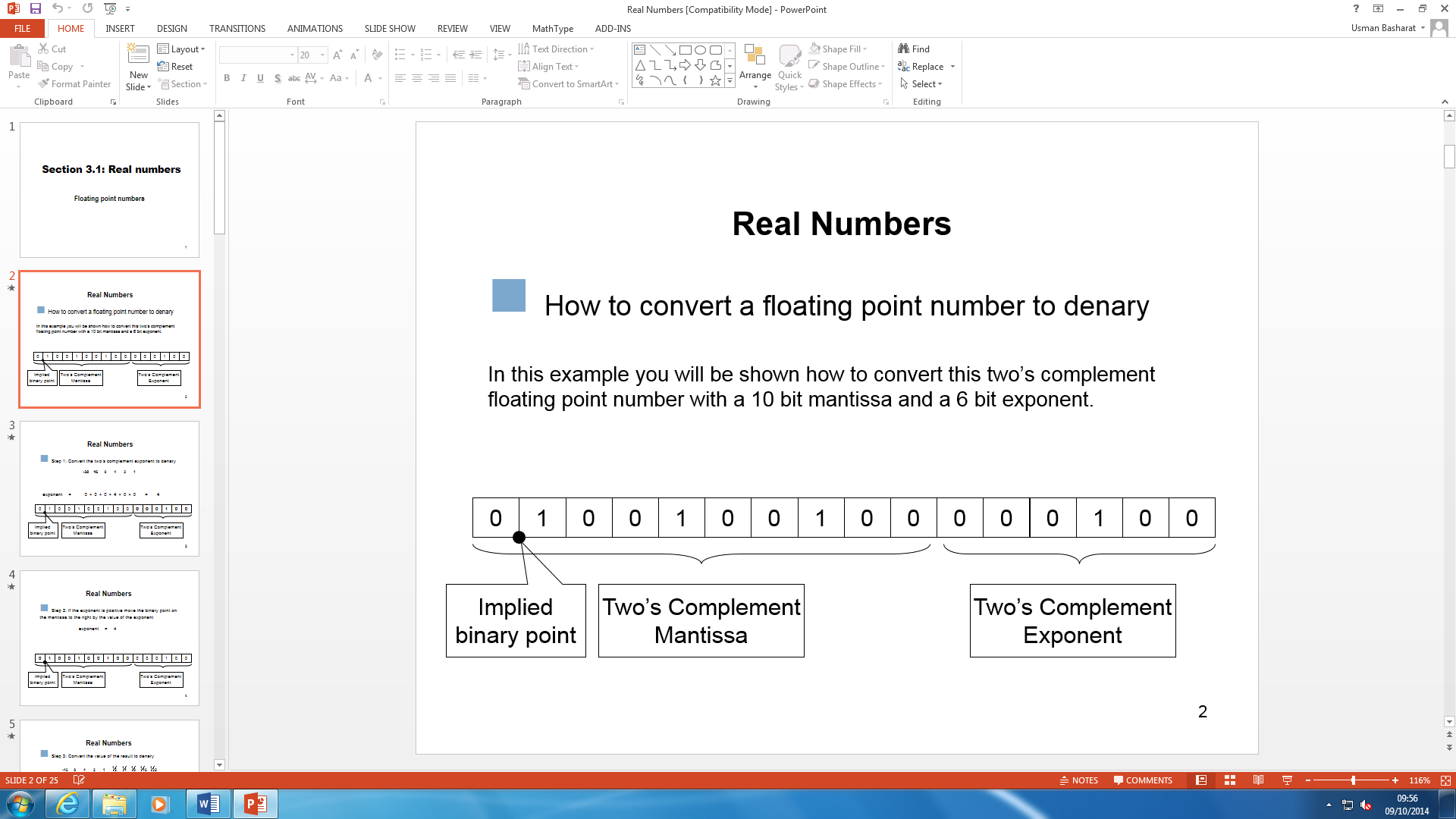
Converting the floating-point numbers would be shown below.

**Converting**

This is the start of how to convert floating point to denary. The first thing you do is understand the picture below. Implied binary point is what is used to separate the point from integers to fixed point. The mantissa and exponent is always added together and they stay the same. This stays as 10-bit mantissa and 6-bit exponent. The place values contribute to the 10 bit and 6 bit. The place value is the following for exponent and mantissa:

**Exponent Mantissa**

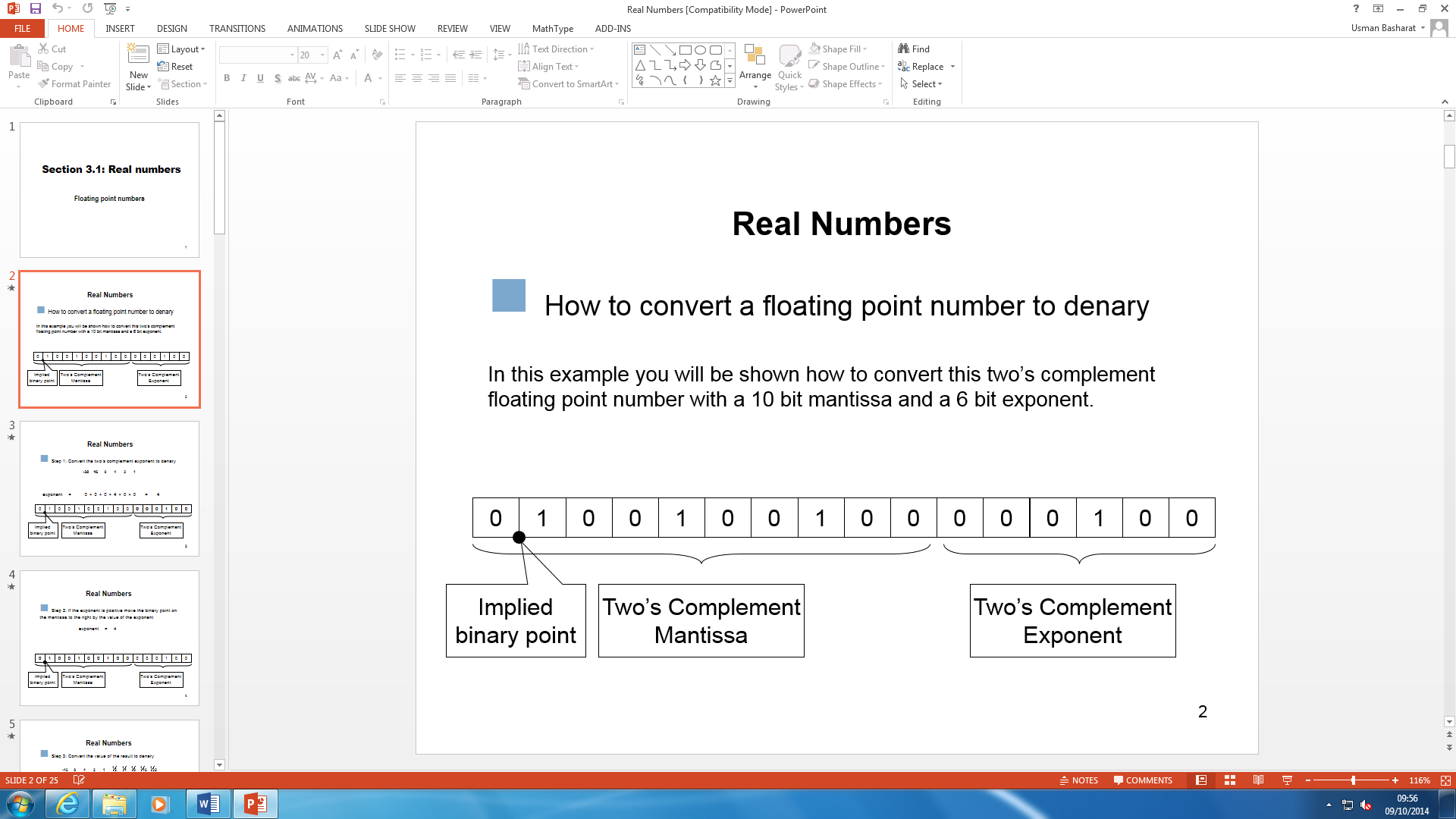
**-32 16 8 4 2 1 -16 8 4 2 1 ½ ¼ 1/8 1/16 1/32**



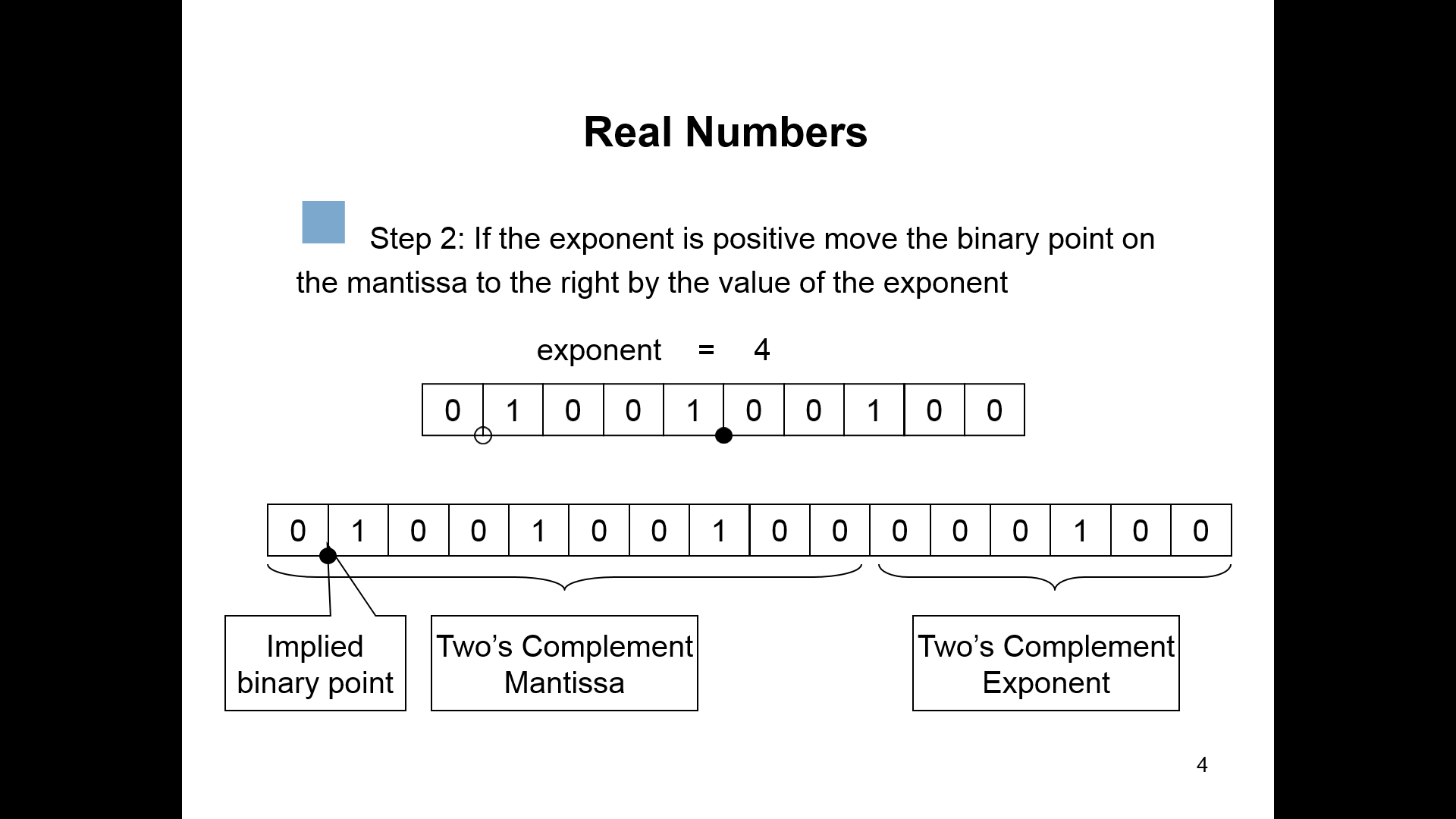
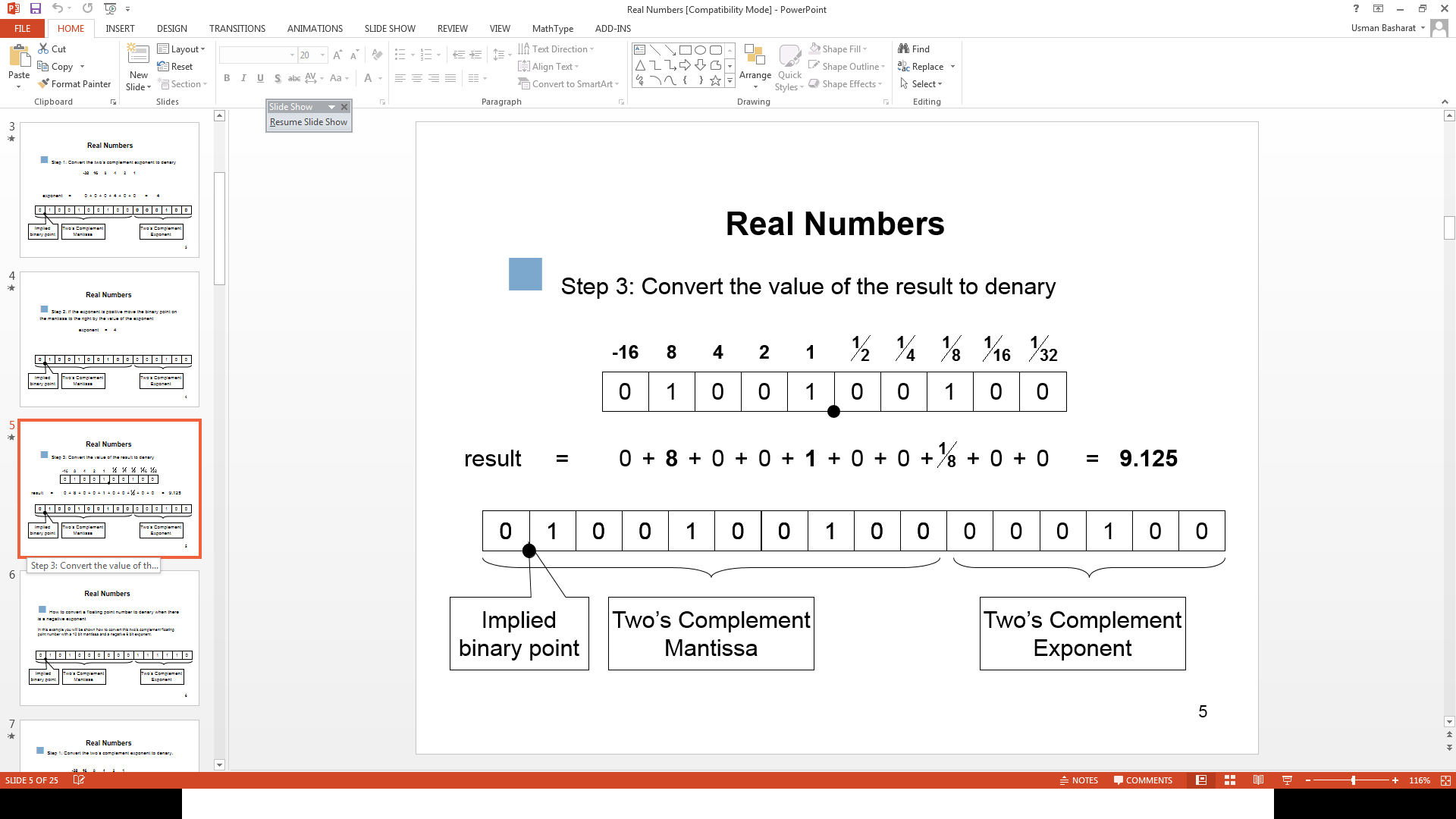
This picture breaks it down of how it is shown.

Once the exponent is added, either positive or negative, you move the implied binary point to the left if negative or right if positive. Once complete, you use the mantissa to add up the rest and you get the binary number. Using fixed-point, the answer can be decimal or normal if the others are not added.

**Examples**



If this was used, first of all, you calculate the exponent. Taking out the first 6, from the right,

This would be 001000. Using the exponent place values, you add them up. This would determine whether it would be go to the right or left. It would be a positive 4. Moving the point 4 places. Once this is complete, you use the mantissa to convert it from fixed-point to denary now.

**Used in 32 bit and 64 bit processors**

They are both different as both are used differently for different processors. 32 bit is used for both processors, Intel and AMD. This was used very early on once the computers were released. So, 32 bit was used for Windows 95, 98 and XP. 64 bit is used for the latest operating system, which are Windows 7, 8 and Vista. The difference for the two is the memory The RAM for it is different, and 64 bit can be used over 4GB whereas 32 bit cannot be used over 4GB. 32 bit and 64 bit are used in three precision that is used in the languages. The three precision is the following:

* **Single precision –** This is used in the C language family. The binary format is used in the 32 bit.
* **Double precision** - This is used in the same as single precision, but it is doubled. Therefore, it uses 64 bit in the binary format.
* **Double extended** – This is extended from the double. Therefore, the binary format would be used more than 64 format.

All of these above are used in either one of the two processors. These two pictures show how they are both represented in 32 bit and 64 bit.

