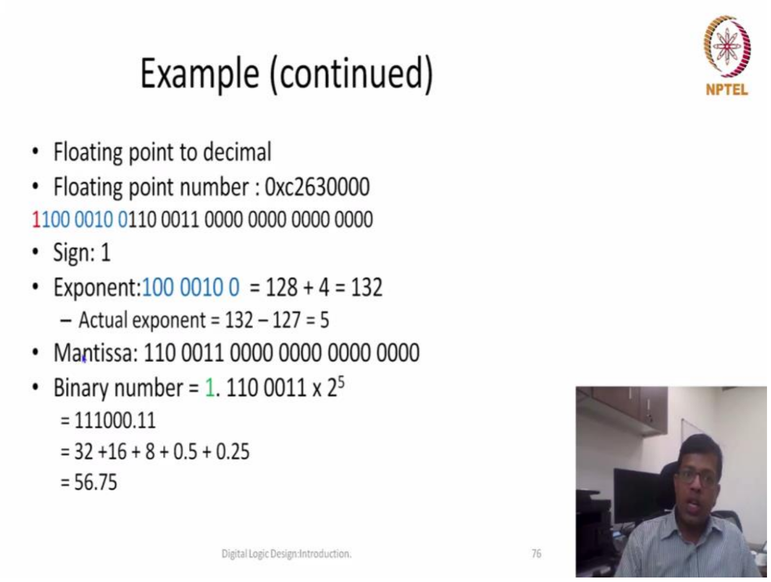
Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture 10
Floating Point Number - 3

Hello everyone, so you know in yesterday's lecture we have understood how to represent Floating Point Numbers in binary and floating-point Numbers we have also seen how to represent Fixed Point numbers and floating-point numbers in binary. Now, we will continue with the same lecture with the same topic. So let us, let us have one more example, which will help us understand let us say we have given with a floating-point number in binary then how to convert it back into decimal number.

Along with that we will also in this in this lecture we will also learn about various other aspects of floating-point numbers and how to increase the precision and decrease the precision and many other things. Let us see. So, let us say we have a floating-point number which is represented in binary now we would like to extract what is the decimal number it represents.

(Refer Slide Time: 01:19)



Example (continued)

- Floating point to decimal
- Floating point number : 0xc2630000

1100 0010 0110 0011 0000 0000 0000 0000

- Sign: 1
- Exponent:100 0010 0 = 128 + 4 = 132
  - Actual exponent = 132 – 127 = 5
- Mantissa: 110 0011 0000 0000 0000 0000
- Binary number = 1. 110 0011 x 2^5
  - = 111000.11
  - = 32 +16 + 8 + 0.5 + 0.25
  - = 56.75

Digital Logic Design:Introduction. 76

So, I am taking it as an example. So, let us say example. For example, the floating-point number is 0 cross C 263 four 0s. So, if such a number is given to us, then how to start first we will represent this, this hexadecimal number into binary. In binary will again further make the groups of 4. So, the group of 4 means, like we will break it into nibbles, so that it is make, make things easy.

So, first is C, C means 12. And then the next nibble is to next level is 6, and the next level is 3 and the rest of the 4 levels are 0. Now, we know now the binary notation, or this floating-point number from our previous lecture, we understand that each binary representation would have exponent assigned and mantissa. And we also know that the 31st bit in a 32-bit number is going to represent sign, and next 8 bits, that means 1000010, and 1, this part would represent exponent and the rest is, is mantissa.

So, I will write all of these things separately sign is 1 and exponent is 1000010 and 0. So, if I put this in decimal, this become a 128 plus 4, so how do I find 128 this is a weight of this number is 1 this is 2, this is 4, 8, 16, 32, 64, 128. So, 128 plus 4 is 132. So, the actual exponent we have to reduce the offset also which we were adding to represent the negative exponents. So, the actual exponent is this is the exponent which we have written in the floating-point binary number, but the actual exponent we have to subtract the offset.

So, in, in case of 32 bit binary 32-bit, floating point number offset is 127. So, let us subtract this 127 from 132 remaining is fine. So, rest of the bits rest of the rest of the how many bits? 23 bits, so the rest of the 23 bits are mantissa. So, in mantissa, we will write it from here 1100011 and then rest of the 0s. So, remember, when we were converting a binary number or a decimal number 2 exponent then we first had to have first we were having significant then we were expecting mantissa.

So, the same step will go in as a as a next step here. So actual, this binary number I can write it like this 1 is a significant which I need to add. So, the significant would be added and then I am writing the mantissa bits and skipping all the 0s because they do not count. So, all the non zeros all the trailing nonzero numbers, I am adding so that means 1100011. So, and exponent I got as 5 so into till 2 is to power 5.

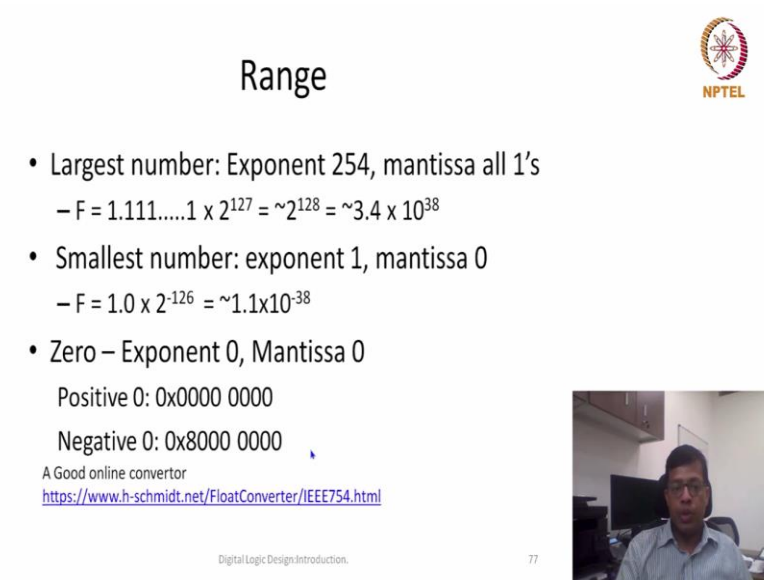
So, this is this is the binary number we can represent or the scientific notation of this, this floating-point number. So, this floating-point number has this binary number as a notation. Now, what is the next step because this is 2 is to power 5, so, let me multiply 2 is to power 5 to this number. So what it will become, so that means this decimal point would be shifted by 5 points 1

2 3 4 5. So, this decimal point would be shifted here, and I can rewrite this number as 11100 decimal point 11.

So, now I can create the create the decimal number which is 32 plus 16 plus 8 plus 0.5 plus 0.25. So, this is 1 by 2 this is 1 by 4 and adding all of them will, will give me 56.75. So, this is how would be a general procedure if I have not even I came to know about a floating point number which is represented in hexadecimal or in binary then I have to go this step by step first extracting sine exponent and mantissa and then re creating the binary number and from there, I will create the decimal number.

Good. So, with this now, we understand how to convert a decimal number to floating point number floating point number 2 decimal numbers. Now, let us see, we were the motivation of this decimal numbers. What do you know the floating point numbers was 2 fold 1, I would like to represent fractions the second thing was I wanted to represent very large numbers as well as very small numbers. So, let us try to see that if I want to write or want to express the largest number or large number how much large it can be, if the notation is that these 32 bit floating point numbers.

(Refer Slide Time: 07:09)



Range

- Largest number: Exponent 254, mantissa all 1's
  - F = 1.111.....1 x 2^{127} = ~2^{128} = ~3.4 x 10^{38}
- Smallest number: exponent 1, mantissa 0
  - F = 1.0 x 2^{-126} = ~1.1x10^{-38}
- Zero – Exponent 0, Mantissa 0

  Positive 0: 0x0000 0000

  Negative 0: 0x8000 0000

A Good online convertor
https://www.h-schmidt.net/FloatConverter/IEEE754.html

Digital Logic Design:Introduction. 77

So, for the largest number, the exponent range for, for these floating-point numbers for 32-bit floating-point numbers are actually 8 bits are assigned. So, 8 bits means 0 to 255. But in previous lecture, we have stated 1 fact that 0 and 255 is not allowed so, that means they are reserved. So, let us not use them, so the maximum exponent we can have is 254. And the minimum exponent we can have is 1, not 0.

So, we will see how 0 and 255 are reserved. So, that would come. Now, the largest exponent is 254. And if I want to write the maximum number, certainly the exponent has to be the largest that is 254. And what about mantissa? So, mantissa, in mantissa, all numbers are 1, all bits are 1, so that would represent the highest number. So, if I write all the 1s, and we will also add the significant part, so, the number will become something like this 1 point, all 1's in mantissa like there are 23 1s and then 2 is power 127.

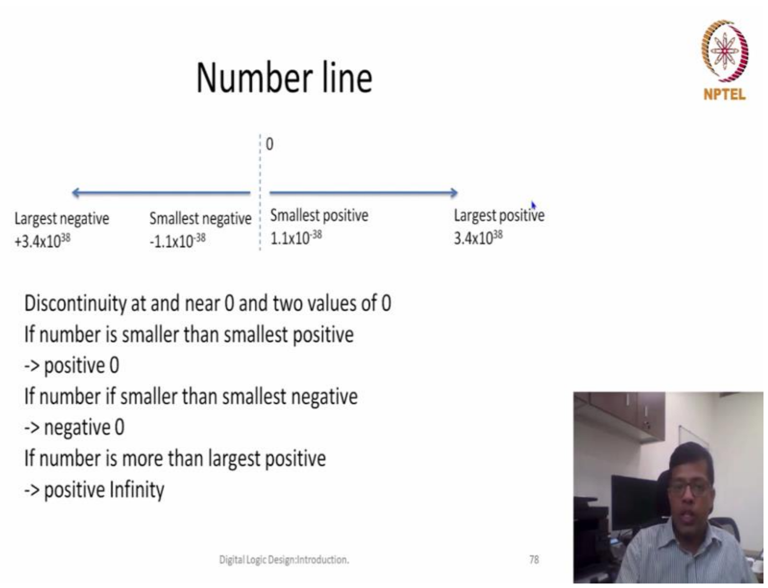
So, why this 127 because I have reduced or subtracted my offset of 127 from 254. So, the value is 254 minus 127 equal to 127. So, if we compute this if I if I see this really close, so this is 1.1111 up to 23 bits, so that is very, very close to 2 actually. So, so if I expand this what does it mean this is 1 plus 1 by 2 plus 1 by 4 plus 1 by 8 plus 1 by 16 plus 1 by 32 plus 1 by 64 plus 1 by 128. So, this is geometric progression.

So, and in the geometric progression if I would like to calculate the sum, it would be equal to 2 so that is how it is approximately 2 is to power 128. So actually 2, 2 bit multiplied with 2 into power 127. So, we will finally have to 2 is to power 128 which is approximately equal to 3.4 into 10 is to power 38. So, which essentially means that around 10 is power 38. So, there are 38 0s at 1 before 1 so that much large number I can represent in single precision floating point numbers. And how many decimal digits so because my mantissa are represent around 23 bits. So, approximately 7 decimal digits could be there as a fractional part in any of the scientific notation. So, what about the smallest number? In case of smallest number my exponent would be 1, because exponent is 1.

So, the real exponent will be 1 minus 127 that means minus 126 mantissa bits, all of the mantissa bits would be 0 because I am trying to find out the minimum number, the number can be represented as 1.0 into 2 is to power 126 minus 126 and this comes to be around 1.1 into 10 is power minus 38. So, this is the smallest number. So, this satisfies a good fraction of number that we see around.

So precision is around 7 decimal digits and maximum number is stand for 38 and minimum is stand for minus 38. Reasonably good number of numbers can be represented using this single precision floating point number. Now what about 0 so he put exponent and mantissa both has 0 the positive there would be 2 0s 1 is positive 0, another is negative 0. So, in our one of the previous lectures, when we were discussing negative numbers, we were we are highly object objecting this thing that there should not be 2 different notions of 0s. But here, we are accepting this, this fact that 0 will have 2 different notions or 2 different representations, why, let us try to understand this thing using line using a number line.

(Refer Slide Time: 11:54)



Number line

Discontinuity at and near 0 and two values of 0
If number is smaller than smallest positive
-> positive 0
If number if smaller than smallest negative
-> negative 0
If number is more than largest positive
-> positive Infinity

Digital Logic Design:Introduction. 78

So, number line is a is a horizontal line, which, which would, which would say that, let us say we will have a 0 studying from this and then the numbers are keep on increasing in the arrow direction. So, if this number line represent what kind of numbers can be represented using single precision floating point number, then we can see that there is a discontinuity. So, the smallest number that can be represented is, is 1.1 into 10 is to power minus 38 and the largest number is 3.4 into 10 is to power 38. So, what does this mean try to understand this that this number 1.1 into 10 is to power 38 is very, very close to 0.

Because it is very, very close, it can be approximated to 0 also in certain scenarios, but still, the fact remains that there is a discontinuity that we cannot represent any numbers smaller than 1.1 into 10 is to power minus 38. Similarly, this is also a fact that any number which is larger than 3.4 into 10 is to power 38, that cannot be represented using this particular notation, no single precision, so there will be kind of an overflow, how do we represent overflow? That is also one question which we need to answer.

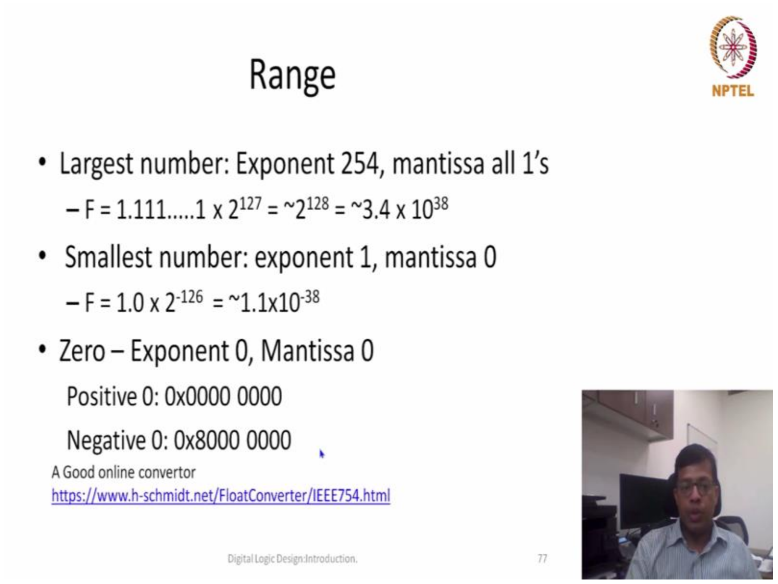
Similarly, this, this line number line can also be extended in the negative direction. So, in the negative direction also, there would be the smallest negative number is minus 1.1 into 10 is to power minus 38. Again, there is a discontinuity and the largest number is 3.4 into 10 10 is to power, 10 is to power minus 38. So, there is a negative sign here will correct this. So, now you see that the number which is smaller than 10 is to power 38 and larger than 0.

What should it be called? Because that number is very, very close to 0, we can call it positive 0, it is more than 0 but lesser than 10 is to power minus 38. So that, that is why we can call it positive 0. So, the positive has a certain meaning here. Positive certainly says that the number is more than 0, but it is less than 10 is to power minus 38. So, that will have at least some meanings. So, either it is completely 0 or it is slightly more than 0 but less than 10 10 is to power minus 38 which cannot be represented in, in the normal representation.

Similarly, there could be a negative 0 also the number is smaller, then minus 10 or minus 38, but more than 0, because it is more than lesser than 0, but, but more than this, so, that means, there is a discontinuity, this number would be represented as negative 0 that means 0 cross 8 7 0s in hexadecimal. So, what we call a number which is larger than this 10 is to power 38, 10 is to power 38 itself is a very large number, if a number is even larger than this what should we call this number? Can we call that infinity?

Yes. So, there could be now 2 infinities one is a positive infinity the other could be negative infinity any number which is larger than the number which can be represented using single precision numbers is positive infinity in the similar way any number which is less than, than the largest negative number is a minus infinity. So, this, this is the way we can understand that what kind of numbers can be represented using, using single precision floating point number.

(Refer Slide Time: 16:36)



Range

- Largest number: Exponent 254, mantissa all 1's
  - F = 1.111.....1 x 2^{127} = ~2^{128} = ~3.4 x 10^{38}
- Smallest number: exponent 1, mantissa 0
  - F = 1.0 x 2^{-126} = ~1.1x10^{-38}
- Zero – Exponent 0, Mantissa 0

  Positive 0: 0x0000 0000

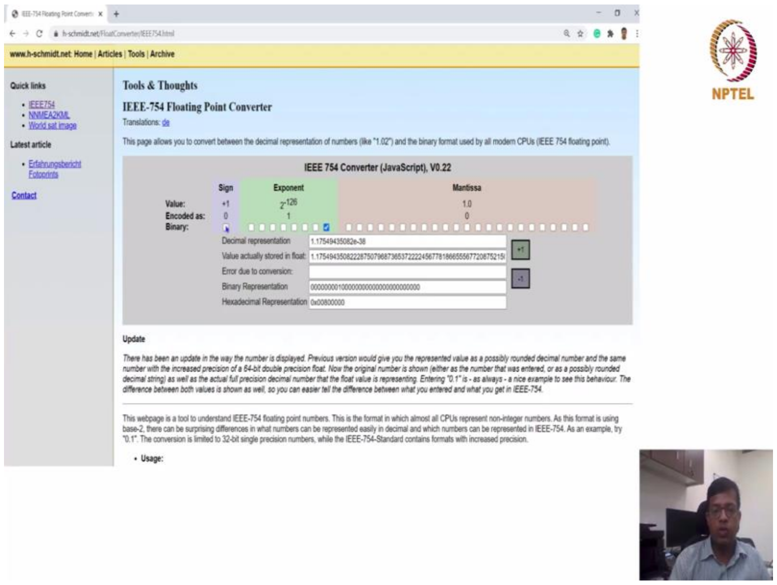
  Negative 0: 0x8000 0000

A Good online convertor
https://www.h-schmidt.net/FloatConverter/IEEE754.html

Digital Logic Design:Introduction. 77

Now, there is a one interesting website which, which tells a little more interesting facts about this this is called this, this is a very nice web browser. This browser h summit dotnet float converter IEEE754. So, this, this, this browser, this website gives quite a good intuitive information that what kind of numbers can be represented and we can play around a little bit. So, I will give a small demonstration of this, this this website.

(Refer Slide Time: 17:08)


IEEE-754 Floating Point Conver
h-schmidt.net/FloatConverter/IEEE754.html
www.h-schmidt.net: Home | Articles | Tools | Archive
Quick links
IEEE754
NNMEA2KML
World sat image
Latest article
Erfahrungsbericht Fotoprints
Contact
+1
-1

Tools & Thoughts

IEEE-754 Floating Point Converter

Translations: de

This page allows you to convert between the decimal representation of numbers (like "1.02") and the binary format used by all modern CPUs (IEEE 754 floating point).

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{-126}	1.0
Encoded as:	0	1	0

Decimal representation: 1.17549435082e-38

Value actually stored in float: 1.1754943508222875079687365372222456778186655567720875215

Error due to conversion:

Binary Representation: 00000000100000000000000000000000

Hexadecimal Representation: 0x00800000

Update

There has been an update in the way the number is displayed. Previous version would give you the represented value as a possibly rounded decimal number and the same number with the increased precision of a 64-bit double precision float. Now the original number is shown (either as the number that was entered, or as a possibly rounded decimal string) as well as the actual full precision decimal number that the float value is representing. Entering "0.1" is - as always - a nice example to see this behaviour. The difference between both values is shown as well, so you can easier tell the difference between what you entered and what you get in IEEE-754.

This webpage is a tool to understand IEEE-754 floating point numbers. This is the format in which almost all CPUs represent non-integer numbers. As this format is using base-2, there can be surprising differences in what numbers can be represented easily in decimal and which numbers can be represented in IEEE-754. As an example, try "0.1". The conversion is limited to 32-bit single precision numbers, while the IEEE-754-Standard contains formats with increased precision.

- Usage:

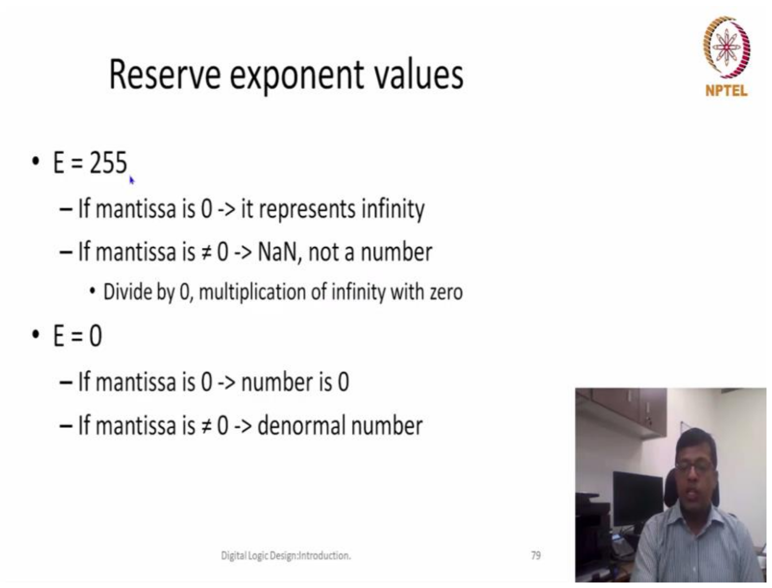
So, we can see here that you can write the numbers in any form so, let us say I would like to write the number in decimal. So, in decimal, I say, let us say the number is 1000. I press the enter button. As soon as I press the Enter button, it says that your number is 1000 in decimal and after that, now binary representation is this and the beautiful thing is that along with the binary representation in this form, it also breaks things into sign, exponent and mantissa. So, it says your exponent is there is a this bit is high and this bit is high and similarly in the mantissa also it specifies that what is the mantissa and this part is essentially significant 1.953125 is, is, is a significant part.

So, similarly this is the hexadecimal representation. Now, I can I can play around with that I can say that this I can toggle the bits become a negative number or I can also see what is the largest number so, in that case, I will make all of these bits as, as 1. And so, I have to keep this as the minimum bit to 54 is the maximum decimal. So, if I if I say, let us say if I put the maximum exponent as 255 then it says the number is infinity.

However, for the 254 this is the number which is being represented the decimal number is 1.7 into 10 is to power 38. Now, if I keep all the mantissa bits as 1, so, that means I have to write all of this as f, one more, yes. So, this says this is the largest floating-point number, which can be represented using single precision numbers. Now, similarly, let us see what would happen if I want to represent the smallest numbers in the case of smallest number all of these would be 0.

So, I want to say all of these are 0s. And I also would like to say that my exponent is, is only 1 you can see the range of numbers which are being reduced. So, you can see that if this is the smallest number this, this number can be represented something like this. Now, this is a positive number and this is a negative number. So, this, this calculator is an interesting tool to play around with the numbers and to see that how we can convert 1 decimal number into floating point number or vice versa. So, with this demo, we can again move forward and look at some more examples more questions.

(Refer Slide Time: 20:37)



Reserve exponent values

- E = 255
  - If mantissa is 0 -> it represents infinity
  - If mantissa is ≠ 0 -> NaN, not a number
    - Divide by 0, multiplication of infinity with zero
- E = 0
  - If mantissa is 0 -> number is 0
  - If mantissa is ≠ 0 -> denormal number

Digital Logic Design:Introduction. 79

So, now we will talk about these reserve bits, we said that two reserve exponents one is 255, another reason 0. So, when exponent 255 that means all bits, so, my exponents are 1, I have we have seen in the in this converter also and we have discussed a few minutes before that the number represents infinity and it represents infinity only mantissa is 0. So, that means all bits of mantissa is 0 then only it represents infinity.

One more number we need to represent that is called not a number. So, consider this situation, let us say you want to divide certain number by 0 what is the result? You have been told in your primary classes, secondary classes the number has to be infinity. But now, let us look at it from a very close perspective. You understand that whenever you are writing 0, this number 0 means the number is smaller than 10 is for minus 38, but more than 0. Now, since you are dividing this number, dividing any number by 0, that means that this particular number could be anything here, most likely we can call it infinity because the number would be more than 10s for 38. So, it cannot be represented so, but let us say the other case 0 by 0.

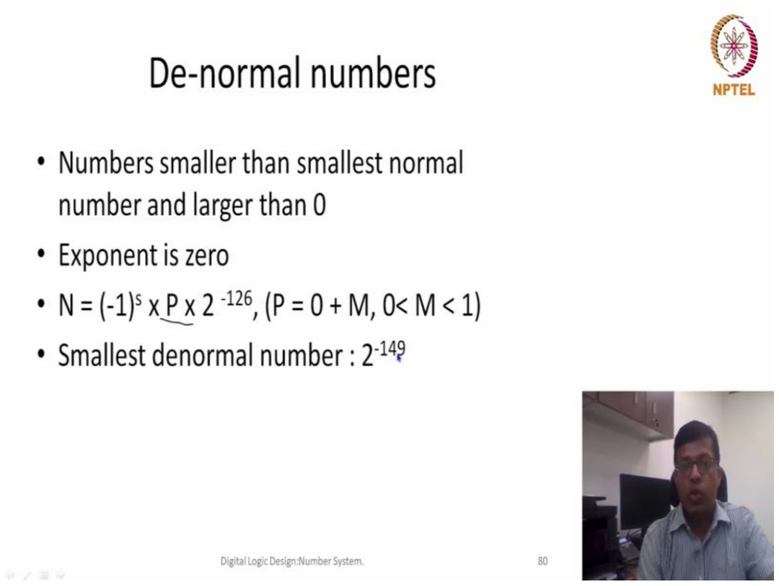
So, 0 by 0 is undefined, because your upper numerator is also smaller than 10 is to power minus 38. Your denominator is also smaller than 10 is to power minus 38. You do not know what would be the output of this division. Similarly, if you multiply infinity with 0, what is the output infinity by, by 0 infinity means it is more than 10 is to power 38. And 0 means it is lesser than 10 is to power 38, you can multiply these 2 numbers, we do not know what is the output.

So, this kind of scenario is called not a number. We do not know these undefined scenarios. So, in this case of undefined, we call we have a special representation, that a mantissa will be not equal to 0, and my exponent is equal to 255. So, this represents a special scenario, which means not a number. And in this not a number, there is also one more special category, which is called signaling and non signaling.

And so that means if we want to raise an error you want to raise an exception, then the most significant bit of your mantissa is turned out as one so that it signals that there is an error. So, that is that is why we have reserved this 255 as a special exponent just to represent all the infinity or not a number of cases. Now, what if my exponent is 0 in my exponent is 0 and mantissa is also 0, we say the number is 0.

And if exponent is 0 mantissa is not equal to 0, then it is a very special case we call it Denormal numbers. So, these Denormal numbers are not supported by all systems they are not supported by all processors. But if some particular processors are supporting it, then its meaning is slightly different from all other numbers.

(Refer Slide Time: 24:39)



De-normal numbers

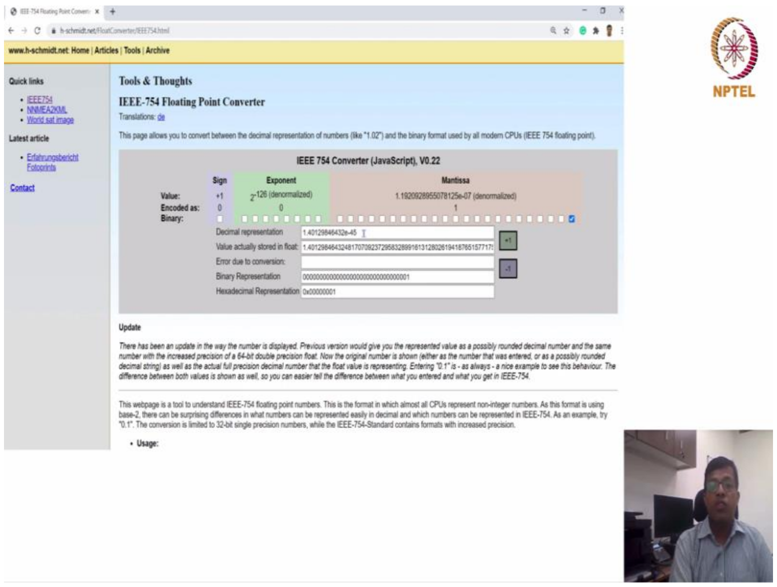
- Numbers smaller than smallest normal number and larger than 0
- Exponent is zero
- $N = (-1)^s \times \underbrace{P}_{} \times 2^{-126}$, $(P = 0 + M, 0 < M < 1)$
- Smallest denormal number : 2^{-149}

Digital Logic Design:Number System. 80

So, in all other numbers whenever we are trying to represent, we, we simply say this, this formula, that minus 1 is to power s and then P and then 2's power whatever exponent was written minus 127. Here, because exponent is written as 0, we will write it directly 2 is to power minus 126. And the leading 1 here, you leading number 1 that in significant we this, this number represents significant as well as, so, this, this number P represent significant as well as mantissa here significant is turned out to be 0 and the mantissa is added.

So, and mantissa we always say it is less than 0 and 1. So, that is why this particular denormal numbers because their calculation is different. So, they are classified as a different number, they are not normal numbers and further because of this, this, this additional complexity some of the computers also do not support these denormal numbers. So, these numbers are of course, smaller than the than the smallest normal numbers. So, this there is smaller than the smallest normal number and they are the smallest than the smallest normal number but still larger than 0. So, if a if a keep on going like this the smallest denormal number is equal to 2 is to power minus 149.

(Refer Slide Time: 26:21)



www.h-schmidt.net: Home | Articles | Tools | Archive

Quick links

- IEEE754
- NNMEA2KML
- World sat image

Latest article

- Erfahrungsbericht Fotoprints

Contact

Tools & Thoughts

IEEE-754 Floating Point Converter

Translations: de

This page allows you to convert between the decimal representation of numbers (like "1.02") and the binary format used by all modern CPUs (IEEE 754 floating point).

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	+1	2^{-126} (denormalized)	1.1920928955078125e-07 (denormalized)
Encoded as:	0	0	1
Binary:	☐	☐☐☐☐☐☐☐☐	☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☑

Decimal representation	1.40129846432e-45
Value actually stored in float:	1.40129846432481707092372958328991613128026194187651577175
Error due to conversion:	
Binary Representation	00000000000000000000000000000001
Hexadecimal Representation	0x00000001

Update

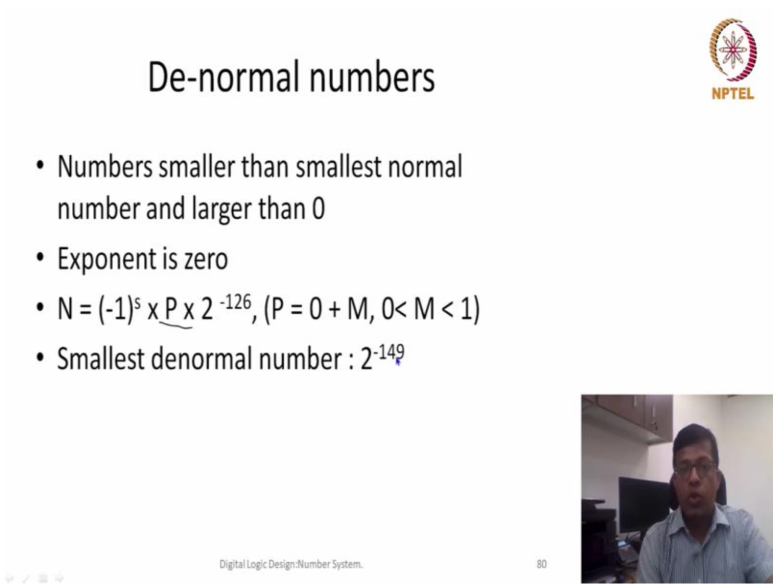
There has been an update in the way the number is displayed. Previous version would give you the represented value as a possibly rounded decimal number and the same number with the increased precision of a 64-bit double precision float. Now the original number is shown (either as the number that was entered, or as a possibly rounded decimal string) as well as the actual full precision decimal number that the float value is representing. Entering "0.1" is - as always - a nice example to see this behaviour. The difference between both values is shown as well, so you can easier tell the difference between what you entered and what you get in IEEE-754.

This webpage is a tool to understand IEEE-754 floating point numbers. This is the format in which almost all CPUs represent non-integer numbers. As this format is using base-2, there can be surprising differences in what numbers can be represented easily in decimal and which numbers can be represented in IEEE-754. As an example, try "0.1". The conversion is limited to 32-bit single precision numbers, while the IEEE-754-Standard contains formats with increased precision.

- **Usage:**

So, that also we can check from here. So, let us say the number is this if I want to say this, so, these are all lesser than 10 is to power minus 38, this 10 is to power minus 39. Similarly, if I can keep on going like this, this these all these numbers are lesser than 10 is to power minus 38. So, let us say only the last bit is 1 rest all of these bits are 0 the smallest number which I can represent is 10 is to power minus 45.

(Refer Slide Time: 26:54)



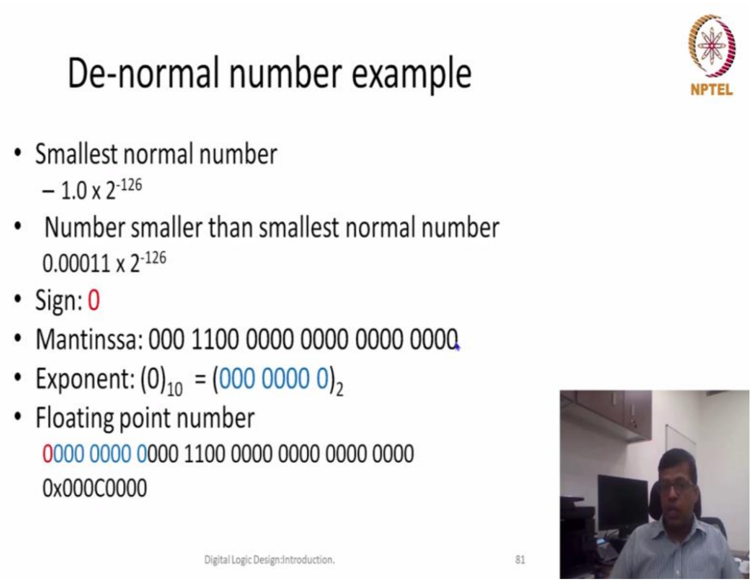
De-normal numbers

- Numbers smaller than smallest normal number and larger than 0
- Exponent is zero
- $N = (-1)^s \times \underline{P} \times 2^{-126}$, $(P = 0 + M, 0< M < 1)$
- Smallest denormal number : 2^{-149}

Digital Logic Design:Number System. 80

Or which is also we can say 2 is to power minus 149. So, this is the smallest number which we can represent even if we take into account denormal numbers.

(Refer Slide Time: 27:08)



De-normal number example

- Smallest normal number
  - 1.0 x 2^{-126}
- Number smaller than smallest normal number
  0.00011 x 2^{-126}
- Sign: 0
- Mantinssa: 000 1100 0000 0000 0000 0000
- Exponent: $(0)_{10}$ = $(000\ 0000\ 0)_2$
- Floating point number
  0000 0000 0000 1100 0000 0000 0000 0000
  0x000C0000

Digital Logic Design:Introduction. 81

So, this was about denormal numbers now, we can take some examples. So, the smallest or the smallest normal number is 1.0 into 2 is to power minus 126 and let us say because the number is smaller, I can represent it using 0.00011 into 2 is to power 126. You see there is no preceding 1 because the number is not normal and why there is no one because it cannot be represented within the limit of normal numbers in my single precision floating point number.

So, the sign is 0 mantissa is 00011 and after that all 0s exponent we clearly said the denormal numbers exponent is 0. So, I can represent this number again I can collect all the bits and the number can be represented using 0000 again 0 again 0 and then 1100. So, similarly, if I need to represent any other denormal numbers I can I can represent that number in my floating-point notations.

So, this explain us that what is the smallest thing which we can represent or how this, this exponent 0 can become a very special exponent and it can be used to express even smaller number, but you see there are two side effects, one the precision will reduce in all other normal numbers will have a precision of 23 bits, but here precision we keep on reducing. So, let us say I want to represent some number as small as 2 is to power minus 149.

Now, the precision is only single bit no more than that. So, that is also one of the side effects of representing the normal numbers plus I have to do a separate calculation a different kind of calculation the offset method is not working out here. So, with this, let us move on to the other topics or other questions. So, what would happen let us say I have a lot of denormal numbers or lot of numbers which are lesser then 10 is to power minus 38.

Or if I have more numbers, which are more than 10 is to power 38. I want to represent mass of universe or I want to do something which involves very, very large numbers. Or let us say we have a third scenario also that the precision of seven decimal digits is not sufficient. What should we do? So, in that case, what we would like to do is we would like to increase the number of precision bits. So, instead of 23 bits, there has to be more bits to represent precision plus exponent also should be represented with more than 8 bits so that larger number or more numbers can be included. A larger size can be included.