Digital System Design

Professor Neeraj Goel

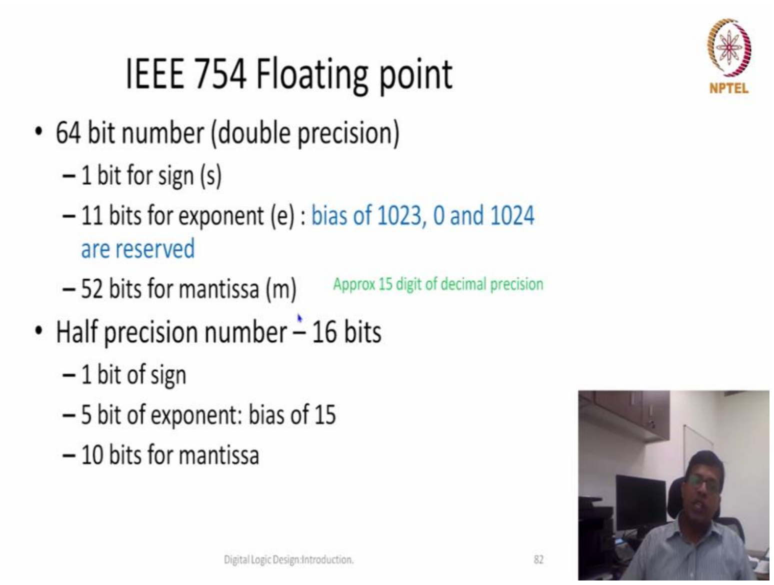
Department of Computer Science Engineering

Indian Institute of Technology, Ropar

Lecture 11

Floating Point Number - 4

(Refer Slide Time: 00:15)



IEEE 754 Floating point

- 64 bit number (double precision)
  - 1 bit for sign (s)
  - 11 bits for exponent (e) : bias of 1023, 0 and 1024 are reserved
  - 52 bits for mantissa (m) Approx 15 digit of decimal precision
- Half precision number – 16 bits
  - 1 bit of sign
  - 5 bit of exponent: bias of 15
  - 10 bits for mantissa

Digital Logic Design:Introduction. 82

So, representing these numbers again IEEE 754 also include another set of numbers which are called Double Precision numbers. When you write a C C++ program there are 2 kinds of floats 1 is called float another is called double. So, the double means it consumes 64 bits and it is actually a double precision number and whenever you write a float that represent 32 bit number, which is equivalent to a single precision floating point number.

Now, in case of double precision floating point number, number of bits we got a 64. So, again out of the 64 bits 1 bit has to be assigned to sign and then IEEE standard says that 11 bits will reserved for the exponent. So, that means and here we say 1023 is the bias and 0 and 1024 are reserved. So, the meaning of reserved bits are same as we have discussed in single precision floating point number 1024 would represent infinity and not a number and 0 would mean either 0 or the denormal numbers.

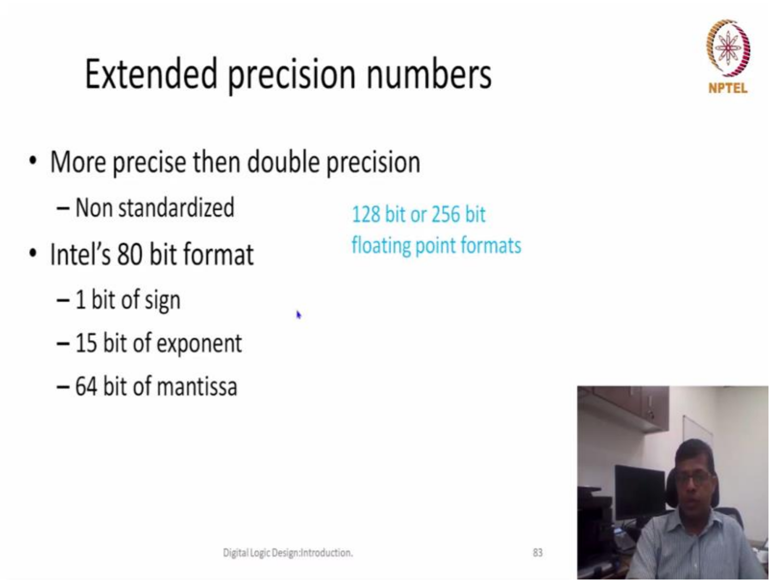
So, and rest of the bits 52 bits for mantissa now, because of that, what do we see we see the precision has increased from 7 digits to 15 digits. So, this, this gives us sufficient precision for most of the mathematical operations. So, similar to this requirement. So, you can keep this as an exercise for you that after this lecture you try to find out yourself that what is the largest double precision floating point number that you can represent and what is this smallest double precision floating point number that you can represent.

So, I will tell you it would be more than decimal for 100 So, tense for number, tense for 100 is a very, very large number it is the number which you can represent using floating point this double precision floating point number is, is more than the whole weight of the universe, the largest number we can think of. So, sometimes we also required to represent to be efficient in computation as well as efficient in space, space means how many bits are required to represent some number.

So, that is why sometimes we also do half precision floating point numbers. So how precision floating-point numbers would be represented in 16 bits. In 16 bits 1 bit would be given as a sign and 5 bits would be given to exponent and a bias of 15 and 10 bits for mantissa. So, rest of the rules are same as what we have discussed for single precision floating point numbers everything else every rule is same.

So, for example, here also 32, 31 and 0 exponent are reserved for, for representing infinity representing 0 respectively. So, what if I would like to have even more precision than double precision floating point numbers? So in that case is for, for initial some time, like the past, in the past history so there was initially like different vendors, they were trying to invent their own their, their own standardization.

(Refer Slide Time: 04:09)



Extended precision numbers

- More precise then double precision
  - Non standardized
- Intel's 80 bit format
  - 1 bit of sign
  - 15 bit of exponent
  - 64 bit of mantissa

128 bit or 256 bit floating point formats

Digital Logic Design:Introduction. 83

So, for example, somebody was designing a calculator, because the part of calculators are not going to be used somewhere else. So, they were defining their own precision. So, let us say your calculator was showing only 10 digits of decimal numbers. So, the precision can be decided accordingly. So similarly, let us say you are designing a computer which require arbitrarily very large precision. So, you could use more number of bits for precision and exponent.

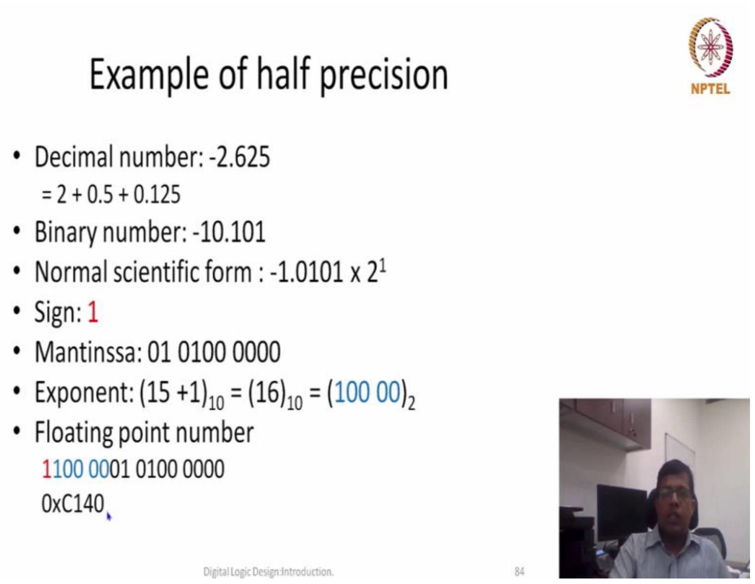
So, one of the one of the example is Intel's 80 bit format standardized, but because Intel is leading the industry for the last 40 years, so whenever you write in C Low double, it will automatically take this 80 bit format. In this 80-bit format, you have more exponent bits you have 15 exponent bits, you mantissa bits is also 64. So, that means precision has increased the number of the range of numbers which you can express is also increased.

So, based on our requirement, we can choose these formats. Then slowly IEEE has given more standardization or like have some people have started using more number of bits. So, for example, there are 128-bit representation of floating-point number or 256 bits of floating-point representation. So, all of these floating-point representations could be custom made for the custom requirement, if we see that we need to represent more, more or a larger number or a higher precise number, we can increase the number of mantissa bits or exponent bits accordingly.

So, they would be required for very specific applications. These applications could be space, it could be nuclear, it could be it could be any scientific application. So, the idea here is that, it gives us freedom to choose how many bits and because your hardware is going to be custom, nobody else is going to use your hardware firmware. So that is why we can these people can decide their own floating-point notations.

But if your hardware and software need to talk to each other, let us say we are writing a piece of software which we need to use in Intel machine as well as AMD machine or in some other machines, then the format has to be inter replaceable, or basically all of them should understand the same format. So, with this, we are almost closed. Let us take one example. So that we can understand that if some arbitrary precision is given or some particular precision is, given some format is given, can I convert that number to floating point number.

(Refer Slide Time: 07:12)



Example of half precision

- Decimal number: -2.625
  = 2 + 0.5 + 0.125
- Binary number: -10.101
- Normal scientific form : -1.0101×2^1
- Sign: 1
- Mantinssa: 01 0100 0000
- Exponent: $(15+1)_{10} = (16)_{10} = (100\ 00)_2$
- Floating point number
  1100 0001 0100 0000
  0xC140

Digital Logic Design:Introduction. 84

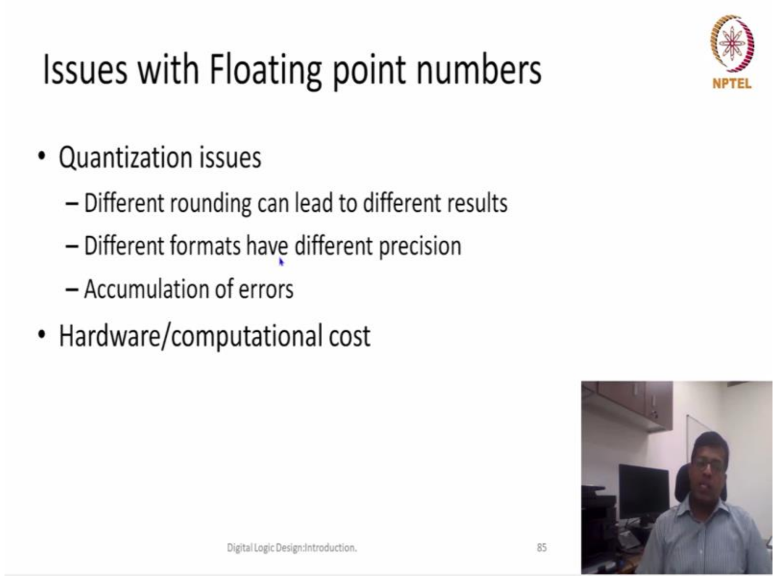
So, let us take it with a quick example. So, let us say we have taken this example in in single precision numbers also, let us do it with high precision. So, we can do it quickly because we have anyway done this in our previous lecture. So, let us say the decimal number is 2.625. And now I can represent that number using a 2 plus 0.5 plus 0.125. Now, we can say the binary number is minus 2 means 10, so binary number is 10 and this is 1, this is 01.

So, the normal scientific form would be minus 1.0101 into 2 is to power 1. So, if I want to represent it in a in a half precision method, so that means the mantissa would be 10 bits can which means 4 plus 4 plus 2, so I have divided it like 2 here and then 4 and then 4. So basically, I have started from the reverse, so that we can complete the number of bits and exponent is 5 bits, and the base offset was 15.

So exponent here was 1, so 1 plus 15 means 16 would be there, and then 16 need to be converted into binary, which is a 5-bit number that is 10000 then we can combine all these sign mantissa and exponent bits and then convert our floating point number. So, that means 1 here and then 10000 and then the mantissa weights 010100. And so again, I would like to emphasize that whenever we are writing number in this, this way, if we write in a group of 4, then converting into hex would be easier for us.

So, this 1100 will become C 0001 is 1 and 0100 is 4 and these 4 0s will be 0. So, this way, we can if we are given any number, we can understand that how to convert that number into a into a floating-point number of half precision, single precision, double precision or whatever is a new thing.

(Refer Slide Time: 09:36)



Issues with Floating point numbers

- Quantization issues
  - Different rounding can lead to different results
  - Different formats have different precision
  - Accumulation of errors
- Hardware/computational cost

Digital Logic Design:Introduction. 85

So, with this, we are we know understand all the various aspects. So, let us also see that when do we use floating point number which number to be used whether to use single precision, double precision or extended double precision and when not to use. So, one thing which has been observed. So, you see that whenever we are representing floating point numbers, let us say if I want to do addition in these floating-point numbers, how the addition would have would have to be done how do we do addition in our scientific form.

So, first we have to keep the exponent as same and then we add the decimal part, fractional, fractional part would be added. So, let us say the 2 numbers will have different exponent first you will set the exponent as same and then you will shift 1 of the number so that the exponent is same, and then you will add the fractional part and the non fractional part. So, if all of those things have to be done in binary, it would require multiple different steps.

You have to further after doing all the computation then again you have to normalize and then represent it in binary. So, all of these steps required so many different all this addition, subtraction multiplication, they require so many steps that the amount of hardware that is acquired or number of cycles, number of amount of time which is acquired, both of them are huge.

Huge means it has to do with respect to something. So, huge means with respect to integer multiplication or integer addition, integer subtraction. So, that is why if, if my application has to be performance oriented, I want to make my application run as fast as possible, I would like to use as many integer calculations as possible, and I would like to minimize the floating point calculations.

So, sometimes we even try to reduce the floating-point multiplication by doing fixed point multiplications. Though precision will reduce, but computation can be done in a fairly fast method. The other thing, which we have to understand with these floating-point numbers that whatever precision we keep, they are not infinite precision. Even in case of double precision, floating point number, the amount of digits are 15.

And we have lot of irrational numbers, where whatever number of digits you have, is, is less. So, for example, you know, pi. So, all of these numbers will have so many digits, and you would say, why does it matter? A digit, which is less significant than the 10 is power minus 15, or 2’s power minus 1024 why it should be significant. So, it becomes significant when it accumulates at many places, we keep on doing competition, and let us say the competition is multiplied with some large number.

The error would be the approximate error would be equal to the multiplication factor. So, this this accumulation, because of multiplication, because of addition, subtraction, every operation will introduce its own error. And those error will get magnified when, when will have this multiplication operations or division operations. So, that is why many times we prefer to use double precision numbers, if that error should be very less, whenever we want to have this quantization error, this is called quantization error, because we are trimming we are saying that we cannot represent smaller than this number.

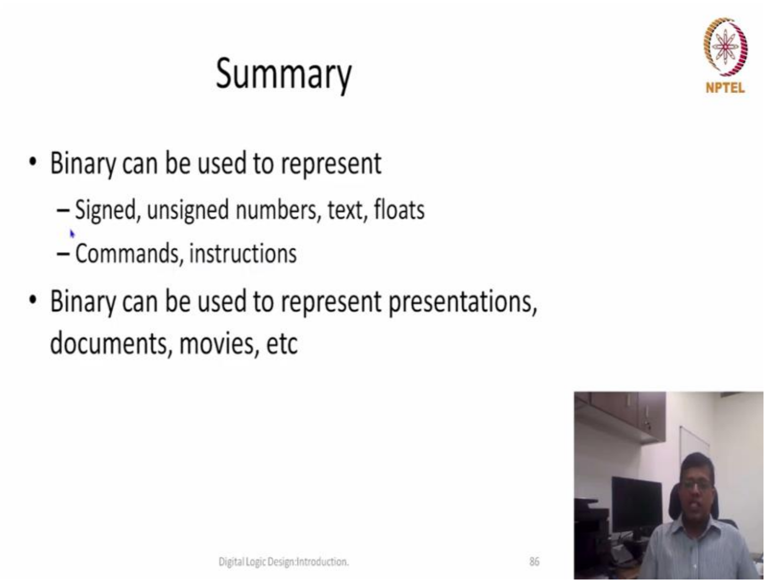
So, whenever we have these quantization numbers or quantization thing then and accumulation is happening then, then he should have single precision we go for double precision numbers so that the amount of error can be reduced. So, and that is how we choose that which, which position we are to choose. So, different if the number is of a higher precision, then it will be a using that we can use that for lesser error.

Now, sometimes we also do these operations that we convert floating point number 2, you also sometimes typecast, it is a double precision number 2 single precision or single precision to double precision. Let us say you are converting a single precision number to a double precision numbers single precision has only 23 bits but are 23 bits of precision but double precision has 52 bits. Now, these 52 bits so that means rest of the bits have to make a 0.

So, if reverse is the case, we do not know what rounding technique we have to use. There are various rounding techniques. So, based on these rounding techniques, sometimes you will keep the MSB as 0 least significant bit as 0 sometimes will keep least significant bit as 1. So, these rounding will also create different results for different scenarios for different conversions. So, all of these things should be reproducible, reproducible means that every competition should give certain definite results.

So, hardware will try to make sure that whatever method is being followed for rounding for precision for accumulation of error that that should be standardized. So, that should be fixed or so that it could be repeatable. So, with this we close this particular topic of floating-point numbers and essentially, we can summarize this whole topic as the numbers depending on our representation, we can have various formats and depending on the requirement, we can have different formats as, as per our requirement. This is also conclusion of our module 1 which means Binary representation. So, as a summary of this module, what do we summarize,

(Refer Slide Time: 16:34)



Summary

- Binary can be used to represent
  - Signed, unsigned numbers, text, floats
  - Commands, instructions
- Binary can be used to represent presentations, documents, movies, etc

Digital Logic Design:Introduction. 86

We can say that a binary number although it is only collections of 0s and 1s, but the same collection of 0s and 1s can be used to represent a sign number or an unsigned number and an unsigned number in a sign number, there could be various possibilities I can represent it using 2's complement 1’s complement or offset method or same magnitude method. So, all these combinations are there and the same 0s and 1s can be used to represent some symbols, some text.

And same 0s 1s can also be used to represent floating point numbers as we have discussed in this, this, this lecture. So, that means, it is not only important to know what is 0 and 1 we should also know that this collection of 0s and 1s is representing what. So, some additional information should be provided it is mostly context dependent in based on the context, you know that this number.

So, for example, whenever you are writing your C C++ programs, so you say specifically, that this is the type of this number, whether this number is an int, whether this number is a character, and somebody in the hardware is taking care of that, if you are representing that number as an integer, it should always be considered as integer unless you type cast it, when you type cast it, then you are asking hardware to not consider this number as a integer.

But let us say consider it as a float, or corrector, or the area of connectors. The interesting thing is these 0s and 1s are not only representing data in various homes, but it also represent applications, instructions, commands, or signals. So, based on these signals, various operations can be performed. And that is what we see during this this whole course, that that 0s and 1s are is a powerful mechanism to do various operations to represent data as well as to represent different control signals, different commands or different instructions. There is at one level of abstraction. At higher level of abstraction, we can also see that these 0s and 1s can be used to represent applications.

So, if you see Microsoft Word, what is it, it is an application, replayed in a binary form 01 form. So, if you see Microsoft Word document, that word document is also a collection of 0s and 1s. And in the Microsoft PowerPoint, let us say in the PowerPoint, I am writing this, this whole array of characters, and some symbols, all of those things are further encapsulated and written within Microsoft PowerPoint.

And you all have you know that all these data so these applications, which means you can say documents, movies, presentations, songs, all of these are binary representation and the application which are understanding which are running them are also binary representation your, your mp3 movie player or mp3 player or a movie player and M player all of these are also some binary application to the extent that your operating system is also a binary representation which, which loads all the computer and works on it.

So, in summary, these 0s and 1s can be used to represent anything and many other times you would try to define your own custom understanding custom representation of these 0s and 1s and because we are working at the lowest level of abstraction in this course, so, we will see that these 0s and 1s are represented either using integers signed unsigned text, ASCII characters floats or Command signals instructions. With this, I would like to close. Thank you very much. We will start with a new module next in the next class. Thank you very much.