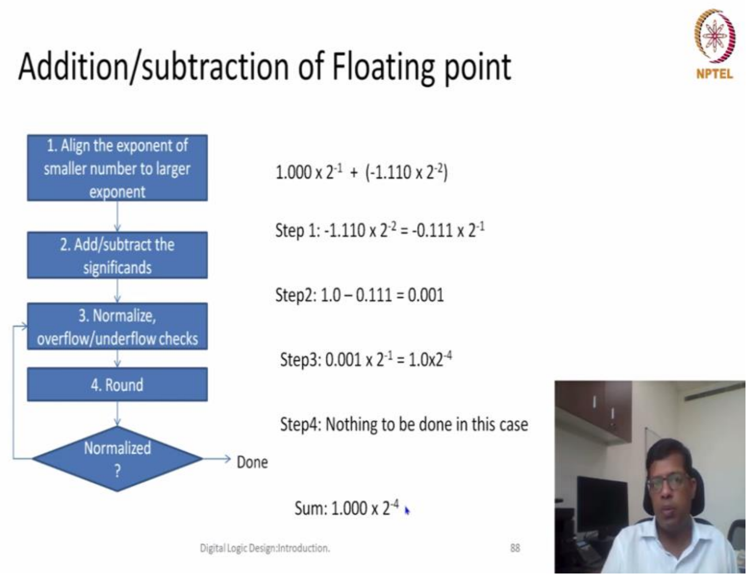
Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture 12
Floating point Number - 5

Hello everyone, this is a supplementary Lecture 2, Module 1 there we will discuss some additional arithmetic of floating-point Numbers. So, in floating point Numbers we have seen how to represent any decimal number into a floating-point number, but we are still curious that how to do multiplication, how to do addition, subtraction in floating point numbers. So, in the supplementary lecture, we will discuss all these things.

Please be clear that this supplementary lecture will is an optional lecture, it is not part of a regular syllabus, the whatever we are teaching here, it would not be given in the exams or quizzes, but it is still worthy to go through this lecture once one it will clear your curiosity, second thing you will get to know at least some aspects that how addition, subtraction is done some time it may be useful in future if required.

(Refer Slide Time: 01:30)



Addition/subtraction of Floating point

1. Align the exponent of smaller number to larger exponent
2. Add/subtract the significands
3. Normalize, overflow/underflow checks
4. Round
Normalized ?
Done

$1.000 \times 2^{-1} + (-1.110 \times 2^{-2})$

Step 1: $-1.110 \times 2^{-2} = -0.111 \times 2^{-1}$

Step2: $1.0 - 0.111 = 0.001$

Step3: $0.001 \times 2^{-1} = 1.0 \times 2^{-4}$

Step4: Nothing to be done in this case

Sum: 1.000×2^{-4}

Digital Logic Design:Introduction. 88

So, let us first see how addition or subtraction would be done in a floating-point number, it would be done in the similar way the way we do addition or subtraction in decimal numbers which are representing as represented in scientific notations. So, first step we should do or what a first step what we do is first we align the exponent to the larger exponent. So, yeah before even first step, the 0th step is because our, our floating point number is represented in a in a 32 bit compact format where some bit is signed some bit is mantissa and some bit is exponent.

So, first of all we extract all of these bits, we extract sign bit, we extract mantissa bits and we also extract exponent bit and the mantissa is also converted into significant. So, after that, you see when we are doing addition, we know that the addition whatever is the larger number, we would be adding the smaller number to larger numbers. So, this the exponent of the smaller number would be aligned or adjusted to the exponent of the larger number. So, how it is done, it is done by shifting the significant or the smaller number in a right direction in a right direction means in right, shifting it right by the difference of 2 exponents.

So, after this alignment is done, now, we need not to worry about the exponent part we only need to add we only need to add the significance. So, if we are performing addition and the sign of both these numbers sign of both these numbers are same, then we have to perform addition. But if the operation is addition, but the sign of these 2 numbers are opposite, let us say one of the number is positive another number is negative, then we have to perform subtraction.

Now, again remember that these, these both of these 2 numbers are represented in signing magnitude form or there is only magnitude which is presented sign is, is present as a separate bit. So, we have to create if we want to use 2's complement arithmetic then we have to create 2's complement notion for both of these numbers and then perform these addition and subtraction. Third thing after doing addition or subtraction then again, we are to normalize, normalize means we are to represent the number in the normal scientific notation or a standard scientific notation.

In standard scientific notation, one, so there has to be only 1 non-zero number that means 1 before decimal point and after that any number of 0s or 1s can be there. So, if there is something required to be done for rounding, that will be done in step 4. So, in Step 3, along with doing normalization so, if there is there is any other correction required. So, for example, because in after normalization it may be the case that the number cannot be represented using the range of floating-point numbers represented using that let us say single precision format or double precision format.

So, we also do overflow and under per check. So, if the number is larger, then whatever is the limit, then we also give a error and we terminate the processor itself or if there is an overflow then also we give the error and return it but if there is no overflow, there is no underflow then we go for the 4th step which is rounding, rounding means the number of bits after let us say number of bits in the mantissa is more than the number of bits which are there in that format, then we do a round over here to remove the extra bits.

Because after rounding we also sometimes add plus 1. So, because of that rounding, there may be again a normalization required. So, if the number is not normalized, we go back and go to step number 3 to normalize. So, to understand this, let us take one small quick example. So, let us say these are the 2 numbers, I am directly representing in the in the scientific format. So, that means I have already extracted my mantissa bits exponent bit as well as sign bits.

So, these 2 numbers are 1.000. So, this is in the format, which is like significant is already added. So, my step 0 is already done, where I have added significant, I have extracted mantissa exponent and this exponent also you see is the processed format. So that means let us say it was a single precision number then 127 is already subtracted from the whatever exponent I get from that 32-bit number.

So, this is the scientific notation or the normal exponent notation of these 2 numbers. Now, you see, one of the number is 2 is to power minus 1 another number is 2 is to power minus 2 and this number is positive number and this number is a negative number. So, if I want to do addition, so, effectively, I need to subtract, but in the first step I first I need to align.

So, in align are which number is the larger number? The larger number is this one, which is 2 is to power minus 1, 2 is to power minus 2 is always lesser than 2 is to power minus 1. So, I need to shift by one bit this, this number I need to shift by one bit. So, in step one, I have to make the power of this number is 2 is to power minus 1 so that means I need to multiply. So, in in a very general case, in a very general case, if the difference of the larger remember the difference of exponent bits of the larger number and the smaller number is n, then I will shift the significant n times or n bits in the right direction.

So, the next step is after now, both of these 2 numbers are aligned. So, that means exponent of both of these 2 numbers 2 is to power minus 1, 2 is to power minus 1 now, I can do I can perform the addition or subtraction. So, because this, this number was minus this was positive.

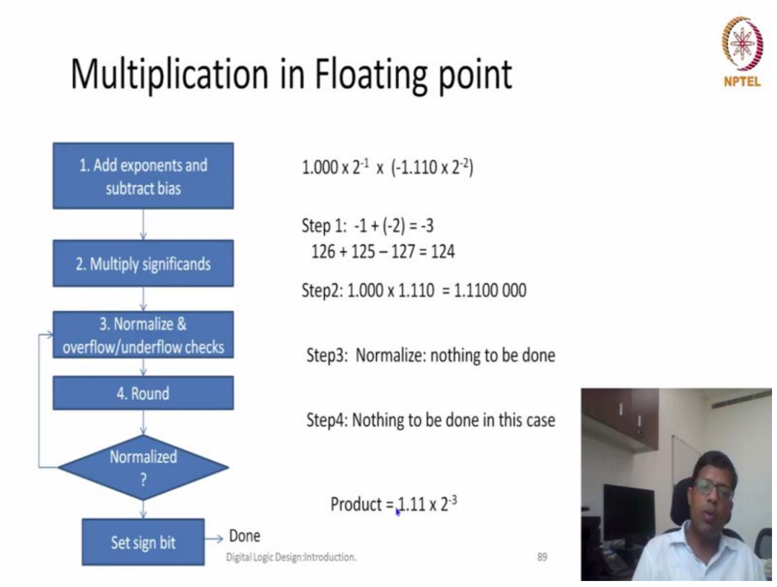
So, the number the, the in step 2 the, the I need to perform this 1.0 minus 0.111. So, this arithmetic I need to perform either using 2's complement or using the direct subtraction.

So, here I am directly writing the results, if we have to perform this using 2's complement, then we have to take 2's complement of these 2 numbers. So, basically, we also need to add 3 0 here. So, the decimal point will match and then we will take 2's complement, but now I am directly writing the result. So, here we also know intuitively that 1.0 minus 0.001 will be 0.001. So, in the third step, we will check whether it is underflow or overflow yes of course, it is not underflow neither overflow.

So, we will go ahead and normalize. In normalization this number needs to be converted into a normal or scientific notation standard scientific notation. So, that means, this has to be shifted in this case this has to be shifted left and whatever number of bits we are shifting that we will add here. So, because we have shifting it by 3. So, we will subtract 3 bits here and it will become 2 is to power minus 4.

And then rounding so, because if we are representing this number as a single precision number where number of mantissa bits are 23. So, we need not to do anything for rounding and this, this can be written as the result 1.0 to 2 is to power minus 4. So, this is how addition or subtraction need to be done.

(Refer Slide Time: 10:39)



Multiplication in Floating point

1. Add exponents and subtract bias
2. Multiply significands
3. Normalize & overflow/underflow checks
4. Round
Normalized ?
Set sign bit
Done
NPTEL

$1.000 \times 2^{-1} \times (-1.110 \times 2^{-2})$

Step 1: $-1 + (-2) = -3$
$126 + 125 - 127 = 124$

Step2: $1.000 \times 1.110 = 1.1100\ 000$

Step3: Normalize: nothing to be done

Step4: Nothing to be done in this case

Product = 1.11×2^{-3}

Digital Logic Design:Introduction. 89

Now, let us see how multiplication needs to be done multiplication will also do in will also do multiplication in a very similar way, the way we do it in our decimal numbers represented in exponent forms. So, try to remember how do we do multiplications in a decimal form. So, first we will try to multiply or we will add the exponents and we will multiply the significance that is the second part and a third will again normalize and 4th we will round.

So, the steps are very similar to whatever we did for addition, and after rounding again if it is not normalized, then we will go ahead and normalize and then we will again round so, and so forth. So, until it is normalized here, one additional step would be that the sign bit would be set after all of these things by seeing if the multiplication of 2 numbers were 2 positive numbers or multiplied or 2 negative numbers were multiplied, then the sign bit would be 1.

So, sign would be 0 that number one a positive if one of the number was saying positive another number was negative, then the multiplication would result in a negative answer. So, sign which would be one. So, sign bit is treated separately in whole process other ways, all of these numbers are treated as magnitude. So, let us try to understand this with an example. Again, I will take the same example.

So 1.000 into 2 is to power minus 1 is multiplied with minus 1.110 into 2 is to power minus 2. So, first step is we are adding the exponent. Now, here, I can add the exponent minus one plus minus 2 is the exponent here, so total exponent is minus 3. Now, again, remember that in our in our standard or IEEE 754 single precision or double precision representation, exponent are represented using biased numbers.

So, if we are using those bias number, we need not to subtract the bias in all the cases, what we can do is we can simply add those 2 numbers and subtract the bias. So, let us say, in the bias case, this would have a bias of this, this, this would be represented using 127 126 because bias was 127 minus 1 is 126. And this would have bias of 120 this would have an exponent of 125. So, we will simply add the 2 exponents and then whatever is the result will subtract bias that means minus 127 whatever is the result that we can keep as an exponent. So, this subtraction of bias, the subtraction of bias need not to be performed when we were doing addition and subtraction because their exponent was a relative term.

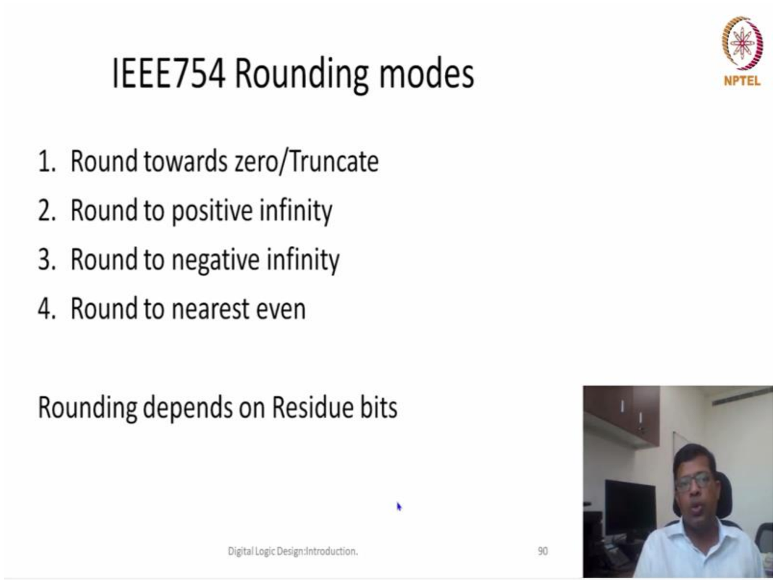
Higher is the exponent that means larger is the number and so when we were doing addition and subtraction, we can simply see the relative difference between 2 exponents, but here because we need to add to exponent that is way one bias is subtracted. Then after that multiplication part, so first we have created these significant 1.000 into 1.110. So then, these 2 would be multiplied. Now, I have directly written the multiplication result, but the way this can be multiplied is so we can remove this decimal point and we can treat both of them as 4 digit number.

So, we can treat both of them as, as a 4-digit number and we will simply multiply and after multiplication the decimal points need to be added after 6, 6 binary digits. So, after 6 bits we have added I think, yeah. So, this need to be added after, after 6 bits. I think this, this 1 0 is redundant here. But yeah, so this has to be added after, after 6, 6 bits because of the standard rule of multiplication. Now after this step, then we go for normalization.

So, because this number was already represented in normalized form, so nothing to be done for normalization. Otherwise during normalization, sometimes we have to shift bits left or right, and there is no overflow or underflow here, so that is why we are not doing anything, but there is a good likelihood that overflow and underflow can occur in multiplication, because multiplication would require this addition of this exponent and sometimes these exponents jump out of the range and then these numbers cannot be represented in the multiplication itself in the range itself. So, we have to say that there is a workflow and we exit from this step itself.

So, after doing normalization then we do rounding here are rounding also need not to be performed because the number of significant bits or number of mantissa bits are still less than the mantissa bits which are required or which are possible in 6 single precision numbers. And here the sign is minus 1 because this number is negative this number is positive. So, we can represent the result. So, this is a negative result, this is also a small mistake here. So, this is how multiplication and addition is done. Now, let us see if rounding need to be performed and how do we do rounding?

(Refer Slide Time: 17:12)



IEEE754 Rounding modes

1. Round towards zero/Truncate
2. Round to positive infinity
3. Round to negative infinity
4. Round to nearest even

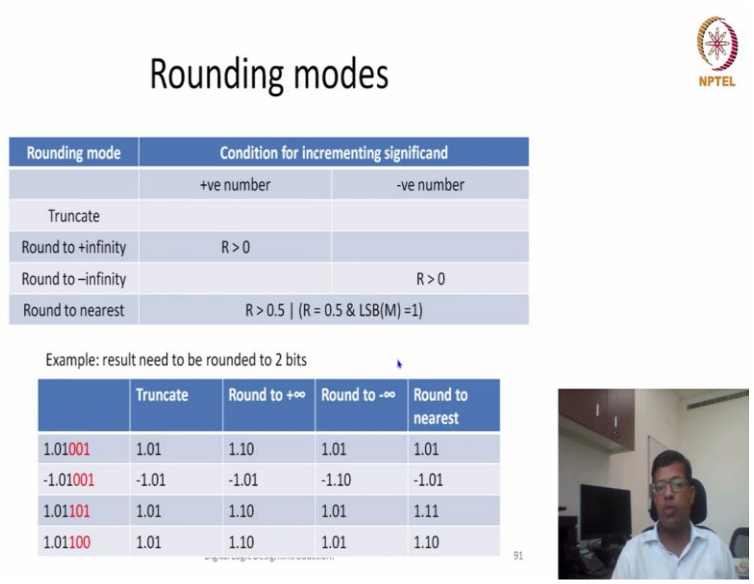
Rounding depends on Residue bits

Digital Logic Design:Introduction. 90

So, IEEE 754 standard it defines that there are 4 possibility of rounding. So, one rounding is round towards 0 this is also called truncate. So truncate means that whatever number of bits after significant bits are there, we simply drop them. The second is Round to work towards positive infinity, the third is Round towards negative infinity and 4th is Round towards nearest even. So, in all the rounding cases so we can divide our mantissa or divide our fractional part into 2 parts.

One is what is significant or what is the what need to be added in our representation, the rest of the bits could be represented as a residue which bits, so, residue bits based on the residue bit, sometimes we need to add plus 1 to the least significant bit, or plus 1, 2, the least significant part of this mantissa. So, let us quickly see that in case of in case of, we will see that how and when we add this plus 1.

(Refer Slide Time: 18:37)



Rounding modes

Rounding mode	Condition for incrementing significand	
	+ve number	-ve number
Truncate		
Round to +infinity	R > 0	
Round to –infinity		R > 0
Round to nearest	R > 0.5 | (R = 0.5 & LSB(M) =1)	

Example: result need to be rounded to 2 bits

	Truncate	Round to +∞	Round to -∞	Round to nearest
1.01001	1.01	1.10	1.01	1.01
-1.01001	-1.01	-1.01	-1.10	-1.01
1.01101	1.01	1.10	1.01	1.11
1.01100	1.01	1.10	1.01	1.10

So, you can classify this into if the number is positive or a number is negative, in case of truncation that means rounding towards 0, so, the original number. So, the original number, whatever is the residue bits, because we are rounding towards 0. So, the number we are generating after rounding should not be bigger or should not be like it. So, whatever it can simply we called like we, we want to go towards 0.

So, that means if the number is positive then also we leave the residue part if it is negative then also we leave the residue part, while in whenever round to towards positive infinity. So that is another case in round towards positive infinity if the number is positive, then we will add the we will increment the significant. So, which essentially means that whenever we want to round towards whenever this mode is there round towards infinity.

That means the number has to be rounded towards infinity means if the number is negative then whatever is the, whatever is the residue part we can leave, because that number will be closer to positive infinity. But in case of the number is positive, then we will increment this significant by plus 1, so that the number is now closer towards infinity. Now, the third mode is around towards negative infinity in this round towards negative infinity, if the number is positive, and then we compare these 2 things either the number if we if we leave, if we truncate, truncate the residue part then it is closer to minus infinity.

So, with that, that means we do not have 1, but if it is a negative number, if it is a negative number, then we add a 1 to the whatever is the residue if residue is more than 0, then we add plus 1 to the significant so that now it is closer to minus infinity. And round to nearest is the most popular one, which is being used. So, this is a very standard method, which we have using our decimal points also in decimal point, the way we do is that either residue is more than 0.5, then we add plus 1.

So, how do we know here that a residue is, is more than 0.5. So that means in my residue bits in binary residue bits, first bit is 1, that means first bit is 1 and any other bit other than first bit of residue is one, that means the number is more than 0.5. So, your number is more than 0.5, we will always add 1 to the significant, but what if number is exactly 0.5, then we will see that we will, if the number is if my significant is odd, then I will add 1 otherwise, I would not add.

So, how to see that if significant is, is odd or even significant means lower MSB bit so low lower significantly, significant bit on my mantissa if it is one, then I will add 1otherwise I will not. So, let us try to understand this with an example. So, let us say that the result needs to be rounded to 2 bits. So, let us say these are the input numbers. Now, I am marking the residue bits in red. So, if it is truncate then I will always remove the residue bits will not do any other.

So, whatever is the number given other than these bits that would be given as the answer. In case of positive infinity. Now 1.01 and then 0.01 would be added. So that means in the least significant place I will add 1 because I have added 1 it will become 1.10. And so here round towards positive infinity, I am not caring what is the size of my, what is the size of my residue, even if it is very small, then I am adding one.

So, it is also very similar to sealing function on my C and truncate could also be seen as the floor function of my in C language round towards minus infinity in minus infinity, because we can see from this table or we can simply see that if I want to run towards minus infinity, because I want this number, the final number to be closer to minus infinity. So, that is why I will leave this residual and not do anything.

In case of around to nearest because this residue is lesser than 0.5. So, it is discarded nothing is added to my C bit. So, in this case, in the negative this number minus 1.01001 rest of the results are same but round towards plus infinity. I am not doing anything because the number is negative but around towards minus infinity, I am adding 0.01 so that the number final number is 1.10. So, consider this case 1.01101.

So, truncate we are leaving these bits and round towards infinity because there is a residue which is more than 0 I am adding 0.01. Round towards minus infinity because the number is positive I am discarding it in case of round to nearest the 101 means the MSB of residue is 1 so that means the number is 0.5. And if any other bit in the residue is 1, so, that means it is more than 0.5. So, I have to add a 1. So, this should be actually 1.10 so, they should be 1.10.

So, the last case if my residues exactly 0.5 or 100 or all the 0s after this 1. So, this is exactly 0.5 in truncate I simply remove this 0.5 in case of plus infinity I will add 0.01 in case of minus infinity I will leave it and in case of nearest even I will see the last bit of my significant is one So, that means it was odd. So, I will add 0.01 to make it even. So, the number would be 1.10. So, one thing while we will add this while we will make this number as evens with there is nothing technical about it it is most more like a historical way of doing things and that is why we add 0.01.

So, this is how Rounding is done and rounding would be done because of multiple reasons. So, whenever number of significant bits of example, when whenever we are converting from double precision floating point number to single precision floating point number we may have to do rounding, as a result of multiplication sometimes we need to do rounding. So, all these rounding's are become important because even the small significant bits are important whenever we are doing multiplication or division.

So, that is why what mode is there it is important. The second thing is how to decide this mode. So, mostly it is the hardware specific feature whenever it is implemented in hardware, hardware, we either it is configurable we need to say that which particular rounding mode is being used. Most in a default or in a typical calculation round to nearest is the most popular mode which is used in the floating-point calculations. So, with this I would like to close. Thank you very much.