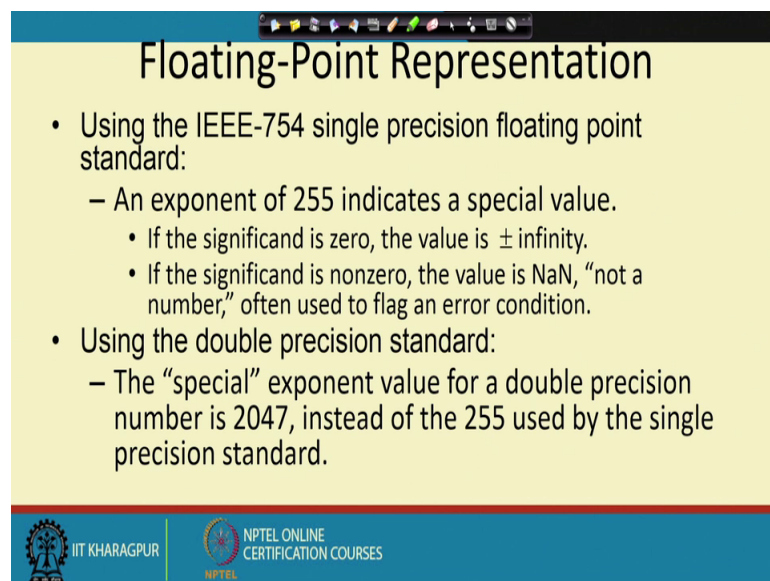


**Digital Circuits**  
**Prof. Santanu Chattopadhyay**  
**Department of Electronics and Electrical Communication Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 07**  
**Number System (Contd.)**



So, in the floating point representation there is one more point regarding these numbers that we are going to store.

(Refer Slide Time: 00:20)



**Floating-Point Representation**

- Using the IEEE-754 single precision floating point standard:
  - An exponent of 255 indicates a special value.
    - If the significand is zero, the value is  $\pm$  infinity.
    - If the significand is nonzero, the value is NaN, “not a number,” often used to flag an error condition.
- Using the double precision standard:
  - The “special” exponent value for a double precision number is 2047, instead of the 255 used by the single precision standard.

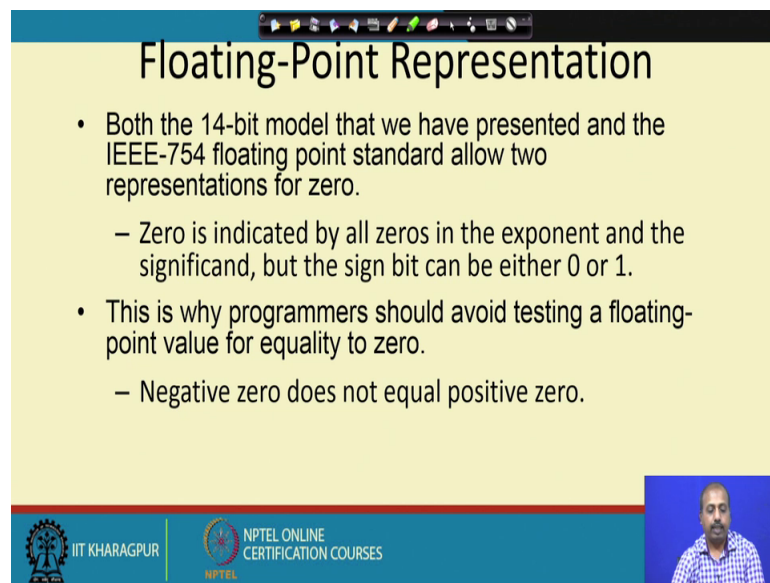
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

Like in IEEE 754 single precision floating point standard. So, an exponent 255 it indicates a special value. So, if the significand part is 0. So, this represent a plus infinity or minus infinity depending upon the sign b ok. So, this has got a special representation of infinity.

On the other hand if the significand part is nonzero, then it means it is a not a number. So, many a time you might have seen while writing programs using floating point numbers. So, it prints the message that not a number in a in like that. So, that actually comes from here. So, if the number that you are getting is of this format that this the exponent part is 255, and the significand part is nonzero then it will be telling that it is a it is not a number. And if it is zero, if the significand part is zero then it is plus infinity or minus infinity depending upon the sign.

In the double precision format also the special exponent value for double precision number is 2047. So, instead of 255, it is 2047 otherwise it is same. So, if you are getting exponent 2047 then this has got a special meaning. So, this maybe if the significand is 0 then it is plus it is plus minus infinity depending upon the sign bit and if it is if the significand is nonzero, then it is not a number. So, this way this IEEE the standard so, it has kept special provision for representing infinity and this not a number type of flags.

(Refer Slide Time: 02:00)



### Floating-Point Representation

- Both the 14-bit model that we have presented and the IEEE-754 floating point standard allow two representations for zero.
  - Zero is indicated by all zeros in the exponent and the significand, but the sign bit can be either 0 or 1.
- This is why programmers should avoid testing a floating-point value for equality to zero.
  - Negative zero does not equal positive zero.

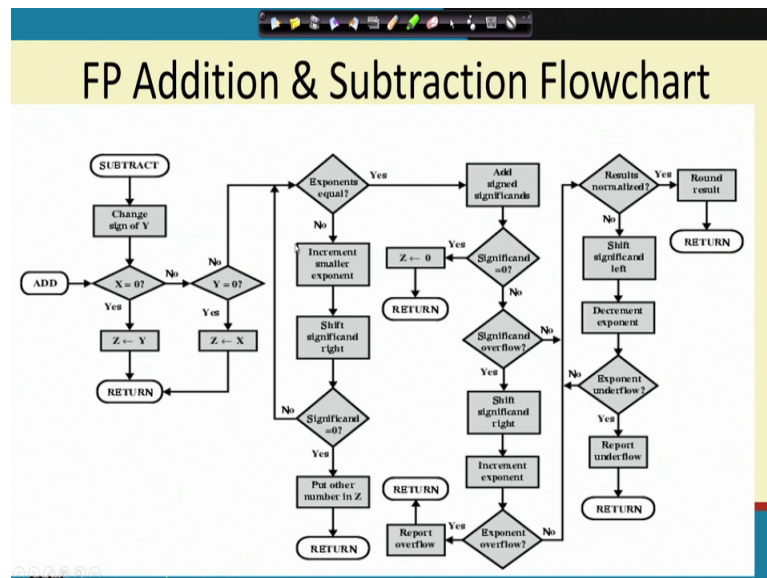
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, another caution that we have discussed in the last class that, both in our model as well as IEEE. So, there is plus 0 and minus 0 that representation is possible. So, we should be careful while writing programs and trying to check between whether a variable b has become 0 or not because this plus 0 and minus 0 they are not same.

So, you can get some surprising results because of that either the check may be successful or the check may be fail, maybe a failure though in both the cases the value is 0 only ok. But the  $x$  equal to 0 in some cases that check may be very check may come out as pass some cases, it may come out as a fail. So, we have to be careful.

So, maybe we will do some other check ok. So, it will normally what we do is it is less than some very very small quantity. So, maybe I can I will write like  $x$  less than point 10 zeros and then 1. So, that way it is a very small number. So, it is it may be considered to be equal to 0; the absolute value of  $x$  of course.

(Refer Slide Time: 03:14)



Next we look into how do we do this addition and subtraction in the floating point format. So, floating point addition and subtraction

(Refer Slide Time: 03:19)

The slide, titled "Floating-Point Representation", contains three bullet points:

- Floating-point addition and subtraction are done using methods analogous to how we perform calculations using pencil and paper.
- The first thing that we do is express both operands in the same exponential power, then add the numbers, preserving the exponent in the sum.
- If the exponent requires adjustment, we do so at the end of the calculation.

The slide footer includes the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a man in a blue shirt.

So, we use methods which are analogous to our paper pencil method. So, first of all the operands must have same exponent otherwise you cannot do the addition subtraction. So, the exponent has to be preserved and the after doing the addition and subtraction. So, we may we may need to normalize and that way the exponent may be changing after the

addition or subtraction. So, if the exponent requires adjustment so, we do so, at the end of the calculation. So, this is the process.

Suppose I am do trying to do an addition of X plus Y. So, if X equal to 0 then the answer is Y directly so, we go back. If the X is not equal to 0 then we check whether Y equal to 0 or not. If Y is also not equal to 0 then we, if Y is equal to 0 in that case the answer is Z. So, it simply goes out otherwise X and Y both are nonzero. So, it comes to this point and it checks whether the exponents are equal or not.

So, if the exponents are equal, I will do not need to do any exponent adjustments. So, I just simply add the add sign significands and if the significand becomes 0, then the answer is 0. If the significand is not is nonzero then there may be a possibility that significand there is an overflow or there is no over flow in the significand with contain containing the number of bits.

If there is no overflow then we have to check whether the result has been normalized or not that is after decimal point, the first digit should be 1 and all before decimal point there should not be any digit. So, all those rules so, it is checking those rules and accordingly if the results are not normalized, then we have to shift the significand towards left, decrement the exponent part and that way this if there is it resulting exponent underflow. So, if it is not so, we and it is result is not yet normalized. So, we may just shift by a number of bits the significant part till the exponent and go on decrementing the exponent.

So, till the exponent becomes this significand is correct as per our rule. So, ultimately when this everything is correct, then there we have far we have got the result then we go then we round up round our do a rounding operation of the result and then it returns. And otherwise if there is an if there is an overflow in the significant part. So, you have to shift the significant right increment the exponent part and if there is no exponent overflow, then we will go for result normalization and go back.

And if this exponent part also overflows; that means, after doing this addition both significant and exponent they have over-flown. So, we cannot represent this addition the added addition result into the given format. So, it will report overflow and it will return. So, there is an overflow in the addition process.

Similarly, for the subtraction part so, we just change the sign of Y, and then rest of the thing is same as we are doing for the addition part. So, this is simply pen and paper type of addition subtraction. So, that is followed here also.

(Refer Slide Time: 06:39)

**Floating-Point addition example**

- Example:
  - Find the sum of  $12_{10}$  and  $1.25_{10}$  using the 14-bit “simple” floating-point model.
  - We find  $12_{10} = 0.1100 \times 2^4$ . And  $1.25_{10} = 0.101 \times 2^1 = 0.000101 \times 2^4$ .
  - Thus, our sum is  $0.110101 \times 2^4$ .

+	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0



So, suppose we are trying to do this so, 12 to the base 10 and 1.25 to the base 10. So, this we are adding using 14 bit simple floating point model that we have. So, 12 is represented as in our normalized model say it is 0.1100 into 2 power 4 and 1.25 is 2.1 there is 0.101 into 2 power 1.

So, I have to make both of their exponents same so, we take both of them 2 exponent 4 larger one definitely. So, after that so, when this is converted into 2 power 4. So, this becomes the representation knows now we add this significant with this significant. So, getting the significant part as this so, our overall sum becomes 0.110101 into 2 power 4. So, here no adjustment is necessary because before decimal we have 0 and after decimal we have one. So, no adjustment is necessary.

(Refer Slide Time: 07:35)

## Floating-Point Multiplication

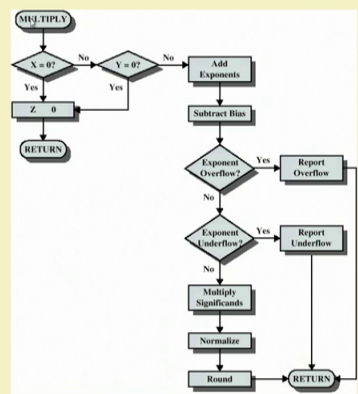
- Floating-point multiplication is also carried out in a manner akin to how we perform multiplication using pencil and paper.
- We multiply the two operands and add their exponents.
- If the exponent requires adjustment, we do so at the end of the calculation.





So, in case of multiplication so, we do similar thing so, by this is also like say paper and pen a paper and pencil type of multiplication that we do. So, we multiply the operands and add the exponents. So, significant parts are multiplied and the exponent parts are added. So, if after that we you or you need to do some adjustment with the exponent. So, we do that adjustment after the calculation.

(Refer Slide Time: 08:00)

## Floating Point Multiplication flowchart



```
graph TD
    Start([MULTIPLY]) --> X0{X = 0?}
    X0 -- Yes --> Z0[Z = 0]
    Z0 --> R1([RETURN])
    X0 -- No --> Y0{Y = 0?}
    Y0 -- Yes --> R1
    Y0 -- No --> AddExp[Add Exponents]
    AddExp --> SubBias[Subtract Bias]
    SubBias --> ExpOverflow{Exponent Overflow?}
    ExpOverflow -- Yes --> R2([Report Overflow])
    ExpOverflow -- No --> ExpUnderflow{Exponent Underflow?}
    ExpUnderflow -- Yes --> R3([Report Underflow])
    ExpUnderflow -- No --> MultSig[Multiply Significands]
    MultSig --> Norm[Normalize]
    Norm --> Round[Round]
    Round --> R4([RETURN])
    R2 --> R4
    R3 --> R4
```

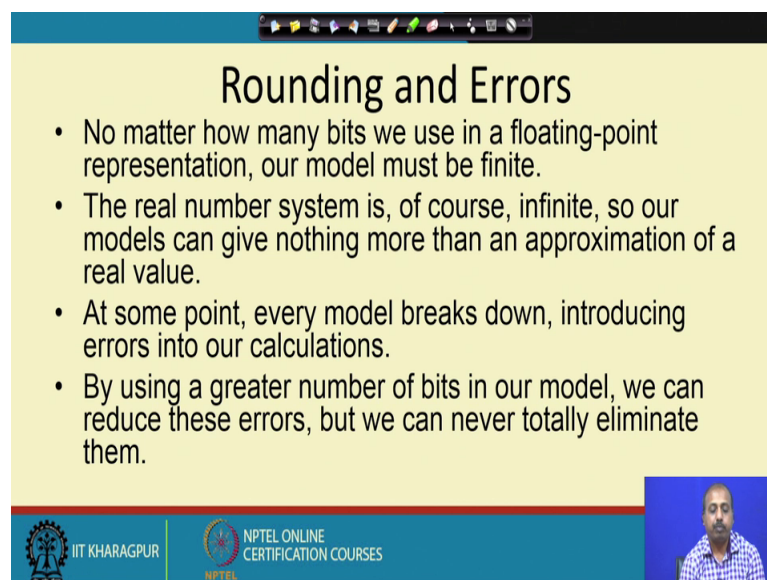


So, this is the multiplication routine. So, we multiply so, first check whether X is equal to 0 or not, if X equal to 0 then we say that we check whether we say that the result Z is

also equal to 0 or if Y equal to 0 then also we say that the result Z is equal to 0 and it returns from there.

And if both are nonzero then we add the exponents because while doing the multiplication the exponents will get added. So, we add the exponent and we subtract the bias and if there is an exponent overflow so, we report overflow. If there is no exponent overflow then we check whether exponent under flows or not, in that case we will report an underflow. And that is not the case then we multiply the significance, they do some normalization and then do rounding and ultimately it return. So, that way this floating point multiplication can take place.

(Refer Slide Time: 08:58)



**Rounding and Errors**

- No matter how many bits we use in a floating-point representation, our model must be finite.
- The real number system is, of course, infinite, so our models can give nothing more than an approximation of a real value.
- At some point, every model breaks down, introducing errors into our calculations.
- By using a greater number of bits in our model, we can reduce these errors, but we can never totally eliminate them.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

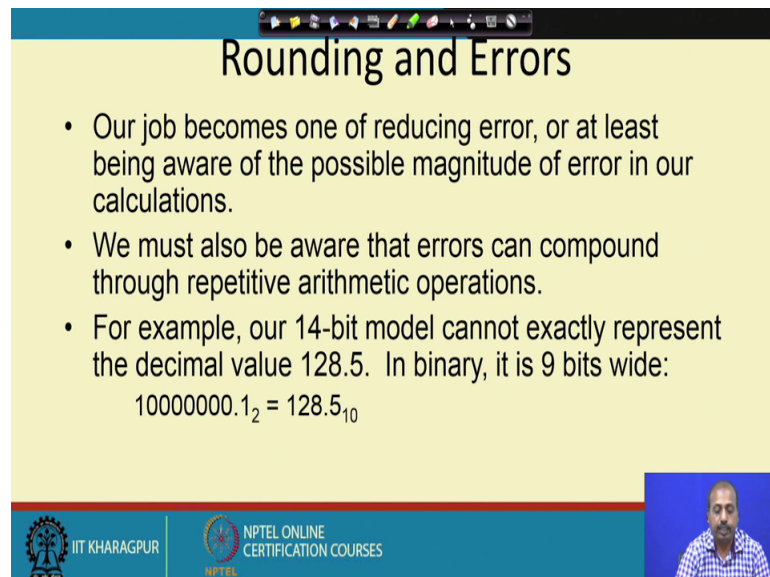
So, you can take a paper pencil type of example and see how to do this thing, and I hope you are already familiar with these multiplications from the school days; So, instead of taking the base 10 so, it is checking base 2, that is the only difference otherwise the rules are same.

So, another very important issue with this floating point representation is the rounding and errors. So, no matter how many bits we use in a floating point representation. So, our model is finite. So, as I was telling that there is this floating point numbers they can go up to infinity. So, this after the decimal point how many digits you are keeping. So, that way it can go up to infinity. So, that way there will be restriction. So, we will have to be

careful and then when there is a restriction. So, we will it will introduce some amount of error.

So, real number system is infinite. So, our models can give a nothing more than an approximation of the real value. So, at some point every model breaks down. So, whatever be the precise the that we allocate for the exponent part and the significant part. So, there is there will be a limit. So, it will introduce some error in the calculation. So, if you use more number of bits, then we will can reduce error, but we can never eliminate it totally. So, these errors will remain it will may not go.

(Refer Slide Time: 10:22)



The slide is titled "Rounding and Errors" and contains the following text:

- Our job becomes one of reducing error, or at least being aware of the possible magnitude of error in our calculations.
- We must also be aware that errors can compound through repetitive arithmetic operations.
- For example, our 14-bit model cannot exactly represent the decimal value 128.5. In binary, it is 9 bits wide:  
 $10000000.1_2 = 128.5_{10}$

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker in the bottom right corner.

So, how to reduce the error? So, we will try to reduce the error or we should be while doing arithmetic with this floating point. So, we should be and we should be having an idea like how much error may get introduced into my calculation and another thing is that, as we are doing some operation repetitively.

So, this may add this may go increase the error part ok. For example, if we are adding all numbers all floating point numbers in an array then as we are adding successive numbers or successive entries of the array. So, this the error also grows.

So, in our 14 bit model we cannot exactly represent the value 128.5. So, in binary it is 9 bit wide so, it is like this. So, in case of floating point representation what will happen is,



I have to represent it as though this 0.1 something and then 2 to the power some value, but that it will not be possible to represent in 14 bits.

(Refer Slide Time: 11:35)

The slide is titled "Rounding and Errors" and contains the following text:

- When we try to express  $128.5_{10}$  in our 14-bit model, we lose the low-order bit, giving a relative error of:

$$\frac{128.5 - 128}{128.5} \approx 0.39\%$$

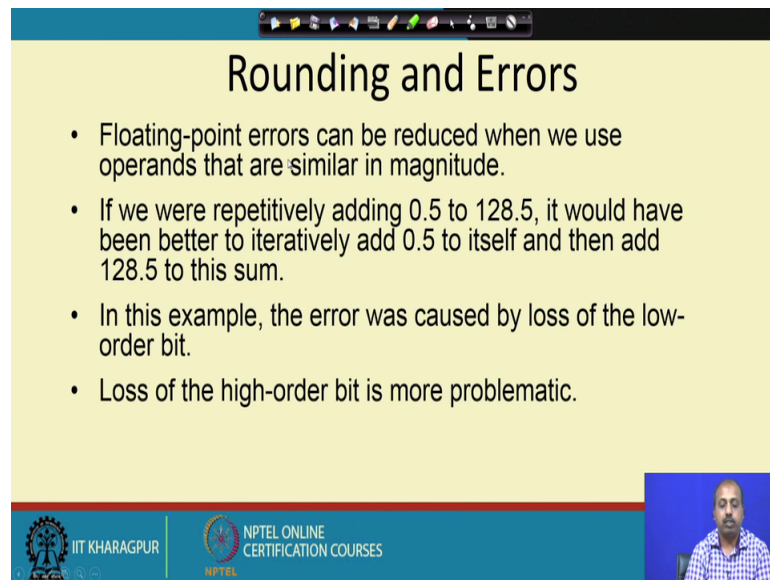
- If we had a procedure that repetitively added 0.5 to 128.5, we would have an error of nearly 2% after only four iterations.

The slide footer includes the IIT KHARAGPUR logo and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset of a speaker is visible in the bottom right corner.

So, what is what will happen is that so, if you represent we try to represent 128.5 in our 14 bit model we will lose the lower order bit. So, it will become 128 only. So, this 0.5 will not be coming, because the last bit will be ignored it will not have so much of space. So, the error that is introduced is about 0.39 percent. So, if we if we. So, so just for representing the number, we are losing 0.39 percent.

So, if we if the go on repeating the process by adding 0.5 to 128.5. So, just we go on adding 0.5 to this again and again, then within 4 iterations. So, it will introduce about 2 percent error in the result. So, that will come because of every time we are doing it so, it is not representable in this system. So, as a result it will introduce some error into the system. So, that is that is how so, these errors can creep in floating point computations.

(Refer Slide Time: 12:35)



The slide is titled "Rounding and Errors" and contains the following bullet points:

- Floating-point errors can be reduced when we use operands that are similar in magnitude.
- If we were repetitively adding 0.5 to 128.5, it would have been better to iteratively add 0.5 to itself and then add 128.5 to this sum.
- In this example, the error was caused by loss of the low-order bit.
- Loss of the high-order bit is more problematic.

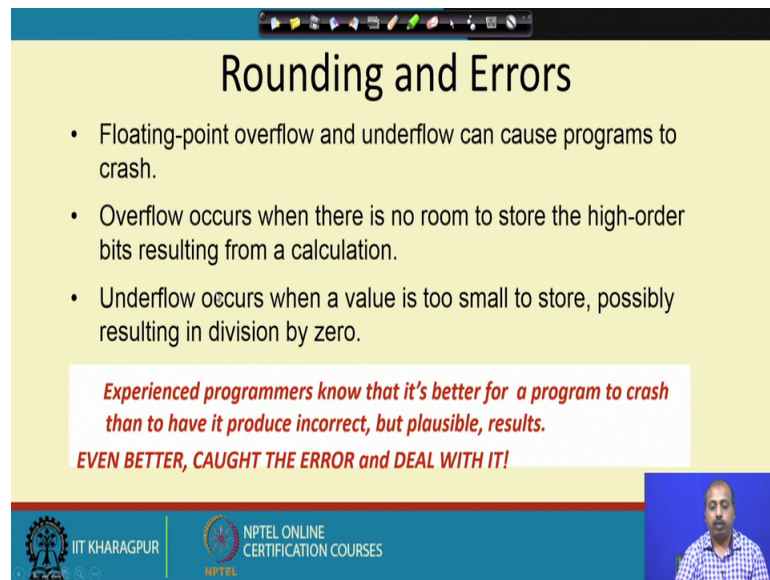
The slide also features a video inset of a man speaking in the bottom right corner, and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

So, floating point errors can be reduced when we use the operands that are similar in magnitude; so, that is one possibility and if we are for example, if we are repeatedly adding 0.5 to 128.5. So, it would be better to iteratively add 0.5 to itself and then add 128.5 to the sum. So, what it says is that since every time we were adding 0.5.

So, you add this 0.5 separately how many whatever be the number of 0.5 is you need to add with 128.5. So, many 0.5's are added together and then the resulting sum is added to 128.5. So, in this way while you are doing this 0.5 additions so, error will not be introduced. So, it is possible that while adding with 128.5 the resulting number. So, the some error will be introduced, but that will be introduced only once. So, that way the so, if the error caused so, the error possibility will reduce.

So, in this particular example so, this error when we are adding 1.5 to 128.5 we lost the lower order bit, but if the values are higher, then we may start losing the higher order bits as well and as a result we can a the situation may become more problematic. So, it may become the number that we are representing maybe far away from the actual number.

(Refer Slide Time: 14:00)



The slide is titled "Rounding and Errors" and features a yellow background. At the top, there is a navigation bar with various icons. The main content consists of three bullet points and a quote. The quote is in red text and reads: "Experienced programmers know that it's better for a program to crash than to have it produce incorrect, but plausible, results. EVEN BETTER, CAUGHT THE ERROR and DEAL WITH IT!". At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video feed of a man in a blue shirt.

## Rounding and Errors

- Floating-point overflow and underflow can cause programs to crash.
- Overflow occurs when there is no room to store the high-order bits resulting from a calculation.
- Underflow occurs when a value is too small to store, possibly resulting in division by zero.

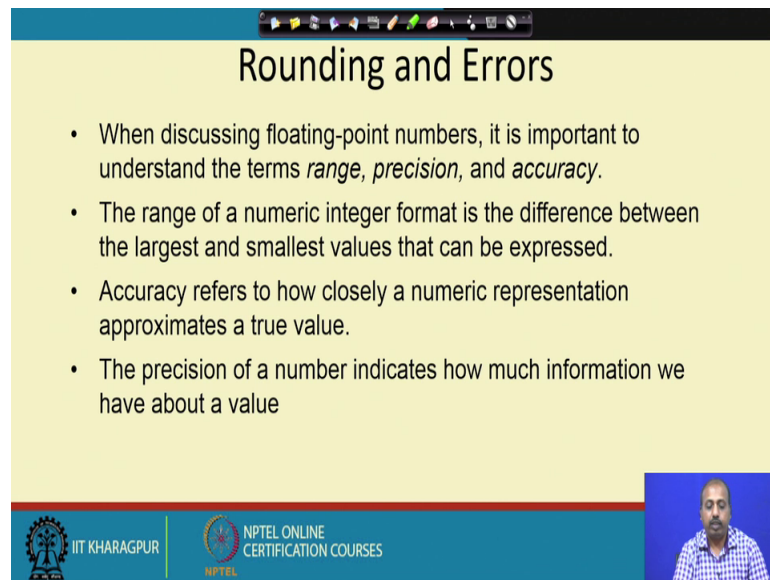
*Experienced programmers know that it's better for a program to crash than to have it produce incorrect, but plausible, results.*  
**EVEN BETTER, CAUGHT THE ERROR and DEAL WITH IT!**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, floating point overflow and underflow can cause programs to crash ok so, this is there. So, though it is a bit of programming side, but it is important because if there is an underflow the number becomes 0 and then when you are doing some division for example, by that number. So, it is a divided by 0 error. So, as a result the program may crash.



Similarly over flow also it may go out of range as a result it may generate some wrong values and then the program may crash. So, this overflow and underflow so, these are the two cases and if we are whenever we are using this floating point representation so, we should be careful. Like if you are writing a program, then it should be careful that this type of cases are caught, and then we avoid the operations using those values, and come up with say some sort of error message ok. So, that may the program does not crash or the system does not crash.

(Refer Slide Time: 15:05)



## Rounding and Errors

- When discussing floating-point numbers, it is important to understand the terms *range*, *precision*, and *accuracy*.
- The range of a numeric integer format is the difference between the largest and smallest values that can be expressed.
- Accuracy refers to how closely a numeric representation approximates a true value.
- The precision of a number indicates how much information we have about a value

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, these floating point numbers. So, we have to there are three important terms with respect to floating point numbers the range, that is the range of values it can represent the precision, with what precision it can do it and the accuracy level. So, range is simple.

So, range of any integer format is the difference between the largest and smallest values that can be expressed. In case of accuracy it refers how closely a numeric representation approximates a true value and this precision means, how many the number how much of information we have about a value. So, apparently it seems that if the precision is higher than the accuracy will also be high, but that is not always the case.

(Refer Slide Time: 15:50)

The slide is titled "Rounding and Errors" and contains the following content:

- Most of the time, greater precision leads to better accuracy, but this is not always true.
  - For example, 3.1333 is a value of pi that is accurate to two digits, but has 5 digits of precision.
- There are other problems with floating point numbers.
- Because of truncated bits, you cannot always assume that a particular floating point operation is commutative or distributive.

Handwritten mathematical formulas on the slide include  $x+y = y+x$  and  $x(y+z) = xy + xz$ .

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a speaker.

For example suppose we are representing the number pi and we have stored the number as 3.1333. Now, you see that value of pi that we have here is accurate only up to two places after decimal ok, but while storing it I am storing value which has got 5 digits of precision. So, 3 and this 1333 so, total 5 digits are stored. So, that way the amount of accuracy and the amount of precision, they are not necessarily same ok. So, they may be different.

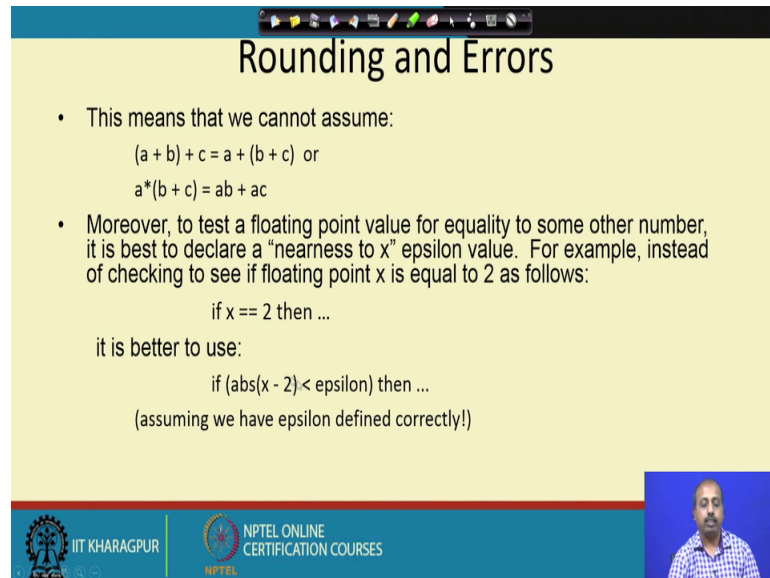
So, there are other problems in floating point numbers also because when your truncate the bits, you cannot always assume that a particular floating point operation is commutative or distributive; like what do you mean by if I have an operation say x plus y ok.

So, this addition operation so, in general x plus y is equal to y plus x. So, this is a commutative operation or x into y plus z is xy plus xz. So, the first one is the commutativity property, second one is the distributive property. So, these operations addition and multiplication say they follow this type of properties.

So, for integers it is true, but for floating point numbers. So, it may or may not be true. So, in reality it may not happen because of this accuracy and all ok. So, there is some of the bits we truncated. So, you so, we cannot always assume that x plus y you will always be equal to x plus y plus x or in particular x into y plus z will be xy plus xz. So, that is

another very problematic situation. So, while handling this floating point number. So, you have to be careful with them.

(Refer Slide Time: 17:45)



**Rounding and Errors**

- This means that we cannot assume:  
 $(a + b) + c = a + (b + c)$  or  
 $a * (b + c) = ab + ac$
- Moreover, to test a floating point value for equality to some other number, it is best to declare a "nearness to x" epsilon value. For example, instead of checking to see if floating point x is equal to 2 as follows:  
if  $x == 2$  then ...  
it is better to use:  
if  $(abs(x - 2) < epsilon)$  then ...  
(assuming we have epsilon defined correctly!)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, for example, so, this a plus b plus c is a plus b plus c. So, if you are doing this addition first and then adding c with it, in another case you are doing b plus c first and then adding a with that then or. So, this is the distributivity property or say a into b plus c is a b plus as I was telling so, here this for floating point number. So, we cannot assume this type of equality is to hold always; like if you are do a plus b first.

So, the result may have may have some got some truncations with that c is added. So, on the other hand when I am being b plus c first so, this gives another this may result in some other type of truncation, with that when a is added the final results may not match of that may be between the two cases ok.

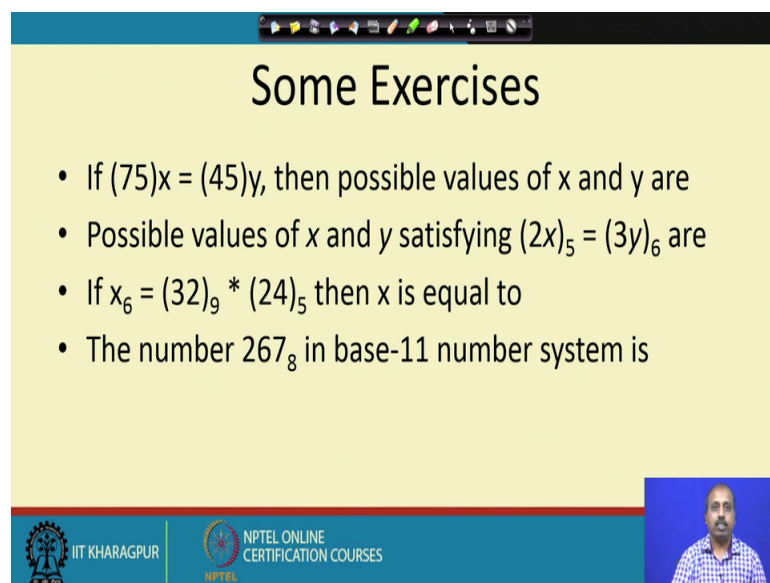
So, this another important thing is that when you are checking the value of a floating point number. So, we have to say like say we want to check say x whether x is equal to 2 or not where x is a floating point number. So, this check may not be correct because this the 2 the number 2 may not be represented with same level of accuracy as x ok.

So, as a result what will happen is that, even if the mathematically this x and 2 should be same, but in the represent from the representation point of view the x and 2 their values may be different.

So, while if you put it in a program. So, you may find that it is creating some problem. So, it is better that you write it like this the absolute value of  $x$  minus 2 is less than sum epsilon so, where epsilon is a very very small positive quantity. So, we find out the difference between  $x$  and 2 take the absolute of that. So, it becomes positive and then we check, whether it is a less than epsilon or not.

So, this way so, normally in floating point problems instead of writing it like this. So, we have to write it in this format to avoid this type of problems ok.

(Refer Slide Time: 19:50)



**Some Exercises**

- If  $(75)_x = (45)_y$ , then possible values of  $x$  and  $y$  are
- Possible values of  $x$  and  $y$  satisfying  $(2x)_5 = (3y)_6$  are
- If  $x_6 = (32)_9 * (24)_5$  then  $x$  is equal to
- The number  $267_8$  in base-11 number system is

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next that finishes our discussion on this floating point this number system. So, next we will be doing a few exercises which are very interesting. So, the first one is says that I have got 2 numbers 75 and 45. Out of this 75 is a base  $x$  number and 45 is a base  $y$  number. So, what are the possible values of  $x$  and  $y$ . So, if we want to do this.

(Refer Slide Time: 20:19)

Handwritten notes on a whiteboard:

$(75)_x = (45)_y$  ✓

$\Rightarrow 7x + 5 = 4y + 5$

$\Rightarrow 7x = 4y$

$x=4$  ✓  
 $y=7$  ✗

$x=8$  ✓  
 $y=14$  ✓

Left side:  $d$   
 $0, 1, 2, \dots, d-1$

So, let us take this one so, 75 to the base  $x$  equal to 45 to the base  $y$  where this 7 5 so, they are the digits of that number system and they are taken to be equal to decimal 7 and decimal 5 ok. So, if I just convert it so, it becomes  $7x + 5 = 4y + 5$ . So, that gives me the condition that  $7x = 4y$  fine.

So, how can I satisfy? So, any value of  $x$  and  $y$  that satisfies this equality so, that is good enough for the mathematical point of view. So, we can say that I will choose  $x$  equal to 4 and  $y$  equal to 7 that satisfies this condition. But this is not a correct result because if you look into the number here 75  $x$  so; that means, this  $x$  must be more than 7, otherwise the digit 7 is not defined for  $x$ . Because if I have got a base  $d$  number system, then the digits are 0, 1, 2 up to  $d - 1$ . So, if 7 is the base then I will have the digits 0, 1, up to 6.

Similarly, for this if so, this  $x$  equal to 4, I cannot get the digit 7 there; similarly if  $y$  equal to 7  $y$  equal to 7 is of course, all right from the point of view. So, this is fine. So, this does not satisfy the relation. So, what else can be satisfied? So, we can have this the next one so, I can take  $x$  equal to 8 and  $y$  equal to 14. So, if I take  $x$  equal to 8 and  $y$  equal to 14, then it is fine ok. So, then this relation is correct. So, we have to so, this is the correct value for this particular problem.



(Refer Slide Time: 22:25)

**Some Exercises**

- If  $(75)_x = (45)_y$ , then possible values of  $x$  and  $y$  are
- Possible values of  $x$  and  $y$  satisfying  $(2x)_5 = (3y)_6$  are  ~~$x=9, y=1$~~
- If  $x_6 = (32)_9 * (24)_5$  then  $x$  is equal to  $(24)_5 = (34)_6$
- The number  $267_8$  in base-11 number system is

$\hookrightarrow \begin{array}{l} 10+x = 18+y \\ \Rightarrow x-y = 8 \\ \underline{4-y = 8} \quad y = -4 \end{array} \quad \begin{array}{l} x = 9 \\ y = 1 \end{array} \quad = 10 \times 4$

IIT KHARAGPUR
 NPTEL ONLINE CERTIFICATION COURSES

So, we will take another example say this one so,  $2x$  to the base 5 equal to  $3y$  to the base 6. So, by a similar logic so, it is  $2$  into the number is  $2$  into  $5$   $10$  plus  $x$  equal to  $3$  into  $6$  that is  $18$  plus  $y$ . So, you get a condition that  $x$  minus  $y$  should be equal to  $8$ . So, if I put any value of  $x$  and  $y$  such that  $x$  minus  $y$  equal to  $8$  then we are done. So, can I take say  $x$  equal to  $9$  and  $y$  equal to  $1$ .

So, that satisfies this relationship, but the problem is this one. So, this  $y$  should be at least  $y$ ,  $y$  should be less than  $6$  and this  $x$ ,  $x$  cannot be  $9$ . So,  $x$  cannot be  $9$  so,  $x$  has to be less than  $5$ . So,  $x$  has to be less than  $5$ .

So, naturally I cannot have any relation that satisfies this particular property. So, I cannot choose any value of so, if I choose  $y$  equal to  $1$  then I will require  $x$  equal to  $9$ . So, which is not set possible so, I can choose  $x$  at most equal to  $4$  fine. So,  $x$  at most since, this is  $5$  so, I can have  $x$  to be at most equal to  $4$ .

So, if I put  $x$  equal to  $4$  ok so,  $4$  minus  $y$  equal to  $8$  or  $y$  also equal to  $4$ . So, that is a possibility. So, I take  $x$  equal to  $4$  and  $y$  equal to  $4$ . So,  $24$  to the base  $5$  equal to  $34$  to the base  $6$  is it correct. So,  $5$  into  $2$   $10$  plus  $4$  sorry then of course, so, this  $x$  minus  $y$  no sorry this is not correct because this is this  $4$  minus  $y$ . So,  $y$  is becoming minus  $4$ .

So, the so, I cannot choose any value of  $x$  and  $y$  which satisfies this relationship. So, this I cannot get any satisfying value for this problem.

Then this one the third problem is simple. So, here we have got 32 to the base 9 and 24 to the base 5. So, you multiply them and then so, for multiplication purpose. So, we have to we can first convert both of them to their decimal values do the multiplication, and then convert it into a base 6 number system. So, you have to you have to go on dividing the number by 6 and see what are the remainders and then the remainders you can represent it like the.

(Refer Slide Time: 25:15)

**Some Exercises**

- If  $(75)_x = (45)_y$ , then possible values of  $x$  and  $y$  are
- Possible values of  $x$  and  $y$  satisfying  $(2x)_5 = (3y)_6$  are
- If  $x_6 = (32)_9 * (24)_5$  then  $x$  is equal to
- The number  $267_8$  in base-11 number system is

Handwritten work on the slide:

$$\begin{array}{r} 2x \\ 64 \\ \hline 128 \\ 48 \\ \hline 183 \end{array}$$

$$2 \times 64 + 6 \times 8 + 7$$

$$= 128 + 48 + 7$$

$$= 183$$

$$\begin{array}{r} 11 \overline{)183} \\ \underline{116} \phantom{0} \\ 67 \\ \underline{66} \\ 1 \end{array} \quad (157)_{11}$$

The slide also features the IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES logos, and a small video inset of the presenter.

Then the fourth one this 267 to the base 8 in the base 11 number system so, here first of all these 267 to the base 8. So, this has to be converted into decimal number system. So, this is 2 into 64 plus 6 into 8 plus 7. So, that is 128 plus 48 plus 7. So, that is this is 55 so, 55 plus 128 so, 128 plus 55. So, this is 3 183 so, we get 183.

Now, this 183 if you divide by 11 so, it is 1 then 7 so, 6 so, 66 then this is 73. So, this is your 7 then again by 11. So, you get 1 and 5 and then 11 0 and 1. So, 157 in the base 11 number system. So, this way you can convert numbers from one number system to another using this division.