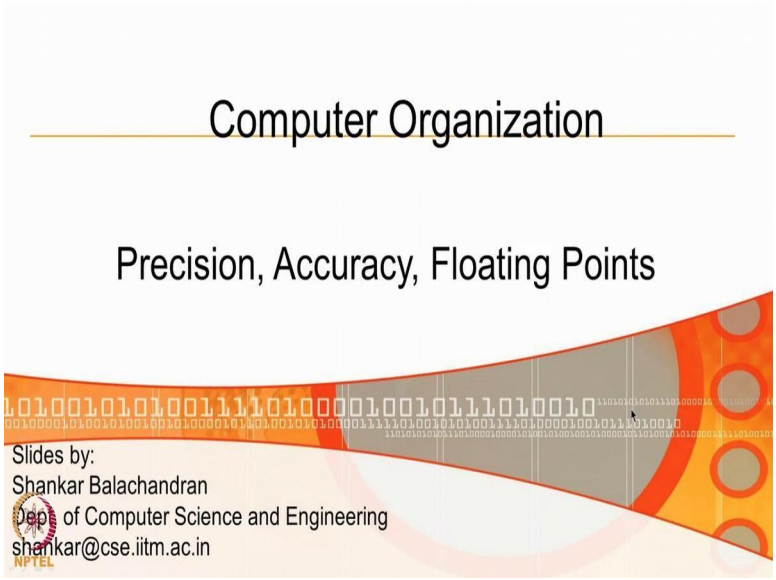
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 08
(Part I)
Floating Point - Precision and Accuracy

Both on your commercial computations like E-commerce and also scientific computations and lot of you know data analytics etcetera, we use floating point extensively right in all media applications for example, there is a need for using floating point in a big way.

(Refer Slide Time: 00:24)



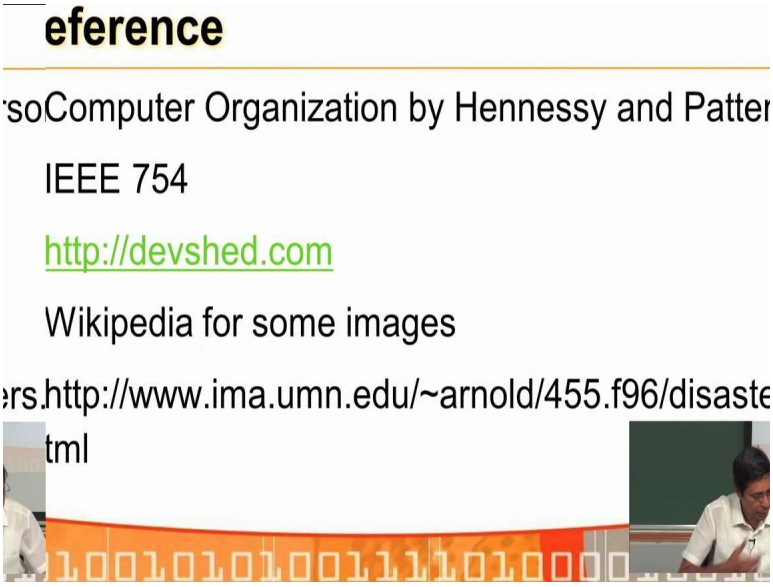
So, there when we use floating point as its I told you that this is not an infinite arithmetic space when you look at systems we have limitations we cannot represent beyond something that is the range right; if you take twos compliment arithmetic with n bits you can only you know the range is minus 2 power n minus 1 to plus 2 power n minus n minus 1 we cannot go beyond that range.

So, because of this finite representation what are the problems that you land it? If you are just looked at we know integer arithmetic, the problems are very simple. So, if the range if I give you n bit and the range is between minus 2 power n minus 1 to plus 2 power n minus n minus 1 either you go above that range or below that range. So, underflow and

overflow were the 2 things that you saw in integer arithmetic, but when we go to floating point the thing becomes much little more complex.

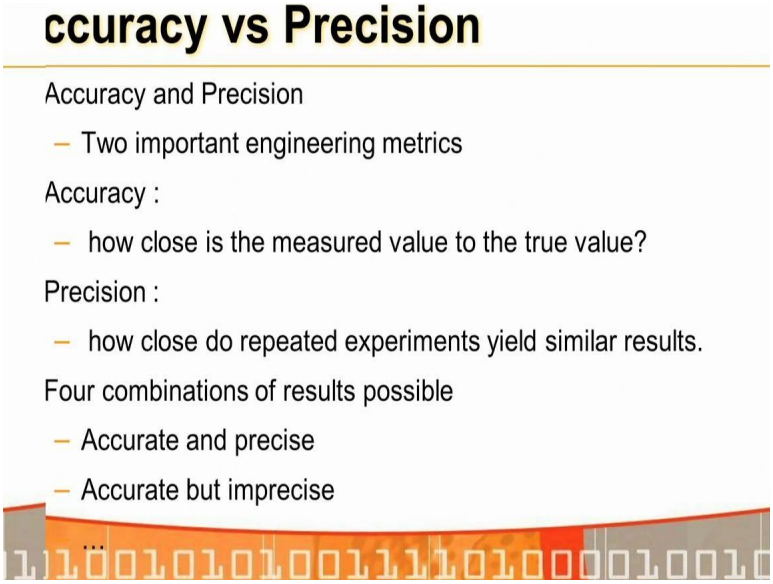
So, let us understand that is a set of very nice examples, so these slide are made by professor Shankar Balachandran is Intel now, but so, I am just borrowing that slides some excellent slides and we will try and understand especially what is the difference between things like what is precision, what is accuracy those things becomes very very important when we look at floating points a arithmetic. So, this is a very very crucial aspect of this course and I hope we will learn that in the proper prospective I will teach and you learn the proper perspective.

(Refer Slide Time: 02:11)



So, these are the references for this. So, Hennasy and Pattersons books on computer organisation I triple E 754 standard then some web pages and Wikipedia.

(Refer Slide Time: 02:36)

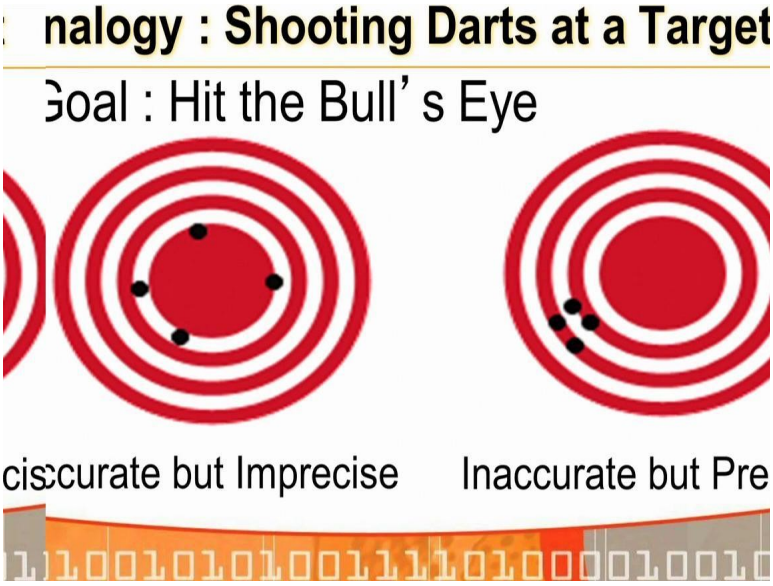


Now let us start the lecture with what do we understand by the term accuracy what did we understand by term precision. Accuracy and precision are 2 very important engineering metrics, what is accuracy how close is the measured value to the true value that is what we mean by accuracy. What do you mean by precision? We keep doing an experiment repeatedly how close are the results right if a experiment. So, if my results are repeated if I repeat the experiments and I get almost the same result then I same way arithmetic is precise right. So, accuracy means how close is the measured value to the actual true value; precision is how close do repeated experiments yield similar result results right. So, these are difference between accuracy and precision.

So, we need both accuracy and precision because we are run a programs say m times every time I want the same answer, I do not want different answer and I run program in machine one and I run it in another machine two I need to have the same answer correct. so that is what I mean by a precise computing environment, and the answer that this computer generation of whichever computer generates how close it this to the actual value that is called accuracy right; are you able to distinguish between these two?

Now there are four combinations of results possible, I will have a accurate result and a precise result I will have an accurate result, but imprecise result. I could have an inaccurate result, but precise in accurate and imprecise.

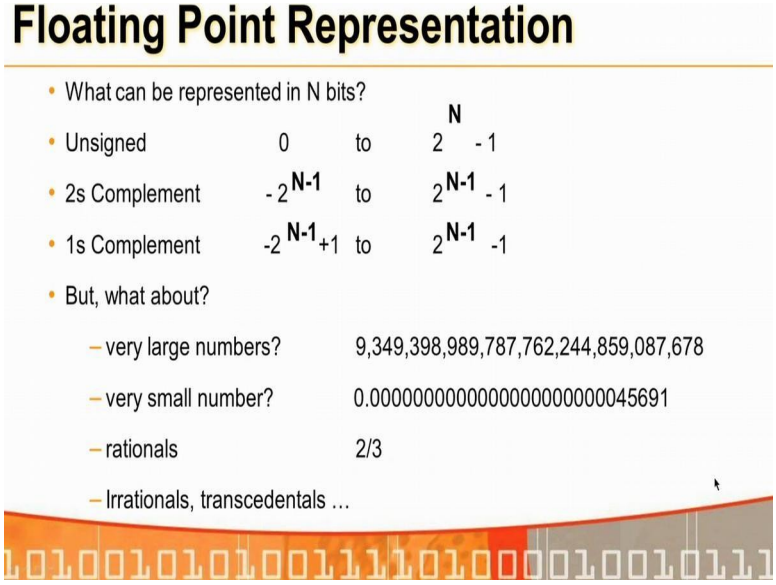
(Refer Slide Time: 04:40)



So, all four combinations are possible; now how so on I we will now see some experiments right I will put those experiments here you can go and check it out very very simple programs, now you can check it out on your real system you can write those c programs and compile and execute it see for yourself how each of these four combinations are possible in floating point arithmetic and what sort of care we take right; are you getting this? This is the difference between an precise accurate fellow and a precise fellow, let us been shown shooting dots on the target right.

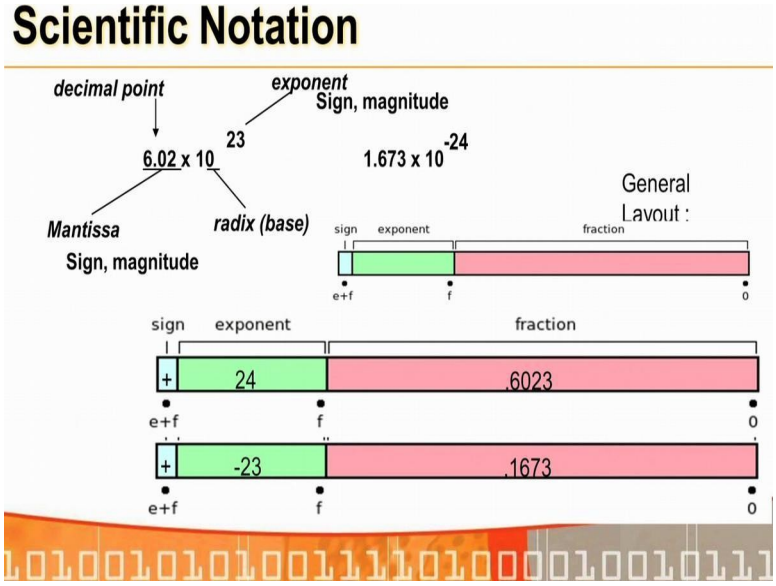
Now, what do you see on the left hand side, it is an accurate hit almost accurate hit on all ground like we were very close to that whatever the middle red circle, but it is imprecise. Different times I hit the dots are far off from each other, but on the other hand when you see on the right hand side it is in accurate, we never the dots never reach the central circle, but precise, right. So, this should have it in mind I will I will give several analogies like that that we proceed. So, where you should understand what is accuracy and what is precision and especially when you start doing you know these are something switch.

(Refer Slide Time: 05:37)



Now, let us look at floating point representation right. So, before going to that what can we do with in which if it is a unsigned, so I can represent anything from 0 to 2 power n minus 1, if it is twos complement just minus 2 power minus 1 to 2 power n minus 1 minus 1, just once complement I can do minus 2 power n minus 1 plus 1 to 2 power n minus 1 minus 1. So, these things we have seen a in your digital course and of course, in your c s 100, but when you want to represent large number say for example, very big one or very very small numbers or rationales irrationals transcendental like pi e right then we need a different representation and that is why and that representation is very crucial because it the next we some standard there are we can come out with infinite trance for represent these numbers. So, there was a need for getting a standard and that is why the; I triple E 754 standard came into place and we started off from that.

(Refer Slide Time: 06:58)



Now, what we will now discusses about the I triple e 754 standard? Now this is overall the I triple E 754 standard. So, I want to what is this numbers 6.02 into 10 power 23, right this is have a grade of number. So, what we need here, we need a sign, a magnitude then a mantissa, decimal point then exponent, and then a radix and base for a exponent right. So, I need. So, I have an exponent I have whatever fraction there and then I need a sign bit so for example, this 6.023 into 10 power 23 can be can be written as 0.6023 into 10 power 24 it is a plus sign right. Similarly your 1.673 into 10 power minus 724 can be written as 0.1673 to the power minus 23 and a sign here plus.

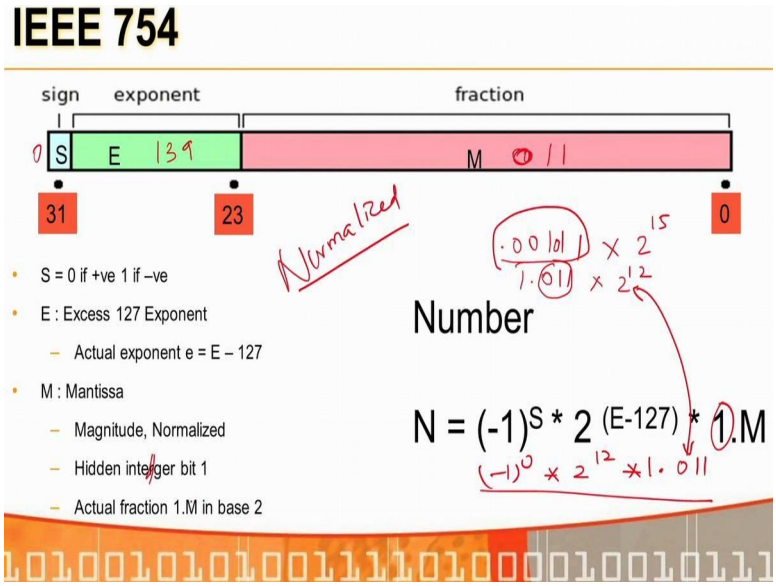
So, there is an exponent there is a mantissa or whatever the fraction and then a signed it, this is the basic representation here, but we want to make this you know look. So, I cloud have see why I say there are infinite base of representing this, I can represent it as 0.6023 into 10 power 24 or I could have even represent it as 6.023 into 10 power 23 or I could put 23 here I could 60.23 and put you know 22 here right. So, on right or I could put 0.062 023 in to 10 power 25, right. So, I want different ways of representing this. So, I need to have some standard if to 2 computers start representing in 2 different ways.

What will happen is a program can never be portable across system. So, if I write a program for say some in Intel machine then I cannot use it in say IBA machine, if I use run for IBA machine I cannot use it as in some where some spark machine or the DC machine, right.

So, I need to have a common floating point representation so that I could

use it I can port my programs. So, you sell E-commerce program, you sell E-commerce software, you sell you know engineering computation software like MAC lab etcetera, etcetera right. So, we need we cannot write different softwares for different architectures right we need right one architecture and compile it for different machines, so that is why this standard has come out, right.

(Refer Slide Time: 09:49)



Now this is the I triple E 754 floating point format that there is one sign bit, this is a single precision when you say float x right this is what you see that, is that 4 byte representation. So, 23 bits 0 to 22, 23 bits are used for M that is the mantissa and there are 23 8 bits are used for the exponent and one bit is used for the sign is that fine right if S is actually 0 then it is positive, S is 1 then it is negative. So, sign bit is very clear if I put 0 there is a positive number 1 is a negative number. Now how do we store the exponent? We store the exponent in what we call as the 127 excess 127 format; whatever is the exponent you want you add 127 to it and store it right. So, by that with the exponent is always a positive number then it is stored here.

So, if I have minus 120 suppose I am I am I want to store say 10 power minus 127, you add 127 to it and store 0 here. So, your exponent value will go from 0 to 2 power 8 to 8 bits na 255 0 to 255 essentially you are storing from minus 127 to plus 127 right, plus

128. So, every time you add 127 to it and store it, right. So, always your exponent is

going to be a positive number exponent that is being stored here is a positive number,

you can store actual representation you can have negative exponents, but since you add 127 it becomes a positive number, and then how do you get the fraction track this is once and zeros. So, you make the fraction as 1.110 or whatever, you do not store that one you store whatever is after that a decimal point. So, what would be the value of this? So, suppose I am storing S E and M the value of this is minus 1 power S, right.

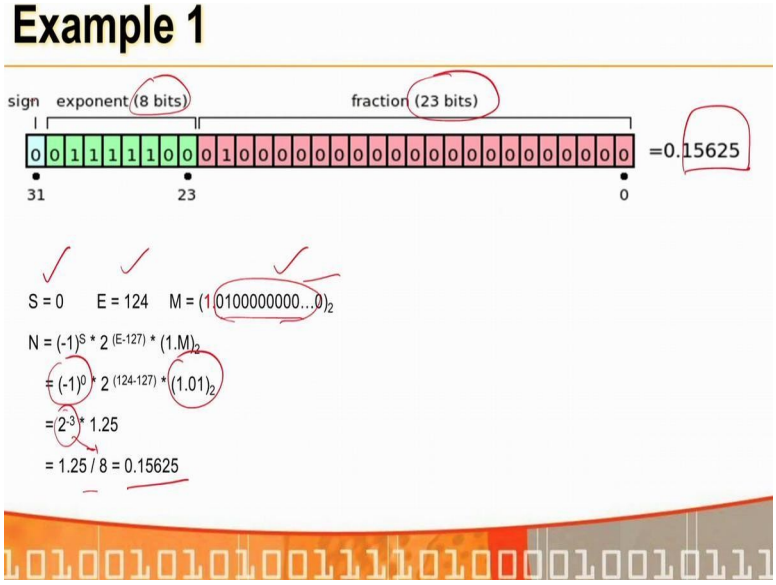
So, if S is 0 then it is 1, if it is a plus then f is 1 it is minus into 2 into E minus 127 because you have stored it in excess 127 format into whatever M here 1.M right. So, whenever I get the binary. So, if I get point 0.00111 I will store it into say some 2 power 15, I will now make it as one point sorry 0.110 let me put I will make it as 1.011 and 3 right into 2 power 12, right, got this; and I will store 0.011 here I will store point 0 sorry not point I will store 011 here and of course, 12 plus 127 as stored 139 here and of course, I will store 0 here 011 is some 3 whatever. So, 139 in 139 I will store it in binary format am not converted it. So, when I want to retrieve it, it is going to be minus 1 power 0 into 2 power 139 minus 127, 12 into this 1 comes here one point whatever I have 01 which is nothing, but this are you able to appreciate this. So, the good thing is that exponent is always positive and mantissa is normalised we use the word called normalized.

Normalized in the sense that I this is the un normalized value I normalization is essentially to bring 1 1 here and one point something. So, to convert your whatever number you have got into the form as one point something else, and the something else is stored in m one is not stored right. So, one is implant here are you able to follow this yes

Student: how do we (Refer Time: 14:39).

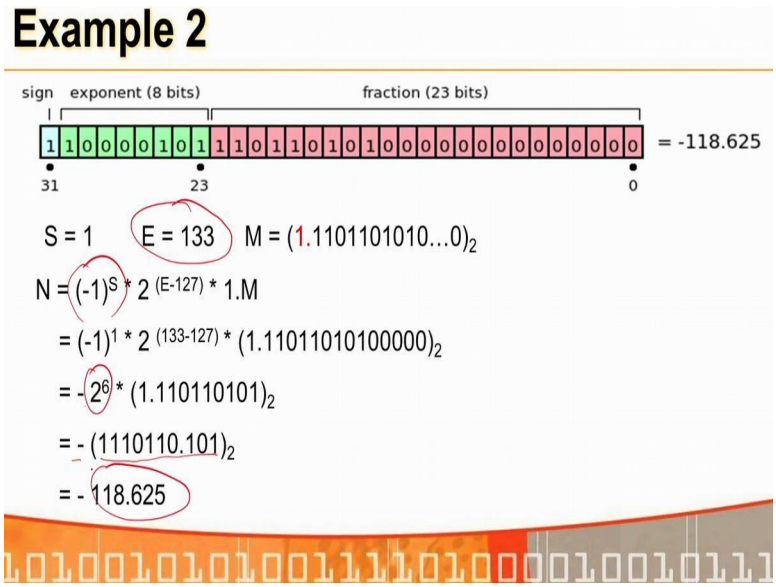
Will come to that, so, M is the mantissa which is the magnitude of the number normalized there is a hidden integer bit, integer bit 1 and of course, the actual fraction is 1.M.

(Refer Slide Time: 14:53)



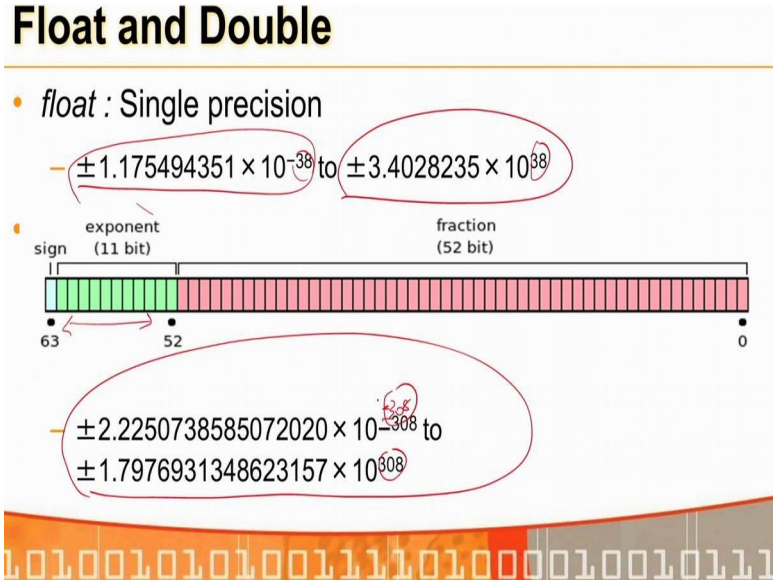
So, let us look at this. So, S is 0, S is 0 here, E is 124 and M is one point whatever this whole thing. So, when we convert this part this is 0.15625 you see here this whole thing. So, how do you convert this minus 1 power 0 because into 2 power 124minus 127 which is minus 3 into 1.01 whatever; this is 2 power minus 3 into one point is it convert 0 1 from binary to decimal which is point 1.25 right. So, 1.25 divided by 8, because 2 power minus 3. So, this is 0.156, right. So, this is how 0.15625 will be represented right. So, you have a 23 bit mantissa 8 bit exponent and a sign bit.

(Refer Slide Time: 15:56)



Now again, so this is again a negative number minus 1 power 1, and E is 133. So, this is 2 power 6 and this 0.1101 whatever is one point this. So, this is now currently minus 1110 whatever you see here into 2 power 6 right I just bring it here, when you convert it this is minus 118.625.

(Refer Slide Time: 16:33)



This capacitive number with a positive exponent negative number can decimal. So, I am just giving you 2 examples very straight forward. So, single precision I can represent numbers like this, right.

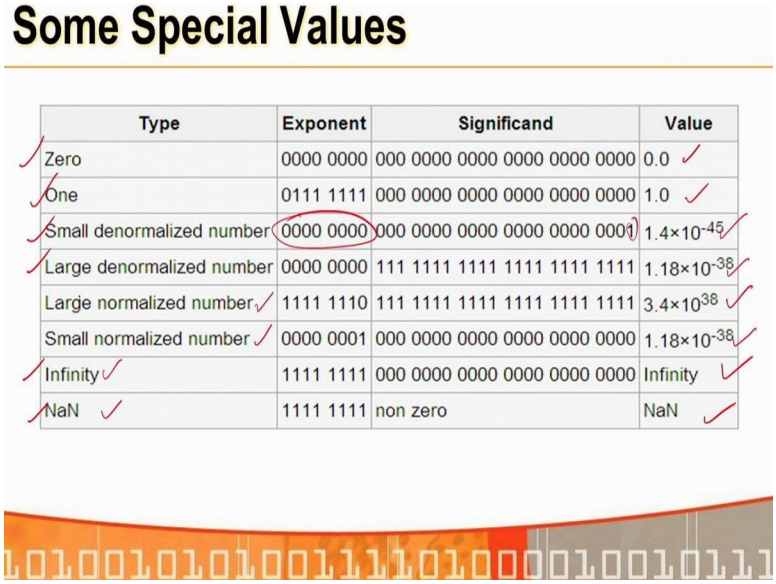
But when I want something like this, right, so, when I want something as biggest this right the single precision I can represent numbers like this, but if I want to have a you know bigger numbers right as you see here then we going for a double precision. So, how does the double precision work? It is 64 bits that is 8 bytes same [FL] no change here, but except that this is 11 bit exponent 52 bit fraction and one sign bit. So, with 11 bit what we can do? We do an excess 1023 like what we did for excess 127 we can do a excess 1023 right so that means, we should have a very big range here.

So, interestingly this is the minimum number and maximum number that could be represent right now just minimum plus or minus 1.175 into 10 power minus 38 to plus or minus 3.4 into 10 power 38, this is the range right that is the plus or minus here plus or minus here and this is the. So, this is binary converted to decimal for a single precision.

So, really when I go for double precession this is this is minus 308. So, this is the range

in which we can represent yeah. So, so please note that my range comes at most 9 times in my exponent right when I go from single to double I increment it by nine almost 9 times yeah 8 times right almost 8 times my exponent can increase 38 to 308 time minus 38 2 minus right. So, this is how floating and floating single precision and double precision are actually represented and enough able to follow, right.

(Refer Slide Time: 19:08)



Now, these are some special values like you asked how is 0 represented. Your exponent is 0 your mantissa is 0 then the value is 0, one how do you represent? Your exponent is this and your mantissa is 0. So, these 2 and there are numbers which we cannot basically normalize at all it, is so small right. So, it is called small denormalized numbers. So, these are all some large denormalized, the smallest and the largest right where I just can put one one here largest is all once. So, if the exponent is 0 sort of this becomes a denormalized number. A largest normalized number is off course this smallest larger normalized number is this, you can you can represent infinity as this and if this is non zero then this is not a number, many I cannot I cannot represent it is out of this range.

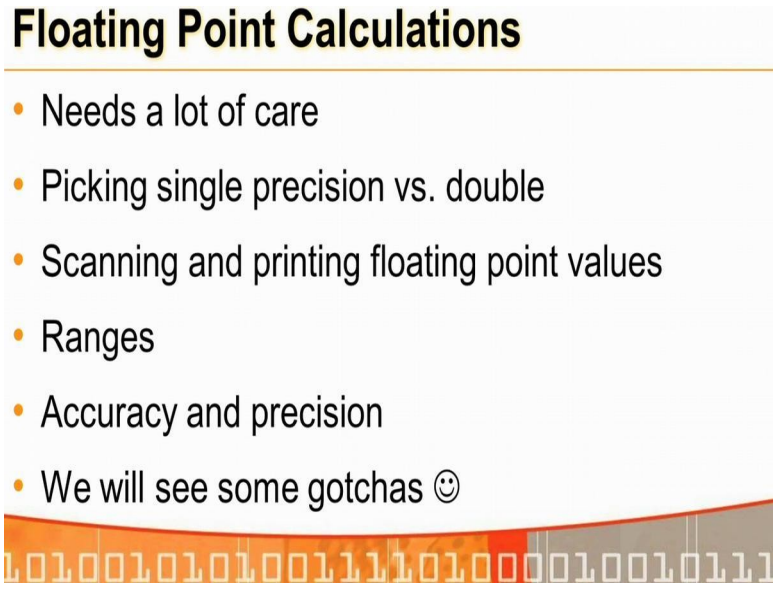
So, these are some special values that we need to keep in mind because there are some numbers which we cannot normalize, because your exponent becomes much smaller than that. So, those numbers we just we put the exponent as 0 and put that number here. So, these are all this is smallest denormalized number and this is the largest denormalized

number, and I cannot express it as one point something into 2 power this because it exists

it is outside that 0 to 255 element in your exercise even in your excess form either below or above right and similarly. So, this is the largest normalised number and smallest normalized number, then I have a representation for infinity and a and something which I cannot represent even denormailzed I cannot do then call it as a not a number right, many times when you start doing floating point computation you will get NaN, what is NaN right this is this is something that I cannot represent in this right.

So, this is 1, 2, 3, 4, 5, 6, 7, 8, this is for single precision of course, the same can be extended for double precision, but you should understand that when I interpret a floating point value if I interpret those 4 bytes that interpretation is not very straight forward there are of course, if you have a non zero exponent and a non zero significant is not just if I except these you know 1, 2, 3, 4, 5, 6, combinations that you have seeing here other than that interpretation is what I have told in the previous slide, but these 6 combinations you should very very clear. So, this is extremely important here some steps we need to keep in mind when we start looking at floating point representation.

(Refer Slide Time: 22:36)

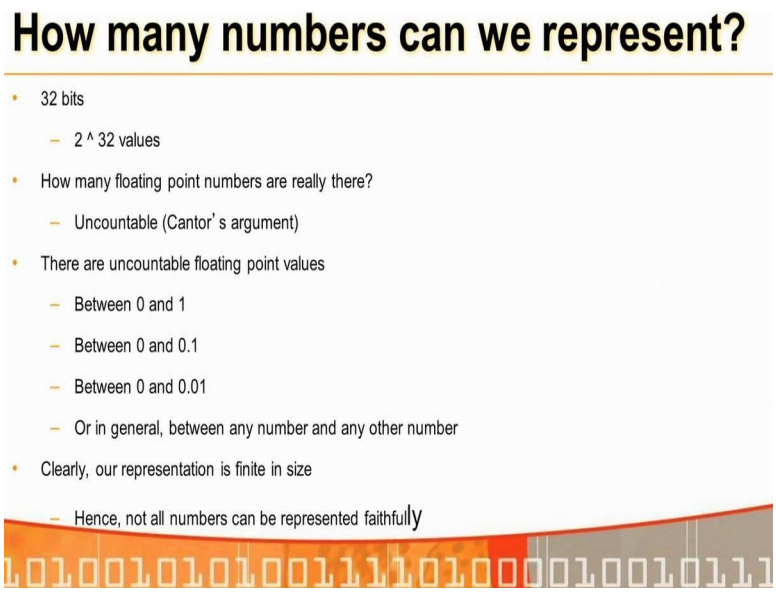


So, now with representation itself we had these type of issues when we start computing using floating point then we need to be much more careful. Now let us take this right we need to be very careful in the sense what sort of accuracy and precision and range I need right I need a bigger range or the small range. So, what sort of accuracy in precision I need

that. So, I have to go and pick one of the decision I need to take this whether should

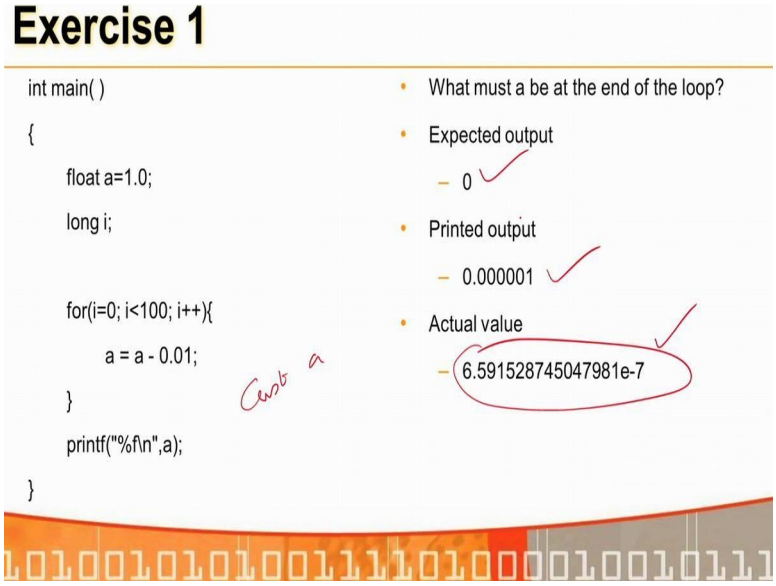
I have a single precision or a double precision and that is also very crucial right and when I scan and print floating point values what is actually there in the memory will not come out, because your print f and scan f also does some jugglery which in floating point when it is printing it that also you should be aware right. Print f it wants to print the things it does some bonding it does some truncation even you are scanning. So, your scan f and print f in your c; we should be bothered about the ranges and accuracy in precision and based on that we should select what we want to do and the results that you get right the results you get you cannot just relay on print f and scan f for the results right then you may your computation can be accurate, but the way it present it can be different. So, we will see some very interesting examples. So, we will see all these cot asks.

(Refer Slide Time: 23:56)



So, first one thing if I have 32 bits, I can represent 2 power 32 as. So, how many floating point numbers are really there uncountable right this cantors argument right if you studied that sign if we are not its uncountable because between 0 and 1, between 0 and 0.1 between 0 and point not 1 and then again keep on right I could have infinite number of floating point values a real life real number line is really a continues line integer line is at this screed line so, but our representation is finite in size. So, I been repenting this I will repeat it because I believe several times I will repeat once it will get into your mind.

(Refer Slide Time: 24:53)



So, now please note that the problem is here that I have infinite numbers infinite floating point numbers, but I have only finite size to represent it and because of that what are the things that are going to happen let us see some examples. What do you expect here? I start with a equal to 1 is a long I is a long number that is fine into this is single precision for I repeat it hundred times and every time I am going to subtract point not one from this. So, what is the answer you will you should get 0 right; note that your expected output is zero, but actually a printed output would be point not, not, not, not, not, one, but the actual value that will be stored in your memory some 6.591 into 10 power minus 7 right. So, you can how do we see you take this a. So, this you can just print this mathematically you calculate this is what you print you will get a real g c c you will get something, something very close to this, but how will you find this.

Student: (Refer Time: 25:59).

Multiply a;

Student: (Refer Time: 26:00)

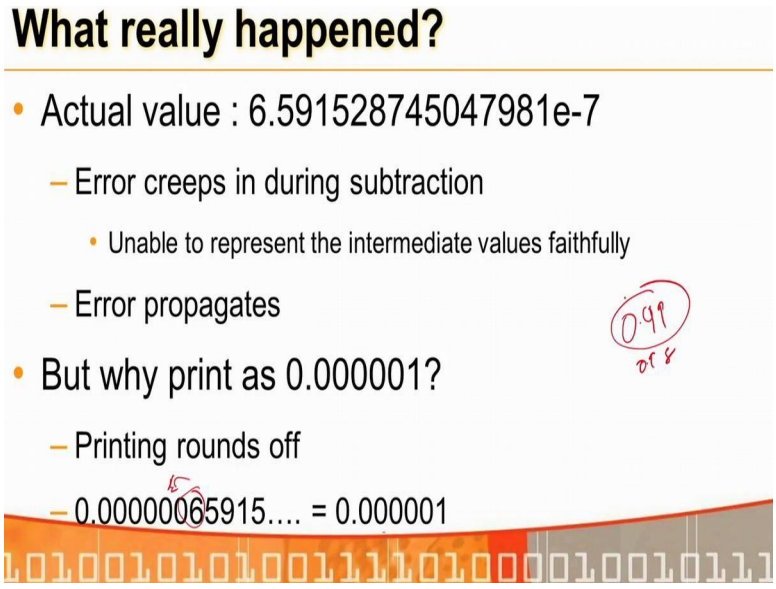
No, no, no, what you do is you cast a as a character pointer right and print those 4 bytes of a and actually find out how what characters are there that as key character as key values right you can cast it as character and print that four characters starting from that point right you can give up a pointer and increment the pointer and print the contents and

from there you calculate exactly what it is you will find out that this is the value that is stored in the system. So, first thing is mathematically it is uncorrected incorrect, but what the computer actually computes this print f does not print that it is printing something else. So, the print f itself has something to go and massage it and give you some answer right. So, what you see as a print out as an answer is not actually what is stored in the memory and it is not the actual mathematically what is stored in the memory is not the correct answer right you see there are 2 levels of inconsistencies that creeping. Imagine that I take this answer I put it into a file and some other time later somebody takes this answer and do further floating point computation already you have got a ironies results and that error still keeps up and becomes a bigger error correct. So, this is something I do not know did you learned this in your c s 100 what is this thought? Yes?

Student: (Refer Time: 27:46).

So a lot of redundancy, but still I will continue now. So, the error actually crept in while we were subtracting, right. So, why because some of the intermediate values they are not able to represent faithfully exactly, and then the error propagates.

(Refer Slide Time: 28:14)



But why the print statement printed it as point not, not, not, 1 because the printing actually rounds it off. So, it says as you see here you see 1, 2, 3, 4, 5, 6, 7. So, this 0.06 actually became one. So, print f actually wrongs the value. So, these are I;

Student: Sir what do you mean by intermediate values.

Like 0.01, right, 1 minus point not 1 is point not 0.99, 0.98 right when I want to represent 0.99 it will not be exactly 0.99 because of your 0.9.

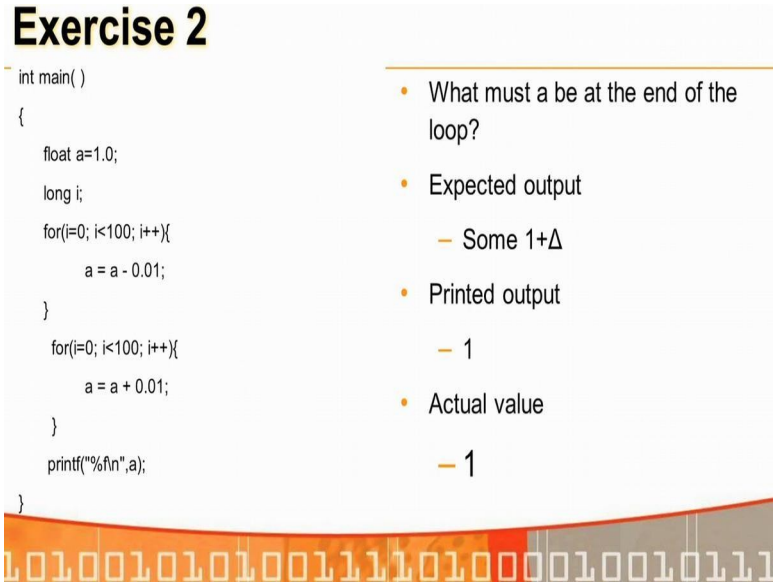
Student: But in our example a 0.99 can be a (Refer Time: 28:55).

Faithfully it coming it will be some 0.999 you know something.

Student: 0 right.

No, non no, this is binary like yeah, yeah, I do not have a decimal number fine binary it will yeah just write those.

(Refer Slide Time: 29:15)



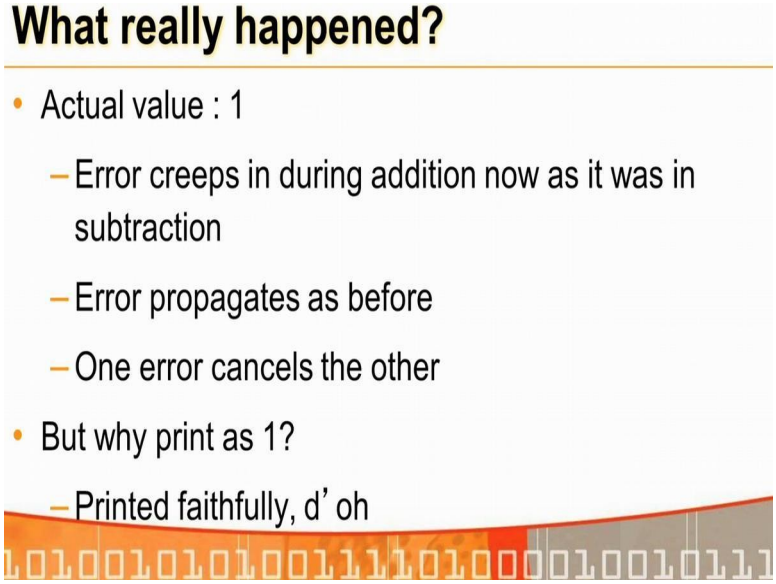
Now let us see what will happen here. So, I am subtracting from a equal to 1 and I am subtracting it 100 times and adding 1. So, the answer should be again 1. So, the expected output is 1, now you want say one will say 1 plus delta. So, actually the printed output is 1.

Student: (Refer Time: 29:40).

Because you expected 1 plus delta, but actually it comes 1 becomes a great, but the actual value stored also 1. So, 2 negatives make a positive. So, the error in subtraction

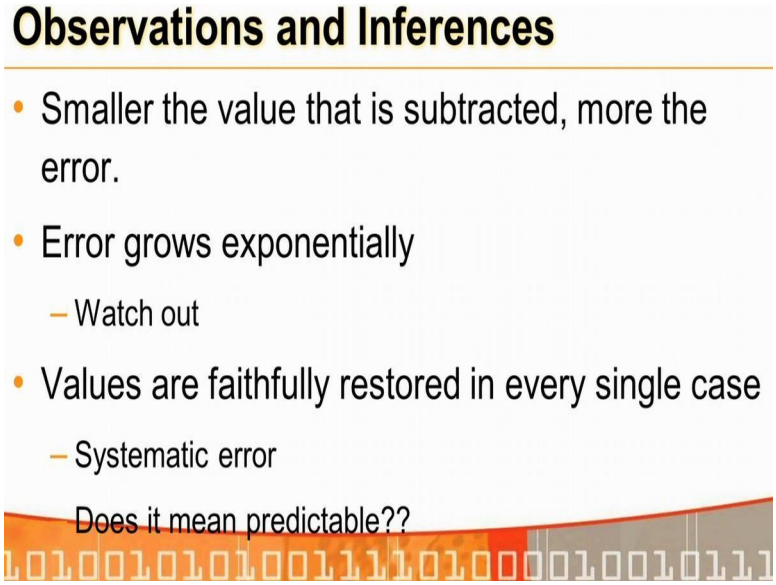
got compensated by the error in addition. So, sometimes error hides each other. So, they complement each other.

(Refer Slide Time: 30:08)



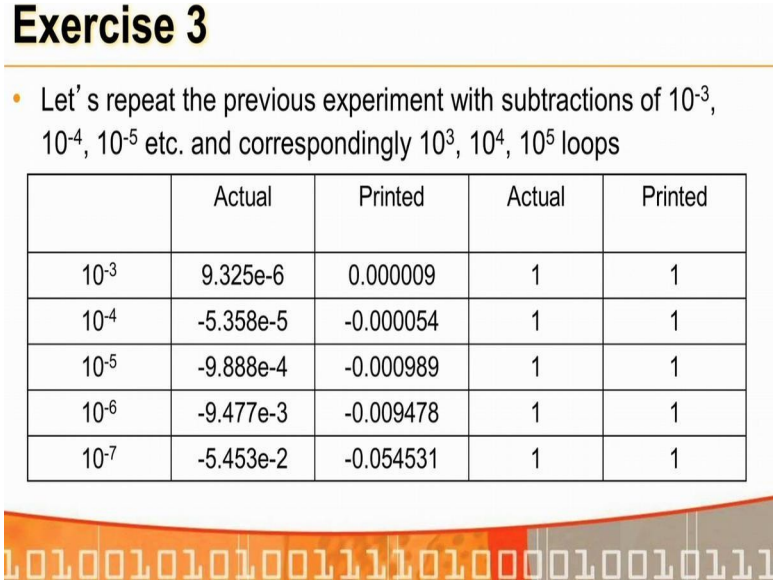
Error creeps in during addition now as it was in subtraction error propagates as before, but one error cancels the other, but why printed as one because now it is printing it faithfully.

(Refer Slide Time: 30:26)



So, please understand these are all the issues that we land up when we start doing these type of floating point propagation.

(Refer Slide Time: 30:27)



So, I will give you simple exercise lets repeat this previous experiments with subtraction of 10 we needed to 10 power minus 2 right minus 0.01, do it thousand times with 10 power minus 3, 10 power minus 4, 10,000 times and hundred thousand times, it is and correspondingly have the loop should be 10 power 3, 10 power 4 and 10 power 5, but actually you will see that the actual value that will be there the actual value that will be there would be you know in the system what will be printed is this for the first experiment that is just subtraction, subtraction plus addition is always the error gets cancelled, right.

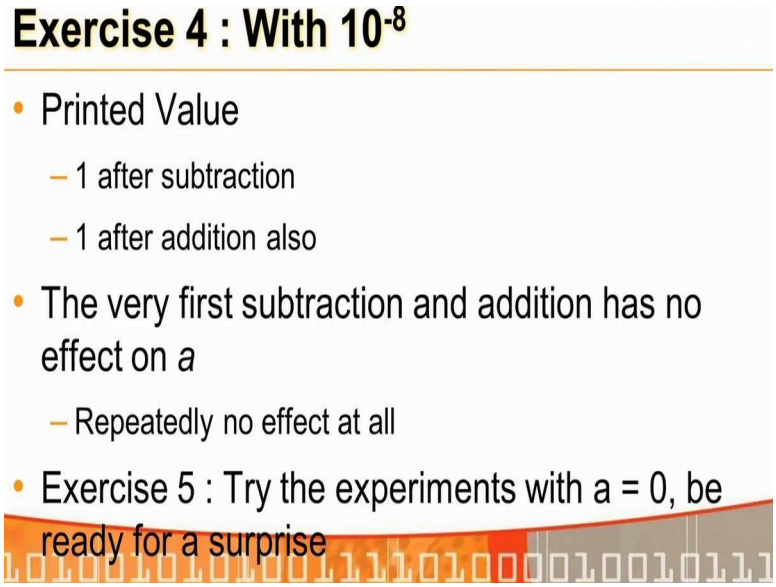
So, please see that smaller the value that is subtracted more the error like that is also very important. So, when I; so, this is becoming 10 power minus 3 this the error was 10 power minus 6 here, it became 10 power minus 3 here, error was 0.00009 which is now become 09.7 eight. So, some orders of magnitude the error has creeping. So, smaller that I am going to subtract 1 minus 0.0000001 then the smaller the thing I subtract more the error right are you able to see this and that error is growing exponentially, say here it was in the seventh place 1 2 3 4 sixth place that is say 10 power 6, here it became 10 power minus 5, here it became 10 power minus 4, 10 power minus 3 every time it is getting multiplied by 10. So, the error also basically grows exponentially and the values are faithfully restored in every single case; that means, what when I subtract and then I add then it is basically faithfully representation so; that means, the error is systematic

right. So, that something happening wrong in that subtraction exactly the opposite thing

happens in the addition to compensate for you. So, does it mean this error is predictable this is some questions that we need to answer?

But just note here that error if I go and become more and more accurate right I want to become more and more accurate by increasing by you know floating point value if I want to keep decrementing by say 10 power minus 5, minus 6 minus 7 then the amount of error that accumulates becomes exponentially, it increases exponentially that is something that we need to keep in mind when we do this in spite of all you are I triple E 754 standards ability right this is something that we need to very much keep in mind.

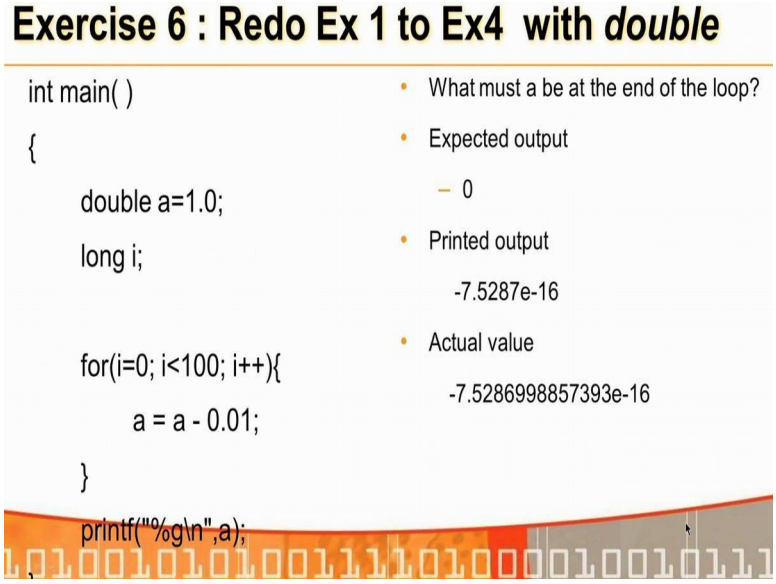
(Refer Slide Time: 33:29)



But what will happen if I do with 10 power minus 8, we have done till 10 power minus 7 interestingly you will find that the printed value is one after subtraction and one after addition also.

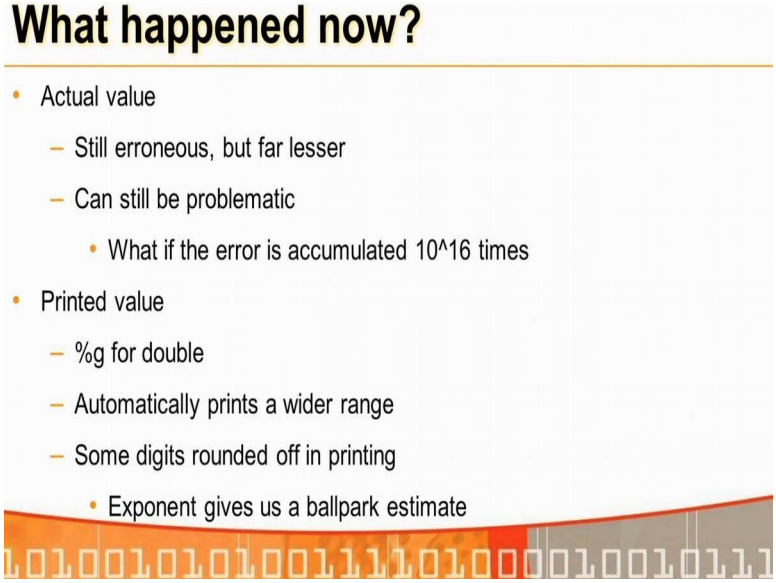
The very first subtraction and addition has no effect on a right if I go for 10 power minus 8. So, you can try the experiments with a starting at 0 subtract and bring and you will get lot more surprises here right are you able to follow this is very interesting.

(Refer Slide Time: 34:17)



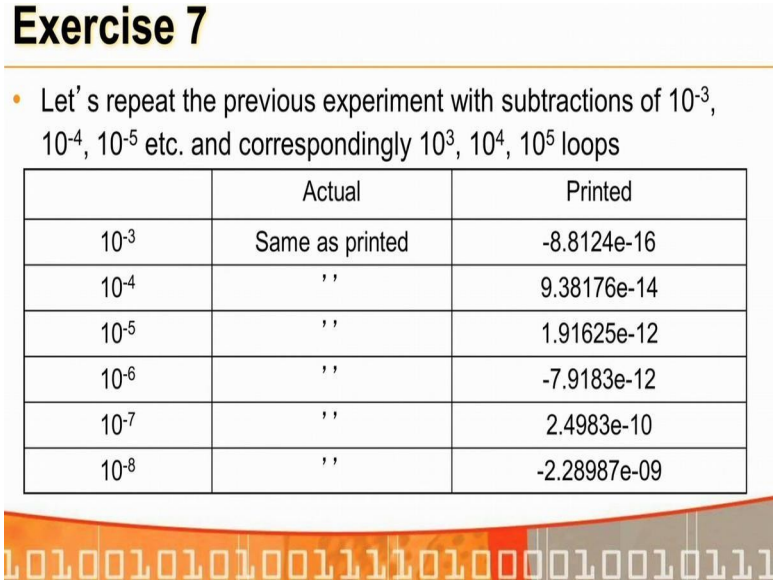
So, let us go and one more little let us go for and redo with this double then actually we will see actually you will see that the actual value is very very less very close to 0, right, it is some 10 power minus 16 right where in the previous case it was 10 power minus 5 or 6 still the answer is ironies, but the error is far lesser, can still be problematic if the error is accumulated 10 power 16 times then it actually becomes 1.

(Refer Slide Time: 34:30)



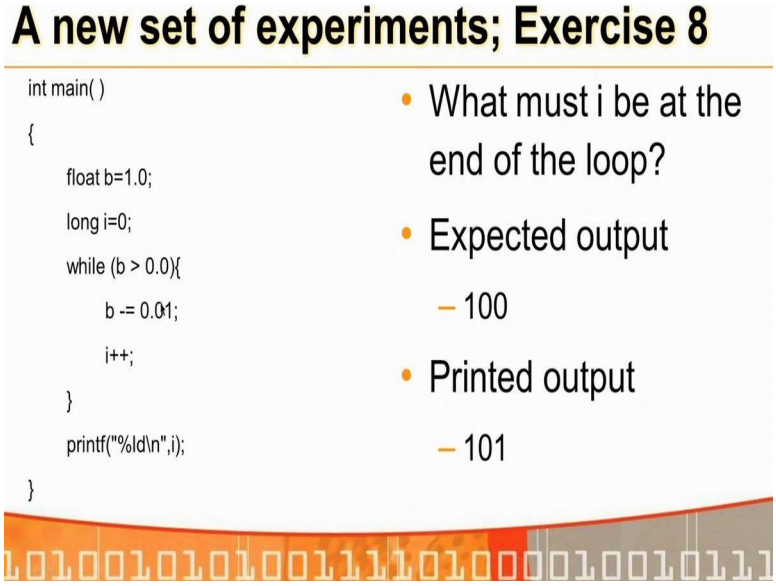
So, and when we print we also used percentage d here if you carefully see here we have used percentage d g, which is the a double printing right. So, it actually prints a wider range right in contrast to percentage f, right.

(Refer Slide Time: 35:04)



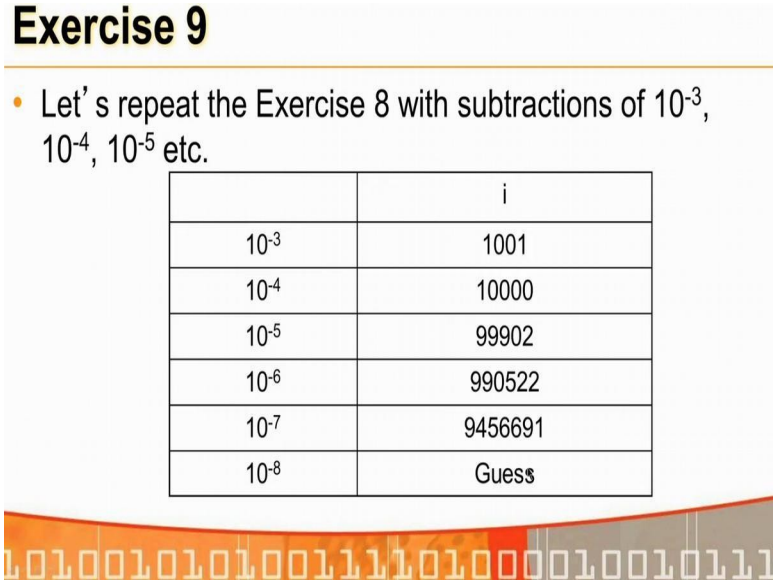
So, this is what we see when as 10 power minus 3 in double and note that here also my error was minus 16 when I started with 10 power minus 3, but if we start decrementing 1 with 10 power minus 8 am getting an error which is 10 power minus 9, but again here the error as I keep decrementing I as a become more and more accurate in my computation my error actually increases exponentially. This 10 power minus 16 this is hundred times that, this is again hundred times that this is again and then it becomes hundred times and 10 times. So, it starts you see some exponential growth here.

(Refer Slide Time: 35:50)



So the last one. So, we will do a new set of experiments like you know we start with 0 and do this what was I be at the end of this loop expected output is 100 right we start with 1 while b is greater than 0.0 do b equal to b minus 0.01 I plus plus. So, the value value of I should be a 100 where the value of I you get would be 101 right because 101 times I need to subtract actually make it less than 0.

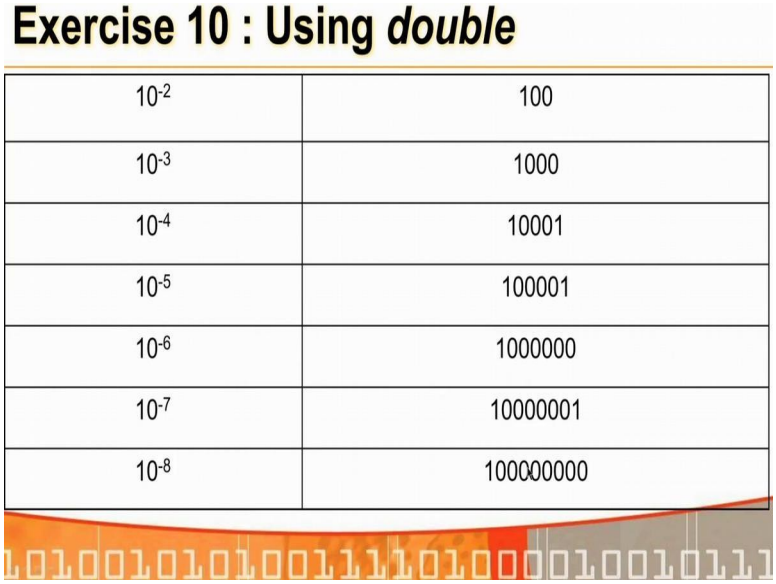
(Refer Slide Time: 36:26)



If you repeat this with 10 power minus 3, 10 power minus 4 and you will actually find that is 1001, 1000 10000 and so on this is not going to be one million it is less than a

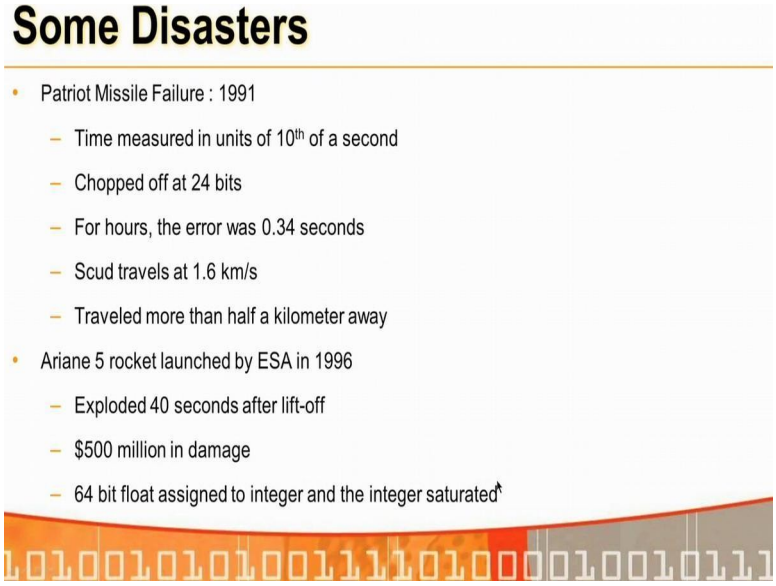
million here sorry less than a lakh here, very quickly it becomes 0 this is less than a million here 10 million, 10 power minus 8 what would be it will be infinite look right similar question.

(Refer Slide Time: 36:51)



Using double precision now I get much closer, right.

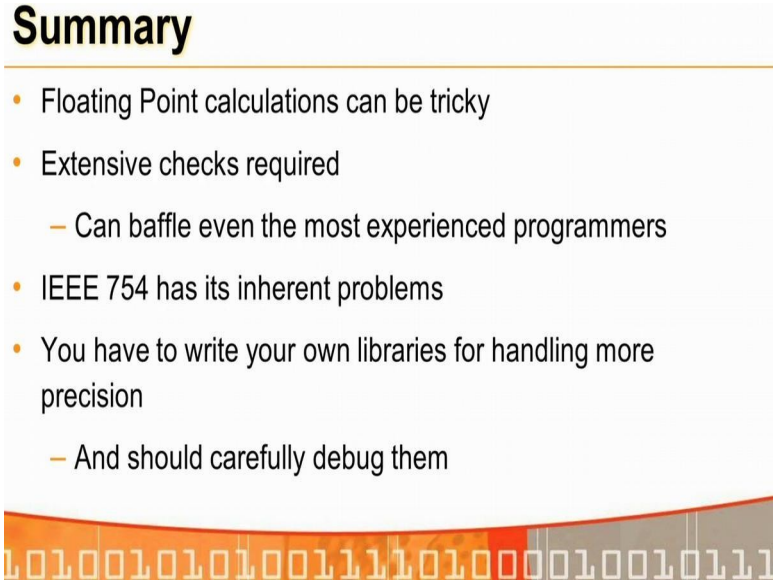
(Refer Slide Time: 36:57)



So, because of the floating points we had all these disasters because of these type of computations, the patriot missile actually became non unpatriotic failed at 1991 there was the time measured was in units of tenth of a second and it well chopped off at 24

bits. So, for hours the error was 0.34 seconds the scud travels at 1.6 kilometres. So, it travelled more than half a kilometre away. So, I want to go and keep something here it went something later and Ariane rocket 5 5 actually launched by ESA in this European space association 1996 exploded 40 seconds after left off because the 64 bit float assigned to integer and the integer saturated; this also 500 dollar 500 million dollar damage. So, that is why we call these types of error as a million dollar error.

(Refer Slide Time: 36:56)



So, floating point calculations are tricky you can you need to do extensive checks, I triple E 754 also has inherent problems as we have seen this. So, you have to write your own libraries when you do floating point you should be extremely careful in this experimentations.

Thank you very much.